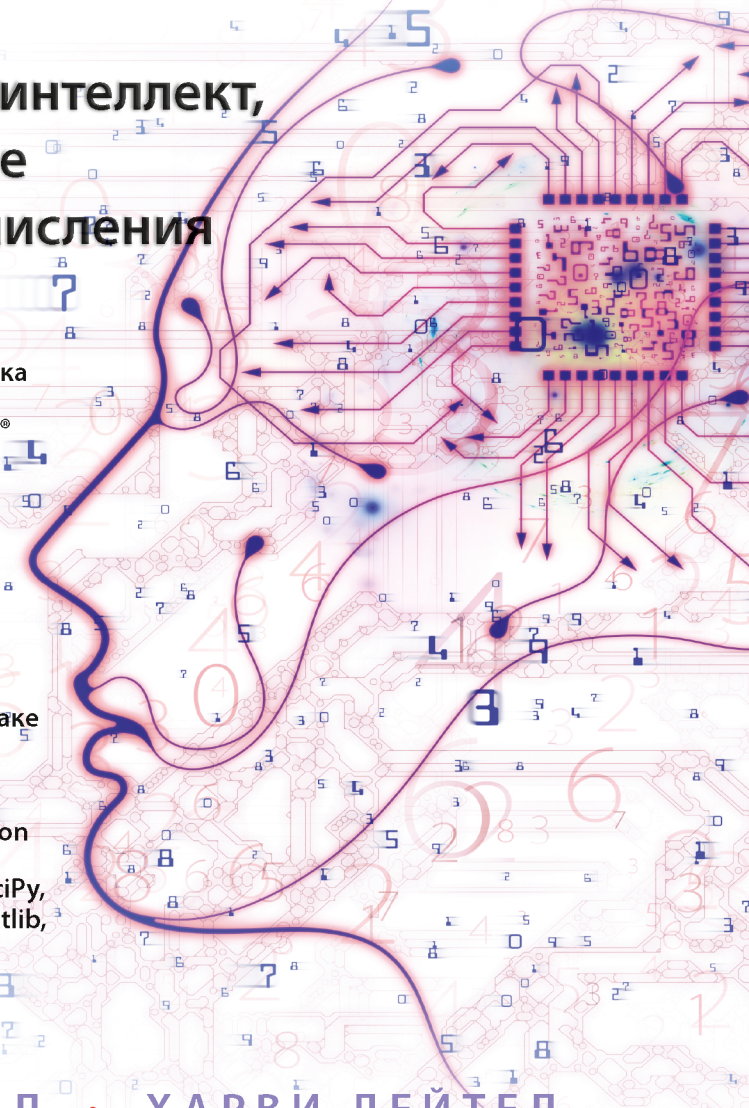
DEITEL® DEVELOPER SERIES

python™

Python®

Искусственный интеллект, большие данные и облачные вычисления

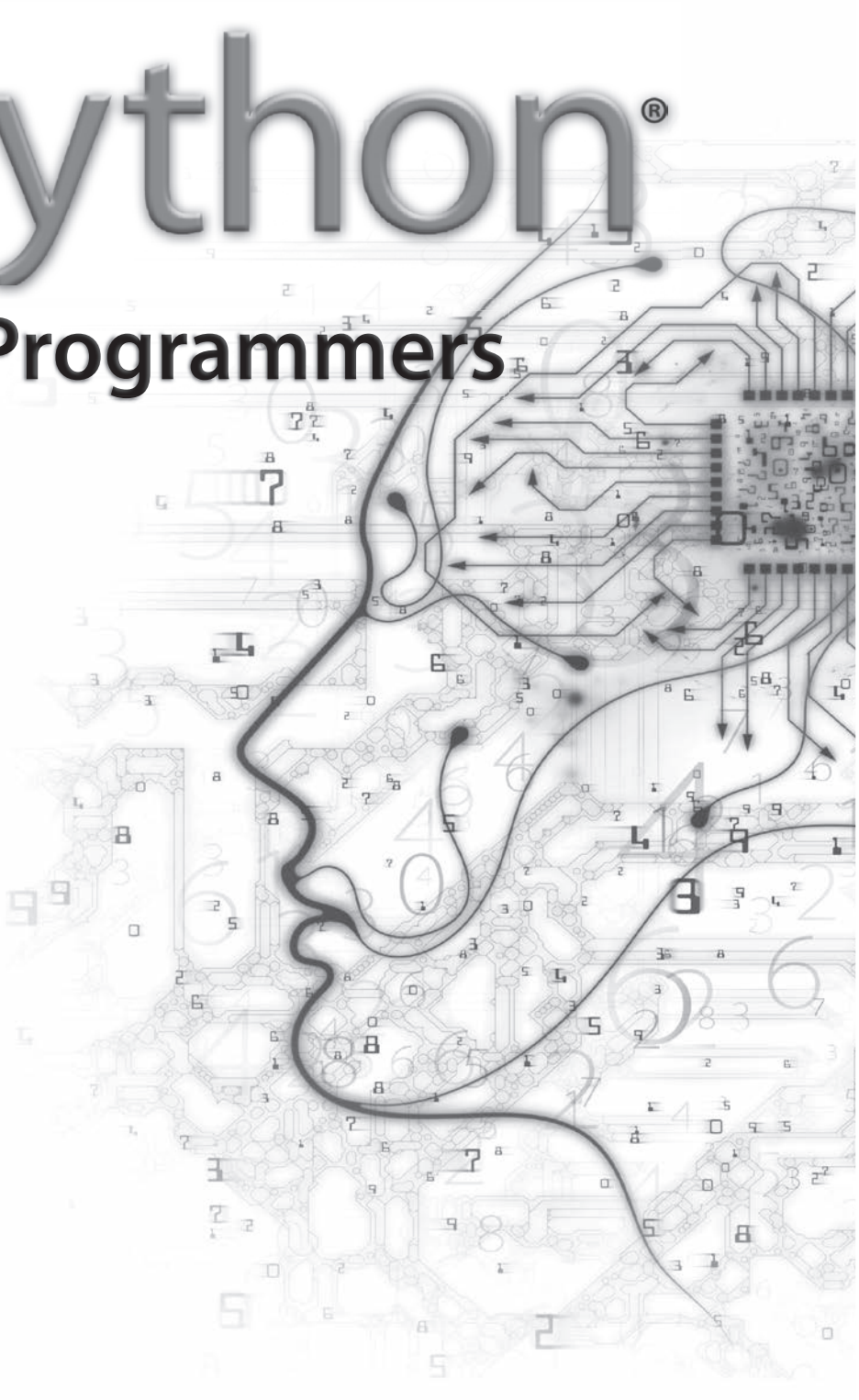


- Обработка естественного языка
- Извлечение данных из Twitter®
- IBM® Watson™
- Машинное обучение с scikit-learn®
- Глубокое обучение с Keras
- Большие данные с Hadoop®, Spark ™, NoSQL и работа в облаке
- Интернет вещей (IoT)
- Стандартная библиотека Python
- Библиотеки NumPy, Pandas, SciPy, NLTK, TextBlob, Tweepy, Matplotlib, Seaborn, Folium и др.

ПОЛ ДЕЙТЕЛ • ХАРВИ ДЕЙТЕЛ

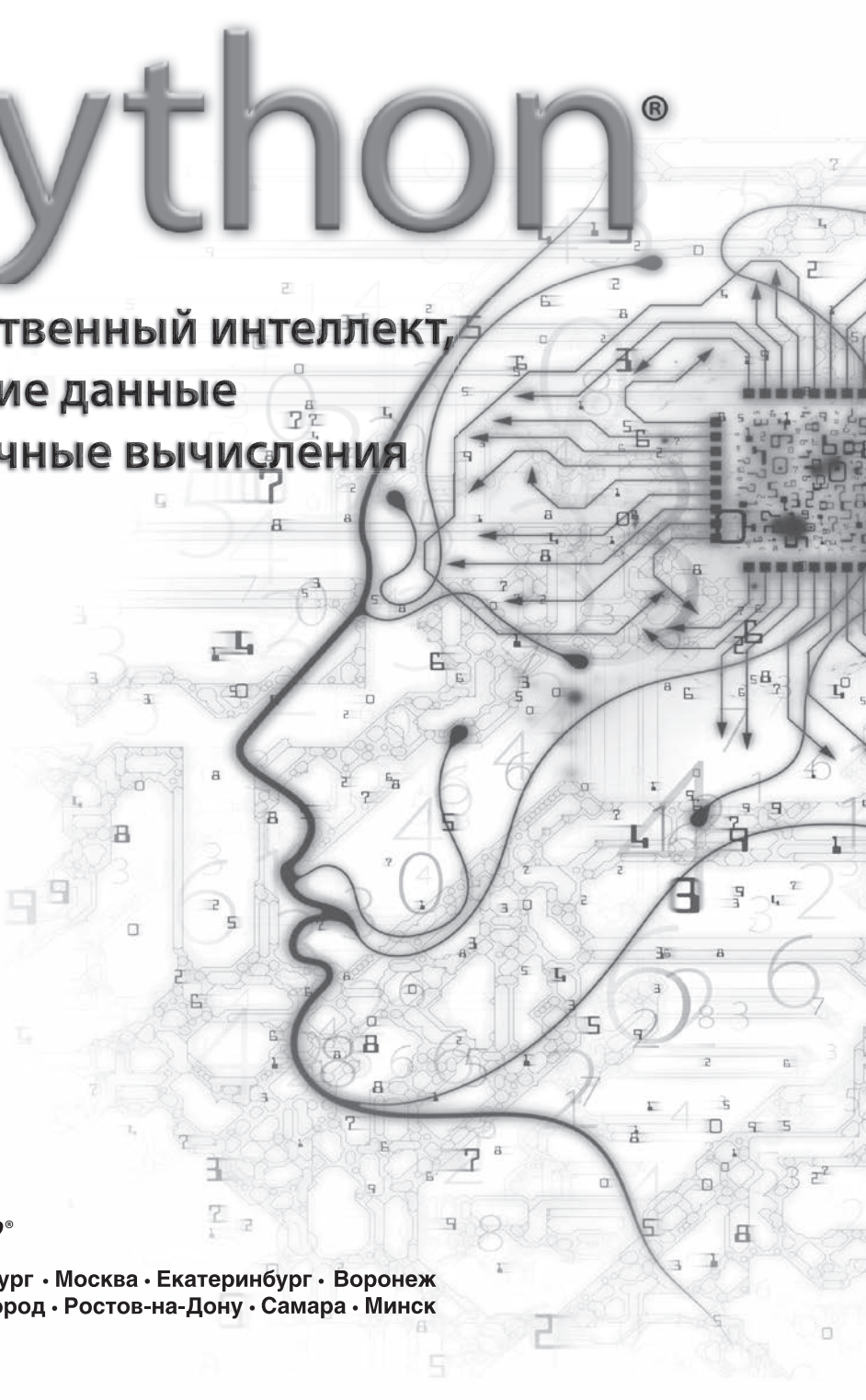
Python®

for Programmers



Python®

Искусственный интеллект, большие данные и облачные вычисления



ПИТЕР®

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону • Самара • Минск

2020

ББК 32.973.2-018.1
УДК 004.43
Д27

Дейтел Пол, Дейтел Харви

Д27 Python: Искусственный интеллект, большие данные и облачные вычисления. — СПб.: Питер, 2020. — 864 с.: ил. — (Серия «Для профессионалов»).

ISBN 978-5-4461-1432-0

Пол и Харви Дейтелы предлагают по-новому взглянуть на Python и использовать уникальный подход, чтобы быстро решить проблемы, стоящие перед современными айтишниками. Вы на практике познакомитесь с революционными вычислительными технологиями и программированием на Python — одном из самых популярных языков.

В вашем распоряжении более пятисот реальных задач — от фрагментов до 40 больших сценариев и примеров с полноценной реализацией. IPython с Jupyter Noteboos позволят быстро освоить современные идиомы программирования Python. Главы 1–5 и фрагменты глав 6–7 сделают понятными примеры решения задач искусственного интеллекта из глав 11–16. Вы познакомитесь с обработкой естественного языка, анализом эмоций в Twitter®, когнитивными вычислениями IBM® Watson™, машинным обучением с учителем в задачах классификации и регрессии, машинным обучением без учителя в задачах кластеризации, распознавания образов с глубоким обучением и сверточными нейронными сетями, рекуррентными нейронными сетями, большими данными с Hadoop®, Spark™ и NoSQL, IoT и многим другим. Вы поработаете (напрямую или косвенно) с облачными сервисами, включая Twitter, Google Translate™, IBM Watson, Microsoft® Azure®, OpenMapQuest, PubNub и др.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.1
УДК 004.43

ISBN 978-0135224335 англ. Authorized translation from the English language edition, entitled PYTHON FOR PROGRAMMERS, 1st Edition by PAUL DEITEL; HARVEY DEITEL, published by Pearson Education, Inc, publishing as Prentice Hall,

ISBN 978-5-4461-1432-0

Краткое содержание

Оглавление

От издательства

Некоторые иллюстрации для лучшего восприятия нужно смотреть в цветном варианте. Мы снабдили их QR-кодами, перейдя по которым, вы можете ознакомиться с цветной версией рисунка.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция). Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

В память о Марвине Мински, отце-основателе искусственного интеллекта.

Мне выпала честь учиться у вас на двух учебных курсах по искусственному интеллекту в Массачусетском технологическом институте.

Вы вдохновляли ваших студентов мыслить, выходя за рамки традиционных представлений.

Харви Дейтел

Предисловие

«Там золото в этих холмах!»[1]

Перед вами книга «Python: Искусственный интеллект, большие данные и облачные вычисления». В ней вы займетесь практическим освоением самых интересных, самых революционных вычислительных технологий, а также программированием на Python — одном из самых популярных языков программирования в мире, лидирующем по темпам развития.

Python обычно сразу приходится по нраву разработчикам. Они ценят Python за выразительность, удобочитаемость, компактность и интерактивную природу. Разработчикам нравится мир разработки с открытым кодом, который порождает стремительно растущую базу программного обеспечения для невероятно широкого спектра прикладных областей.

Уже много десятилетий в мире действуют определенные тенденции. Компьютерное оборудование становится быстрее, дешевле и компактнее. Скорость доступа к интернету растет и дешевеет. Качественное программное обеспечение становится более массовым и практически бесплатным (или почти бесплатным) благодаря движению «открытого кода». Вскоре «интернет вещей» объединит десятки миллиардов устройств любых видов, которые только можно представить. Они порождают колоссальные количества данных на быстро растущих скоростях и объемах.

---

[1] Источник неизвестен, часто ошибочно приписывается Марку Твену.

В современных вычислениях большинство последних новшеств связано с данными — data science, аналитика *данных*, большие *данные*, реляционные базы *данных* (SQL), базы *данных* NoSQL и NewSQL... Все эти темы будут рассматриваться в книге в сочетании с инновационным подходом к программированию на Python.

Спрос на квалификацию в области data science

В 2011 году Глобальный институт McKinsey опубликовал отчет «Большие данные: новый рубеж для инноваций, конкуренции и производительности». В отчете было сказано: «Только Соединенные Штаты сталкиваются с нехваткой от 140 тысяч до 190 тысяч специалистов, обладающих глубокими аналитическими познаниями, а также 1,5 миллиона менеджеров и аналитиков, которые бы анализировали большие данные и принимали решения на основании полученных результатов»[1]. Такое положение дел сохраняется. В отчете за август 2018 года «LinkedIn Workforce Report» сказано, что в Соединенных Штатах существует нехватка более 150 тысяч специалистов в области data science[2]. В отчете IBM, Burning Glass Technologies и Business-Higher Education Forum за 2017 года говорится, что к 2020 году в Соединенных Штатах будут существовать сотни тысяч вакансий, требующих квалификации в области data science[3].

Модульная структура

Модульная структура книги обеспечивает потребности разных профессиональных аудиторий.

Главы 1–10 посвящены программированию на языке Python. Каждая из этих глав содержит краткий раздел «Введение в data science»; в этих разделах будут представлены такие темы, как искусственный интеллект, основные характеристики описательной статистики, метрики, характеризующие положение центра распределения и разброс, моделирование, статические и динамические визуализации, работа с файлами CSV, применение Pandas для исследования и первичной обработки данных, временные ряды и простая линейная регрес-

---

[1] https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/Big%20data%20The%20next%20frontier%20for%20innovation/MGI_big_data_full_report.ashx (с. 3).

[2] https://economicgraph.linkedin.com/resources/linkedin-workforce-report-august-2018.

[3] https://www.burning-glass.com/wp-content/uploads/The_Quant_Crunch.pdf (с. 3).

сия. Эти разделы подготовят вас к изучению data science, искусственного интеллекта, больших данных и облачных технологий в главах 11–16, в которых вам представится возможность применить реальные наборы данных в полноценных практических примерах.

После описания Python в главах 1–5 и некоторых ключевых частей глав 6–7 вашей подготовки будет достаточно для основных частей практических примеров в главах 11–16. Раздел «Зависимость между главами» данного предисловия поможет преподавателям спланировать свои профессиональные курсы в контексте уникальной архитектуры книги.

Главы 11–16 переполнены занимательными, современными примерами. В них представлены практические реализации по таким темам, как обработка естественного языка, глубокий анализ данных Twitter, когнитивные вычисления на базе IBM Watson, машинное обучение с учителем для решения задач классификации и регрессии, машинное обучение без учителя для решения задач кластеризации, глубокое обучение на базе сверточных нейронных сетей, глубокое обучение на базе рекуррентных нейронных сетей, большие данные с Hadoop, Spark и баз данных NoSQL, «интернет вещей» и многое другое. Попутно вы освоите широкий спектр терминов и концепций data science, от кратких определений до применения концепций в малых, средних и больших программах. Подробное оглавление книги даст вам представление о широте изложения.

Ключевые особенности

- **Простота**: в каждом аспекте книги мы ставили на первое место простоту и ясность. Например, для обработки естественного языка мы используем простую и интуитивную библиотеку TextBlob вместо более сложной библиотеки NLTK. При описании глубокого обучения мы отдали предпочтение Keras перед TensorFlow. Как правило, если для решения какой-либо задачи можно было воспользоваться несколькими разными библиотеками, мы выбирали самый простой вариант.
- **Компактность**: большинство из 538 примеров этой книги невелики — они состоят всего из нескольких строк кода с немедленным интерактивным откликом от IPython. Также в книгу включены 40 больших сценариев и подробных практических примеров.
- **Актуальность**: мы прочитали множество книг о программировании Python и data science; просмотрели или прочитали около 15 000 статей, исследовательских работ, информационных документов, видеороликов, публикаций в блогах, сообщений на форумах и документов. Это позво-

лило нам «держать руку на пульсе» сообществ Python, компьютерных технологий, data science, AI, больших данных и облачных технологий.

Быстрый отклик: исследования и эксперименты с IPython

- Если вы хотите учиться по этой книге, лучше всего читать текст и параллельно выполнять примеры кода. В этой книге используется *интерпретатор IPython*, который предоставляет удобный интерактивный режим с немедленным откликом для быстрых исследований и экспериментов с Python и обширным набором библиотек.
- Большая часть кода представлена в виде небольших интерактивных сеансов IPython. IPython немедленно читает каждый фрагмент кода, написанный вами, обрабатывает его и выводит результаты. Мгновенная обратная связь помогает сосредоточиться, повышает эффективность обучения, способствует быстрой прототипизации и ускоряет процесс разработки.
- В наших книгах на первый план всегда выходит живой код и ориентация на полноценные работоспособные программы с реальным вводом и выводом. «Волшебство» IPython как раз и заключается в том, что он превращает фрагменты в код, который «оживает» с каждой введенной строкой. Такой результат повышает эффективность обучения и поощряет эксперименты.

Основы программирования на Python

- Прежде всего в книге достаточно глубоко и подробно излагаются основы Python.
- В ней рассматриваются модели программирования на языке Python — процедурное программирование, программирование в функциональном стиле и объектно-ориентированное программирование.
- Мы стараемся наглядно выделять текущие идиомы.
- Программирование в функциональном стиле используется везде, где это уместно. На диаграмме в главе 4 перечислены ключевые средства программирования в функциональном стиле языка Python с указанием глав, в которых они впервые рассматриваются.

538 примеров кода

- Увлекательное, хотя и непростое введение в Python подкрепляется 538 реальными примерами — от небольших фрагментов до основательных прак-

тических примеров из области компьютерной теории, data science, искусственного интеллекта и больших данных.

- Мы займемся нетривиальными задачами из области искусственного интеллекта, больших данных и облачных технологий, такими как обработка естественного языка, глубокий анализ данных Twitter, машинное обучение, глубокое обучение, Hadoop, MapReduce, Spark, IBM Watson, ключевые библиотеки data science (NumPy, pandas, SciPy, NLTK, TextBlob, spaCy, Textatistic, Tweepy, Scikit-learn, Keras), ключевые библиотеки визуализации (Matplotlib, Seaborn, Folium) и т. д.

Объяснения вместо математических выкладок

- Мы стараемся сформулировать концептуальную сущность математических вычислений и использовать ее в своих примерах. Для этого применяются такие библиотеки, как statistics, NumPy, SciPy, pandas и многие другие, скрывающие математические сложности от пользователя. Таким образом, вы сможете пользоваться такими математическими методами, как линейная регрессия, даже не владея математической теорией, на которой они базируются. В примерах машинного обучения мы стараемся создавать объекты, которые выполнят все вычисления за вас.

Визуализации

- 67 статических, динамических, анимированных и интерактивных визуализаций (диаграмм, графиков, иллюстраций, анимаций и т. д.) помогут вам лучше понять концепции.
- Вместо того чтобы подолгу объяснять низкоуровневое графическое программирование, мы сосредоточимся на высокоуровневых визуализациях, построенных средствами Matplotlib, Seaborn, pandas и Folium (для интерактивных карт).
- Визуализации используются как учебный инструмент. Например, закон больших чисел наглядно демонстрируется динамической моделью бросков кубиков и построением гистограммы. С увеличением количества бросков процент выпадений каждой грани постепенно приближается к 16,667% (1/6), а размеры столбцов, представляющих эти проценты, постепенно выравниваются.
- Визуализации чрезвычайно важны при работе с большими данными: они упрощают исследование данных и получение воспроизводимых результа-

тов исследований, когда количество элементов данных может достигать миллионов, миллиардов и более. Часто говорят, что одна картинка стоит тысячи слов[1] — в мире больших данных визуализация может стоить миллиардов, триллионов и даже более записей в базе данных. Визуализации позволяют взглянуть на данные «с высоты птичьего полета», увидеть их «в перспективе» и составить о них представление. *Описательные статистики* полезны, но иногда могут увести в ошибочном направлении. Например, квартет Энскомба[2] демонстрирует посредством визуализаций, что *серьезно различающиеся* наборы данных могут иметь *почти одинаковые* показатели описательной статистики.

- Мы приводим код визуализаций и анимаций, чтобы вы могли реализовать собственные решения. Также анимации предоставляются в виде файлов с исходным кодом и документов Jupyter Notebook, чтобы вам было удобно настраивать код и параметры анимаций, заново выполнить анимации и понаблюдать за эффектом изменений.

Опыт работы с данными

- В разделах «Введение в data science» и практических примерах из глав 11–16 вы получите полезный опыт работы с данными.
- Мы будем работать со многими реальными базами данных и источниками данных. В интернете существует огромное количество бесплатных открытых наборов данных, с которыми вы можете экспериментировать. На некоторых сайтах, упомянутых нами, приводятся ссылки на сотни и тысячи наборов данных.
- Многие библиотеки, которыми вы будете пользоваться, включают популярные наборы данных для экспериментов.
- В книге мы рассмотрим действия, необходимые для получения данных и подготовки их к анализу, анализ этих данных различными средствами, настройки моделей и эффективных средств передачи результатов, особенно посредством визуализации.

GitHub

- GitHub — превосходный ресурс для поиска открытого кода, который вы сможете интегрировать в свои проекты (а также поделиться своим кодом

---

[1] https://en.wikipedia.org/wiki/A_picture_is_worth_a_thousand_words.

[2] https://ru.wikipedia.org/wiki/Квартет_Энскомба.

с сообществом). Также GitHub является важнейшим элементом арсенала разработчика с функциональностью контроля версий, которая помогает командам разработчиков управлять проектами с открытым (и закрытым) кодом.

- Мы будем использовать множество разнообразных библиотек Python и data science, распространяемых с открытым кодом, а также программных продуктов и облачных сервисов — бесплатных, имеющих пробный период и условно-бесплатных. Многие библиотеки размещаются на GitHub.

Практические облачные вычисления

- Большая часть аналитики больших данных выполняется в облачных средах, позволяющих легко динамически масштабировать объем аппаратных и программных ресурсов, необходимых вашему приложению. Мы будем работать с различными облачными сервисами (напрямую или опосредованно), включая Twitter, Google Translate, IBM Watson, Microsoft Azure, OpenMapQuest, geopy, Dweet.io и PubNub.
- Мы рекомендуем пользоваться бесплатными, имеющими пробный период или условно-бесплатными сервисами. Предпочтение отдается тем сервисам, которые не требуют ввода данных кредитной карты, чтобы избежать получения больших счетов. Если же вы решили использовать сервис, требующий ввода данных кредитной карты, убедитесь в том, что с выбранного вами бесплатного уровня не происходит автоматический переход на платный уровень.

Базы данных, большие данные и инфраструктура больших данных

- По данным IBM (ноябрь 2016 года), 90% мировых данных было создано за последние два года[1]. Факты показывают, что скорость создания данных стремительно растет.
- По данным статьи AnalyticsWeek за март 2016 года, в течение 5 лет к интернету будет подключено более 50 миллиардов устройств, а к 2020 году в мире будет ежесекундно производиться 1,7 мегабайт новых данных на каждого человека[2]!

---

[1] https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf.

[2] https://analyticsweek.com/content/big-data-facts/.

- В книге рассматриваются основы работы с реляционными базами данных и использования SQL с SQLite.
- Базы данных — критический элемент инфраструктуры больших данных для хранения и обработки больших объемов информации. Реляционные базы данных предназначены для обработки структурированных данных — они не приспособлены для неструктурированных и полуструктурированных данных в приложениях больших данных. По этой причине с развитием больших данных были созданы базы данных NoSQL и NewSQL для эффективной работы с такими данными. Мы приводим обзор NoSQL и NewSQL, а также практический пример работы с документной базой данных MongoDB в формате JSON. MongoDB — самая популярная база данных NoSQL.
- Оборудование и программная инфраструктура больших данных рассматриваются в главе 16.

Практические примеры из области искусственного интеллекта

- В практических примерах глав 11–15 представлены темы искусственного интеллекта, включая обработку естественного языка, глубокий анализ данных Twitter для анализа эмоциональной окраски, когнитивные вычисления на базе IBM Watson, машинное обучение с учителем, машинное обучение без учителя и глубокое обучение. В главе 16 представлено оборудование больших данных и программная инфраструктура, которые позволяют специалистам по компьютерным технологиям и теоретикам data science реализовать ультрасовременные решения на базе искусственного интеллекта.

Встроенные коллекции: списки, кортежи, множества, словари

- Как правило, в наши дни *самостоятельная* реализация структур данных не имеет особого смысла. В книге приведено подробное, состоящее из двух глав описание встроенных структур данных Python — списков, кортежей, словарей и множеств, успешно решающих большинство задач структурирования данных.

Программирование с использованием массивов NumPy и коллекций pandas Series/DataFrame

- Мы также уделили особое внимание трем ключевым структурам данных из библиотек с открытым кодом: массивам NumPy, коллекциям pandas `Series` и pandas `DataFrame`. Эти коллекции находят широкое применение

в data science, компьютерной теории, искусственном интеллекте и больших данных. NumPy обеспечивает эффективность на два порядка выше, чем у встроенных списков Python.

- В главу 7 включено подробное описание массивов NumPy. Многие библиотеки, например pandas, построены на базе NumPy. В разделах «Введение в data science» в главах 7–9 представлены коллекции pandas `Series` и `DataFrame`, которые хорошо работают в сочетании с массивами NumPy, а также используются в оставшихся главах.

Работа с файлами и сериализация

- В главе 9 рассказывается об обработке текстовых файлов, а затем показано, как сериализовать объекты в популярном формате JSON (JavaScript Object Notation). JSON часто используется в части, посвященной data science.
- Многие библиотеки data science предоставляют встроенные средства для работы с файлами и загрузки наборов данных в программы Python. Кроме простых текстовых файлов, мы также займемся обработкой файлов в популярном формате CSV (значения, разделенные запятыми) с использованием модуля `csv` стандартной библиотеки Python и средств библиотеки data science pandas.

Объектно-базированное программирование

- Мы стараемся использовать многочисленные классы, упакованные сообществом разработки с открытым кодом Python в библиотеки классов. Прежде всего мы разберемся в том, какие библиотеки существуют, как выбрать библиотеки, подходящие для ваших приложений, как создать объекты существующих классов (обычно в одной-двух строках кода) и пустить их в дело. Объектно-базированный стиль программирования позволяет быстро и компактно строить впечатляющие приложения, что является одной из важных причин популярности Python.
- Этот подход позволит вам применять машинное обучение, глубокое обучение и другие технологии искусственного интеллекта для быстрого решения широкого спектра интересных задач, включая такие задачи когнитивных вычислений, как распознавание речи и «компьютерное зрение».

Объектно-ориентированное программирование

- Разработка собственных классов — важнейшая составляющая объектно-ориентированного программирования наряду с наследованием, полимор-

физмом и утиной типизацией. Эти составляющие рассматриваются в главе 10.

- В главе 10 рассматривается модульное тестирование с использованием doctest и интересного моделирования процесса тасования и раздачи карт.
- Для целей глав 11–16 хватает нескольких простых определений пользовательских классов. Вероятно, в коде Python вы будете применять объектно-базированное программирование чаще, чем полноценное объектно-ориентированное программирование.

Воспроизводимость результатов

- В науке вообще и в data science в частности существует потребность в воспроизведении результатов экспериментов и исследований, а также эффективном распространении этих результатов. Для этого обычно рекомендуется применять документы Jupyter Notebook.
- Воспроизводимость результатов рассматривается в книге в контексте методов программирования и программных средств, таких как документы Jupyter Notebook и Docker.

Эффективность

- В нескольких примерах используется средство профилирования `%timeit` для сравнения эффективности разных подходов к решению одной задачи. Также рассматриваются такие средства, относящиеся к эффективности, как выражения-генераторы, сравнение массивов NumPy со списками Python, эффективность моделей машинного обучения и глубокого обучения, эффективность распределенных вычислений Hadoop и Spark.

Большие данные и параллелизм

В этой книге вместо написания собственного кода параллелизации мы поручим таким библиотекам, как Keras на базе TensorFlow, и таким инструментам больших данных, как Hadoop и Spark, провести параллелизацию за вас. В эру больших данных/искусственного интеллекта колоссальные требования к вычислительным мощностям приложений, работающих с большими массивами данных, заставляют нас задействовать полноценный параллелизм, обеспечиваемый многоядерными процессорами, графическими процессорами (GPU), тензорными процессорами (TPU) и гигантскими компьютерными *кластерами* в облаке. Некоторые задачи больших данных могли требовать параллельной работы тысяч процессоров для быстрого анализа огромных объемов данных.

Зависимость между главами

Допустим, вы — преподаватель, составляющий план лекций для профессиональных учебных курсов, или разработчик, решающий, какие главы следует читать в первую очередь. Тогда этот раздел поможет вам принять оптимальные решения. Главы лучше всего читать (или использовать для обучения) по порядку. Тем не менее для большей части материала разделов «Введение в data science» в конце глав 1–10 и практических примеров в главах 11–16 необходимы только главы 1–5 и небольшие части глав 6–10.

Часть 1: Основы Python

Мы рекомендуем читать все главы по порядку:

- В главе 1 «Компьютеры и Python» представлены концепции, которые закладывают фундамент для программирования на языке Python в главах 2–10 и практических примеров больших данных, искусственного интеллекта и облачных сервисов в главах 11–16. В главе также приведены результаты пробных запусков интерпретатора IPython и документов Jupyter Notebook.
- В главе 2 «Введение в программирование Python» изложены основы программирования Python с примерами кода, демонстрирующими ключевые возможности языка.
- В главе 3 «Управляющие команды» представлены *управляющие команды* Python и простейшие возможности обработки списков.
- В главе 4 «Функции» представлены пользовательские функции, методы моделирования с генерированием случайных чисел и основы *работы с кортежами*.
- В главе 5 «Последовательности: списки и кортежи» встроенные списки и кортежи Python описаны более подробно. Также в ней начинается изложение азов *программирования в функциональном стиле*.

Часть 2: Структуры данных Python, строки и файлы

Ниже приведена сводка зависимостей между главами для глав 6–9; предполагается, что вы уже прочитали главы 1–5.

- Глава 6 «Словари и множества» — раздел «6.4. Введение в data science» этой главы не зависит от материала главы.

- Глава 7 «NumPy и программирование, ориентированное на массивы» — для раздела «7.14. Введение в data science» необходимо знание словарей (глава 6) и массивов (глава 7).
- Глава 8 «Подробнее о строках» — для раздела «8.13. Введение в data science» необходимо знание необработанных строк и регулярных выражений (разделы 8.11–8.12), а также коллекций pandas `Series` и `DataFrame` из раздела 7.14.
- Глава 9 «Файлы и исключения» — для изучения сериализации JSON полезно знать основы работы со словарями (раздел 6.2). Кроме того, раздел «9.12. Введение в data science» требует знания встроенной функции `open` и команды `with` (раздел 9.3), а также коллекций pandas `Series` и `DataFrame` из раздела 7.14.

Часть 3: Нетривиальные аспекты Python

Ниже приведена сводка зависимостей между главами для глав 10; предполагается, что вы уже прочитали главы 1–5.

- Глава 10 «Объектно-ориентированное программирование» — раздел «Введение в data science» требует знания возможностей `DataFrame` из раздела 7.14. Преподаватели, которые намерены ограничиваться рассмотрением только классов и объектов, могут изложить материал разделов 10.1–10.6. Для преподавателей, которые собираются изложить более сложные темы (наследование, полиморфизм, утиная типизация), могут представлять интерес разделы 10.7–10.9. В разделах 10.10–10.15 изложены дополнительные перспективы.

Часть 4: Искусственный интеллект, облачные технологии и практические примеры больших данных

Ниже приведена сводка зависимостей между главами для глав 11–16; предполагается, что вы уже прочитали главы 1–5. Большинство глав 11–16 также требует знания словарей из раздела 6.2.

- В главе 11 «Обработка естественного языка» используются возможности pandas `DataFrame` из раздела 7.14.
- В главе 12 «Глубокий анализ данных Twitter» используются возможности pandas `DataFrame` из раздела 7.14, метод строк `join` (раздел 8.9), основы работы с JSON (раздел 9.5), `TextBlob` (раздел 11.2) и словарные облака

(раздел 11.3). Некоторые примеры требуют определения классов с наследованием (глава 10).

- В главе 13 «IBM Watson и когнитивные вычисления» используется встроенная функция `open` и команда `with` (раздел 9.3).
- В главе 14 «Машинное обучение: классификация, регрессия и кластеризация» используются основные средства работы с массивами NumPy и метод `unique` (глава 7), возможности pandas `DataFrame` из раздела 7.14, а также функция `subplots` библиотеки Matplotlib (раздел 10.6).
- В главе 15 «Глубокое обучение» используются основные средства работы с массивами NumPy (глава 7), метод строк `join` (раздел 8.9), общие концепции машинного обучения из главы 14 и функциональность из практического примера главы 14 «Классификация методом k ближайших соседей и набор данных Digits».
- В главе 16 «Большие данные: Hadoop, Spark, NoSQL и IoT» используется метод строк `split` (раздел 6.2.7), объект Matplotlib `FuncAnimation` из раздела 6.4, коллекции pandas `Series` и `DataFrame` из раздела 7.14, метод строк `join` (раздел 8.9), модуль JSON (раздел 9.5), игнорируемые слова NLTK (раздел 11.2.13), аутентификация Twitter из главы 12, класс Tweepy `StreamListener` для потоковой передачи твитов, а также библиотеки geopy и folium. Некоторые примеры требуют определения классов с применением наследования (глава 10), но вы можете просто повторить наши определения классов без чтения главы 10.

Документы Jupyter Notebook

Для вашего удобства мы предоставили примеры кода книги в файлах с исходным кодом Python (`.py`) для использования с интерпретатором командной строки IPython, а также файлы Jupyter Notebook (`.ipynb`), которые можно загрузить в браузере и выполнить.

Jupyter Notebook — бесплатный проект с открытым кодом, который позволяет объединять текст, графику, аудио, видео и функциональность интерактивного программирования для быстрого и удобного ввода, редактирования, выполнения, отладки и изменения кода в браузере. Фрагмент статьи «Что такое Jupyter?»:

> *«Jupyter стал фактическим стандартом для научных исследований и анализа данных. Вычисления упаковываются вместе с аргументами, позволяя вам строить "вычислительные нарративы"; ...это упрощает проблему*

распространения работоспособного кода между коллегами и участниками сообщества»[1].

Наш опыт показывает, что эта среда прекрасно подходит для обучения и быстрой прототипизации. По этой причине мы используем документы Jupyter Notebook вместо традиционных интегрированных сред (IDE), таких как Eclipse, Visual Studio, PyCharm или Spyder. Ученые и специалисты уже широко применяют Jupyter для распространения результатов своих исследований. Поддержка Jupyter Notebook предоставляется через традиционные механизмы сообщества с открытым кодом[2] (см. раздел «Поддержка Jupyter» в этом предисловии). За подробным описанием установки обращайтесь к разделу «Приступая к работе» после предисловия, а информация о запуске примеров книги приведена в разделе 1.5.

Совместная работа и обмен результатами

Работа в команде и распространение результатов исследований играют важную роль для разработчиков, которые занимают или собираются занять должность, связанную с аналитикой данных, в коммерческих, правительственных или образовательных организациях:

- Созданные вами документы Notebook удобно распространять среди участников команды простым копированием файлов или через GitHub.
- Результаты исследований, включая код и аналитику, могут публиковаться в виде статических веб-страниц при помощи таких инструментов, как nbviewer (https://nbviewer.jupyter.org) и GitHub, — оба ресурса автоматически визуализируют документы Notebook в виде веб-страниц.

Воспроизводимость результатов: веский аргумент в пользу Jupyter Notebook

В области data science и научных дисциплин вообще эксперименты и исследования должны быть воспроизводимыми. Об этом неоднократно упоминалось в литературе:

- Публикация Дональда Кнута «Грамотное программирование» в 1992 году[3].

---

[1] https://www.oreilly.com/ideas/what-is-jupyter.

[2] https://jupyter.org/community.

[3] *Knuth D.* Literate Programming (PDF), The Computer Journal, British Computer Society, 1992.

- Статья «Языково-независимый воспроизводимый анализ данных с применением грамотного программирования»[1], в которой сказано: «Lir-вычисления (грамотные воспроизводимые вычисления) базируются на концепции грамотного программирования, предложенной Дональдом Кнутом».

По сути, воспроизводимость отражает полное состояние среды, использованной для получения результатов: оборудование, программное обеспечение, коммуникации, алгоритмы (особенно код), данные и *родословная* данных (источник и линия происхождения).

Docker

В главе 16 используется Docker — инструмент для упаковки программного кода в контейнеры, содержащие все необходимое для удобного, воспроизводимого и портируемого выполнения этого кода между платформами. Некоторые программные пакеты, используемые в главе 16, требуют сложной подготовки и настройки. Для многих из них можно бесплатно загрузить готовые контейнеры Docker. Это позволяет избежать сложных проблем установки и запускать программные продукты локально на настольном или портативном компьютере. Docker предоставляет идеальную возможность быстро и удобно приступить к использованию новых технологий.

Docker также помогает обеспечить воспроизводимость. Вы можете создавать специализированные контейнеры Docker с нужными версиями всех программных продуктов и всех библиотек, использованных в исследовании. Это позволит другим разработчикам воссоздать использованную вами среду, а затем повторить вашу работу и получить ваши результаты. В главе 16 мы используем Docker для загрузки и выполнения контейнера, заранее настроенного для программирования и запуска Spark-приложений больших данных на базе Jupyter Notebook.

IBM Watson и когнитивные вычисления

На ранней стадии исследований, проводимых для этой книги, мы распознали быстро растущий интерес к IBM Watson. Мы проанализировали предложения конкурентов и обнаружили, что политика Watson «без ввода данных кредит-

[1] http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0164023.

ной карты» для бесплатных уровней является одной из самых удобных для наших читателей.

IBM Watson — платформа когнитивных вычислений, применяемая в широком спектре реальных сценариев. Системы когнитивных вычислений моделируют функции человеческого мозга по выявлению закономерностей и принятию решений для «обучения» с поглощением большего объема данных[1,2,3]. В книге Watson уделяется значительное внимание. Мы используем бесплатный пакет Watson Developer Cloud: Python SDK, который предоставляет различные API для взаимодействия с сервисами Watson на программном уровне. С Watson интересно работать, и эта платформа помогает раскрыть ваш творческий потенциал.

Сервисы Watson уровня Lite и практический пример Watson

Чтобы способствовать обучению и экспериментам, IBM предоставляет бесплатные lite-уровни для многих своих API[4]. В главе 13 будут опробованы демонстрационные приложения для многих сервисов Watson[5]. Затем мы используем lite-уровни сервисов Watson Text to Speech, Speech to Text и Translate для реализации приложения-переводчика. Пользователь произносит вопрос на английском языке, приложение преобразует речь в английский текст, переводит текст на испанский язык и зачитывает испанский текст. Собеседник произносит ответ на испанском языке (если вы не говорите на испанском, мы предоставили аудиофайл, который вы можете использовать). Приложение быстро преобразует речь в испанский текст, переводит текст на английский и зачитывает ответ на английском. Круто!

Подход к обучению

«Python: Искусственный интеллект, большие данные и облачные вычисления» содержит обширную подборку примеров, позаимствованных из многих областей. Мы рассмотрим некоторые интересные примеры с реальными наборами данных. В книге основное внимание уделяется принципам качественного

---

1 http://whatis.techtarget.com/definition/cognitive-computing.

2 https://en.wikipedia.org/wiki/Cognitive_computing.

3 https://www.forbes.com/sites/bernardmarr/2016/03/23/what-everyone-should-know-about-cognitive-computing.

4 Всегда проверяйте последние условия предоставления сервиса на сайте IBM, так как условия и сервисы могут меняться со временем.

5 https://console.bluemix.net/catalog/.

проектирования программных продуктов, а на передний план выходит ясность кода.

538 примеров кода

538 примеров, приведенных в книге, содержат приблизительно 4000 строк кода. Это относительно небольшой объем для книги такого размера, что отчасти объясняется выразительностью языка Python. Кроме того, наш стиль программирования подразумевает, что мы по возможности используем полнофункциональные библиотеки классов; эти библиотеки берут на себя большую часть работы.

160 таблиц/иллюстраций/визуализаций

В книгу включено множество таблиц, графиков, а также статических, динамических и интерактивных визуализаций.

Житейская мудрость программирования

В материал книги *интегрируется* житейская мудрость программирования, основанная на девяти десятилетиях (в сумме) авторского опыта программирования и преподавания.

- *Хороший стиль программирования* и *общепринятые идиомы Python* помогают создавать более понятные, более четкие и простые в сопровождении программы.
- Описание *распространенных ошибок программирования* снижает вероятность того, что эти ошибки будут допущены читателями.
- *Советы по предотвращению ошибок* с рекомендациями по выявлению дефектов и исключению их из программ. Во многих советах описываются приемы, которые препятствуют изначальному проникновению ошибок в ваши программы.
- *Советы по быстродействию*, в которых выделяются возможности для ускорения работы ваших программ или сокращения объема занимаемой памяти.
- *Наблюдения из области программирования*, в которых выделяются архитектурные и проектировочные аспекты правильного построения программных продуктов (особенно для больших систем).

Программные продукты, используемые в книге

Программные продукты, используемые в книге, доступны для Windows, macOS и Linux, и их можно бесплатно загрузить из интернета. Для написания примеров используется бесплатный дистрибутив Anaconda Python. Он включает большую часть Python, библиотек визуализации и data science, которые вам понадобятся, а также интерпретатор IPython, Jupyter Notebook и Spyder — одну из самых лучших интегрированных сред Python для data science. Для разработки программ, приведенных в книге, используется только IPython и Jupyter Notebook. В разделе «Приступая к работе» после предисловия обсуждается установка Anaconda и других продуктов, необходимых для работы с нашими примерами.

Документация Python

Следующая документация особенно пригодится вам во время работы с книгой:

- Справочник по языку Python:

  https://docs.python.org/3/reference/index.html

- Стандартная библиотека Python:

  https://docs.python.org/3/library/index.html

- Список документации Python:

  https://docs.python.org/3/

Ответы на вопросы

Несколько популярных форумов, посвященных Python и программированию вообще:

- python-forum.io
- https://www.dreamincode.net/forums/forum/29-python/
- StackOverflow.com

Кроме того, многие разработчики открывают форумы по своим инструментариям и библиотекам. Управление и сопровождение многих библиотек, используемых в книге, осуществляется через *github.com*. Для некоторых библиотек поддержка предоставляется через вкладку Issues на странице GinHub этих

библиотек. Если вы не найдете ответ на свои вопросы, посетите веб-страницу этой книги на сайте

http://www.deitel.com[1]

Поддержка Jupyter

Поддержка Jupyter Notebook предоставляется на следующих ресурсах:

- Project Jupyter Google Group:
  https://groups.google.com/forum/#!forum/jupyter
- Jupyter-чат в реальном времени:
  https://gitter.im/jupyter/jupyter
- GitHub
  https://github.com/jupyter/help
- StackOverflow:
  https://stackoverflow.com/questions/tagged/jupyter
- Jupyter for Education Google Group (для преподавателей, использующих Jupyter в ходе обучения):
  https://groups.google.com/forum/#!forum/jupyter-education

Приложения

Чтобы извлечь максимум пользы из материала, выполняйте каждый пример кода параллельно с соответствующим описанием в книге. На веб-странице книги на сайте

http://www.deitel.com

предоставляются:

- исходный код Python (файлы .py), подготовленный к загрузке, и документы Jupyter Notebook (файлы .ipynb) для примеров кода;
- видеоролики, поясняющие использование примеров кода с IPython и документами Jupyter Notebook. Эти инструменты также описаны в разделе 1.5;

---

[1] Наш сайт сейчас проходит серьезную переработку. Если вы не найдете нужную информацию, обращайтесь к нам по адресу deitel@deitel.com.

- сообщения в блогах и обновления книги.

За инструкциями по загрузке обращайтесь к разделу «Приступая к работе» после предисловия.

Как связаться с авторами книги

Мы ждем ваши комментарии, критические замечания, исправления и предложения по улучшению книги. С вопросами, найденными опечатками и предложениями обращайтесь по адресу: deitel@deitel.com.

Или ищите нас в соцсетях:

- Facebook® (http://www.deitel.com/deitelfan)
- Twitter® (@deitel)
- LinkedIn® (http://linkedin.com/company/deitel-&-associates)
- YouTube® (http://youtube.com/DeitelTV)

Благодарности

Спасибо Барбаре Дейтел (Barbara Deitel) за долгие часы, проведенные в интернете за поиском информации по проекту. Нам повезло работать с группой профессионалов из издательства Pearson. Мы высоко ценим все усилия и 25-летнее наставничество нашего друга и профессионала Марка Л. Тауба (Mark L. Taub), вице-президента издательской группы Pearson IT Professional Group. Марк со своей группой работает над всеми нашими профессиональными книгами, видеоуроками и учебными руководствами из сервиса Safari (https://learning.oreilly.com/). Они также являются спонсорами наших обучающих семинаров в Safari. Джули Наил (Julie Nahil) руководила выпуском книги. Мы выбрали иллюстрацию для обложки, а дизайн обложки был разработан Чати Презертсит (Chuti Prasertshith).

Мы хотим выразить свою благодарность своим редакторам. Патрисия Байрон-Кимболл (Patricia Byron-Kimball) и Меган Джейкоби (Meghan Jacoby) подбирали научных рецензентов и руководили процессом рецензирования. Держась в рамках жесткого графика, редакторы рецензировали нашу работу, делились многочисленными замечаниями для повышения точности, полноты и актуальности материала.

Научные редакторы

Книга

Дэниел Чен (Daniel Chen), специалист по data science, Lander Analytics

Гаррет Дансик (Garrett Dancik), доцент кафедры компьютерных наук/биоинформатики, Университет Восточного Коннектикута

Пранш у Гупта (Pranshu Gupta), доцент кафедры компьютерных наук, Университет Десалс

Дэвид Куп (David Koop), доцент кафедры data science, содиректор по учебным программам, Университет Массачусетса в Дартмуте

Рамон Мата-Толедо (Ramon Mata-Toledo), профессор кафедры компьютерных наук, Университет Джеймса Мэдисона

Шьямал Митра (Shyamal Mitra), старший преподаватель кафедры компьютерных наук, Техасский университет в Остине

Элисон Санчес (Alison Sanchez), доцент кафедры экономики, Университет Сан-Диего

Хосе Антонио Гонсалес Секо (José Antonio González Seco), IT-консультант

Джейми Уайтакер (Jamie Whitacre), независимый консультант в области data science

Элизабет Уикс (Elizabeth Wickes), преподаватель, школа информатики, Университет штата Иллинойс

Черновик

Ирен Бруно (Dr. Irene Bruno), доцент кафедры информатики и информационных технологий, Университет Джорджа Мэйсона

Ланс Брайант (Lance Bryant), доцент кафедры математики, Шиппенсбургский университет

Дэниел Чен (Daniel Chen), специалист по data science, Lander Analytics

Гаррет Дансик (Garrett Dancik), доцент кафедры компьютерных наук/биоинформатики, Университет Восточного Коннектикута

Марша Дэвис (Dr. Marsha Davis), декан математического факультета, Университет Восточного Коннектикута

Роланд ДеПратти (Roland DePratti), доцент кафедры компьютерных наук, Университет Восточного Коннектикута

Шьямал Митра (Shyamal Mitra), старший преподаватель, Техасский университет в Остине

Марк Поли (Dr. Mark Pauley), старший научный сотрудник на кафедре биоинформатики, школа междисциплинарной информатики, Университет штата Небраска в Омахе

Шон Рейли (Sean Raleigh), доцент кафедры математики, заведующий кафедрой data science, Вестминстерский колледж

Элисон Санчес (Alison Sanchez), доцент кафедры экономики, Университет Сан-Диего

Харви Сай (Dr. Harvey Siy), доцент кафедры компьютерных наук, информатики и информационных технологий, Университет штата Небраска в Омахе

Джейми Уайтакр (Jamie Whitacre), независимый консультант в области data science

Мы будем благодарны за ваши комментарии, критику, исправления и предложения по улучшению. Если у вас возникают какие-либо вопросы, обращайтесь по адресу *deitel@deitel.com*.

Добро пожаловать в увлекательный мир разработки Python с открытым кодом. Надеемся, вам понравится эта книга, посвященная разработке современных приложений Python с использованием IPython и Jupyter Notebook и затрагивающая вопросы data science, искусственного интеллекта, больших данных и облачных технологий. Желаем успеха!

Пол и Харви Дейтелы

Об авторах

Пол Дж. Дейтел (Paul J. Deitel), генеральный и технический директор компании Deitel & Associates, Inc., окончил Массачусетский технологический институт (MIT), более 38 лет занимается компьютерами. Пол — один из самых опытных преподавателей языков программирования, он ведет учебные курсы для разработчиков с 1992 года. Он провел сотни занятий по всему миру для корпоративных клиентов, включая Cisco, IBM, Siemens, Sun Microsystems (сейчас Oracle), Dell, Fidelity, NASA (Космический центр имени Кеннеди), Национальный центр прогнозирования сильных штормов, ракетный полигон Уайт-Сэндз, Rogue Wave Software, Boeing, Nortel Networks, Puma, iRobot и многих других. Пол и его соавтор, д-р Харви М. Дейтел, являются авторами всемирно известных бестселлеров — учебников по языкам программирования, предназначенных для начинающих и для профессионалов, а также видеокурсов.

Харви М. Дейтел (Dr. Harvey M. Deitel), председатель и главный стратег компании Deitel & Associates, Inc., имеет 58-летний опыт работы в области информационных технологий. Он получил степени бакалавра и магистра Массачусетского технологического института и степень доктора философии Бостонского университета — он изучал компьютерные технологии во всех этих программах до того, как в них появились отдельные программы компьютерных наук. Харви имеет огромный опыт преподавания в колледже и занимал должность председателя отделения информационных технологий Бостонского колледжа. В 1991 году вместе с сыном — Полом Дж. Дейтелом — он основал компанию Deitel & Associates, Inc. Харви с Полом написали несколько десятков книг и выпустили десятки видеокурсов LiveLessons. Написанные ими книги получили международное признание и были изданы на японском, немецком, русском, испанском, французском, польском, итальянском, упрощенном китайском, традиционном китайском, корейском, португальском, греческом, турецком языках и на языке урду. Дейтел провел сотни семинаров по программированию в крупных корпорациях, академических институтах, правительственных и военных организациях.

О компании Deitel® & Associates, Inc.

Компания Deitel & Associates, Inc., основанная Полом Дейтелом и Харви Дейтелом, получила международное признание в области авторских разработок и корпоративного обучения. Компания специализируется на языках программирования, объектных технологиях, интернете и веб-программировании. В число клиентов компании входят многие ведущие корпорации, правительственные агентства, военные и образовательные учреждения. Компания предоставляет учебные курсы, проводимые на территории клиента по всему миру для многих языков программирования и платформ.

Благодаря своему 44-летнему партнерскому сотрудничеству с Pearson/Prentice Hall, компания Deitel & Associates, Inc., публикует передовые учебники по программированию и профессиональные книги в печатном и электронном виде, видеокурсы **LiveLessons** (доступны для покупки на https://www.informit.com), **Learning Paths** и интерактивные обучающие семинары в режиме реального времени в службе Safari (https://learning.oreilly.com) и интерактивные мультимедийные курсы **Revel™**.

Чтобы связаться с компанией Deitel & Associates, Inc. и авторами или запросить план-проспект или предложение по обучению, напишите: deitel@deitel.com

Чтобы узнать больше о корпоративном обучении Дейтелов, посетите

http://www.deitel.com/training.

Желающие приобрести книги Дейтелов, могут сделать это на

https://www.amazon.com.

Крупные заказы корпораций, правительства, военных и академических учреждений следует размещать непосредственно на сайте Pearson. Для получения дополнительной информации посетите

https://www.informit.com/store/sales.aspx.

Приступая к работе

В этом разделе собрана информация, которую следует просмотреть перед чтением книги. Обновления будут публиковаться на сайте http://www.deitel.com.

Загрузка примеров кода

Файл examples.zip с кодом примеров книги можно загрузить на нашей вебстранице книги на сайте:

http://www.deitel.com

Щелкните на ссылке Download Examples, чтобы сохранить файл на вашем компьютере. Многие браузеры помещают загруженный файл в папку Downloads вашей учетной записи. Когда загрузка завершится, найдите файл в своей системе и распакуйте папку examples в папку Documents вашей учетной записи:

- Windows: C:\Users\YourAccount\Documents\examples
- macOS или Linux: ~/Documents/examples

В большинстве операционных систем имеется встроенная программа распаковки архивов. Также можно воспользоваться внешней программой-архиватором — например, 7-Zip (www.7-zip.org) или WinZip (www.winzip.com).

Структура папки examples

В этой книге приводятся примеры трех типов:

- отдельные фрагменты кода для интерактивной среды IPython;
- законченные приложения, называемые *сценариями*;
- документы Jupyter Notebook — удобной интерактивной среды для браузера, в которой можно писать и выполнять код, а также чередовать код с текстом, графикой и видео.

Все варианты продемонстрированы в примерах из раздела 1.5.

Каталог examples содержит одну вложенную папку для каждой главы. Этим папкам присвоены имена вида ch##, где ## — двузначный номер главы от 01 до 16 — например, ch01. Кроме глав 13, 15 и 16, папка каждой главы содержит следующие элементы:

- snippets_ipynb — папка с файлами Jupyter Notebook этой главы;
- snippets_py — папка с файлами с исходным кодом Python, в которых хранятся все представленные фрагменты кода, разделенные пустой строкой. Вы можете скопировать эти фрагменты в IPython или в созданные вами новые документы Jupyter Notebook;
- файлы сценариев и используемые ими файлы.

Глава 13 содержит одно приложение. Главы 15 и 16 объясняют, где найти нужные файлы в папках ch15 и ch16 соответственно.

Установка Anaconda

В книге используется дистрибутив Anaconda Python, отличающийся простотой установки. В него входит практически все необходимое для работы с примерами, в том числе:

- интерпретатор IPython;
- большинство библиотек Python и data science, используемых в книге;
- локальный сервер Jupyter Notebook для загрузки и выполнения документов;

- другие программные пакеты, такие как Spyder IDE (Integrated Development Environment), — в книге используются только среды IPython и Jupyter Notebook.

Программу установки Python 3.x Anaconda для Windows, macOS и Linux можно загрузить по адресу:

https://www.anaconda.com/download/

Когда загрузка завершится, запустите программу установки и выполните инструкции на экране. Чтобы установленная копия Anaconda работала правильно, не перемещайте ее файлы после установки.

Обновление Anaconda

Затем проверьте актуальность установки Anaconda. Откройте окно командной строки в своей системе:

- В macOS откройте приложение Terminal из подкаталога `Utilities` в каталоге Applications.
- В Windows откройте командную строку Anaconda Prompt из меню Пуск. Когда вы делаете это для обновления Anaconda (как в данном случае) или для установки новых пакетов (см. ниже), выполните Anaconda Prompt с правами администратора: щелкните правой кнопкой мыши и выберите команду Запуск от имени администратора. (Если вы не можете найти команду Anaconda Prompt в меню Пуск, найдите ее при помощи поля поиска в нижней части экрана.)
- В Linux откройте терминал или командную оболочку своей системы (зависит от дистрибутива Linux).

В окне командной строки своей системы выполните следующие команды, чтобы обновить установленные пакеты Anaconda до последних версий:

```
conda update conda
conda update --all
```

Менеджеры пакетов

Приведенная выше команда conda запускает менеджер пакетов `conda` — один из двух основных менеджеров пакетов Python, используемых в книге (другой — `pip`). Пакеты содержат файлы, необходимые для установки отдельных

библиотек или инструментов Python. В книге `conda` будет использоваться для установки дополнительных пакетов, если только не окажется, что эти пакеты недоступны в `conda`; в этом случае будет использоваться `pip`. Некоторые разработчики предпочитают пользоваться исключительно `pip`, потому что эта программа в настоящее время поддерживает больше пакетов. Если у вас возникнут проблемы с установкой пакетов из `conda`, попробуйте использовать `pip`.

Установка программы статического анализа кода Prospector

Для анализа кода Python можно воспользоваться аналитической программой Prospector, которая проверяет ваш код на наличие типичных ошибок и помогает улучшить его. Чтобы установить программу Prospector и используемые ею библиотеки Python, выполните следующую команду в окне командной строки:

```
pip install prospector
```

Установка jupyter-matplotlib

В книге для построения некоторых анимаций используется библиотека визуализации Matplotlib. Чтобы использовать анимации в документах Jupyter Notebook, необходимо установить программу `ipympl`. В терминале в командной строке Anaconda или в оболочке, открытой ранее, последовательно выполните следующие команды[1]:

```
conda install -c conda-forge ipympl
conda install nodejs
jupyter labextension install @jupyter-widgets/jupyterlab-manager
jupyter labextension install jupyter-matplotlib
```

Установка других пакетов

Дистрибутив Anaconda включает приблизительно 300 популярных пакетов Python и data science, включая NumPy, Matplotlib, pandas, Regex, BeautifulSoup, requests, Bokeh, SciPy, SciKit-Learn, Seaborn, Spacy, sqlite, statsmodels и многие другие. Количество дополнительных пакетов, которые вам придется уста-

[1] https://github.com/matplotlib/jupyter-matplotlib.

навливать в книге, будет небольшим, и мы будем приводить инструкции по установке в таких местах. В документации новых пакетов объясняется, как их следует устанавливать.

Получение учетной записи разработчика Twitter

Если вы намереваетесь использовать главу «Глубокий анализ данных Twitter» и все примеры на базе Twitter в последующих главах, подайте заявку на получение учетной записи разработчика Twitter. Сейчас Twitter требует регистрации для получения доступа к их API. Чтобы подать заявку на создание учетной записи разработчика, заполните и отправьте форму по адресу:

https://developer.twitter.com/en/apply-for-access

Twitter проверяет каждую заявку. На момент написания книги заявки на создание личных учетных записей разработчиков утверждались практически немедленно, а на утверждение заявок от компаний требовалось от нескольких дней до нескольких недель. Успешное утверждение не гарантировано.

Необходимость подключения к интернету в некоторых главах

При использовании этой книги вам может понадобиться подключение к интернету для установки дополнительных библиотек. В некоторых главах вы будете регистрироваться для создания учетных записей в облачных сервисах (в основном для использования их бесплатных уровней). Некоторые сервисы требуют кредитных карт для подтверждения личности. В отдельных случаях будут использоваться платные сервисы. Тогда мы воспользуемся кредитом, который предоставляется фирмой-поставщиком, так что вы сможете опробовать сервис без каких-либо затрат. Предупреждение: некоторые облачные сервисы начинают выставлять счета после их настройки. Когда вы закончите анализ примеров с использованием таких сервисов, без промедления удалите выделенные ресурсы.

Различия в выводе программ

При выполнении наших примеров вы можете заметить некоторые различия между приведенными и вашими результатами:

- Из-за различий в выполнении вычислений в формате с плавающей точкой (например, `-123.45`, `7.5` или `0.0236937`) в разных операционных системах вы можете заметить незначительные различия в выводе, особенно в младших разрядах дробной части.
- Когда мы приводим выходные данные, отображаемые в разных окнах, мы обрезаем границы окон для экономии места.

Получение ответов на вопросы

На форумах в интернете вы сможете общаться с другими программистами Python и получать ответы на свои вопросы. Некоторые популярные форумы по тематике Python и программирования в целом:

- python-forum.io
- StackOverflow.com
- https://www.dreamincode.net/forums/forum/29-python/

Кроме того, многие фирмы-разработчики предоставляют форумы, посвященные их инструментам и библиотекам. Управление большинством упоминаемых в книге библиотек и их сопровождение осуществляется на *github.com*. Некоторые разработчики, занимающиеся сопровождением библиотек, предоставляют поддержку на вкладке Issues страницы GitHub библиотеки. Если вам не удастся найти ответ на свой вопрос в интернете, обращайтесь к веб-странице книги на сайте[1]

http://www.deitel.com

Теперь все готово к чтению книги. Надеемся, она вам понравится!

[1] В настоящее время наш сайт проходит глобальную реконструкцию. Если вы не найдете какую-то необходимую информацию, напишите нам по адресу deitel@deitel.com.

1

Компьютеры и Python

В этой главе...

- Наиболее интересные тенденции в области современных вычислительных технологий.
- Краткий обзор основ объектно-ориентированного программирования.
- Сильные стороны Python.
- Знакомство с важнейшими библиотеками Python и data science, используемыми в этой книге.
- Интерактивный режим интерпретатора IPython и выполнение кода Python.
- Выполнение сценария Python для анимации гистограммы.
- Создание и тестирование веб-оболочки Jupyter Notebook для выполнения кода Python.
- Большие данные: с каждым днем еще больше.
- Анализ больших данных на примере популярного мобильного навигационного приложения.
- Искусственный интеллект: пересечение компьютерной теории и data science.

1.1. Введение

Знакомьтесь: Python — один из наиболее широко используемых языков программирования, а согласно *индексу PYPL* (Popularity of Programming Languages) — самый популярный в мире[1].

В этом разделе представлены терминология и концепции, закладывающие основу для программирования на языке Python, которому будут посвящены главы 2–10, и для практических примеров из области больших данных, искусственного интеллекта и облачных технологий, описываемых в главах 11–16.

Читатели узнают, почему язык Python стал таким популярным, познакомятся со стандартной библиотекой Python и различными библиотеками data science, благодаря которым не придется заново «изобретать велосипед». Эти библиотеки используются для создания объектов. С их помощью можно решать серьезные задачи при умеренном объеме программного кода.

Затем будут рассмотрены три примера, демонстрирующие различные способы выполнения кода Python:

- В первом примере вы будете выполнять команды Python в оболочке IPython в интерактивном режиме и сразу же видеть результаты.
- Во втором примере мы выполним серьезное приложение Python, отображающее анимированную гистограмму результатов бросков шестигранного кубика. Это позволит увидеть «закон больших чисел» в действии. Забегая чуть вперед, отметим, что в главе 6 это же приложение будет построено с использованием библиотеки визуализации Matplotlib.
- В последнем примере будет продемонстрирована работа с документами Jupyter Notebook с использованием JupyterLab — интерактивной оболочки на базе веб-браузера, обеспечивающей удобство в процессе формирования и выполнения инструкции Python. К слову, в документы Jupyter Notebook можно включать текст, изображения, звуковые данные, видеоролики, анимации и программный код.

В прошлом большинство компьютерных приложений выполнялось на автономных (то есть не объединенных в сеть) компьютерах. Современные приложения могут создаваться с расчетом на взаимодействие с миллиардами компьютеров по всему миру через интернет. Мы расскажем об облачных

[1] https://pypl.github.io/PYPL.html (на январь 2019 г.).

технологиях и концепции IoT (Internet of Things), закладывающих основу для современных приложений (подробнее об этом в главах 11–16).

Вы, кроме того, узнаете, насколько велики большие данные и как они стремительно становятся еще больше. Затем будет представлен вариант анализа больших данных на примере мобильного навигационного приложения Waze. Это приложение многие современные технологии использует для построения динамических инструкций, помогающих быстро и безопасно добраться до конечной точки маршрута. При описании этих технологий будет указано, где многие из них используются в этой книге. Глава завершается разделом «Введение в data science», посвященным важнейшей области пересечения компьютерной теории и дисциплины data science — искусственному интеллекту.

1.2. Основы объектных технологий

Потребность в новых, более мощных программах стремительно растет, поэтому очень важно, чтобы программные продукты строились по возможности быстро, корректно и эффективно. *Объекты* (точнее, *классы*, на основе которых строятся объекты), по сути, представляют собой программные компоненты, *пригодные для повторного использования*. Объектом может быть что угодно: дата, время, аудио, видео, автомобиль, человек и т. д. Практически каждое *существительное* может быть адекватным образом представлено программным объектом, обладающим *атрибутами* (например, имя, цвет и размер) и *способностями к выполнению определенных действий* (например, вычисление, перемещение и обмен данными). Разработчики программ используют модульную структуру и объектно-ориентированную методологию проектирования с большей эффективностью, чем некогда популярные методологии вроде «структурного программирования». Объектно-ориентированные программы зачастую более понятны, их проще исправлять и модифицировать.

Автомобиль как объект

Чтобы лучше понять суть объектов и их внутреннее устройство, воспользуемся простой аналогией. Представьте, что вы *ведете автомобиль и нажимаете педаль газа, чтобы набрать скорость*. Но что должно произойти до того, как вы получите возможность водить автомобиль? Прежде всего автомобиль нужно *изготовить*. Изготовление любого автомобиля начинается с инженерных чертежей (ка́лек), подробно описывающих устройство автомобиля. На этих чертежах даже показано устройство педали акселератора. За этой педа-

лью *скрываются* сложные механизмы, которые при приведении в действие ускоряют автомобиль. Аналогично и педаль тормоза «скрывает» механизмы, тормозящие автомобиль, руль «скрывает» механизмы, поворачивающие автомобиль, и т. д. Благодаря этому люди, не имеющие понятия о внутреннем устройстве автомобиля, могут легко им управлять.

Невозможно готовить пищу на кухне, существующей лишь на листе бумаги; точно так же нельзя водить автомобиль, существующий лишь в чертежах. Прежде чем вы сядете за руль машины, ее нужно *воплотить* в металл на основе чертежей. Воплощенный в металле автомобиль имеет *реальную* педаль газа, с помощью которой он может ускоряться, но (к счастью!) не может делать это самостоятельно — *нажимает* на педаль газа водитель.

Методы и классы

Воспользуемся примером с автомобилем для иллюстрации некоторых ключевых концепций объектно-ориентированного программирования. Для выполнения задачи в программе требуется *метод*, «скрывающий» инструкции программы, которые фактически выполняют задание. Метод скрывает эти инструкции от пользователя подобно тому, как педаль газа автомобиля скрывает от водителя механизмы, вызывающие ускорение автомобиля. В Python программная единица, именуемая *классом*, включает методы, выполняющие задачи класса. Например, класс, представляющий банковский счет, может включать три метода, один из которых выполняет *пополнение* счета, второй — *вывод средств* со счета, а третий — *запрос* текущего баланса. Класс напоминает концепцию автомобиля, представленного чертежами, включая чертежи педали газа, рулевого колеса и других механизмов.

Создание экземпляра класса

Итак, чтобы управлять автомобилем, его сначала необходимо изготовить по чертежам; точно так же перед выполнением задач, определяемых методами этого класса, сначала необходимо *создать объект класса*. Этот процесс называется *созданием экземпляра*. Полученный при этом объект называется *экземпляром* класса.

Повторное использование

Подобно тому как один и тот же чертеж можно *повторно использовать* для создания многих автомобилей, один класс можно *повторно использовать* для

создания многих объектов. Повторное использование существующих классов при построении новых классов и программ экономит время и силы разработчика, облегчает создание более надежных и эффективных систем, поскольку ранее созданные классы и компоненты прошли тщательное *тестирование*, *отладку* и *оптимизацию производительности*. Подобно тому как концепция *взаимозаменяемых частей* легла в основу индустриальной революции, повторно используемые классы — двигатель прогресса в области создания программ, который был вызван внедрением объектной технологии.

В Python для создания программ обычно применяется *принцип компоновки из готовых блоков*. Дабы заново не «изобретать велосипед», используйте качественные готовые компоненты там, где это возможно. Повторное использование программного кода — важнейшее преимущество объектно-ориентированного программирования.

Сообщения и вызовы методов

Нажимая педаль газа, вы тем самым отсылаете автомобилю *сообщение* с запросом на выполнение определенной задачи (ускорение движения). Подобным же образом *отсылаются и сообщения объекту*. Каждое сообщение реализуется вызовом метода, который «сообщает» методу объекта о необходимости выполнения определенной задачи. Например, программа может вызвать метод *deposit* объекта банковского счета для его пополнения.

Атрибуты и переменные экземпляра класса

Любой автомобиль не только способен выполнять определенные задачи, но и обладает и характерными *атрибутами*: цвет, количество дверей, запас топлива в баке, текущая скорость и пройденное расстояние. Атрибуты автомобиля, как и его возможности по выполнению определенных действий, представлены на инженерных диаграммах как часть проекта (например, им могут соответствовать одометр и указатель уровня бензина). При вождении автомобиля его атрибуты существуют вместе с ним. Каждому автомобилю присущ *собственный* набор атрибутов. Например, каждый автомобиль «знает» о том, сколько бензина осталось в его баке, но ему ничего не известно о запасах горючего в баках *других* автомобилей.

Объект, как и автомобиль, имеет собственный набор атрибутов, которые он «переносит» с собой при использовании этого объекта в программах. Эти атрибуты определены в качестве части объекта класса. Например, объект банков-

ского счета bank-account имеет *атрибут баланса*, представляющий количество средств на банковском счете. Каждый объект bank-account «знает» о количестве средств на собственном счете, но *ничего* не «знает» о размерах *других* банковских счетов. Атрибуты определяются с помощью других переменных *экземпляра класса*. Между атрибутами и методами класса (и его объектов) существует тесная связь, поэтому классы хранят вместе свои атрибуты и методы.

Наследование

С помощью *наследования* можно быстро и просто создать новый класс объектов. При этом новый класс (называемый *подклассом*) наследует характеристики существующего класса (называемого *суперклассом*) — возможно, частично изменяя их или добавляя новые характеристики, уникальные для этого класса. Если вспомнить аналогию с автомобилем, то «трансформер» *является* объектом *более обобщенного* класса «автомобиль» с *конкретной* особенностью: у него может подниматься или опускаться крыша.

Объектно-ориентированный анализ и проектирование

Вскоре вы начнете писать программы на Python. Как же вы будете формировать код своих программ? Скорее всего, как и большинство других программистов, вы включите компьютер и начнете вводить с клавиатуры исходный код программы. Подобный подход годится при создании маленьких программ (вроде тех, что представлены в начальных главах книги), но что делать при создании крупного программного комплекса, который, например, управляет тысячами банкоматов крупного банка? А если вы руководите командой из 1000 программистов, занятых разработкой системы управления воздушным движением следующего поколения? В таких крупных и сложных проектах нельзя просто сесть за компьютер и строка за строкой вводить код.

Чтобы выработать наилучшее решение, следует провести процесс детального *анализа требований* к программному проекту (то есть определить, *что* должна делать система) и разработать *проектное решение*, которое будет соответствовать этим требованиям (то есть определить, *как* система будет выполнять поставленные задачи). В идеале вы должны тщательно проанализировать проект (либо поручить выполнение этой задачи коллегам-профессионалам) еще до написания какого-либо кода. Если этот процесс подразумевает применение объектно-ориентированного подхода, то, значит, мы имеем дело с *процессом OOAD* (object-oriented analysis and design, *объектно-ориентированный анализ и проектирование*). Языки программирования, подобные Python, тоже

называются объектно-ориентированными. Программирование на таком языке — *объектно-ориентированное программирование* (*ООП*) — позволяет реализовать результат объектно-ориентированного проектирования в форме работоспособной системы.

1.3. Python

Python — объектно-ориентированный сценарный язык, официально опубликованный в 1991 году. Он был разработан Гвидо ван Россумом (Guido van Rossum) из Национального исследовательского института математики и компьютерных наук в Амстердаме.

Python быстро стал одним из самых популярных языков программирования в мире. Он пользуется особой популярностью в среде образования и научных вычислений[1], а в последнее время превзошел язык программирования R в качестве самого популярного языка обработки данных[2,3,4]. Назовем основные причины популярности Python, пояснив, почему каждому стоит задуматься об изучении этого языка[5,6,7]:

- Python — бесплатный общедоступный проект с открытым кодом, имеющий огромное сообщество пользователей.
- Он проще в изучении, чем такие языки, как C, C++, C# и Java, что позволяет быстро включиться в работу как новичкам, так и профессиональным разработчикам.
- Код Python проще читается, чем большинство других популярных языков программирования.
- Он широко применяется в образовательной области[8].
- Он повышает эффективность труда разработчика за счет обширной подборки стандартных и сторонних библиотек с открытым кодом, так что

---

1 https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017.
2 https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html.
3 https://www.r-bloggers.com/data-science-job-report-2017-r-passes-sas-but-python-leaves-them-both-behind/.
4 https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017.
5 https://dbader.org/blog/why-learn-python.
6 https://simpleprogrammer.com/2017/01/18/7-reasons-why-you-should-learn-python/.
7 https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017.
8 *Tollervey N.* Python in Education: Teach, Learn, Program (O'Reilly Media, Inc., 2015).

программисты могут быстрее писать код и решать сложные задачи с минимумом кода (подробнее см. раздел 1.4).

- Существует множество бесплатных приложений Python с открытым кодом.
- Python — популярный язык веб-разработки (Django, Flask и т. д.).
- Он поддерживает популярные парадигмы программирования — процедурную, функциональную и объектно-ориентированную[1]. Обратите внимание: мы начнем описывать средства функционального программирования в главе 4, а потом будем использовать их в последующих главах.
- Он упрощает параллельное программирование — с asyncio и async/await вы можете писать однопоточный параллельный код[2], что существенно упрощает сложные по своей природе процессы написания, отладки и сопровождения кода[3].
- Существует множество возможностей для повышения быстродействия программ на языке Python.
- Python используется для построения любых программ от простых сценариев до сложных приложений со множеством пользователей, таких как Dropbox, YouTube, Reddit, Instagram или Quora[4].
- Python — популярный язык для задач искусственного интеллекта, а эта область в последнее время стремительно развивается (отчасти благодаря ее особой связи с областью data science).
- Он широко используется в финансовом сообществе[5].
- Для программистов Python существует обширный рынок труда во многих областях, особенно в должностях, связанных с data science, причем вакансии Python входят в число самых высокооплачиваемых вакансий для программистов[6,7]. Ближайший конкурент Python — *R*, популярный язык

---

1 https://en.wikipedia.org/wiki/Python_(programming_language).

2 https://docs.python.org/3/library/asyncio.html.

3 https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017.

4 https://www.hartmannsoftware.com/Blog/Articles_from_Software_Fans/Most-Famous-Software-Programs-Written-in-Python.

5 *Kolanovic M., Krishnamachari R.* Big Data and AI Strategies: Machine Learning and Alternative Data Approach to Investing (J. P. Morgan, 2017).

6 https://www.infoworld.com/article/3170838/developer/get-paid-10-programming-languages-to-learn-in-2017.html.

7 https://medium.com/@ChallengeRocket/top-10-of-programming-languages-with-the-highest-salaries-in-2017-4390f468256e.

программирования с открытым кодом для разработки статистических приложений и визуализации. Сейчас Python и *R* — два наиболее широко применяемых языка data science.

Anaconda

Мы будем использовать дистрибутив Python Anaconda — он легко устанавливается в Windows, macOS и Linux, поддерживая последние версии Python, интерпретатора IPython (раздел 1.5.1) и Jupyter Notebooks (раздел 1.5.3). Anaconda также включает другие программные пакеты и библиотеки, часто используемые в программировании Python и data science, что позволяет разработчикам сосредоточиться на коде Python и аспектах data science, не отвлекаясь на возню с проблемами установки. Интерпретатор IPython[1] обладает интересными возможностями, позволяющими проводить исследования и эксперименты с Python, стандартной библиотекой Python и обширным набором сторонних библиотек.

Дзен Python

Мы придерживаемся принципов из статьи Тима Петерса (Tim Peters) «*The Zen of Python*», в которой приведена квинтэссенция идеологии проектирования от создателя Python Гвидо ван Россума. Этот список также можно просмотреть в IPython, воспользовавшись командой `import this`. Принципы «дзена Python» определяются в предложении об улучшении языка Python (PEP, Python Enhancement Proposal) 20, согласно которому «PEP — проектный документ с информацией для сообщества Python или с описанием новой возможности Python, его процессов или окружения»[2].

1.4. Библиотеки

В этой книге мы будем по возможности пользоваться существующими библиотеками (включая стандартные библиотеки Python, библиотеки data science и некоторые сторонние библиотеки) — их применение способствует повышению эффективности разработки программных продуктов. Так, вместо того чтобы писать большой объем исходного кода (дорогостоящий и длительный процесс), можно просто создать объект уже существующего библиотечного класса посредством всего одной команды Python. Библиотеки позволяют решать серьезные задачи с минимальным объемом кода.

---

[1] https://ipython.org/.

[2] https://www.python.org/dev/peps/pep-0001/.

1.4.1. Стандартная библиотека Python

Стандартная библиотека Python предоставляет богатую функциональность обработки текстовых/двоичных данных, математических вычислений, программирования в функциональном стиле, работы с файлами/каталогами, хранения данных, сжатия/архивирования данных, криптографии, сервисных функций операционной системы, параллельного программирования, межпро-

НЕКОТОРЫЕ МОДУЛИ СТАНДАРТНОЙ БИБЛИОТЕКИ PYTHON, ИСПОЛЬЗУЕМЫЕ В КНИГЕ

`collections` — дополнительные структуры данных помимо списков, кортежей, словарей и множеств.

`csv` — обработка файлов с данными, разделенными запятыми.

`datetime`, `time` — операции с датой и временем.

`decimal` — вычисления с фиксированной и плавающей точкой, включая финансовые вычисления.

`doctest` — простое модульное тестирование с использованием проверочных тестов и ожидаемых результатов, закодированных в doc-строках.

`json` — обработка формата JSON (JavaScript Object Notation) для использования с веб-сервисами и базами данных документов NoSQL.

`math` — распространенные математические константы и операции.

`os` — взаимодействие с операционной системой.

`queue` — структура данных, работающая по принципу «первым зашел, первым вышел» (FIFO).

`random` — псевдослучайные числа.

`re` — регулярные выражения для поиска по шаблону.

`sqlite3` — работа с реляционной базой данных SQLite.

`statistics` — функции математической статистики (среднее, медиана, дисперсия, мода и т. д.).

`string` — обработка строк.

`sys` — обработка аргументов командной строки; потоки стандартного ввода, стандартного вывода; стандартный поток ошибок.

`timeit` — анализ быстродействия.

цессных взаимодействий, сетевых протоколов, работы с JSON/XML/другими форматами данных в интернете, мультимедиа, интернационализации, построения графического интерфейса, отладки, профилирования и т. д. В таблице выше перечислены некоторые модули стандартной библиотеки Python, которые будут использоваться в примерах.

1.4.2. Библиотеки data science

У Python сформировалось огромное и стремительно развивающееся сообщество разработчиков с открытым кодом во многих областях. Одним из самых серьезных факторов популярности Python стала замечательная подборка библиотек, разработанных сообществом с открытым кодом. Среди прочего мы стремились создавать примеры и практические примеры, которые бы послужили увлекательным и интересным введением в программирование Python, но при этом одновременно познакомили читателей с практическими аспектами data science, ключевыми библиотеками data science и т. д. Вы не поверите, какие серьезные задачи можно решать всего в нескольких строках кода. В следующей таблице перечислены некоторые популярные библиотеки data science. Многие из них будут использоваться нами в примерах. Для визуализации будут использоваться Matplotlib, Seaborn и Folium, но существует и немало других (хорошая сводка библиотек визуализации Python доступна по адресу http://pyviz.org/).

ПОПУЛЯРНЫЕ БИБЛИОТЕКИ PYTHON, ИСПОЛЬЗУЕМЫЕ В DATA SCIENCE

Научные вычисления и статистика

NumPy (Numerical Python) — в Python нет встроенной структуры данных массива. В нем используются списки — удобные, но относительно медленные. NumPy предоставляет высокопроизводительную структуру данных `ndarray` для представления списков и матриц, а также функции для обработки таких структур данных.

SciPy (Scientific Python) — библиотека SciPy, встроенная в NumPy, добавляет функции для научных вычислений: интегралы, дифференциальные уравнения, расширенная обработка матриц и т. д. Сайт scipy.org обеспечивает поддержку SciPy и NumPy.

StatsModels — библиотека, предоставляющая функциональность оценки статистических моделей, статистических критериев и статистического анализа данных.

Анализ и управление данными

pandas — чрезвычайно популярная библиотека для управления данными. В pandas широко используется структура ndarray из NumPy. Ее две ключевые структуры данных — `Series` (одномерная) и `DataFrames` (двумерная).

Визуализация

Matplotlib — библиотека визуализации и построения диаграмм с широкими возможностями настройки. Среди прочего в ней поддерживаются обычные графики, точечные диаграммы, гистограммы, контурные и секторные диаграммы, графики поля направлений, полярные системы координат, трехмерные диаграммы и текст.

Seaborn — высокоуровневая библиотека визуализации, построенная на базе Matplotlib. Seaborn добавляет более качественное оформление и дополнительные средства визуализации, а также позволяет строить визуализации с меньшим объемом кода.

Машинное обучение, глубокое обучение и обучение с подкреплением

scikit-learn — ведущая библиотека машинного обучения. Машинное обучение является подмножеством области искусственного интеллекта, а глубокое обучение — подмножеством машинного обучения, ориентированным на нейронные сети.

Keras — одна из самых простых библиотек глубокого обучения. Keras работает на базе TensorFlow (Google), CNTK (когнитивный инструментарий Microsoft для глубокого обучения) или Theano (Монреальский университет).

TensorFlow — самая популярная библиотека глубокого обучения от Google. TensorFlow использует графические процессоры или тензорные процессоры Google для повышения быстродействия. TensorFlow играет важную роль в областях искусственного интеллекта и аналитики больших данных с их огромными требованиями к вычислительным мощностям. Мы будем использовать версию Keras, встроенную в TensorFlow.

OpenAI Gym — библиотека и среда для разработки, тестирования и сравнения алгоритмов обучения с подкреплением.

Обработка естественного языка (NLP, Natural Language Processing)

NLTK (Natural Language Toolkit) — используется для задач обработки естественного языка (NLP).

TextBlob — объектно-ориентированная NLP-библиотека обработки текста, построенная на базе библиотек NLTK и NLP с паттернами. TextBlob упрощает многие задачи NLP.

Gensim — похожа на NLTK. Обычно используется для построения индекса для коллекции документов и последующего определения его схожести с остальными документами в индексе.

1.5. Первые эксперименты: использование IPython и Jupyter Notebook

В этом разделе мы опробуем интерпретатор IPython[1] в двух режимах:

- В *интерактивном режиме* будем вводить небольшие фрагменты кода Python и немедленно просматривать результаты их выполнения.
- В *режиме сценариев* будет выполняться код, загруженный из файла с расширением `.py` (сокращение от Python). Такие файлы, называемые *сценариями* или *программами*, обычно имеют большую длину, чем фрагменты кода, используемые в интерактивном режиме.

Затем вы научитесь использовать браузерную среду Jupyter Notebook для написания и выполнения кода Python[2].

1.5.1. Использование интерактивного режима IPython как калькулятора

Попробуем использовать интерактивный режим IPython для вычисления простых арифметических выражений.

---

[1] Прежде чем читать этот раздел, выполните инструкции из раздела «Приступая к работе» и установите дистрибутив Python Anaconda, содержащий интерпретатор IPython.

[2] Jupyter поддерживает многие языки программирования, для чего следует установить соответствующее «ядро». За дополнительной информацией обращайтесь по адресу https://github.com/jupyter/jupyter/wi.

Запуск IPython в интерактивном режиме

Сначала откройте окно командной строки в своей системе:

- В macOS откройте **Терминал** из папки **Utilities** в папке **Applications**.
- В Windows запустите **командную строку Anaconda** из меню Пуск.
- В Linux откройте **Терминал** или командный интерпретатор своей системы (зависит от дистрибутива Linux).

В окне командной строки введите команду `ipython` и нажмите Enter (или *Return*). На экране появится сообщение вроде (зависит от платформы и версии IPython):

```
Python 3.7.0 | packaged by conda-forge | (default, Jan 20 2019, 17:24:52)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Текст `"In [1]:"` — *приглашение*, означающее, что IPython ожидает вашего ввода. Вы можете ввести `?` для получения справки или перейти непосредственно к вводу фрагментов, чем мы сейчас и займемся.

Вычисление выражений

В интерактивном режиме можно вычислять выражения:

```
In [1]: 45 + 72
Out[1]: 117

In [2]:
```

После того как вы введете `45 + 72` и нажмете Enter, IPython *читает* фрагмент, *вычисляет* его и *выводит* результат в `Out[1]`[1]. Затем IPython выводит приглашение `In [2]`, означающее, что среда ожидает от вас ввода второго фрагмента. Для каждого нового фрагмента IPython добавляет 1 к числу в квадратных скобках. Каждое приглашение `In [1]` в книге указывает на то, что мы начали новый интерактивный сеанс. Обычно мы будем делать это для каждого нового раздела главы.

---

[1] В следующей главе вы увидите, что в некоторых ситуациях приглашение `Out[]` не выводится.

Создадим более сложное выражение:

```
In [2]: 5 * (12.7 - 4) / 2
Out[2]: 21.75
```

В Python звездочка (`*`) обозначает операцию умножения, а косая черта (`/`) — операцию деления. Как и в математике, круглые скобки определяют порядок вычисления, так что сначала будет вычислено выражение в круглых скобках `(12.7 - 4)` и получен результат `8.7`. Затем вычисляется результат `5 * 8.7`, который равен `43.5`. После этого вычисляется выражение `43.5 / 2`, а полученный результат `21.75` выводится средой IPython в `Out[2]`. Числа, такие как 5, 4 и 2, называются целыми числами. Числа с дробной частью, такие как `12.7`, `43.5` и `21.75`, называются *числами с плавающей точкой*.

Выход из интерактивного режима

Чтобы выйти из интерактивного режима, можно воспользоваться одним из нескольких способов:

- Ввести команду `exit` в приглашении `In []` и нажать Enter для немедленного выхода.
- Нажать клавиши <Ctrl>+d (или <control>+d). На экране появляется предложение подтвердить выход `"Do you really want to exit ([y]/n)?"`. Квадратные скобки вокруг `y` означают, что этот ответ выбирается по умолчанию — нажатие клавиши Enter отправляет ответ по умолчанию и завершает интерактивный режим.
- Нажать <Ctrl>+d (или <control>+d) дважды (только в macOS и Linux).

1.5.2. Выполнение программы Python с использованием интерпретатора IPython

В этом разделе выполним сценарий с именем RollDieDynamic.py, который будет написан в главе 6. *Расширение* .py указывает на то, что файл содержит исходный код Python. Сценарий RollDieDynamic.py моделирует бросок шестигранного кубика. Он отображает цветную диаграмму с анимацией, которая в динамическом режиме отображает частоты выпадения всех граней.

Переход в папку с примерами этой главы

Сценарий находится в папке ch01 исходного кода книги. В разделе «Приступая к работе» папка examples была распакована в папку Documents вашей учетной записи пользователя. У каждой главы существует папка с исходным кодом этой главы. Этой папке присвоено имя ch##, где ## — номер главы от 01 до 16. Откройте окно командной строки вашей системы и введите команду `cd` («change directory»), чтобы перейти в папку ch01:

- В macOS/Linux введите команду `cd ~/Documents/examples/ch01` и нажмите Enter.
- В Windows введите команду `cd`

  C:\Users\YourAccount\Documents\examples\ch01

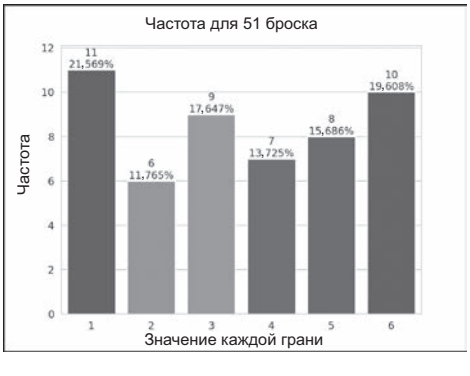
и нажмите Enter.

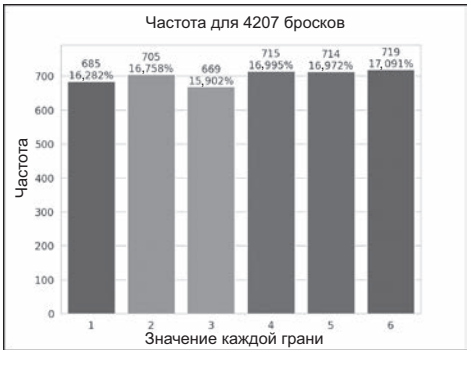
Выполнение сценария

Чтобы выполнить сценарий, введите следующую команду в командной строке и нажмите Enter:

```
ipython RollDieDynamic.py 6000 1
```

Сценарий открывает окно, в котором отображается его визуализация. Числа `6000` и `1` сообщают сценарию, сколько бросков должно быть сделано и сколько кубиков нужно бросать каждый раз. В данном случае диаграмма будет обновляться `6000` раз по одному кубику за раз.





Для шестигранного кубика значения от 1 до 6 выпадают с равной вероятностью — вероятность каждого результата равна 1/6, или около 16,667%. Если бросить кубик 6000 раз, каждая грань выпадет около 1000 раз. Бросок кубиков, как и бросок монетки, является *случайным процессом*, поэтому некоторые грани могут выпасть менее 1000 раз, а другие — более 1000 раз. Мы сделали несколько снимков экрана во время выполнения сценария. Сценарий использует случайно сгенерированные значения, поэтому ваши результаты будут другими. Поэкспериментируйте со сценарием; попробуйте заменить значение 1 на 100, 1000 и 10 000. Обратите внимание: с увеличением количества бросков частоты сходятся к величине 16,667%. В этом проявляется действие «закона больших чисел».

Создание сценария

Обычно исходный код Python создается в редакторе для ввода текста. Вы вводите программу в редакторе, вносите все необходимые исправления и сохраняете ее на своем компьютере. *Интегрированные среды разработки* (*IDE, Integrated Development Environments*) предоставляют средства, обеспечивающие поддержку всего процесса разработки программного обеспечения: редакторы, отладчики для поиска логических ошибок, нарушающих работу программы, и т. д. Среди популярных интегрированных сред Python можно выделить Spyder (входящий в комплект Anaconda), PyCharm и Visual Studio Code.

Проблемы, которые могут возникнуть во время выполнения программы

Далеко не все программы работают с первой попытки. Например, программа может попытаться выполнить деление на 0 (недопустимая операция в Python). В этом случае программа выдаст сообщение об ошибке. Если это произойдет в сценарии, то вы снова возвращаетесь к редактору, вносите необходимые изменения и повторно выполняете сценарий, чтобы определить, решило ли исправление проблему(-ы).

Такие ошибки, как деление на 0, происходят во время выполнения программы, поэтому они называются *ошибками времени выполнения*. *Фатальные ошибки* времени выполнения приводят к немедленному завершению программы. При возникновении *нефатальной ошибки* программа может продолжить выполнение, часто — с получением ошибочных результатов.

1.5.3. Написание и выполнение кода в Jupyter Notebook

Дистрибутив Anaconda, установленный вами в разделе «Приступая к работе», включает *Jupyter Notebook* — интерактивную браузерную среду, в которой можно писать и выполнять код, а также комбинировать его с текстом, изображениями и видео. Документы Jupyter Notebook широко применяются в сообществе data science в частности и в более широком научном сообществе в целом. Они рассматриваются как предпочтительный механизм проведения аналитических исследований данных и распространения *воспроизводимых* результатов. Среда Jupyter Notebook поддерживает все больше языков программирования.

Для вашего удобства весь исходный код книги также предоставляется в формате документов Jupyter Notebook, которые вы можете просто загружать и выполнять. В этом разделе используется интерфейс *JupyterLab*, который позволяет управлять файлами документов Notebook и другими файлами, используемыми в них (например, графическими изображениями и видеороликами). Как вы убедитесь, JupyterLab также позволяет легко писать код, выполнять его, просматривать результаты, вносить изменения и снова выполнять его.

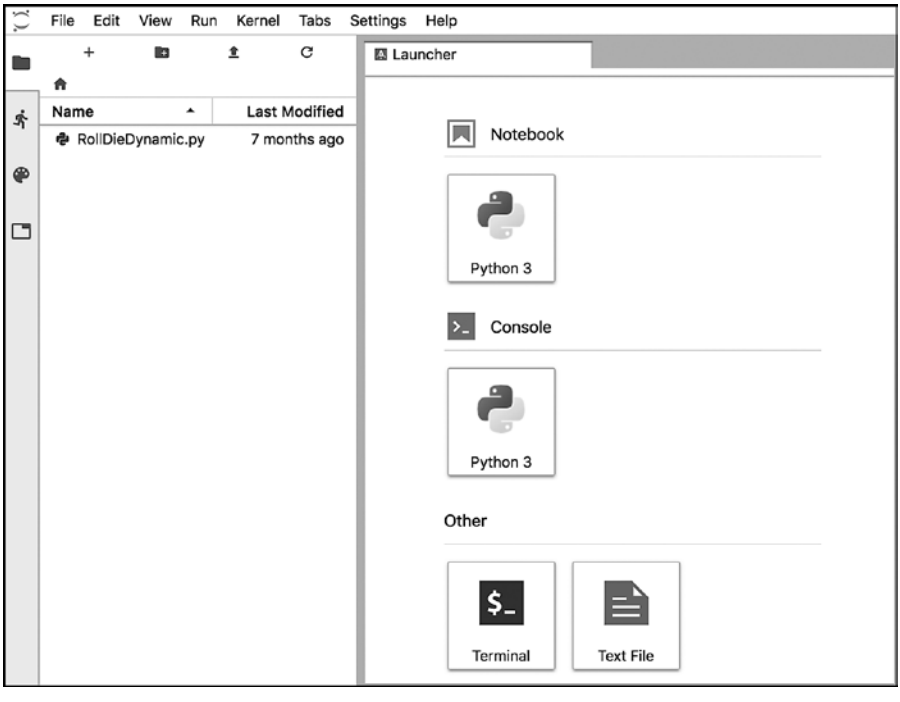
Вы увидите, что программирование в Jupyter Notebook имеет много общего с работой в IPython, более того, документы Jupyter Notebooks используют IPython по умолчанию. В этом разделе вы создадите документ Notebook, добавите в него код из раздела 1.5.1 и выполните этот код.

Открытие JupyterLab в браузере

Чтобы открыть JupyterLab, перейдите в папку примеров ch01 архива примеров в терминале, окне командной строки или приглашении Anaconda (раздел 1.5.2), введите следующую команду и нажмите Enter (или *Return*):

```
jupyter lab
```

Команда запускает на вашем компьютере сервер Jupyter Notebook и открывает JupyterLab в браузере по умолчанию; содержимое папки ch01 отображается на вкладке File Browser в левой части интерфейса JupyterLab:



Сервер Jupyter Notebook позволяет загружать и выполнять документы Jupyter Notebook в браузере. На вкладке JupyterLab Files вы можете сделать двойной щелчок на файле, чтобы открыть его в правой части окна, где в настоящее время отображается вкладка Launcher. Каждый файл, который вы открываете, отображается на отдельной вкладке в этой части окна. Если вы случайно закроете браузер, то сможете повторно открыть JupyterLab — для этого достаточно ввести в браузере следующий адрес:

```
http://localhost:8888/lab
```

Создание нового документа Notebook Jupyter

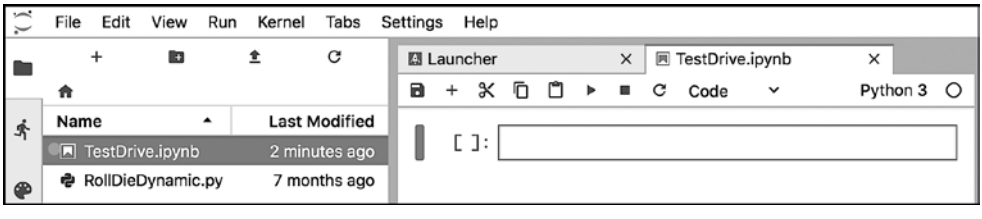
На вкладке Launcher в Notebook щелкните на кнопке Python 3, чтобы создать новый документ Jupyter Notebook с именем Untitled.ipynb. В этом документе можно вводить и выполнять код Python 3. Расширение файла .ipynb является сокращением от «IPython Notebook» — исходное название Jupyter Notebook.

Переименование документа Notebook

Переименуйте Untitled.ipynb в TestDrive.ipynb:

1. Щелкните правой кнопкой мыши на вкладку Untitled.ipynb и выберите Rename Notebook.
2. Измените имя на TestDrive.ipynb и нажмите RENAME.

Верхняя часть JupyterLab должна выглядеть так:

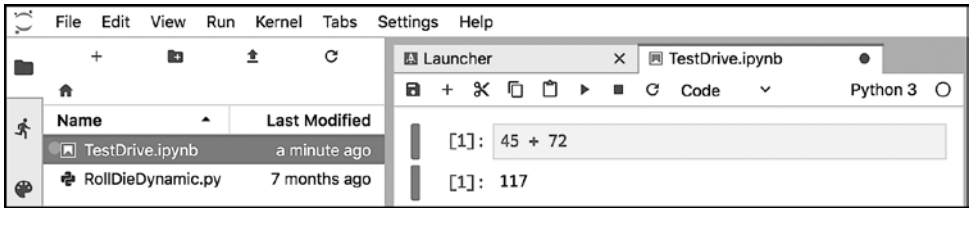


Вычисление выражений

Рабочей единицей в документе Notebook является *ячейка*, в которой вводятся фрагменты кода. По умолчанию новый документ Notebook состоит из одной ячейки (прямоугольник в документе TestDrive.ipynb), но вы можете добавить в нее и другие ячейки. Обозначение [ ]: слева от ячейки показывает, где Jupyter Notebook будет выводить номер фрагмента ячейки после ее выполнения. Щелкните в ячейке и введите следующее выражение:

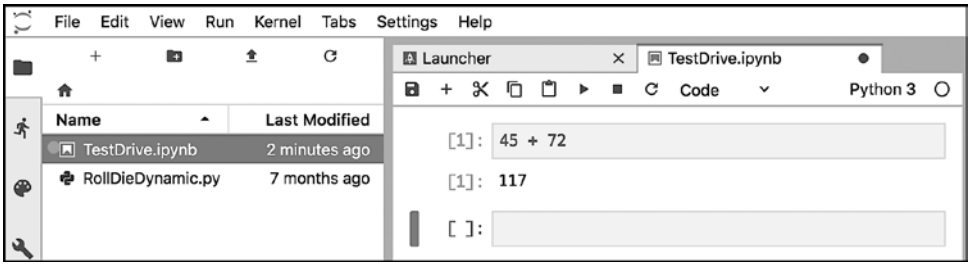
```
45 + 72
```

Чтобы выполнить код текущей ячейки, нажмите Ctrl + Enter (или control + Enter). JupyterLab выполняет код в IPython, после чего выводит результаты под ячейкой:



Добавление и выполнение дополнительной ячейки

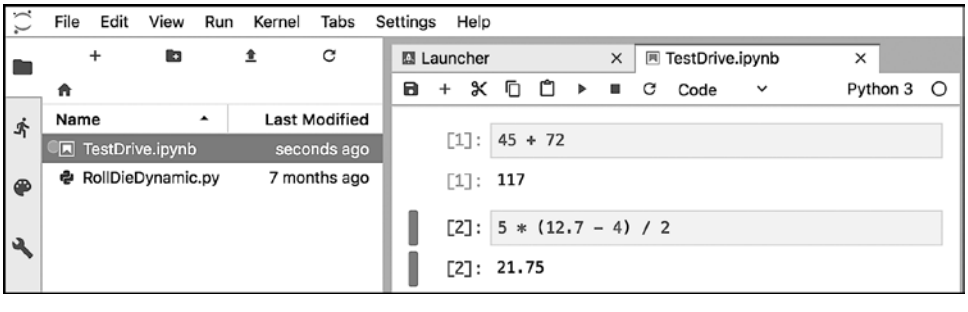
Попробуем вычислить более сложное выражение. Сначала щелкните на кнопке + на панели инструментов над первой ячейкой документа — она добавляет новую ячейку под текущей:



Щелкните в новой ячейке и введите выражение

```
5 * (12.7 - 4) / 2
```

и выполните код ячейки комбинацией клавиш Ctrl + Enter (или control + Enter):



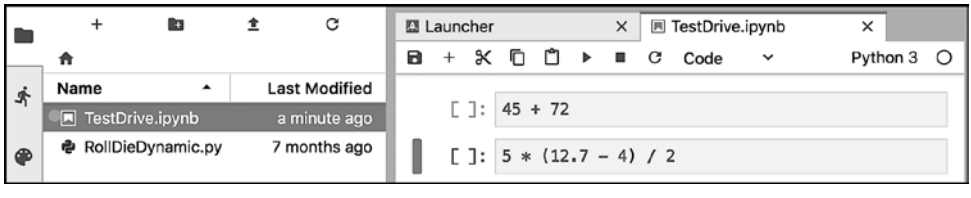
Сохранение документа Notebook

Если ваш документ Notebook содержит несохраненные изменения, то знак X на вкладке заменяется на ●. Чтобы сохранить документ Notebook, откройте меню File в JupyterLab (не в верхней части окна браузера) и выберите команду Save Notebook.

Документы Notebook с примерами каждой главы

Для удобства примеры каждой главы также предоставляются в форме готовых к исполнению документов Notebook без вывода. Вы сможете проработать их фрагмент за фрагментом и увидеть результат при выполнении каждого фрагмента.

Чтобы показать, как загрузить существующий документ Notebook и выполнить его ячейки, выполним сброс блокнота TestDrive.ipynb, чтобы удалить выходные данные и номера фрагментов. Документ вернется в исходное состояние, аналогичное состоянию примеров последующих глав. В меню Kernel выберите команду Restart Kernel and Clear All Outputs..., затем щелкните на кнопке RESTART. Эта команда также полезна в тех случаях, когда вы захотите повторно выполнить фрагменты документа Notebook. Документ должен выглядеть примерно так:



В меню File выберите команду Save Notebook. Щелкните на кнопке X на вкладке TestDrive.ipynb, чтобы закрыть документ.

Открытие и выполнение существующего документа Notebook

Запустив JupyterLab из папки примеров конкретной главы, вы сможете открывать документы Notebook из любой папки или любой из его вложенных папок. Обнаружив нужный документ, откройте его двойным щелчком. Снова откройте файл TestDrive.ipynb. После того как документ Notebook будет открыт, вы сможете выполнить каждую его ячейку по отдельности, как это делалось ранее в этом разделе, или же выполнить всю книгу сразу. Для этого откройте меню Run и выберите команду Run All Cells. Все ячейки выполняются по порядку, а выходные данные каждой ячейки выводятся под этой ячейкой.

Закрытие JupyterLab

Завершив работу с JupyterLab, закройте вкладку в браузере, а затем в терминале, командном интерпретаторе или приглашении Anaconda, из которого была запущена оболочка JupyterLab, дважды нажмите Ctrl + c (или control + c).

Советы по работе с JupyterLab

Возможно, следующие советы пригодятся вам при работе с JupyterLab:

- Если требуется вводить и выполнять много фрагментов, то можно выполнить текущую ячейку *и* добавить новую комбинацией клавиш Shift + Enter вместо Ctrl + Enter (или control + Enter).
- В более поздних главах отдельные фрагменты, которые вы вводите в документах Jupyter Notebook, будут состоять из многих строк кода. Чтобы в каждой ячейке выводились номера строк, откройте в JupyterLab меню View и выберите команду Show line numbers.

О работе с JupyterLab

JupyterLab содержит много возможностей, которые будут полезны в вашей работе. Мы рекомендуем ознакомиться с вводным курсом JupyterLab от создателей Jupyter по адресу:

https://jupyterlab.readthedocs.io/en/stable/index.html

Краткий обзор открывается по ссылке Overview в разделе GETTING STARTED. В разделе USER GUIDE приведены вводные уроки по работе с интерфейсом JupyterLab, работе с файлами, текстовым редактором, документами Notebook и др.

1.6. Облачные вычисления и «интернет вещей»

1.6.1. Облачные вычисления

В наши дни все больше вычислений осуществляется «в облаке», то есть выполняется в распределенном виде в интернете по всему миру. Многие приложения, используемые в повседневной работе, зависят от *облачных сервисов*, использующих масштабные кластеры вычислительных ресурсов (компьютеры, процессоры, память, дисковое пространство и т. д.) и баз данных, взаимодействующих по интернету друг с другом и с приложениями, которыми вы пользуетесь. Сервис, доступ к которому предоставляется по интернету, называется *веб-сервисом*. Как вы увидите, использование облачных сервисов в Python часто сводится к простому созданию программных объектов и взаимодействию с ними. Созданные объекты используют веб-сервисы, которые связываются с облаком за вас.

В примерах глав 11–16 вы будете работать со многими сервисами на базе облака:

- В главах 12 и 16 веб-сервисы Twitter будут использоваться (через библиотеку Python Tweepy) для получения информации о конкретных пользователях Twitter, поиска твитов за последние семь дней и получения потока твитов в процессе их появления (то есть в реальном времени).
- В главах 11 и 12 библиотека Python TextBlob применяется для перевода текста с одного языка на другой. Во внутренней реализации TextBlob использует веб-сервис Google Translate для выполнения перевода.
- В главе 13 будут использоваться сервисы IBM Watson: Text to Speech, Speech to Text и Translate. Мы реализуем приложение-переводчик, позволяющее зачитать вопрос на английском языке, преобразовать речь в текст, переводить текст на испанский и зачитывать испанский текст. Затем пользователь произносит ответ на испанском (если вы не говорите на испанском, используйте предоставленный аудиофайл), приложение преобразует речь в текст, переводит текст на английский и зачитывает ответ на английском. Благодаря демонстрационным приложениям IBM Watson мы также поэкспериментируем в главе 13 с некоторыми другими облачными сервисами Watson.
- В главе 16 мы поработаем с сервисом Microsoft Azure HDInsight и другими веб-сервисами Azure в ходе реализации приложений больших данных на базе технологий Apache Hadoop и Spark. Azure — группа облачных сервисов от компании Microsoft.
- В главе 16 веб-сервис Dweet.io будет использоваться для моделирования подключенного к интернету термостата, который публикует показания температуры. Также при помощи сервиса на базе веб-технологий мы создадим панель, отображающую динамически изменяемый график температуры и предупреждающую пользователя о том, что температура стала слишком высокой или слишком низкой.
- В главе 16 веб-панель будет использована для наглядного представления моделируемого потока оперативных данных с датчиков веб-сервиса PubNub. Также будет построено приложение Python для визуального представления моделируемого потока изменений биржевых котировок PubNub.

В большинстве случаев мы будем создавать объекты Python, которые взаимодействуют с веб-сервисами за вас, скрывая технические подробности обращений к этим сервисам по интернету.

Гибриды

Методология разработки *гибридных приложений* позволяет быстро разрабатывать мощные продукты за счет объединения веб-сервисов (часто бесплатных) и других форм информационных поставок — как мы поступим при разработке приложения-переводчика на базе IBM Watson.

Одно из первых гибридных приложений объединяло списки продаваемых объектов недвижимости, предоставляемые сайтом http://www.craigslist.org, с картографическими средствами Google Maps; таким образом строились карты с местоположением домов, продаваемых или сдаваемых в аренду в заданной области.

На сайте ProgrammableWeb (http://www.programmableweb.com/) представлен каталог более 20 750 веб-сервисов и почти 8000 гибридных приложений. Также здесь имеются учебные руководства и примеры кода работы с веб-сервисами и создания собственных гибридных приложений. По данным сайта, к числу наиболее часто используемых веб-сервисов принадлежат Facebook, Google Maps, Twitter и YouTube.

1.6.2. «Интернет вещей»

Интернет уже нельзя рассматривать как сеть, объединяющую *компьютеры*, — сегодня уже говорят об *«интернете вещей»*, или *IoT* (*Internet of Things*). *Вещью* считается любой объект, обладающий IP-адресом и способностью отправлять, а в некоторых случаях и автоматически получать данные по интернету. Вот несколько примеров:

- автомобиль с транспондером для оплаты дорожных сборов;
- мониторы доступности места для парковки в гараже;
- кардиомониторы в человеческом организме;
- мониторы качества воды;
- интеллектуальный датчик, сообщающий о расходе энергии;
- датчики излучения;
- устройства отслеживания товаров на складе;
- мобильные приложения, способные отслеживать ваше местонахождение и перемещения;

- умные термостаты, регулирующие температуру в комнате на основании прогнозов погоды и текущей активности в доме;
- устройства для умных домов.

По данным statista.com, в наши дни уже используются свыше 23 миллиардов IoT-устройств, а к 2025 году их число может превысить 75 миллиардов[1].

1.7. Насколько велики большие данные?

Для специалистов по компьютерной теории и data science данные играют не менее важную роль, чем написание программ. По данным IBM, в мире ежедневно создаются приблизительно 2,5 квинтиллиона байт (2,5 *экзабайта*) данных[2], а 90% мировых данных были созданы за последние два года[3]. По материалам IDC, к 2025 году ежегодный объем глобального генерирования данных достигнет уровня 175 *зеттабайт* (175 триллионов гигабайт, или 175 миллиардов терабайт)[4]. Рассмотрим наиболее популярные единицы объема данных.

Мегабайт (Мбайт)

Один мегабайт составляет приблизительно 1 миллион (а на самом деле 2^{20}) байт. Многие файлы, используемыми нами в повседневной работе, занимают один или несколько мегабайтов. Примеры:

- Аудиофайлы в формате MP3 — минута качественного звука в формате MP3 занимает от 1 до 2,4 Мбайт[5].
- Фотографии — фотография в формате JPEG, сделанная на цифровую камеру, может занимать от 8 до 10 Мбайт.
- Видеоданные — на камеру смартфона можно записывать видео с разным разрешением. Каждая минута видео может занимать много мегабайтов. Например, на одном из наших iPhone приложение настроек камеры со-

---

1 https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

2 https://www.ibm.com/blogs/watson/2016/06/welcome-to-the-world-of-a-i/.

3 https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf.

4 https://www.networkworld.com/article/3325397/storage/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html.

5 https://www.audiomountain.com/tech/audio-file-size.html.

общало, что видео в разрешении 1080p при 30 кадрах в секунду (FPS) занимает 130 Мбайт/мин, а видео в разрешении 4K при 30 FPS занимает 350 Мбайт/мин.

Гигабайт (Гбайт)

Один гигабайт составляет приблизительно 1000 мегабайт (а точнее, 2^{30} байт). Двуслойный диск DVD способен вмещать до 8,5 Гбайт[1], что соответствует:

- до 141 часа аудио в формате MP3;
- приблизительно 1000 фотографий на 16-мегапиксельную камеру;
- приблизительно 7,7 минуты видео 1080p при 30 FPS, или
- приблизительно 2,85 минуты видео 4K при 30 FPS.

Современные диски наивысшей емкости Ultra HD Blu-ray вмещают до 100 Гбайт видеоданных[2]. Потоковая передача фильма в разрешении 4K может происходить со скоростью от 7 до 10 Гбайт (с высокой степенью сжатия).

Терабайт (Тбайт)

Один терабайт составляет около 1000 гигабайт (а точнее, 2^{40} байт). Емкость современного диска для настольных компьютеров достигает 15 Тбайт[3], что соответствует:

- приблизительно 28 годам аудио в формате MP3;
- приблизительно 1,68 миллиона фотографий на 16-мегапиксельную камеру;
- приблизительно 226 часов видео 1080p при 30 FPS;
- приблизительно 84 часов видео 4K при 30 FPS.

У Nimbus Data наибольший объем диска SSD составляет 100 Тбайт; он вмещает в 6,67 раза больше данных, чем приведенные выше примеры аудио-, фото- и видеоданных для 15-терабайтных дисков[4].

---

[1] https://en.wikipedia.org/wiki/DVD.

[2] https://en.wikipedia.org/wiki/Ultra_HD_Blu-ray.

[3] https://www.zdnet.com/article/worlds-biggest-hard-drive-meet-western-digitals-15tb-monster/.

[4] https://www.cinema5d.com/nimbus-data-100tb-ssd-worlds-largest-ssd/.

Петабайт, экзабайт и зеттабайт

Почти 4 миллиарда людей в интернете ежедневно создают около 2,5 квинтиллиона байт данных[1], то есть 2500 петабайт (1 петабайт составляет около 1000 терабайт) или 2,5 экзабайта (1 экзабайт составляет около 1000 петабайт). По материалам статьи из *AnalyticsWeek* за март 2016 года, через пять лет к интернету будет подключено свыше 50 миллиардов устройств (в основном через «интернет вещей», который рассматривается в разделах 1.6.2 и 16.8), а к 2020 году *на каждого человека на планете* ежесекундно будет производиться 1,7 мегабайта новых данных[2]. Для современного уровня населения (приблизительно 7,7 миллиарда людей[3]) это соответствует приблизительно:

- 13 петабайт новых данных в секунду;
- 780 петабайт в минуту;
- 46 800 петабайт (46,8 экзабайт) в час;
- 1123 экзабайта в день, то есть 1,123 зеттабайта (ЗБ) в день (1 зеттабайт составляет около 1000 экзабайт).

Это эквивалентно приблизительно 5,5 миллиона часов (более 600 лет) видео в разрешении 4K в день, или приблизительно 116 миллиардам фотографий в день!

Дополнительная статистика по большим данным

На сайте https://www.internetlive-stats.com доступна различная занимательная статистика, в том числе количество:

- поисков в Google;
- твитов;
- просмотров роликов на YouTube;
- загрузок фотографий на Instagram за сегодняшний день.

Вы можете получить доступ и к иной интересующей вас статистической информации. Например, по данным сайта видно, что за 2018 год создано более 250 *миллиардов* твитов.

---

[1] https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf.

[2] https://analyticsweek.com/content/big-data-facts/.

[3] https://en.wikipedia.org/wiki/World_population.

Приведем еще несколько интересных фактов из области больших данных:

- Ежечасно пользователи YouTube загружают 24 000 часов видео, а за день на YouTube просматривается почти 1 миллиард часов видео[1].
- Каждую секунду в интернете проходит объем трафика 51 773 Гбайт (или 51,773 Тбайт), отправляется 7894 твита, проводится 64 332 поиска в Google[2].
- На Facebook ежедневно ставятся 800 миллионов **лайков**[3], отправляются 60 миллионов эмодзи[4] и проводятся свыше 2 миллиардов поисков в более чем 2,5 триллиона постов Facebook, накопившихся с момента появления сайта[5].
- Компания Planet использует 142 спутника, которые раз в сутки фотографируют весь массив суши на планете. Таким образом, ежедневно к имеющимся данным добавляется 1 миллион изображений и 7 Тбайт новых данных. Компания вместе с партнерами применяет машинное обучение для повышения урожайности, контроля численности кораблей в портах и контроля за исчезновением лесов. В том, что касается потери леса в джунглях Амазонки, Уилл Маршалл (Will Marshall), исполнительный директор Planet, сказал: «Прежде мы могли обнаружить большую дыру на месте амазонских джунглей лишь спустя несколько лет после ее образования. Сегодня возможно каждый день подсчитывать число деревьев, растущих на планете»[6].

У Domo, Inc. есть хорошая инфографика «Данные никогда не спят 6.0», которая наглядно показывает, сколько данных генерируется *каждую минуту*[7]:

- 473 400 твитов;
- 2 083 333 фотографии публикуются в Snapchat;
- 97 222 часа видео просматриваются в сервисе Netflix;
- отправляются 12 986 111 текстовых сообщений;
- 49 380 сообщений добавляется в Instagram;

---

1 https://www.brandwatch.com/blog/youtube-stats/.
2 http://www.internetlivestats.com/one-second.
3 https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook.
4 https://mashable.com/2017/07/17/facebook-world-emoji-day/.
5 https://techcrunch.com/2016/07/27/facebook-will-make-you-talk/.
6 https://www.bloomberg.com/news/videos/2017-06-30/learning-from-planet-s-shoe-boxed-sized-satellites-video, 30 июня 2017 г.
7 https://www.domo.com/learn/data-never-sleeps-6.

- по Skype проходят 176 220 звонков;
- в Spotify прослушиваются 750 000 песен;
- в Google проводятся 3 877 140 поисков;
- в YouTube просматриваются 4 333 560 видеороликов.

Непрерывный рост вычислительной мощности

Данных становится все больше и больше, а вместе с ними растет вычислительная мощность, необходимая для их обработки. Производительность современных компьютеров часто измеряется в количестве *операций с плавающей точкой в секунду* (*FLOPS*). Вспомним, что в начале и середине 1990-х скорость самых быстрых суперкомпьютеров измерялась в гигафлопсах (10^9 FLOPS). К концу 1990-х компания Intel выпустила первые суперкомпьютеры с производительностью уровня терафлопс (10^{12} FLOPS). В начале и середине 2000-х скорости достигали сотен терафлопс, после чего в 2008 году компания IBM выпустила первый суперкомпьютер с производительностью уровня петафлопс (10^{15} FLOPS). В настоящее время самый быстрый суперкомпьютер IBM Summit, расположенный в Оукриджской национальной лаборатории Министерства энергетики США, обладает производительностью в 122,3 петафлопс[1].

Распределенные вычисления могут связывать тысячи персональных компьютеров в интернете для достижения еще более высокой производительности. В конце 2016 года сеть Folding@home — распределенная сеть, участники которой выделяют ресурсы своих персональных компьютеров для изучения заболеваний и поиска новых лекарств[2], — была способна достигать суммарной производительности свыше 100 петафлопс. Такие компании, как IBM, в настоящее время работают над созданием суперкомпьютеров, способных достигать производительности уровня экзафлопс (10^{18} FLOPS[3]).

Квантовые компьютеры, разрабатываемые в настоящее время, теоретически могут функционировать на скорости, в 18 000 000 000 000 000 000 раз превышающей скорость современных «традиционных» компьютеров[4]! Это число настолько огромно, что квантовый компьютер теоретически может выполнить за одну секунду неизмеримо больше вычислений, чем было выполнено всеми

---

[1] https://en.wikipedia.org/wiki/FLOPS.

[2] https://en.wikipedia.org/wiki/Folding@home.

[3] https://www.ibm.com/blogs/research/2017/06/supercomputing-weather-model-exascale/.

[4] https://medium.com/@n.biedrzycki/only-god-can-count-that-fast-the-world-of-quantum-computing-406a0a91fcf4.

компьютерами мира с момента появления первого компьютера. Эта почти невообразимая вычислительная мощность ставит под угрозу технологию криптовалют на базе технологии блокчейна (например, Bitcoin). Инженеры уже работают над тем, как адаптировать блокчейн для столь значительного роста вычислительных мощностей[1].

Мощь суперкомпьютеров постепенно проникает из исследовательских лабораторий, где для достижения такой производительности тратятся огромные суммы денег, в бюджетные коммерческие компьютерные системы и даже настольные компьютеры, ноутбуки, планшеты и смартфоны.

Стоимость вычислительной мощности продолжает снижаться, особенно с применением облачных вычислений. Прежде люди спрашивали: «Какой вычислительной мощностью должна обладать моя система, чтобы справиться с *пиковой* нагрузкой?». Сегодня этот подход заменился другим: «Смогу ли я быстро получить в облаке ресурсы, *временно* необходимые для выполнения моих наиболее насущных вычислительных задач?». Таким образом, все чаще вы платите только за то, что фактически используется вами для решения конкретной задачи.

Обработка данных требует значительных затрат электроэнергии

Объем данных от устройств, подключенных к интернету, стремительно растет, а обработка этих данных требует колоссальных объемов энергии. Так, энергозатраты на обработку данных в 2015 году росли на 20% в год и составляли приблизительно от 3 до 5% мирового производства энергии. Но к 2025 году общие затраты на обработку данных должны достичь 20%[2].

Другим серьезным потребителем электроэнергии является криптовалюта Bitcoin на базе блокчейна. На обработку всего одной транзакции Bitcoin расходуется примерно столько же энергии, сколько расходуется средним американским домом за неделю! Расходы энергии обусловлены процессом, который используется «майнерами» Bitcoin для подтверждения достоверности данных транзакции[3].

---

[1] https://singularityhub.com/2017/11/05/is-quantum-computing-an-existential-threat-to-blockchain-technology/.

[2] https://www.theguardian.com/environment/2017/dec/11/tsunami-of-data-could-consume-fifth-global-electricity-by-2025.

[3] https://motherboard.vice.com/en_us/article/ywbbpm/bitcoin-mining-electricity-consumption-ethereum-energy-climate-change.

По некоторым оценкам, за год транзакции Bitcoin расходуют больше энергии, чем экономика многих стран[1]. Так, в совокупности Bitcoin и Ethereum (другая популярная криптовалюта и платформа на базе блокчейна) расходуют больше энергии, чем Израиль, и почти столько же, сколько расходует Греция[2].

В 2018 году компания Morgan Stanley предсказывала, что «затраты электроэнергии на создание криптовалют в этом году могут превзойти прогнозируемые компанией глобальные затраты на электротранспорт в 2025 году»[3]. Такая ситуация становится небезопасной, особенно с учетом огромного интереса к блокчейновым технологиям даже за пределами стремительно развивающихся криптовалют. Сообщество блокчейна работает над решением проблемы[4,5].

Потенциал больших данных

Вероятно, в ближайшие годы в области больших данных будет наблюдаться экспоненциальный рост. В перспективе количество вычислительных устройств достигнет 50 миллиардов, и трудно представить, сколько еще их появится за несколько ближайших десятилетий. Очень важно, чтобы всеми этими данными могли пользоваться не только бизнес, правительственные организации, военные, но и частные лица.

Интересно, что некоторых из лучших работ по поводу больших данных, data science, искусственного интеллекта и т. д. были написаны в авторитетных коммерческих организациях: J. P. Morgan, McKinsey и др. Привлекательность больших данных для большого бизнеса бесспорна, особенно если принять во внимание стремительно развивающиеся результаты. Многие компании осуществляют значительные инвестиции в области технологий, описанных в книге (большие данные, машинное обучение, глубокое обучение, обработка естественных языков, и т. д.), и добиваются ценных результатов. Это заставляет конкурентов также вкладывать средства, что стремительно повышает спрос на профессионалов с опытом работы в области больших данных и теории вычислений. По всей вероятности, эта тенденция сохранится в течение многих лет.

---

1 https://digiconomist.net/bitcoin-energy-consumption.

2 https://digiconomist.net/ethereum-energy-consumption.

3 https://www.morganstanley.com/ideas/cryptocurrencies-global-utilities.

4 https://www.technologyreview.com/s/609480/bitcoin-uses-massive-amounts-of-energy-but-theres-a-plan-to-fix-it/.

5 http://mashable.com/2017/12/01/bitcoin-energy/

1.7.1. Анализ больших данных

Аналитическая обработка данных — зрелая, хорошо проработанная научная и профессиональная дисциплина. Сам термин «анализ данных» появился в 1962 году[1], хотя люди применяли статистику для анализа данных в течение тысяч лет, еще со времен Древнего Египта[2]. Анализ больших данных следует считать более современным явлением — так, термин «большие данные» появился около 2000 года[3]. Назовем четыре основные характеристики больших данных, обычно называемые «четырьмя V»[4,5]:

1. Объем (Volume) — количество данных, производимых в мире, растет с экспоненциальной скоростью.
2. Скорость (Velocity) — темпы производства данных, их перемещения между организациями и изменения данных стремительно растут[6,7,8].
3. Разнообразие (Variety) — некогда данные в основном были алфавитно-цифровыми (то есть состояли из символов алфавита, цифр, знаков препинания и некоторых специальных знаков). В наши дни они также включают графику, аудио, видео и данные от стремительно растущего числа датчиков «интернета вещей», установленных в жилищах, коммерческих организациях, транспортных средствах, городах и т. д.
4. Достоверность (Veracity) — характеристика, указывающая на то, являются ли данными полными и точными, и, как следствие, позволяющая ответить на вопросы вроде: «Можно ли на них полагаться при принятии критических решений?», «Насколько реальна информация?» и пр.

Большая часть данных сейчас создается в цифровой форме, относится к самым *разным* типам, достигает невероятных *объемов* и перемещается с потрясающей *скоростью*. Закон Мура и связанные с ним наблюдения позволяют нам организовать экономичное хранение данных, ускорить их обработку и перемещение — и все это на скоростях, экспоненциально растущих со временем. Хранилища цифровых данных обладают настолько огромной емкостью, низ-

---

[1] https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/.

[2] https://www.flydata.com/blog/a-brief-history-of-data-analysis/.

[3] https://bits.blogs.nytimes.com/2013/02/01/the-origins-of-big-data-an-etymological-detective-story/.

[4] https://www.ibmbigdatahub.com/infographic/four-vs-big-data.

[5] Во многих статьях и публикациях в этот список добавляется много других «слов на букву V».

[6] https://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/.

[7] https://whatis.techtarget.com/definition/3Vs.

[8] https://www.forbes.com/sites/brentdykes/2017/06/28/big-data-forget-volume-and-variety-focus-on-velocity.

кой стоимостью и выдающейся компактностью, что мы сейчас можем удобно и экономично хранить *все* создаваемые цифровые данные[1]. Таковы наиболее существенные характеристики больших данных. Но давайте двинемся дальше.

Следующая цитата Ричарда Хемминга (Richard W. Hamming), пусть и относящаяся к 1962 году, задает тон для всей этой книги:

«Целью вычислений является понимание сути, а не числа»[2].

Область data science производит новую, более глубокую, неочевидную и более ценную аналитическую информацию в потрясающем темпе. Инфраструктура больших данных рассматривается в главе 16 на практических примерах использования баз данных NoSQL, программирования Hadoop MapReduce, Spark, потокового программирования IoT и т. д. Чтобы получить некоторое представление о роли больших данных в промышленности, правительственных и научных организациях, взгляните на следующий график[3]:

http://mattturck.com/wp-content/uploads/2018/07/Matt_Turck_FirstMark_Big_Data_Landscape_2018_Final.png

1.7.2. Data Science и большие данные изменяют ситуацию: практические примеры

Область data science стремительно развивается, потому что она производит важные результаты, которые действительно влияют на ситуацию. Некоторые примеры практического применения data science и больших данных перечислены в таблице. Надеемся, эти примеры, а также некоторые другие примеры из книги вдохновят читателей на поиск новых сценариев использования данных в работе. Аналитика больших данных приводила к повышению прибыли, улучшению отношений с клиентами и даже к повышению процента побед у спортивных команд при сокращении затрат на игроков[4,5,6].

---

[1] http://www.lesk.com/mlesk/ksg97/ksg.html. [К статье Майкла Леска (Michael Lesk) нас привела статья: https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/.]

[2] Hamming, R. W., *Numerical Methods for Scientists and Engineers* (New York, NY., McGraw Hill, 1962). [К книге Хемминга и его цитате нас привела следующая статья: https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/.]

[3] Turck, M., and J. Hao, «Great Power, Great Responsibility: The 2018 Big Data & AI Landscape», http://mattturck.com/bigdata2018/.

[4] Sawchik, T., *Big Data Baseball: Math, Miracles, and the End of a 20-Year Losing* Streak (New York, Flat Iron Books, 2015).

[5] Ayres, I., *Super Crunchers* (Bantam Books, 2007), с. 7–10.

[6] Lewis, M., *Moneyball: The Art of Winning an Unfair Game* (W. W. Norton & Company, 2004).

Некоторые примеры практического применения data science и больших данных

автоматизированное генерирование подписей к изображениям
автоматизированное генерирование субтитров
автоматизированные инвестиции
автоматизированные суда
автономные автомобили
агенты по обслуживанию клиентов
адаптированная диета
анализ пространственных данных
анализ социальных графов
анализ тональности высказываний
беспилотные летательные аппараты
борьба с похищением личных данных
борьба с фишингом
ведение электронных историй болезни
визуализация данных
визуальный поиск продуктов
выявление мошенничества
выявление рыночных тенденций
выявление спама
выявление сходства
генерирование музыки
геномика и здравоохранение
географические информационные системы (GIS)
глубокий анализ данных
голосовой поиск
диагностика рака груди
диагностика сердечных заболеваний
диагностика/лечение рака
диагностическая медицина
динамическое построение маршрута
динамическое ценообразование
дистанционная медицина
игры
идентификация вызывающего абонента
идентификация жертв стихийных бедствий
иммунотерапия
индивидуализированная медицина
интеллектуальные системы управления движением
«интернет вещей» (IoT) и мониторинг медицинских устройств
«интернет вещей» (IoT) и прогнозы погоды
картирование головного мозга
картография
качество обслуживания клиентов
кибербезопасность
классификация рукописного текста
личные помощники
маркетинг
маркетинговая аналитика
минимизация рисков
наем и тренерская работа в спорте
обнаружение аномалий
обнаружение вредоносных программ
обнаружение новых вирусов
обнаружение эмоций
обслуживание клиентов
оценка заемщика
оценка недвижимости
оценка успеваемости студентов

перевод естественного языка
перевод иностранных языков
персонализация лекарственных средств
персонализация покупок
повышение урожайности
поиск новых препаратов
поиск попутчиков
помощь инвалидам
построение рекомендуемых маршрутов
построение сводки текста
предиктивный анализ
предотвращение вспышек болезней
предотвращение злоупотреблений опиатами
предотвращение краж
предотвращение оттока клиентов
предотвращение террористических актов
преступления: предиктивная полицейская деятельность
преступления: предотвращение
преступления: прогнозирование места
преступления: прогнозирование рецидивизма
прогнозирование биржевого рынка
прогнозирование вспышек болезней
прогнозирование выживания при раке
прогнозирование погоды
прогнозирование продаж, зависящих от погоды
прогнозирование регистрации абитуриентов
прогнозирование результатов лечения
прогнозирование рисков автострахования
программы лояльности
профилактическая медицина
распознавание выражения лица
распознавание голоса
распознавание изображений
редактирование генома CRISPR
рекомендательные системы
роботизированные финансовые советники
секвенирование человеческого генома
системы GPS
складской учет
снижение частоты повторной госпитализации
сокращение выбросов углерода
сокращение загрязнения окружающей среды
сокращение избыточного резервирования
сокращение энергопотребления
социальная аналитика
удержание клиентов
удовлетворение запросов потребителей
улучшение безопасности
улучшение результатов лечения
умные города
умные датчики
умные дома
умные помощники
умные термостаты
услуги на основе определения местоположения
установление страховых тарифов
фитнес-мониторинг
чтение языка знаков
экономика совместного потребления

1.8. Практический пример: использование больших данных в мобильном приложении

Навигационное GPS-приложение Google Waze с его 90 миллионами активных пользователей в месяц[1] стало одним из самых успешных приложений, использующих большие данные. Ранние устройства и приложения GPS-навигации использовали статические карты и координаты GPS для определения оптимального маршрута к конечной точке, но при этом не могли динамически адаптироваться к изменяющейся дорожной обстановке.

Waze обрабатывает огромные объемы *краудсорсинговых* данных, то есть данных, непрерывно поставляемых пользователями и их устройствами по всему миру. Поступающие данные анализируются для определения оптимального маршрута к месту назначения за минимальное время. Для решения этой задачи Waze использует подключение вашего смартфона к интернету. Приложение автоматически отправляет обновленную информацию о местоположении на свои серверы (при условии что вы разрешите это делать). Данные используются для динамического изменения маршрута на основании текущей дорожной обстановки и для настройки карт. Пользователи также сообщают другую информацию: о пробках, строительстве, препятствиях, транспорте на аварийных полосах, полицейских постах, ценах на бензин и др. Обо всем этом Waze затем оповещает других водителей.

Для предоставления своего сервиса Waze использует множество технологий. Ниже приведен список технологий, которые с большой вероятностью используются этим сервисом (подробнее о некоторых из них см. главы 11–16). Итак:

- Многие приложения, создаваемые в наши дни, используют и некоторые программные продукты с открытым кодом. В этой книге мы будем пользоваться многими библиотеками и служебными программами с открытым кодом.
- Waze передает информацию через интернет. В наши дни такие данные часто передаются в формате JSON (JavaScript Object Notation); мы представим этот формат в главе 9 и будем использовать его в последующих главах. Данные JSON обычно скрываются от вас используемыми библиотеками.

[1] https://www.waze.com/brands/drivers/.

- Waze использует синтез речи, для того чтобы передавать пользователю рекомендации и сигналы, и распознавание речи для понимания речевых команд. Мы поговорим о функциональности синтеза и распознавания речи IBM Watson в главе 13.
- После того как приложение Waze преобразует речевую команду на естественном языке в текст, оно должно выбрать правильное действие, что требует обработки естественного языка (NLP). Мы представим технологию NLP в главе 11 и используем ее в нескольких последующих главах.
- Waze отображает динамические визуализации (в частности, оповещения и карты). Также Waze предоставляет возможность взаимодействия с картами: перемещения, увеличения/уменьшения и т. д. Мы рассмотрим динамические визуализации (с использованием Matplotlib и Seaborn) и отображением интерактивных карт (с использованием Folium) в главах 12 и 16.
- Waze использует ваш мобильный телефон / смартфон как устройство потоковой передачи IoT. Каждое подобное устройство представляет собой GPS-датчик, постоянно передающий данные Waze по интернету. В главе 16 мы представим IoT и будем работать с моделируемыми потоковыми датчиками IoT.
- Waze получает IoT-потоки от миллионов устройств мобильной связи одновременно. Приложение должно обрабатывать, сохранять и анализировать эти данные для обновления карт на ваших устройствах, для отображения и речевого воспроизведения оповещений и, возможно, для обновления указаний для водителя. Все это требует массово-параллельных возможностей обработки данных, реализованных на базе компьютерных кластеров в облаке. В главе 16 мы представим различные инфраструктурные технологии больших данных для получения потоковых данных, сохранения этих больших данных в соответствующих базах данных и их обработки с применением программ и оборудования, обладающего массово-параллельными вычислительными возможностями.
- Waze использует возможности искусственного интеллекта для решения задач анализа данных, позволяющих спрогнозировать лучший маршрут на основании полученной информации. В главах 14 и 15 машинное обучение и глубокое обучение соответственно используются для анализа огромных объемов данных и формирования прогнозов на основании этих данных.

- Вероятно, Waze хранит свою маршрутную информацию в графовой базе данных. Такие базы данных способны эффективно вычислять кратчайшие маршруты. Графовые базы данных, такие как Neo4J, будут представлены в главе 16.
- Многие машины в наше время оснащаются устройствами, позволяющими им «видеть» другие машины и препятствия. Например, они используются для реализации автоматизированных систем торможения и являются ключевым компонентом технологии автономных («необитаемых») автомобилей. Вместо того чтобы полагаться на пользователей, сообщающих о препятствиях и машинах, стоящих на обочине, навигационные приложения пользуются камерами и другими датчиками, применяют методы распознавания изображений с глубоким обучением для анализа изображений «на ходу» и автоматической передачи информации об обнаруженных объектах. Глубокое обучение в области распознавания изображений рассматривается в главе 15.

1.9. Введение в data science: искусственный интеллект — на пересечении компьютерной теории и data science

Когда ребенок впервые открывает глаза, «видит» ли он лица своих родителей? Имеет ли он представление о том, что такое лицо, — или хотя бы о том, что такое форма? Как бы то ни было, ребенок должен «познать» окружающий мир. Именно этим занимается искусственный интеллект (AI, Artificial Intelligence). Он обрабатывает колоссальные объемы данных и обучается по ним. Искусственный интеллект применяется для игр, для реализации широкого спектра приложений распознавания изображений, управления автономными машинами, обучения роботов, предназначенных для выполнения новых операций, диагностики медицинских состояний, перевода речи на другие языки практически в реальном времени, создания виртуальных собеседников, способных отвечать на произвольные вопросы на основании огромных баз знаний, и многих других целей. Кто бы мог представить всего несколько лет назад, что автономные машины с искусственным интеллектом появятся на дорогах, став вскоре обыденным явлением? А сегодня в этой области идет острая конкуренция! Конечной целью всех этих усилий является *общий искусственный интеллект*, то есть искусственный интеллект, способный действовать разумно на уровне человека. Многим эта мысль кажется пугающей.

Вехи развития искусственного интеллекта

Несколько ключевых точек в развитии искусственного интеллекта особенно сильно повлияли на внимание и воображение людей. Из-за них общественность начала думать о том, что искусственный интеллект становится реальностью, а бизнес стал искать возможности коммерческого применения AI:

- В 1997 году в ходе шахматного поединка между компьютерной системой *IBM DeepBlue* и гроссмейстером Гарри Каспаровым DeepBlue стал первым компьютером, победившим действующего чемпиона мира в условиях турнира[1]. Компания IBM загрузила в DeepBlue сотни тысяч записей партий гроссмейстеров[2]. Вычислительная мощь DeepBlue позволяла оценивать до 200 миллионов ходов в секунду[3]! Это классический пример использования больших данных. К слову, по окончании состязания компания IBM получила премию Фредкина, учрежденную Университетом Карнеги — Меллона, который в 1980 году предложил 100 000 долларов создателю первого компьютера, который победит действующего чемпиона мира по шахматам[4].
- В 2011 году суперкомпьютер *IBM Watson* победил двух сильнейших игроков в телевизионной викторине Jeopardy! в матче с призовым фондом в 1 000 000 долларов. Watson одновременно использовал сотни методов анализа языка для поиска правильных ответов в 200 миллионах страницах контента (включая всю «Википедию»), для хранения которых требовалось 4 терабайта[5][6]. Для тренировки Watson использовались **методы машинного обучения** и **глубокого обучения**[7]. В главе 13 обсуждается IBM Watson, а в главе 14 — машинное обучение.
- Го — настольная игра, созданная в Китае тысячи лет назад[8], — обычно считалась одной из самых сложных из когда-либо изобретенных игр с 10^{170} потенциально возможными позициями на доске[9]. Чтобы представить, на-

---

1 https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov.
2 https://ru.wikipedia.org/wiki/Deep_Blue.
3 Ibid.
4 https://articles.latimes.com/1997/jul/30/news/mn-17696.
5 https://www.techrepublic.com/article/ibm-watson-the-inside-story-of-how-the-jeopardy-winning-supercomputer-was-born-and-what-it-wants-to-do-next/.
6 https://en.wikipedia.org/wiki/Watson_(computer).
7 https://www.aaai.org/Magazine/Watson/watson.php, AI Magazine, осень 2010 г.
8 http://www.usgo.org/brief-history-go.
9 https://www.pbs.org/newshour/science/google-artificial-intelligence-beats-champion-at-worlds-most-complicated-board-game.

сколько огромно это число, поясним, что, по некоторым оценкам, известная часть Вселенной содержит (всего лишь) от 10^{78} до 10^{87} атомов[1,2]! Программа *AlphaGo*, созданная группой DeepMind (принадлежит Google), использовала *методы глубокого обучения с двумя нейронными сетями и одержала победу над чемпионом Европы Фан Хуэем*. Заметим, го считается намного более сложной игрой, чем шахматы. Нейронные сети и глубокое обучение рассматриваются в главе 15.

- Позднее компания Google обобщила искусственный интеллект *AlphaGo* для создания AlphaZero — игрового искусственного интеллекта, который *самостоятельно обучается играть в другие игры*. В декабре 2017 года программа AlphaZero узнала правила и научилась играть в шахматы менее чем за 4 часа, используя методы обучения с подкреплением. Затем программа победила шахматную программу Stockfish 8, которая являлась чемпионом мира, в матче из 100 партий, причем все партии завершились ее победой или ничьей. После *обучения* го в течение всего 8 часов AlphaZero смогла играть со своим предшественником AlphaGo и победила в 60 партиях из 100[3].

Личные воспоминания

Один из авторов этой книги, Харви Дейтел, будучи студентом бакалавриата Массачусетского технологического университета в середине 1960-х проходил магистерский курс у Марвина Мински (Marvin Minsky), одного из основателей дисциплины искусственного интеллекта (AI). Вот что вспоминает об этом Харви:

> *Профессор Мински раздавал курсовые проекты. Он предложил нам поразмышлять над тем, что такое интеллект и как заставить компьютер сделать что-то разумное. Наша оценка по этому курсу будет почти полностью зависеть от этого проекта. Полная свобода выбора!*
>
> *Я исследовал стандартизированные IQ-тесты, проводимые в учебных заведениях для оценки интеллектуальных способностей учеников. Будучи математиком по своей природе, я решил взяться за часто встречающуюся в IQ-тестах задачу по предсказанию следующего числа в серии чисел произ-*

[1] https://www.universetoday.com/36302/atoms-in-the-universe/.

[2] https://en.wikipedia.org/wiki/Observable_universe#Matter_content.

[3] https://www.theguardian.com/technology/2017/dec/07/alphazero-google-deepmind-ai-beats-champion-program-teaching-itself-to-play-four-hours.

вольной длины и сложности. Я использовал интерактивный Lisp, работающий на одной из ранних моделей DEC PDP-1, и моя программа предсказания чисел справлялась с довольно сложными задачами, выходившими далеко за рамки тех, что встречались в IQ-тестах. Возможности Lisp по рекурсивной обработке списков произвольной длины были именно тем, что было нужно для выполнения требований проекта. Замечу, Python также поддерживает рекурсию и обобщенную работку со списками (глава 5).

Я опробовал программу на многих сокурсниках из MIT. Они выдумывали числовые серии и вводили их в моей программе. PDP-1 некоторое время «думала» (иногда довольно долго) и почти всегда выдавала правильный ответ. Но потом возникли трудности. Один из моих сокурсников ввел последовательность 14, 23, 34 и 42. Программа взялась за работу. PDP-1 долгое время размышляла, но так и не смогла предсказать следующее число. Я тоже не мог этого сделать. Сокурсник предложил мне немного подумать и пообещал открыть ответ на следующий день; он утверждал, что это простая последовательность. Все мои усилия были напрасными.

На следующий день сокурсник сообщил, что следующее число — 57, но я не понял почему. Он снова предложил подумать до завтра, а на следующий день заявил, что следующее число — 125. Это нисколько не помогло, я был в замешательстве. Видя это, сокурсник пояснил, что последовательность состояла... из номеров сквозных улиц с двусторонним движением на Манхэттене. Я закричал: «Нечестно!», но он парировал — задача соответствует заданному критерию. Я смотрел на мир с точки зрения математика — его взгляд был шире. За прошедшие годы я опробовал эту последовательность на друзьях, родственниках и коллегах. Несколько жителей Манхэттена дали правильный ответ. Для таких задач моей программе было нужно нечто большее, чем математические знания, — ей требовались (потенциально огромные) знания о мире.

Watson и Big Data открывают новые возможности

Харви продолжает:

Когда мы с Полом начали работать над этой книгой о Python, нас сразу же привлекла история о том, как суперкомпьютер IBM Watson использовал большие данные и методы искусственного интеллекта (в частности, обработку естественного языка (NLP) и машинное обучение), чтобы одержать победу над двумя сильнейшими игроками в Jeopardy! Мы поняли, что Watson, вероятно, сможет решать такие задачи, потому что в его память были загружены планы городов мира и много других подобных данных. Все это

усилило наш интерес к глубокому изучению больших данных и современным методам искусственного интеллекта и помогло сформировать структуру глав 11–16 этой книги.

Следует заметить, что все практические примеры реализации data science в главах 11–16 либо уходят корнями к технологиям искусственного интеллекта либо описывают программы и оборудование больших данных, позволяющие специалистам по компьютерной теории и data science эффективно реализовать революционные решения на базе AI.

AI: область, в которой есть задачи, но нет решений

В течение многих десятилетий искусственный интеллект рассматривался как область, в которой есть задачи, но *нет* решений. Дело в том, что после того, как конкретная задача была решена, люди начинают говорить: «Что ж, это не интеллект, а просто компьютерная программа, которая указывает компьютеру, что нужно делать». Однако благодаря методам машинного обучения (глава 14) и глубокого обучения (глава 15) мы уже не программируем компьютер для решения *конкретных* задач. Вместо этого мы предлагаем компьютерам решать задачи, обучаясь по данным, и обычно по очень большим объемам данных.

Для решения многих самых интересных и сложных задач применялись методы глубокого обучения. Компания Google (одна из многих подобных себе) ведет тысячи проектов глубокого обучения, причем их число быстро растет[1,2]. По мере изложения материала книги мы расскажем вам о многих ультрасовременных технологиях искусственного интеллекта, больших данных и облачных технологиях.

1.10. Итоги

В этой главе представлена терминология и концепции, закладывающие фундамент для программирования на языке Python, которому будут посвящены главы 2–10, а также практические примеры применения больших данных, искусственного интеллекта и облачных технологий (подробнее об этом в главах 11–16). Мы обсудили концепции объектно-ориентированного программирования и выяснили, почему язык Python стал настолько популярным. Также в этой главе рассмотрена стандартная библиотека Python и различные

[1] http://theweek.com/speedreads/654463/google-more-than-1000-artificial-intelligence-projects-works.
[2] https://www.zdnet.com/article/google-says-exponential-growth-of-ai-is-changing-nature-of-compute/.

библиотеки data science, избавляющие программистов от необходимости «изобретать велосипед». В дальнейшем эти библиотеки будут использоваться для создания программных объектов, с которыми вы будете взаимодействовать для решения серьезных задач при умеренном объеме кода. В главе рассмотрены и три примера, показывающие, как выполнять код Python с помощью интерпретатора IPython и Jupyter Notebook. Мы представили облачные технологии и концепцию «интернета вещей» (IoT), закладывающие основу для современных приложений, которые будут разрабатываться в главах 11–16.

Также мы обсудили, насколько велики «большие данные» и с какой скоростью они становятся еще больше; рассмотрели пример использования больших данных в мобильном навигационном приложении Waze, в котором многие современные технологии задействованы для генерирования динамических указаний по выбору маршрута, по возможности быстро и безопасно приводящих вас к конечной точке маршрута. Мы пояснили, в каких главах книги используются многие из этих технологий. Глава завершается разделом «Введение в data science», в котором рассматривается ключевая область на стыке компьютерной теории и data science — искусственный интеллект.

2

Введение в программирование Python

В этой главе...

- Ввод фрагментов кода и немедленный просмотр результатов в интерактивном режиме IPython.
- Написание простых команд и сценариев Python.
- Создание переменных для хранения данных.
- Знакомство со встроенными типами данных.
- Арифметические операторы и операторы сравнения, их приоритеты.
- Строки в одинарных, двойных и тройных кавычках.
- Использование встроенной функции `print` для вывода текста.
- Использование встроенной функции `input` для запроса данных с клавиатуры и получения этих данных для использования в программе.
- Преобразование текста в целочисленные значения встроенной функцией `int`.
- Использование операторов сравнения и команды `if` для принятия решений о выполнении команды или группы команд.
- Объекты и динамическая типизация в Python.
- Получение типа объекта встроенной функцией `type`.

2.1. Введение

В этой главе представлены основы программирования на Python, а также приведены примеры, демонстрирующие ключевые возможности языка. Предполагается, что вы прочитали раздел с описанием экспериментов с IPython в главе 1, где представлен интерпретатор IPython и продемонстрированы его возможности для вычисления простых арифметических выражений.

2.2. Переменные и команды присваивания

Напомним, мы использовали интерактивный режим IPython как калькулятор для выражений:

```
In [1]: 45 + 72
Out[1]: 117
```

Теперь создадим переменную `x` для хранения целого числа 7:

```
In [2]: x = 7
```

Фрагмент [2] является *командой* (statement). Каждая команда определяет выполняемую операцию. Предыдущая команда создает `x` и использует *знак равенства* (`=`), для того чтобы назначить `x` значение. Большинство команд завершается в конце строки, хотя команда также может охватывать сразу несколько строк. Следующая команда создает переменную `y` и присваивает ей значение 3:

```
In [3]: y = 3
```

Теперь значения `x` и `y` могут использоваться в выражениях:

```
In [4]: x + y
Out[4]: 10
```

Вычисления в командах присваивания

Следующая команда складывает значения переменных `x` и `y`, присваивая результат переменной `total`, значение которой затем выводится:

```
In [5]: total = x + y

In [6]: total
Out[6]: 10
```

Знак = не является оператором. Часть справа от знака = всегда выполняется первой, после чего результат присваивается переменной слева от знака.

Стиль Python

Руководство по стилю для кода Python[1] помогает писать код, соответствующий общепринятым соглашениям об оформлении кода Python. Руководство рекомендует вставлять по одному пробелу с каждой стороны от знака присваивания = и бинарных операторов (таких как +), чтобы код лучше читался.

Имена переменных

Имя переменной, такое как `x`, является *идентификатором*. Каждый идентификатор может состоять из букв, цифр и символов подчеркивания (_), но не может начинаться с цифры. В языке Python учитывается *регистр символов*, так что идентификаторы `Number` и `number` считаются *различными*, потому что один начинается с прописной буквы, а другой — со строчной.

Типы

Каждое значение в Python обладает *типом*, который сообщает, какие данные представляет это значение. Чтобы получить информацию о типе значения, воспользуйтесь встроенной *функцией* Python *type*:

```
In [7]: type(x)
Out[7]: int

In [8]: type(10.5)
Out[8]: float
```

Переменная `x` содержит целое число 7 (из фрагмента [2]), поэтому Python выводит `int` (сокращение от «integer»). Значение `10.5` является числом с плавающей точкой, поэтому Python выводит строку `float`.

2.3. Арифметические операторы

В табл. 2.1 перечислены *арифметические операторы*, среди которых встречаются некоторые знаки, не используемые в алгебре.

[1] https://www.python.org/dev/peps/pep-0008/.

Таблица 2.1. Арифметические операторы Python

Операция Python	Арифметический оператор	Алгебраическое выражение	Выражение Python
Сложение	+	$f + 7$	`f + 7`
Вычитание	–	$p - c$	`p - c`
Умножение	*	$b \cdot m$	`b * m`
Возведение в степень	**	x^y	`x ** y`
Деление	/	x/y, или $\frac{x}{y}$, или $x \div y$	`x / y`
Целочисленное деление	//	$[x/y]$, или $\left\lfloor \frac{x}{y} \right\rfloor$, или $[x \div y]$	`x // y`
Остаток от деления	%	$r \bmod s$	`r % s`

Умножение (*)

В Python в качестве *оператора умножения* используется знак * (*звездочка*):

```
In [1]: 7 * 4
Out[1]: 28
```

Возведение в степень (**)

Оператор возведения в степень (**) возводит одно значение в степень, заданную другим значением:

```
In [2]: 2 ** 10
Out[2]: 1024
```

Для вычисления квадратного корня можно воспользоваться показателем степени `1/2` (то есть `0.5`):

```
In [3]: 9 ** (1 / 2)
Out[3]: 3.0
```

Деление (/) и деление с округлением (//)

Оператор деления (/) делит числитель на знаменатель; результатом является число с плавающей точкой:

```
In [4]: 7 / 4
Out[4]: 1.75
```

Операция целочисленного деления (`//`) делит числитель на знаменатель; результатом является наибольшее *целое число*, не превышающее результат. Python *отсекает* дробную часть:

```
In [5]: 7 // 4
Out[5]: 1

In [6]: 3 // 5
Out[6]: 0

In [7]: 14 // 7
Out[7]: 2
```

При обычном делении деление –13 на 4 дает результат `-3.25`:

```
In [8]: -13 / 4
Out[8]: -3.25
```

Целочисленное деление дает ближайшее целое число, *не большее* `-3.25`, то есть –4:

```
In [9]: -13 // 4
Out[9]: -4
```

Исключения и трассировка

Деление на нуль оператором `/` или `//` запрещено, а при попытке выполнения такой операции происходит исключение (подробнее об исключениях см. главу 9) — признак возникшей проблемы:

```
In [10]: 123 / 0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-10-cd759d3fcf39> in <module>()
----> 1 123 / 0

ZeroDivisionError: division by zero
```

Сообщая об исключении, Python выдает *трассировку стека*. В трассировке указано, что произошло исключение типа `ZeroDivisionError`, — большинство имен исключений заканчивается суффиксом `Error`. В интерактивном режиме номер фрагмента, вызвавшего исключения, задается числом `10` в строке

```
<ipython-input-10-cd759d3fcf39> in <module>()
```

Строка, начинающаяся с ----->, содержит код, приведший к исключению. Иногда фрагменты содержат более одной строки кода — 1 справа от -----> означает, что исключение возникло в строке 1 внутри фрагмента. Последняя строка содержит имя возникшего исключения, за которым следует двоеточие (:) и сообщение об ошибке с расширенной информацией об исключении:

```
ZeroDivisionError: division by zero
```

Исключение также происходит при попытке использования еще не созданной переменной. Следующий фрагмент пытается прибавить 7 к неопределенной переменной z, что приводит к исключению NameError:

```
In [11]: z + 7
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-f2cdbf4fe75d> in <module>()
----> 1 z + 7

NameError: name 'z' is not defined
```

Оператор вычисления остатка от деления

Оператор вычисления остатка от деления в языке Python (%) получает остаток от целочисленного деления левого операнда на правый:

```
In [12]: 17 % 5
Out[12]: 2
```

В данном случае при делении 17 на 5 мы получаем частное 3 и остаток 2. Этот оператор чаще всего используется с целыми числами, но также может использоваться с другими числовыми типами:

```
In [13]: 7.5 % 3.5
Out[13]: 0.5
```

Линейная форма

Алгебраическая запись вида

$$\frac{a}{b}$$

обычно не поддерживается компиляторами или интерпретаторами. По этой причине алгебраические выражения должны записываться в *линейной форме*

с использованием операторов Python. Приведенное выше выражение должно быть записано в виде `a / b` (или `a//b` для целочисленного деления), чтобы все операторы и операнды выстраивались в одну прямую линию.

Группировка выражений с использованием круглых скобок

Круглые скобки используются для группировки выражений Python, как это происходит в алгебраических выражениях. Например, следующий код умножает на `10` результат выражения `5 + 3`:

```
In [14]: 10 * (5 + 3)
Out[14]: 80
```

Без круглых скобок результат будет *другим*:

```
In [15]: 10 * 5 + 3
Out[15]: 53
```

Круглые скобки *избыточны*, если при их удалении будет получен *тот же* результат.

Правила приоритета операторов

Python применяет операторы в арифметических выражениях с соблюдением *правил приоритета операторов*. Обычно эти правила совпадают с правилами, действующими в алгебре:

1. Выражения в круглых скобках вычисляются первыми, так что при помощи круглых скобок можно обеспечить любой нужный вам порядок вычисления. Круглые скобки обладают наивысшим уровнем приоритета. В выражениях с *вложенными круглыми скобками*, например `(a / (b - c))`, сначала выполняются *внутренние* выражения в круглых скобках, то есть `(b - c)`.
2. Затем выполняются операции возведения в степень. Если выражение содержит несколько операций возведения в степень, Python выполняет их справа налево.
3. Затем выполняются операции умножения, деления и вычисления остатка. Если выражение содержит несколько операций умножения, деления, целочисленного деления и вычисления остатка, Python применяет их слева направо. Операции умножения, деления и вычисления остатка имеют «одинаковые уровни приоритета».

4. Операции сложения и вычитания выполняются в последнюю очередь. Если выражение содержит несколько операций сложения и вычитания, Python применяет их слева направо. Сложение и вычитание тоже имеют одинаковые уровни приоритета.

За полным списком операторов и их приоритетов (по возрастанию) обращайтесь по адресу:

https://docs.python.org/3/reference/expressions.html#operator-precedence

Группировка операторов

Говоря о том, что Python применяет некоторые операторы слева направо, мы подразумеваем *группировку* операторов. Например, в выражении

```
a + b + c
```

операторы сложения (+) применяются слева направо, как если бы в выражении присутствовали круглые скобки `(a + b) + c`. Все операторы Python с одинаковым приоритетом группируются слева направо, кроме операторов возведения в степень (**), которые группируются справа налево.

Избыточные круглые скобки

Избыточные круглые скобки могут использоваться для группировки подвыражений, чтобы смысл выражения стал более понятным. Например, в квадратном многочлене

```
y = a * x ** 2 + b * x + c
```

можно для наглядности расставить круглые скобки:

```
y = (a * (x ** 2)) + (b * x) + c
```

Разбиение сложного выражения на серию нескольких команд с более короткими, простыми выражениями также улучшает читаемость кода.

Типы операндов

Каждый арифметический оператор может использоваться как с целыми числами, так и с числами с плавающей точкой. Если оба операнда являются целыми числами, то результат также является целым числом, кроме оператора

деления (/), который всегда дает число с плавающей точкой. Если оба операнда являются числами с плавающей точкой, то результат является числом с плавающей точкой. Выражения, в которых задействовано целое число и число с плавающей точкой, называются *выражениями со смешанным типом* — они всегда дают результаты с плавающей точкой.

2.4. Функция print и строки, заключенные в одинарные и двойные кавычки

Встроенная *функция* `print` выводит свой аргумент(-ы) в строке текста:

```
In [1]: print('Welcome to Python!')
Welcome to Python!
```

В этом случае аргументом `'Welcome to Python!'` является строка — последовательность символов, заключенная в одинарные кавычки (`'`). В отличие от вычисления выражений в интерактивном режиме, перед выводимым текстом не ставится префикс `Out[1]`. Кроме того, `print` не выводит кавычки, в которые заключена строка, хотя мы скоро покажем, как выводить кавычки в строках.

Строка также может быть заключена в двойные кавычки (`"`):

```
In [2]: print("Welcome to Python!")
Welcome to Python!
```

Программисты Python обычно предпочитают одинарные кавычки. Когда функция `print` выполнит свою задачу, она переводит экранный курсор в начало следующей строки.

Вывод списка элементов, разделенных запятыми

Функция `print` также может получать список аргументов, разделенных запятыми:

```
In [3]: print('Welcome', 'to', 'Python!')
Welcome to Python!
```

Каждый аргумент выводится, отделяясь от следующего аргумента пробелом; в данном случае будет выведен тот же результат, что и в двух предыдущих фрагментах. В примере выводится список строк, разделенных запятыми, но

значения могут относиться к любому типу. В следующей главе мы покажем, как предотвратить автоматическую вставку пробелов между значениями или использовать другой разделитель вместо пробела.

Вывод многострочного текста одной командой

Если в строке встречается обратный слеш (\), она является *управляющим символом*, а в сочетании с непосредственно следующим за ней символом образует *управляющую последовательность*. Так, комбинация `\n` представляет управляющую последовательность для *символа новой строки*, который приказывает функции `print` переместить курсор вывода на следующую строку. В следующем фрагменте для создания многострочного вывода используются три символа новой строки:

```
In [4]: print('Welcome\nto\n\nPython!')
Welcome
to

Python!
```

Другие управляющие последовательности

В табл. 2.2 перечислены часто используемые управляющие последовательности.

Таблица 2.2. Наиболее часто используемые управляющие последовательности Python

Управляющая последовательность	Описание
`\n`	Вставляет в строку символ новой строки. При выводе для каждого символа новой строки экранный курсор перемещается в начало следующей строки
`\t`	Вставляет символ горизонтальной табуляции. При выводе для каждого символа табуляции экранный курсор перемещается к следующей позиции табуляции
`\\`	Вставляет символ обратного слеша
`\"`	Вставляет символ двойной кавычки
`\'`	Вставляет символ одиночной кавычки

Игнорирование разрывов строк

Очень длинную строку (или длинную команду) также можно разбить при выводе; если строка завершается символом \, то разрыв строки игнорируется:

```
In [5]: print('this is a longer string, so we \
   ...: split it over two lines')
this is a longer string, so we split it over two lines
```

Интерпретатор собирает части в одну строку, которая уже не содержит внутренних разрывов. Хотя в строке в приведенном фрагменте присутствует символ обратного слеша, он не является управляющим символом, потому что за ним не следует другой символ.

Вывод значения в выражении

Вычисления также можно выполнять прямо в командах `print`:

```
In [6]: print('Sum is', 7 + 3)
Sum is 10
```

2.5. Строки в тройных кавычках

Ранее мы представили строки, заключаемые в одинарные кавычки (`'`) или в двойные кавычки (`"`). *Строки в тройных кавычках* начинаются и завершаются тремя двойными кавычками (`"""`) или тремя одинарными кавычками (`'''`). *Руководство по стилю для кода Python* рекомендует использовать три двойные кавычки (`"""`). Используйте их для создания:

- многострочных строк;
- строк, содержащих одинарные или двойные кавычки;
- *doc-строк* — рекомендуемого способа документирования целей некоторых компонентов.

Включение кавычек в строки

Строка, заключенная в одинарные кавычки, можно содержать символы двойных кавычек:

```
In [1]: print('Display "hi" in quotes')
Display "hi" in quotes
```

но не одинарные кавычки:

```
In [2]: print('Display 'hi' in quotes')
  File "<ipython-input-2-19bf596ccf72>", line 1
    print('Display 'hi' in quotes')
                    ^
SyntaxError: invalid syntax
```

если только вы не используете управляющую последовательность `\'`:

```
In [3]: print('Display \'hi\' in quotes')
Display 'hi' in quotes
```

Во фрагменте [2] синтаксическая ошибка происходит из-за того, что в строке, заключенной в одинарные кавычки, встречается одинарная кавычка. IPython выводит информацию о строке кода, которая стала причиной синтаксической ошибки, а позиция ошибки обозначается символом ^. Также выводится сообщение о недопустимом синтаксисе (`SyntaxError: invalid syntax`). Строка, заключенная в двойные кавычки, может содержать символы одинарных кавычек:

```
In [4]: print("Display the name O'Brien")
Display the name O'Brien
```

но не двойные кавычки, если только вы не используете управляющую последовательность `\"`:

```
In [5]: print("Display \"hi\" in quotes")
Display "hi" in quotes
```

Чтобы обойтись без использования последовательностей `\'` и `\"` внутри строк, такие строки можно заключить в тройные кавычки:

```
In [6]: print("""Display "hi" and 'bye' in quotes""")
Display "hi" and 'bye' in quotes
```

Многострочные строки

Следующий фрагмент присваивает многострочную строку, заключенную в тройные кавычки, переменной `triple_quoted_string`:

```
In [7]: triple_quoted_string = """This is a triple-quoted
   ...: string that spans two lines"""
```

IPython знает, что строка остается незавершенной, пока до нажатия Enter не будет введена закрывающая последовательность `"""`. По этой причине

IPython выводит *приглашение* `...:`, в котором можно ввести следующую часть многострочной строки. Это продолжается до тех пор, пока вы не введете последовательность `"""` и не нажмете Enter. Следующий фрагмент выводит `triple_quoted_string`:

```
In [8]: print(triple_quoted_string)
This is a triple-quoted
string that spans two lines
```

Python хранит многострочные строки со встроенными символами новой строки. Если вы используете переменную `triple_quoted_string` для вычисления, вместо того чтобы выводить ее, IPython выводит строку в одинарных кавычках с символом `\n` в той позиции, где во фрагменте [7] была нажата клавиша Enter. Кавычки, которые выводит IPython, показывают, что `triple_quoted_string` является строкой, — они не входят в содержимое строки:

```
In [9]: triple_quoted_string
Out[9]: 'This is a triple-quoted\nstring that spans two lines'
```

2.6. Получение ввода от пользователя

Встроенная *функция* `input` запрашивает данные у пользователя и получает их:

```
In [1]: name = input("What's your name? ")
What's your name? Paul

In [2]: name
Out[2]: 'Paul'

In [3]: print(name)
Paul
```

Выполнение этого фрагмента происходит так:

- Сначала функция `input` выводит свой строковый аргумент (подсказку), чтобы пользователь знал, какие данные ему следует ввести, и ожидает ответа пользователя. Мы ввели строку `Paul` и нажали Enter. Полужирный шрифт использован для того, чтобы отличить ввод пользователя от выводимого текста подсказки.
- Затем функция `input` возвращает эти символы в виде строки, которая может использоваться в программе. В данном случае строка присваивается переменной `name`.

Фрагмент [2] выводит значение `name`. При вычислении `name` значение переменной выводится в одинарных кавычках в виде `'Paul'`, потому что это строка. При выводе `name` (в фрагменте [3]) строка отображается без кавычек. Если пользователь ввел кавычки, они становятся частью строки:

```
In [4]: name = input("What's your name? ")
What's your name? 'Paul'

In [5]: name
Out[5]: "'Paul'"

In [6]: print(name)
'Paul'
```

Функция input всегда возвращает строку

Рассмотрим следующие фрагменты, которые пытаются прочитать два числа и сложить их:

```
In [7]: value1 = input('Enter first number: ')
Enter first number: 7

In [8]: value2 = input('Enter second number: ')
Enter second number: 3

In [9]: value1 + value2
Out[9]: '73'
```

Вместо того чтобы сложить числа `7` и `3` и получить `10`, Python «складывает» строковые значения `'7'` и `'3'`, получая *строку* `'73'`. Такое слияние называется *конкатенацией строк*. Эта операция создает новую строку, состоящую из значения левого операнда, за которым следует значение правого операнда.

Получение целого числа от пользователя

Если вам понадобится целое число, преобразуйте строку в число *функцией* `int`:

```
In [10]: value = input('Enter an integer: ')
Enter an integer: 7

In [11]: value = int(value)

In [12]: value
Out[12]: 7
```

Код во фрагментах [10] и [11] можно объединить:

```
In [13]: another_value = int(input('Enter another integer: '))
Enter another integer: 13

In [14]: another_value
Out[14]: 13
```

Переменные `value` и `another_value` теперь содержат целые числа. При их суммировании будет получен целочисленный результат (вместо конкатенации):

```
In [15]: value + another_value
Out[15]: 20
```

Если строка, переданная `int`, не может быть преобразована в целое число, то происходит ошибка `ValueError`:

```
In [16]: bad_value = int(input('Enter another integer: '))
Enter another integer: hello
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-cd36e6cf8911> in <module>()
----> 1 bad_value = int(input('Enter another integer: '))

ValueError: invalid literal for int() with base 10: 'hello'
```

Функция `int` также может преобразовать значение с плавающей точкой в целое число:

```
In [17]: int(10.5)
Out[17]: 10
```

Для преобразования строк в числа с плавающей точкой используется встроенная *функция* `float`.

2.7. Принятие решений: команда if и операторы сравнения

Условие представляет собой логическое выражение со значением «истина» (`True`) или «ложь» (`False`). Следующее условие сравнивает числа `7` и `4` и проверяет, какое из них больше:

```
In [1]: 7 > 4
Out[1]: True

In [2]: 7 < 4
Out[2]: False
```

`True` и `False` — ключевые слова Python. Использование ключевого слова в качестве идентификатора приводит к ошибке `SyntaxError`. Каждое из ключевых слов `True` и `False` начинается с буквы верхнего регистра.

В табл. 2.3 перечислены *операторы сравнения*, часто используемые в условиях.

Таблица 2.3. Операторы сравнения, часто используемые Python в условиях

Алгебраический оператор	Оператор Python	Пример условия	Смысл
>	>	x > y	x больше y
<	<	x < y	x меньше y
≥	>=	x >= y	x больше или равно y
≤	<=	x <= y	x меньше или равно y
=	==	x == y	x равно y
≠	!=	x != y	x не равно y

Операторы >, <, >= и <= обладают одинаковым приоритетом. Операторы == и != имеют равный приоритет, более низкий, чем у операторов >, <, >= и <=. Присутствие пробела между символами оператора считается синтаксической ошибкой:

```
In [3]: 7 > = 4
  File "<ipython-input-3-5c6e2897f3b3>", line 1
    7 > = 4
        ^
SyntaxError: invalid syntax
```

Другая синтаксическая ошибка происходит при изменении порядка символов в операторах !=, >= и <= (то есть если они записываются в виде =!, => и =<).

Принятие решений в командах if: сценарии

Рассмотрим простую версию *команды* `if`, которая использует условие для принятия решения о том, должна ли выполняться команда (или группа команд). В этом примере программа получает два целых числа от пользователя и сравнивает их шестью последовательными командами `if`, по одной для каждого оператора сравнения. Если условие команды `if` истинно (`True`), то соответствующая команда `print` выполняется; в противном случае она пропускается.

Интерактивный режим IPython удобен для выполнения коротких фрагментов и немедленного просмотра результатов. Если несколько команд должны выполняться как единое целое, обычно вы записываете их в виде *сценария*, который хранится в файле с разрешением .py (сокращение от Python) — например, fig02_01.py для сценария из этого примера. Сценарии также называются программами. За инструкциями по поиску и выполнению сценариев из книги обращайтесь к главе 1.

Каждый раз, когда вы выполняете сценарии, три из шести условий будут истинными. Чтобы продемонстрировать этот факт, мы выполняем сценарий три раза — в одном первое целое число *меньше* второго, во втором два целых числа *равны*, а в третьем первое целое число *больше* второго. Три результата выполнения приведены после сценария.

Каждый раз, когда мы приводим в книге сценарий (вроде показанного ниже), мы сначала кратко опишем его, а затем объясним код сценария после листинга. Номера строк приводятся для вашего удобства — они не являются частью Python. Интегрированная среда позволяет вам выбрать, нужно ли выводить номера строк. Чтобы выполнить этот пример, перейдите в папку примеров этой главы ch02 и введите команду:

```
ipython fig02_01.py
```

Или, если вы уже запустили IPython, используйте команду:

```
run fig02_01.py
 1 # fig02_01.py
 2 """Сравнение целых чисел командами if и операторами сравнения."""
 3
 4 print('Enter two integers, and I will tell you',
 5       'the relationships they satisfy.')
 6
 7 # чтение первого числа
 8 number1 = int(input('Enter first integer: '))
 9
```

```
10 # чтение второго числа
11 number2 = int(input('Enter second integer: '))
12
13 if number1 == number2:
14     print(number1, 'is equal to', number2)
15
16 if number1 != number2:
17     print(number1, 'is not equal to', number2)
18
19 if number1 < number2:
20     print(number1, 'is less than', number2)
21
22 if number1 > number2:
23     print(number1, 'is greater than', number2)
24
25 if number1 <= number2:
26     print(number1, 'is less than or equal to', number2)
27
28 if number1 >= number2:
29     print(number1, 'is greater than or equal to', number2)
```

```
Enter two integers and I will tell you the relationships they satisfy.
Enter first integer: 37
Enter second integer: 42
37 is not equal to 42
37 is less than 42
37 is less than or equal to 42
```

```
Enter two integers and I will tell you the relationships they satisfy.
Enter first integer: 7
Enter second integer: 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

```
Enter two integers and I will tell you the relationships they satisfy.
Enter first integer: 54
Enter second integer: 17
54 is not equal to 17
54 is greater than 17
54 is greater than or equal to 17
```

Комментарии

Строка 1 начинается с символа # (решетка), который указывает, что остаток строки представляет собой *комментарий*:

```
# fig02_01.py
```

Для удобства мы начинаем каждый сценарий с комментария, обозначающего имя файла сценария. Комментарий также может располагаться справа от кода и продолжаться до конца строки.

Doc-строки

В *«Руководстве по стилю для кода Python»* указано, что каждый сценарий должен начинаться с doc-строки, объясняющей назначение сценария, как в строке 2 приведенного листинга:

```
"""Сравнение целых чисел командами if и операторами сравнения."""
```

В более сложных сценариях doc-строка часто занимает несколько строк. В последующих главах doc-строки будут использоваться для описания компонентов сценариев, определяемых вами, — например, новых функций и новых типов, называемых классами. Также мы покажем, как обращаться к doc-строкам из справочной системы IPython.

Пустые строки

Строка 3 остается пустой. Пустые строки и пробелы упрощают чтение кода. Пустые строки, пробелы и символы табуляции совместно называются *пропусками* (white space). Python игнорирует большинство пропусков, — хотя, как вы увидите, некоторые отступы необходимы.

Разбиение длинной команды по строкам

Строки 4–5

```
print('Enter two integers, and I will tell you',
      'the relationships they satisfy.')
```

выводят инструкции для пользователя. Эти инструкции не помещаются в одной строке, поэтому мы разбили их на две части. Напомним, что `print` может выводить несколько значений, которые передаются при вызове в списке, разделенном запятыми, — `print` отделяет каждое значение от следующего пробелом.

Обычно команды записываются по одной на строку. Длинную команду можно разбить на несколько строк при помощи символа продолжения `\`. Python также позволяет разбивать длинные строки с круглыми скобками без использования символов продолжения (как в строках 4–5). Этот способ разбиения

строк считается предпочтительным согласно «*Руководству по стилю для кода Python*». Всегда выбирайте точки разбивки, которые выглядят логично, — например, после запятой в предшествующем вызове `print` или перед оператором в длинном выражении.

Получение целых значений от пользователя

Затем строки 8 и 11 используют встроенные функции `input` и `int`, для того чтобы запросить и прочитать два целочисленных значения от пользователя.

Команды if

Команда `if` в строках 13–14

```
if number1 == number2:
    print(number1, 'is equal to', number2)
```

использует оператор сравнения `==` для определения того, равны ли значения переменных `number1` и `number2`. Если они равны, то условие истинно (`True`), и строка 14 выводит строку текста, которая сообщает, что значения равны. Если какие-либо условия остальных команд `if` истинны (строки 16, 19, 22, 25 и 28), то соответствующая команда `print` выводит строку текста.

Каждая команда `if` состоит из ключевого слова `if`, проверяемого условия и двоеточия (`:`), за которым следует снабженное отступом тело команды, которое называется *набором* (suite). Каждый набор должен состоять из одной или нескольких команд. Заметьте, что пропущенное двоеточие (`:`) после условия — весьма распространенная синтаксическая ошибка.

Отступы в наборах

Python требует, чтобы команды в наборах снабжались отступами. «*Руководство по стилю для кода Python*» рекомендует использовать отступы, состоящие из четырех пробелов — это соглашение используется в книге. В следующей главе будет показано, что неправильные отступы могут привести к ошибкам.

Путаница с == и =

Использование знака равенства (`=`) вместо оператора равенства (`==`) в условии команды `if` — еще одна распространенная синтаксическая ошибка.

Чтобы этого не происходило, == стоит читать «равно», а = — «присваивается». В следующей главе будет показано, что использование == вместо = в команде присваивания может привести к неочевидным ошибкам.

Сцепленные сравнения

Объединение сравнений в цепочку позволяет проверить, принадлежит ли значение некоторому диапазону. Следующее сравнение проверяет, принадлежит ли x диапазону от 1 до 5 включительно:

```
In [1]: x = 3

In [2]: 1 <= x <= 5
Out[2]: True

In [3]: x = 10

In [4]: 1 <= x <= 5
Out[4]: False
```

Приоритет упоминавшихся ранее операторов

Ниже представлен приоритет операторов, упоминавшихся в этой главе.

Таблица 2.4. Приоритет операторов, упоминавшихся в главе 2

Операторы	Группировка	Тип
()	слева направо	круглые скобки
**	справа налево	возведение в степень
* / // %	слева направо	умножение, деление, целочисленное деление, остаток
+ -	слева направо	сложение, вычитание
> <= < >=	слева направо	меньше, меньше или равно, больше, больше или равно
== !=	слева направо	равно, не равно

В табл. 2.4 операторы перечисляются сверху вниз в порядке убывания приоритета. Если вы пишете выражения, содержащие несколько операторов, стоит убедиться в том, что они выполняются в ожидаемом порядке, — в этом вам поможет диаграмма приоритета операторов, расположенная по адресу:

https://docs.python.org/3/reference/expressions.html#operator-precedence

2.8. Объекты и динамическая типизация

Все значения, такие как `7` (целое число), `4.1` (число с плавающей точкой) и `'dog'`, являются объектами. Каждый объект обладает типом и значением:

```
In [1]: type(7)
Out[1]: int

In [2]: type(4.1)
Out[2]: float

In [3]: type('dog')
Out[3]: str
```

Значением объекта являются данные, хранящиеся в объекте. В приведенных выше фрагментах показаны объекты встроенных типов `int` (целые числа), `float` (числа с плавающей точкой) и `str` (строки).

Переменные содержат ссылки на объекты

Когда вы присваиваете объект переменной, это означает, что имя переменной *связывается* с объектом. После этого переменная может использоваться в коде для обращения к значению объекта:

```
In [4]: x = 7

In [5]: x + 10
Out[5]: 17

In [6]: x
Out[6]: 7
```

После присваивания во фрагменте [4] переменная `x` содержит ссылку на целочисленный объект, содержащий значение `7`. Как видно из фрагмента [6], фрагмент [5] не изменяет значение `x`. Значение `x` можно изменить командой:

```
In [7]: x = x + 10

In [8]: x
Out[8]: 17
```

Динамическая типизация

В Python используется *динамическая типизация* — это означает, что тип объекта, на который ссылается переменная, определяется во время выполнения

кода. Чтобы убедиться в этом, можно заново связать переменную `x` с разными объектами и проверить их типы:

```
In [9]: type(x)
Out[9]: int

In [10]: x = 4.1

In [11]: type(x)
Out[11]: float

In [12]: x = 'dog'

In [13]: type(x)
Out[13]: str
```

Уборка мусора

Python создает объекты в памяти и удаляет их из памяти по мере необходимости. После выполнения фрагмента `[10]` переменная `x` ссылается на объект `float`. Целочисленный объект из фрагмента `[7]` уже не связан с переменной. Как будет показано далее, Python автоматически удаляет такие объекты из памяти. Благодаря этому процессу, называемому *уборкой мусора*, вы можете быть уверены в том, что для создаваемых новых объектов будет доступна свободная память.

2.9. Введение в data science: основные описательные статистики

В data science статистические показатели часто используются для описания и обобщения данных. Начнем с представления нескольких *описательных статистик*:

- *минимум* — наименьшее значение в коллекции значений;
- *максимум* — наибольшее значение в коллекции значений;
- *диапазон* — диапазон значений от минимума до максимума;
- *количество* — количество значений в коллекции;
- *сумма* — сумма всех значений в коллекции.

О том, как определяются *количество* и *сумма*, будет рассказано в следующей главе. *Дисперсионные характеристики* (также называемые *характеристиками*

изменчивости), такие как *диапазон*, помогают определить степень разброса значений. В следующих главах также будут представлены и другие дисперсионные характеристики, включая *дисперсию* и *стандартное отклонение*.

Определение минимума среди трех значений

Для начала посмотрим, как определить минимум среди трех значений вручную. Следующий сценарий запрашивает и вводит три значения, находит наименьшее значение при помощи команд `if` и выводит результат:

```
 1 # fig02_02.py
 2 """Находит наименьшее из трех значений."""
 3
 4 number1 = int(input('Enter first integer: '))
 5 number2 = int(input('Enter second integer: '))
 6 number3 = int(input('Enter third integer: '))
 7
 8 minimum = number1
 9
10 if number2 < minimum:
11     minimum = number2
12
13 if number3 < minimum:
14     minimum = number3
15
16 print('Minimum value is', minimum)
```

```
Enter first integer: 12
Enter second integer: 27
Enter third integer: 36
Minimum value is 12
```

```
Enter first integer: 27
Enter second integer: 12
Enter third integer: 36
Minimum value is 12
```

```
Enter first integer: 36
Enter second integer: 27
Enter third integer: 12
Minimum value is 12
```

После того как пользователь введет три значения, программа поочередно обрабатывает эти значения:

- Сначала мы предполагаем, что `number1` содержит наименьшее значение. В строке 8 оно присваивается переменной `minimum`. Конечно, нельзя ис-

ключать, что настоящее наименьшее значение хранится в `number2` или `number3`, поэтому каждую из этих переменных необходимо сравнить с `minimum`.

- Первая команда `if` (строки 10–11) проверяет условие `number2 < minimum`. Если это условие `истинно`, то `number2` присваивается переменной `minimum`.
- Вторая команда `if` (строки 13–14) проверяет условие `number3 < minimum`. Если это условие `истинно`, то `number3` присваивается переменной `minimum`.

После этого переменная `minimum` содержит наименьшее значение, поэтому программа выводит результат. Мы выполнили сценарий три раза, чтобы убедиться в том, что она всегда находит наименьшее значение независимо от того, будет ли оно введено первым, вторым или третьим.

Определение минимума и максимума с использованием встроенных функций min и max

Python содержит много встроенных функций для решения стандартных задач. Встроенные функции `min` и `max` вычисляют, соответственно, минимум и максимум для коллекции значений:

```
In [1]: min(36, 27, 12)
Out[1]: 12

In [2]: max(36, 27, 12)
Out[2]: 36
```

Функции `min` и `max` могут содержать любое количество аргументов.

Определение диапазона значений в коллекции

Диапазон значений представляет собой интервал от минимума до максимума. В данном случае коллекция имеет диапазон значений от 12 до 36. Дисциплина data science в значительной мере посвящена получению информации о данных. Описательные статистические показатели являются важнейшей частью этого процесса, но вы также должны понимать, как интерпретируется статистика. Например, если у вас есть 100 чисел с диапазоном от 12 до 36, эти числа могут быть равномерно распределены в этом диапазоне. Или, наоборот, коллекция может состоять из 99 значений 12 и одного значения 36 или 99 значений 36 и одного значения 12.

Программирование в функциональном стиле: свертка

В этой книге будут представлены различные средства *программирования в функциональном стиле*. Они позволяют писать код более компактный, понятный и простой в *отладке*, то есть в процессе поиска и исправления ошибок. Функции `min` и `max` являются примерами концепции программирования в функциональном стиле, которая называется *сверткой* (reduction). Они сокращают коллекцию значений до *одного* значения. Также к сверткам относятся такие характеристики, как сумма, среднее, дисперсия и стандартное отклонение коллекции значений. Вы также узнаете, как определять собственные свертки.

Далее в разделах «Введение в data science»

В следующих двух главах мы продолжим обсуждение основных описательных статистик, оценивающих параметры, характеризующие *положение центра распределения*, включая *математическое ожидание*, *медиану* и *моду*, а также *дисперсионные характеристики*, включая *дисперсию* и *стандартное отклонение*.

2.10. Итоги

В этой главе мы продолжили обсуждение арифметических вычислений. Переменные использовались для хранения значений и последующего использования. Мы представили арифметические операторы Python и показали, что все выражения должны записываться в линейной форме. Встроенная функция `print` была использована для вывода данных. В этой главе создавались строки в одинарных, двойных и тройных кавычках. Строки в тройных кавычках используются для создания многострочных строк и для встраивания одинарных или двойных кавычек в строки.

При помощи функции `input` программа может запрашивать и получать ввод с клавиатуры от пользователя. Функции `int` и `float` преобразуют строки в числовые значения. Мы представили операторы сравнения Python. Они были использованы в сценарии, который получает два целых числа от пользователя и сравнивает их значения серией команд `if`.

Мы обсудили динамическую типизацию Python и воспользовались встроенной функцией `type` для вывода типа объекта. В завершающей части главы мы представили основные описательные статистики «минимум» и «максимум», использовав их для вычисления диапазона коллекции значений. В следующей главе будут представлены управляющие конструкции Python.

3

Управляющие команды

В этой главе...

- Принятие решений в командах `if`, `if...else` и `if...elif...else`.
- Многократное выполнение команд в циклах `while` и `for`.
- Сокращенные команды присваивания.
- Использование команды `for` и встроенной функции `range` для выполнения действий с сериями значений.
- Повторение, управляемое контрольным значением, в циклах `while`.
- Создание составных условий с логическими операторами `and`, `or` и `not`.
- Прерывание цикла командой `break`.
- Переход к следующей итерации цикла командой `continue`.
- Использование средств программирования в функциональном стиле для написания сценариев, более компактных и понятных, более простых в отладке и параллелизации.

3.1. Введение

В этой главе представлены управляющие команды Python — `if`, `if...else`, `if...elif...else`, `while`, `for`, `break` и `continue`. Команда `for` будет использоваться для повторения, управляемого последовательностью, — вы увидите, что количество элементов в последовательности определяет количество итераций `for`. Встроенная функция `range` будет использоваться для генерирования последовательностей целых чисел.

Мы также представим механизм повторения, управляемого контрольным значением, в циклах `while`, тип `Decimal` из стандартной библиотеки Python для точных финансовых вычислений. Данные форматируются в соответствующих строках с использованием различных спецификаторов. Также будет продемонстрировано применение логических операторов `and`, `or` и `not` для создания составных условий. В разделе «Введение в data science» рассматриваются характеристики, описывающие положение центра распределения — математическое ожидание, медиана и мода, — на примере модуля `statistics` из стандартной библиотеки Python.

3.2. Управляющие команды

В Python имеются три команды выбора, которые исполняют код в зависимости от какого-либо условия, значение которого может быть истинным (`True`) или ложным (`False`).

- Команда `if` выполняет действие, если условие `истинно`, и не делает ничего, если условие `ложно`.
- Команда `if...else` выполняет действие, если условие `истинно`, или выполняет другое действие, если условие `ложно`.
- Команда `if...elif...else` выполняет одно из нескольких возможных действий в зависимости от истинности или ложности *нескольких* условий.

Всюду, где может размещаться одно действие, может размещаться и группа действий.

В Python поддерживаются две *команды перебора в цикле* — `while` и `for`:

- Команда `while` повторяет действие (или группу действий), пока условие остается `истинным`.

- Команда `for` повторяет действие (или группу действий) для каждого элемента в последовательности элементов.

Ключевые слова

Слова `if`, `elif`, `else`, `while`, `for`, `True` и `False` являются ключевыми словами Python. Использование ключевого слова в качестве идентификатора (например, имени переменной) является синтаксической ошибкой. Ключевые слова Python перечислены в табл. 3.1.

Таблица 3.1. Ключевые слова Python

`and`	`as`	`assert`	`async`	`await`
`break`	`class`	`continue`	`def`	`del`
`elif`	`else`	`except`	`False`	`finally`
`for`	`from`	`global`	`if`	`import`
`in`	`is`	`lambda`	`None`	`nonlocal`
`not`	`or`	`pass`	`raise`	`return`
`True`	`try`	`while`	`with`	`yield`

3.3. Команда if

Выполним команду Python `if`:

```
In [1]: grade = 85

In [2]: if grade >= 60:
   ...:     print('Passed')
   ...:
Passed
```

Условие `grade>=60` истинно, поэтому снабженная отступом команда `print` в наборе `if` выводит `'Passed'`.

Отступы наборов

Отступ у набора обязателен; если пропустить его, происходит ошибка `IndentationError`:

```
In [3]: if grade >= 60:
   ...: print('Passed')  # неправильный отступ
  File "<ipython-input-3-f42783904220>", line 2
    print('Passed')  # неправильный отступ
        ^
IndentationError: expected an indented block
```

Ошибка `IndentationError` также происходит и в том случае, если набор содержит более одной команды, но эти команды имеют отступы *разной* величины:

```
In [4]: if grade >= 60:
   ...:     print('Passed')  # отступ в 4 пробела
   ...:   print('Good job!)  # неправильный отступ в 2 пробела
  File <ipython-input-4-8c0d75c127bf>, line 3
    print('Good job!)  # неправильный отступ в 2 пробела
    ^
IndentationError: unindent does not match any outer indentation level
```

Иногда сообщения об ошибках выглядят невразумительно. Того факта, что Python обращает ваше внимание на ту или иную строку, обычно достаточно для того, чтобы вы поняли суть проблемы. Старайтесь последовательно применять соглашения об отступах в своем коде — программы с неравномерными отступами труднее читать.

Любое выражение может интерпретироваться как True или как False

Решения могут приниматься на основании *любых* выражений. Ненулевые значения интерпретируются как `True`. Нуль интерпретируется как `False`.

```
In [5]: if 1:
   ...:     print('Nonzero values are true, so this will print')
   ...:
Nonzero values are true, so this will print

In [6]: if 0:
   ...:     print('Zero is false, so this will not print')

In [7]:
```

Строки, содержащие символы, интерпретируются как `True`; пустые строки (`''`, `""` или `"""""""`) интерпретируются как `False`.

Путаница с == и =

Использование оператора проверки равенства == вместо команды присваивания = может привести к коварным ошибкам. Например, в этом сеансе фрагмент [1] определял переменную grade одновременно с присваиванием:

```
grade = 85
```

Если вместо этого случайно написать:

```
grade == 85
```

переменная grade окажется неопределенной, а вы получите сообщение об ошибке NameError. Если переменная grade была определена до предшествующей команды, то grade == 85 просто дает результат True или False без выполнения присваивания, что является логической ошибкой.

3.4. Команды if...else и if...elif...else

Команда if...else выполняет разные наборы в зависимости от того, истинно или ложно условие.

```
In [1]: grade = 85

In [2]: if grade >= 60:
   ...:     print('Passed')
   ...: else:
   ...:     print('Failed')
   ...:
Passed
```

Приведенное выше условие истинно, поэтому набор if выводит сообщение 'Passed'. Обратите внимание: если нажать Enter после ввода print('Passed'), то IPython снабжает следующую строку отступом из четырех пробелов. Вы должны удалить эти четыре пробела, чтобы набор else: правильно выравнивался по букве i в if.

Следующий код присваивает значение 57 переменной grade, а затем снова включает команду if...else для демонстрации того, что набор else выполняется только при ложном условии:

```
In [3]: grade = 57

In [4]: if grade >= 60:
   ...:     print('Passed')
   ...: else:
   ...:     print('Failed')
   ...:
Failed
```

Для перемещения между фрагментами текущего интерактивного сеанса используются клавиши ↑ и ↓. При нажатии Enter заново выполняется отображаемый фрагмент. Присвойте `grade` значение 99, дважды нажмите клавишу ↑, чтобы вызвать код из фрагмента [4], после чего нажмите Enter, чтобы снова выполнить этот код как фрагмент [6]. Каждому фрагменту, к которому возвращаетесь, присваивается новый идентификатор:

```
In [5]: grade = 99

In [6]: if grade >= 60:
   ...:     print('Passed')
   ...: else:
   ...:     print('Failed')
   ...:
Passed
```

Условные выражения

Иногда в наборах команды `if...else` переменной присваиваются разные значения в зависимости от условия:

```
In [7]: grade = 87

In [8]: if grade >= 60:
   ...:     result = 'Passed'
   ...: else:
   ...:     result = 'Failed'
   ...:
```

После этого можно вывести или получить значение этой переменной:

```
In [9]: result
Out[9]: 'Passed'
```

Такие команды, как в фрагменте [8], можно записать в виде компактного *условного выражения*:

```
In [10]: result = ('Passed' if grade >= 60 else 'Failed')

In [11]: result
Out[11]: 'Passed'
```

Круглые скобки необязательны, но они ясно показывают, что команда присваивает значение условного выражения переменной `result`. Сначала Python вычисляет условие `grade >= 60`:

- Если условие `истинно`, то фрагмент [10] присваивает `result` значение выражения *слева* от `if`, а именно `'Passed'`. Часть `else` не выполняется.
- Если условие `ложно`, то фрагмент [10] присваивает `result` значение выражения *справа* от `else`, а именно `'Failed'`.

Условное выражение также можно напрямую выполнить в интерактивном режиме:

```
In [12]: 'Passed' if grade >= 60 else 'Failed'
Out[12]: 'Passed'
```

Несколько команд в наборе

В следующем коде набор `else` команды `if`...`else` содержит две команды:

```
In [13]: grade = 49
In [14]: if grade >= 60:
    ...:     print('Passed')
    ...: else:
    ...:     print('Failed')
    ...:     print('You must take this course again')
    ...:
Failed
You must take this course again
```

В этом случае значение `grade` меньше 60, поэтому будут выполнены *обе* команды в наборе `else`.

Если второй вызов `print` не снабжен отступом, он не входит в набор `else`. Следовательно, эта команда будет выполняться *всегда*, что приведет к странному и некорректному выводу:

```
In [15]: grade = 100

In [16]: if grade >= 60:
    ...:     print('Passed')
```

```
    ...: else:
    ...:     print('Failed')
    ...: print('You must take this course again')
    ...:
Passed
You must take this course again
```

Команда if...elif...else

Команда `if`...`elif`...`else` позволяет проверить несколько возможных вариантов. Следующий код выводит «A» для значений `grade`, больших либо равных 90, «B» — для значений из диапазона 80–89, «C» — для значений 70–79, «D» — для значений 60–69 и «F» — для других значений. Выполняется только действие для *первого* условия `True`. Фрагмент [18] выводит `C`, потому что значение `grade` равно 77:

```
In [17]: grade = 77

In [18]: if grade >= 90:
    ...:     print('A')
    ...: elif grade >= 80:
    ...:     print('B')
    ...: elif grade >= 70:
    ...:     print('C')
    ...: elif grade >= 60:
    ...:     print('D')
    ...: else:
    ...:     print('F')
    ...:
C
```

Первое условие — `grade >= 90` — `ложно`, поэтому команда `print('A')` пропускается. Второе условие — `grade >= 80` — тоже `ложно`, поэтому команда `print('B')` пропускается. Третье условие — `grade >= 70` — `истинно`, поэтому команда `print('C')` выполняется. Затем весь оставшийся код в команде `if`...`elif`...`else` пропускается. Конструкция `if`...`elif`...`else` работает быстрее серии команд `if`, потому что проверка условий останавливается, как только будет найдено `истинное` условие.

Секция else необязательна

Наличие секции `else` в команде `if`...`elif`...`else` необязательно. Ее присутствие позволяет вам обработать значения, не удовлетворяющие *никаким* из условий.

Если команда `if...elif` без `else` проверяет значение, с которым ни одно из условий не будет `истинным`, то программа не выполняет ни один из наборов — будет выполнена следующая команда после `if...elif`. Если секция `else` присутствует, то она должна следовать *за* последней секцией `elif`; в противном случае происходит ошибка `SyntaxError`.

Логические ошибки

Код с неправильным отступом в фрагменте [16] является примером *нефатальной логической ошибки*. Код выполняется, но выдает неправильные результаты. При наличии *фатальной логической ошибки в сценарии* происходит исключение (например, ошибка `ZeroDivisionError` при попытке деления на 0), Python выдает трассировку стека и завершает сценарий. *Фатальная ошибка в интерактивном режиме* завершает только текущий фрагмент — далее IPython ожидает следующего ввода.

3.5. Команда while

Команда `while` позволяет повторить одно или несколько действий, пока условие остается `истинным`. Воспользуемся командой `while` для нахождения первой степени 3, превышающей 50:

```
In [1]: product = 3

In [2]: while product <= 50:
   ...:     product = product * 3
   ...:

In [3]: product
Out[3]: 81
```

Фрагмент [3] получает значение `product`, то есть 81 — первую степень 3, превышающую 50. Где-то в наборе `while` значение `product` должно изменяться, чтобы условие цикла в какой-то момент стало `ложным`; в противном случае программа зацикливается. В приложениях, запущенных из терминала, приглашения Anaconda или командного интерпретатора, зациклившуюся программу можно прервать нажатием клавиш Ctrl + c или control + c. Также в IDE обычно имеется кнопка на панели инструментов или команда меню для прерывания программы.

3.6. Команда for

Команда `for` позволяет *повторить* действие или несколько действий для каждого элемента *последовательности* элементов. Например, строка представляет собой последовательность отдельных символов. Выведем строку `'Programming'`, разделяя ее символы двумя пробелами:

```
In [1]: for character in 'Programming':
   ...:     print(character, end='  ')
   ...:
P  r  o  g  r  a  m  m  i  n  g
```

Команда `for` выполняется так:

- В начале выполнения команды символ `'P'` из `'Programming'` присваивается *управляющей* переменной между ключевыми словами `for` и `in` — в данном случае `character`.
- Затем выполняется команда в наборе, которая выводит значение `character` с двумя пробелами (подробнее об этом — чуть ниже).
- После выполнения набора Python присваивает `character` следующий элемент последовательности (то есть `'r'` в `'Programming'`), после чего снова выполняет набор.
- Выполнение продолжается, пока в последовательности остаются символы для обработки. В нашем примере команда завершается после вывода буквы `'g'` с двумя пробелами.

Использование управляющей переменной в наборе, как это делается в нашем примере с выводом значения, достаточно распространено, но не является обязательным.

Аргумент end функции print

Встроенная функция `print` выводит свой аргумент(-ы), после чего перемещает курсор на следующую строку. Это поведение можно изменить при помощи аргумента `end`. Например, команда

```
print(character, end='  ')
```

выводит значение `character`, за которым следуют два пробела. Это сделано для того, чтобы все символы выводились в *одной* строке. В Python `end` называется *ключевым аргументом*, но `end` не является ключевым словом Python.

Ключевые аргументы также иногда называются *именованными аргументами*. Ключевой аргумент `end` не является обязательным. Если вы не включите его, то `print` по умолчанию использует символ новой строки (`'\n'`). *Руководство по стилю для кода Python* рекомендует не окружать пробелами знак `=` у ключевого аргумента.

Ключевой аргумент sep функции print

Ключевой аргумент `sep` (сокращение от «separator», то есть «разделитель») задает строку, которая появляется *между* элементами, выводимыми `print`. Если этот аргумент не задан, то `print` по умолчанию использует пробел. Выведем три числа, разделяя их запятой и пробелом (вместо одного пробела):

```
In [2]: print(10, 20, 30, sep=', ')
10, 20, 30
```

Чтобы отказаться от вывода пробелов по умолчанию, используйте `sep=''` (то есть пустую строку).

3.6.1. Итерируемые объекты, списки и итераторы

Последовательность справа от ключевого слова `in` команды `for` должна быть *итерируемым объектом*, то есть объектом, из которого команда `for` может брать элементы по одному, пока не будет обработан последний элемент. Помимо строк, Python поддерживает и другие типы итерируемых последовательностей. Один из самых распространенных примеров — *список*, то есть последовательность разделенных запятыми элементов, заключенная в квадратные скобки (`[` и `]`). Следующий код суммирует пять целых чисел в списке:

```
In [3]: total = 0
In [4]: for number in [2, -3, 0, 17, 9]:
   ...:     total = total + number
   ...:
In [5]: total
Out[5]: 25
```

У каждой последовательности существует *итератор*. Команда `for` использует итератор незаметно для вас, чтобы получать каждый элемент последовательности, пока не останется ни одного необработанного элемента. Итератор напоминает закладку в книге — он всегда знает текущую позицию последовательности, чтобы вернуть следующий элемент по требованию. Списки подробно

рассматриваются в главе 5. Вы увидите, что порядок элементов в списке важен, а элементы списка *изменяемы*.

3.6.2. Встроенная функция range

Используем команду `for` и встроенную *функцию* `range` для выполнения ровно 10 итераций с выводом значений от 0 до 9:

```
In [6]: for counter in range(10):
   ...:     print(counter, end=' ')
   ...:
0 1 2 3 4 5 6 7 8 9
```

Вызов функции `range(10)` создает итерируемый объект, который представляет последовательность целых чисел от 0 и до значения аргумента (10), *не включая* последний, в данном случае 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Команда `for` прекращает работу при завершении обработки последнего целого числа, выданного `range`. Итераторы и итерируемые объекты — два примера средств *программирования в функциональном стиле* в языке Python. Другие средства этой категории еще встретятся вам в книге.

Ошибки смещения на 1

Одна из распространенных ошибок смещения на 1 происходит, когда разработчик полагает, что значение аргумента `range` включается в сгенерированную последовательность. Например, если передать 9 в аргументе `range` при попытке сгенерировать последовательность от 0 до 9, то `range` сгенерирует только числа от 0 до 8.

3.7. Расширенное присваивание

Конструкция *расширенного присваивания* сокращает выражения присваивания, у которых слева и справа от знака присваивания `=` встречается одно имя переменной, как в случае с `total` в примере:

```
for number in [1, 2, 3, 4, 5]:
    total = total + number
```

Фрагмент [2] реализует этот цикл с использованием *команды расширенного присваивания* (`+=`):

```
In [1]: total = 0

In [2]: for number in [1, 2, 3, 4, 5]:
   ...:     total += number  # number прибавляется к total
   ...:

In [3]: total
Out[3]: 15
```

Выражение `+=` в фрагменте [2] сначала прибавляет значение `number` к текущему значению `total`, а затем сохраняет новое значение в `total`. В табл. 3.2 приведены примеры расширенного присваивания.

Таблица 3.2. Примеры расширенного присваивания в Python

Расширенное присваивание	Пример выражения	Объяснение	Присваивает
Предполагается: `c = 3, d = 5, e = 4, f = 2, g = 9, h = 12`			
`+=`	`c += 7`	`c = c + 7`	10 переменной `c`
`-=`	`d -= 4`	`d = d - 4`	1 переменной `d`
`*=`	`e *= 5`	`e = e * 5`	20 переменной `e`
`**=`	`f **= 3`	`f = f ** 3`	8 переменной `f`
`/=`	`g /= 2`	`g = g / 2`	4,5 переменной `g`
`//=`	`g //= 2`	`g = g // 2`	4 переменной `g`
`%=`	`h %= 9`	`h = h % 9`	3 переменной `h`

3.8. Повторение, управляемое последовательностью; отформатированные строки

В этом и следующем разделе решаются две задачи вычисления среднего. Поставлена следующая задача:

> *Группа из 10 студентов пишет контрольную работу. Они получили следующие оценки (целые числа в диапазоне 0–100): 98, 76, 71, 87, 83, 90, 57, 79, 82, 94. Требуется вычислить среднюю оценку группы.*

Приведенный ниже сценарий для решения этой задачи ведет накапливаемую сумму оценок, вычисляет среднее значение и выводит результат. Мы поместили 10 оценок в список, но оценки также можно ввести с клавиатуры (этим мы займемся в следующей главе) или прочитать их из файла (как будет показано в главе 9). О том, как загрузить данные из баз данных SQL и NoSQL, будет рассказано в главе 16.

```
 1 # class_average.py
 2 """Вычисление средней оценки с повторением, управляемым последовательностью."""
 3
 4 # Фаза инициализации
 5 total = 0  # Сумма оценок
 6 grade_counter = 0
 7 grades = [98, 76, 71, 87, 83, 90, 57, 79, 82, 94]  # Список из 10 оценок
 8
 9 # Фаза обработки
10 for grade in grades:
11     total += grade  # Прибавить текущую оценку к накапливаемой сумме
12     grade_counter += 1  # Еще одна оценка была обработана
13
14 # Завершающая фаза
15 average = total / grade_counter
16 print(f'Class average is {average}')
```

```
Class average is 81.7
```

Строки 5–6 создают переменные `total` и `grade_counter` и инициализируют каждую из них через 0. Строка 7

```
grades = [98, 76, 71, 87, 83, 90, 57, 79, 82, 94]  # Список из 10 оценок
```

создает переменную `grades` и инициализирует ее списком из 10 оценок.

Команда `for` обрабатывает каждую оценку из списка `grades`. Строка 11 прибавляет текущую `оценку` к сумме. Затем строка 12 увеличивает на единицу переменную `grade_counter`, в которой отслеживается количество обработанных на данный момент оценок. *Руководство по стилю для кода Python* рекомендует включать пустую строку до и после каждой управляющей команды (как в строках 8 и 13). Когда команда `for` завершается, строка 15 вычисляет среднее значение, а строка 16 выводит его. Позднее в этой главе будет приведен более компактный способ вычисления среднего значения по элементам списка средствами программирования в функциональном стиле.

Форматные строки

В строке 16 простая *форматная строка* (сокращенно *f-строка*) используется для форматирования результата этого сценария; значение `average` вставляется в строку:

```
f'Class average is {average}'
```

Буква `f` перед открывающей кавычкой строки означает, что это форматная строка. Чтобы указать, куда должно вставляться значение, используйте поля в фигурных скобках (`{` и `}`). Поле

```
{average}
```

преобразует значение переменной `average` в строковое представление, а затем меняет `{average}` *заменяющим текстом*. Выражения в полях могут содержать значения, переменные или другие выражения (например, вычисления или вызовы функций). В строке 16 также можно было воспользоваться выражением `total / grade_counter` вместо `average`, но в этом случае строка 15 стала бы лишней.

3.9. Повторение, управляемое контрольным значением

Обобщим задачу вычисления средней оценки. Новая формулировка задачи выглядит так:

> *Разработать программу вычисления средней оценки, обрабатывающей произвольное количество оценок при каждом запуске программы.*

В формулировке не сказано, как выглядят оценки и сколько их будет, поэтому оценки будут вводиться пользователем. Программа обрабатывает произвольное количество оценок. Пользователь вводит оценки по одной до исчерпания списка оценок, а потом вводит *контрольное значение* (также называемое «*сторожевым*», «*фиктивным*» *значением* или «*флагом*»). Оно сообщает программе о том, что оценок больше не будет.

Реализация повторения, управляемого контрольным значением

Следующий сценарий решает проблему вычисления средней оценки с использованием повторения, управляемого контрольным значением. Обратите внимание на проверку возможного деления на ноль. Без этой проверки в программе могла бы возникнуть фатальная логическая ошибка. В главе 9 мы напишем программы, которые распознают такие исключения и выполняют соответствующие действия.

```
 1 # class_average_sentinel.py
 2 """Вычисление средней оценки с повторением, управляемым контрольным значением."""
 3
 4 # Фаза инициализации
 5 total = 0  # Сумма оценок
 6 grade_counter = 0  # Количество введенных оценок
 7
 8 # Фаза обработки
 9 grade = int(input('Enter grade, -1 to end: '))  # Получение оценки
10
11 while grade != -1:
12     total += grade
13     grade_counter += 1
14     grade = int(input('Enter grade, -1 to end: '))
15
16 # Фаза завершения
17 if grade_counter != 0:
18     average = total / grade_counter
19     print(f'Class average is {average:.2f}')
20 else:
21     print('No grades were entered')
```

```
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 72
Enter grade, -1 to end: -1
Class average is 85.67
```

Логика повторения, управляемого контрольным значением

При повторении, управляемом контрольным значением, программа читает первое значение (строка 9) перед входом в команду `while`. Значение, введенное в строке 9, определяет, должно ли управление передаваться в набор `while` (строки 12–14). Если условие в строке 11 ложно, то это означает, что пользователь ввел контрольное значение (–1), и набор выполняться не будет, потому что пользователь не ввел ни одной оценки. Если условие истинно, то набор выполняется, значение `grade` прибавляется к `total`, а значение `grade_counter` увеличивается.

Затем строка 14 получает следующую оценку от пользователя и условие (строка 11) проверяется снова, на этот раз для последней оценки, введенной пользователем. Значение `grade` всегда вводится непосредственно перед проверкой условия `while` в программе; это позволяет определить, является ли только что введенное значение контрольным, перед обработкой этого значения как оценки.

Если введено контрольное значение, то цикл завершается и программа не прибавляет –1 к `total`. В цикле, управляемом контрольным значением, который получает данные от пользователя, подсказки (строки 9 и 14) должны напоминать пользователю о контрольном значении.

Форматирование средней оценки до двух знаков

В этом примере средняя оценка форматируется до двух знаков в дробной части. В форматной строке за выражением в заполнителе может следовать двоеточие (`:`) и *форматный спецификатор*, который описывает, как должен форматироваться заменяющий текст. Форматный спецификатор `.2f` (строка 19) форматирует `среднее значение` как число с плавающей точкой (`f`) с двумя цифрами в дробной части (`.2`). В данном примере сумма оценок была равна 257, что при делении на 3 дает 85,666666666.... Форматирование `среднего значения` со спецификатором `.2f` *округляет* его до сотых, в результате чего заменяющий текст принимает вид `85.67`. Среднее значение, имеющее только одну цифру в дробной части, будет форматироваться с *завершающим нулем* (например, `85.50`). Отметим, что в главе 8 рассматриваются другие средства форматирования строк.

3.10. Подробнее о встроенной функции range

Функция `range` также существует в версиях с двумя и тремя аргументами. Как вы уже видели, версия `range` с одним аргументом генерирует последовательность целых чисел от 0 до значения аргумента, не включая последний. Версия `range` с двумя аргументами генерирует последовательность целых чисел от значения первого аргумента до второго аргумента, не включая последний:

```
In [1]: for number in range(5, 10):
   ...:     print(number, end=' ')
   ...:
5 6 7 8 9
```

Версия `range` с тремя аргументами генерирует последовательность целых чисел от значения первого аргумента до значения второго аргумента, исключая последний, с приращением, *определяемым* значением третьего аргумента:

```
In [2]: for number in range(0, 10, 2):
   ...:     print(number, end=' ')
   ...:
0 2 4 6 8
```

Если третий аргумент отрицателен, то последовательность генерируется от значения первого аргумента *до* значения второго аргумента, исключая последний, с уменьшением, *определяемым* значением третьего аргумента:

```
In [3]: for number in range(10, 0, -2):
   ...:     print(number, end=' ')
   ...:
10 8 6 4 2
```

3.11. Использование типа Decimal для представления денежных сумм

В этом разделе представлена функциональность `Decimal` для точных финансовых вычислений. Если вы работаете в банковской сфере или в другой области, требующей вычислений «до цента», то вам, безусловно, стоит поглубже изучить возможности `Decimal`.

Для большинства научных и других математических применений, в которых используются числа с дробной частью, встроенные в Python числа с плавающей точкой работают достаточно хорошо. Например, применительно к «нормальной» температуре тела 36,6 °C точность в два-три и более знака после запятой не нужна. При отображении цифровым термометром значения температуры тела, равного 36,6 °C, фактическая температура может быть равна, к примеру, 36,5999473210643 °C. Суть в том, что точность 36,6 °C достаточна для большинства ситуаций, в которых используется температура тела.

Значения с плавающей точкой хранятся в двоичном формате (двоичная система счисления была представлена в главе 1). Некоторые значения с плавающей точкой могут быть представлены в двоичном виде только с некоторой погрешностью. Для примера возьмем переменную `amount` с денежной суммой `112,31`. Если вы выведете ее в программе, то все выглядит так, словно она содержит именно то значение, которое ей было присвоено:

```
In [1]: amount = 112.31

In [2]: print(amount)
112.31
```

Но если вывести его с точностью до 20 знаков в дробной части, то вы увидите, что реальное значение с плавающей точкой не равно точно `112,31` — это лишь приближенное значение:

```
In [3]: print(f'{amount:.20f}')
112.31000000000000227374
```

Тем не менее во многих областях требуется *точное* представление чисел с дробной частью. Такие организации, как банки с миллионами и даже миллиардами операций в день, должны проводить их «с точностью до цента». Числа с плавающей точкой могут представить с идеальной точностью некоторые, но отнюдь не все денежные суммы.

Стандартная библиотека Python[1] предоставляет много готовых решений, которые вы можете использовать в своем коде Python, чтобы вам не приходилось изобретать велосипед. Для финансовых вычислений и других областей, требующих точного представления и обработки чисел с дробной частью, стандартная библиотека Python предоставляет тип `Decimal`, который использует специальную схему кодирования чисел для решения проблемы идеальной точности. Эта схема увеличивает затраты памяти на хранение чисел и требует большего времени для выполнения вычислений, но при этом обеспечивает точность, необходимую для финансовых вычислений. Банкам также приходится учитывать и другие аспекты, например использование *справедливого алгоритма округления* при вычислении ежедневных процентов по счетам. Тип `Decimal` предоставляет такие возможности[2].

Импортирование типа Decimal из модуля decimal

Мы уже использовали многие *встроенные типы* в примерах — `int` (для целых чисел, таких как `10`), `float` (для чисел с плавающей точкой, таких как `7.5`) и `str` (для строк, таких как `'Python'`). Тип `Decimal` не встроен в Python. Вместо этого он является частью стандартной библиотеки Python, разделенной на

[1] https://docs.python.org/3.7/library/index.html.

[2] Дополнительная информация о возможностях модуля decimal доступна по адресу https://docs.python.org/3.7/library/decimal.html.

функциональные группы, называемые *модулями*. Модуль `decimal` определяет тип `Decimal` и его функциональность.

Чтобы использовать тип `Decimal`, сначала необходимо импортировать весь модуль `decimal`:

```
import decimal
```

и ссылаться на тип `Decimal` в форме `decimal.Decimal` или же обозначить конкретную импортируемую часть функциональности конструкцией `from...import`:

```
In [4]: from decimal import Decimal
```

Эта команда `импортирует` из модуля `decimal` только тип `Decimal`, чтобы вы могли использовать его в своем коде. Другие формы импортирования будут рассмотрены начиная со следующей главы.

Создание значений Decimal

Обычно значение `Decimal` создается на базе строки:

```
In [5]: principal = Decimal('1000.00')

In [6]: principal
Out[6]: Decimal('1000.00')

In [7]: rate = Decimal('0.05')

In [8]: rate
Out[8]: Decimal('0.05')
```

В дальнейшем мы будем использовать переменные `principal` и `rate` в вычислении сложных процентов.

Вычисления с Decimal

Тип `Decimal` поддерживает стандартные арифметические операторы `+`, `-`, `*`, `/`, `//`, `**` и `%`, а также соответствующие формы расширенного присваивания:

```
In [9]: x = Decimal('10.5')

In [10]: y = Decimal('2')

In [11]: x + y
```

```
Out[11]: Decimal('12.5')

In [12]: x // y
Out[12]: Decimal('5')

In [13]: x += y

In [14]: x
Out[14]: Decimal('12.5')
```

Вы можете выполнять арифметические операции между `Decimal` и целыми числами, но не между `Decimal` и числами с плавающей точкой.

Формулировка задачи вычисления сложных процентов

Воспользуемся типом `Decimal` для *точных* финансовых вычислений на примере сложного процента. Рассмотрим следующую задачу:

> *На сберегательный счет вносится сумма $1000 под 5% годовых. Предполагая, что весь процент остается на счете, требуется вычислить и вывести сумму на счете на конец каждого года на ближайшие 10 лет. Искомая величина вычисляется по формуле:*
>
> $a = p(1 + r)^n$,
>
> *где* p — исходная внесенная сумма (основной капитал);
>
> r — годовой процент;
>
> n — количество лет;
>
> a — сумма на счете на конец n-го года.

Вычисление сложного процента

Для решения этой задачи мы воспользуемся переменными `principal` и `rate`, определенными во фрагментах [5] и [7], и командой `for`, вычисляющей проценты за 10 лет нахождения суммы на счете. Для каждого года цикл выводит отформатированную строку с номером года и сумму на счете на конец каждого года:

```
In [15]: for year in range(1, 11):
    ...:     amount = principal * (1 + rate) ** year
    ...:     print(f'{year:>2}{amount:>10.2f}')
    ...:
 1   1050.00
```

```
 2   1102.50
 3   1157.62
 4   1215.51
 5   1276.28
 6   1340.10
 7   1407.10
 8   1477.46
 9   1551.33
10   1628.89
```

Алгебраическое выражение $(1 + r)^n$ из формулировки записывается в форме

```
(1 + rate) ** year,
```

где переменная `rate` представляет r, а переменная `year` представляет n.

Форматирование года и суммы на счете

Команда

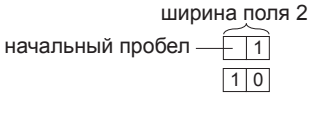
```
print(f'{year:>2}{amount:>10.2f}')
```

использует форматную строку с двумя заполнителями для форматирования выходных данных цикла.

Заполнитель

```
{year:>2}
```

использует форматный спецификатор `>2`, чтобы показать, что значение года должно быть *выровнено по правому краю* (`>`) поля ширины 2 — *ширина поля* определяет количество позиций символов для вывода значения. Для года, состоящего из одной цифры (1–9), спецификатор формата `>2` выводит пробел, за которым следует значение, таким образом, это гарантирует правильное выравнивание лет в первом столбце. Следующая диаграмма показывает, как числа 1 и 10 форматируются с шириной поля 2:



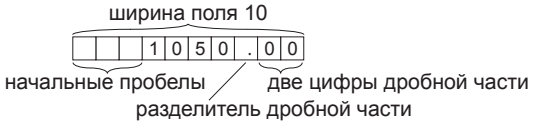
Знак `<` *выравнивает* значение *по левому краю*.

Форматный спецификатор `10.2f` в заполнителе

```
{amount:>10.2f}
```

форматирует `amount` как число с плавающей точкой (`f`), выровненное по правому краю (`>`) поля ширины 10, и двумя знаками в дробной части (`.2`). Такое форматирование суммы *обеспечивает вертикальное выравнивание сумм по разделителю дробной части*, как обычно делается с денежными суммами.

В десяти позициях вывода три крайних правых символа предназначены для разделителя-точки и двух цифр дробной части. Остальные семь позиций заполняются начальными пробелами и цифрами целой части. В этом примере все суммы состоят из четырех цифр, так что каждое число форматируется с тремя *начальными пробелами*. На следующей диаграмме показано форматирование для значения `1050.00`:



3.12. Команды break и continue

Команды `break` и `continue` изменяют логику выполнения цикла. Выполнение команды `break` в `while` или `for` приводит к немедленному выходу из этой команды. В следующем коде `range` генерирует целочисленную последовательность 0–99, но цикл завершается при достижении `number` значения 10:

```
In [1]: for number in range(100):
   ...:     if number == 10:
   ...:         break
   ...:     print(number, end=' ')
   ...:
0 1 2 3 4 5 6 7 8 9
```

В сценарии выполнение продолжится со следующей команды после цикла `for`. Каждая из команд `while` и `for` может содержать необязательную секцию `else`, которая выполняется только в том случае, если цикл завершается нормально, то есть не в результате `break`.

При выполнении команды `continue` в цикле `while` или `for` оставшаяся часть набора цикла пропускается. В цикле `while` затем проверяется условие для

определения того, должен ли цикл продолжаться. В цикле for обрабатывается следующий элемент последовательности (если он есть):

```
In [2]: for number in range(10):
   ...:     if number == 5:
   ...:         continue
   ...:     print(number, end=' ')
   ...:
0 1 2 3 4 6 7 8 9
```

3.13. Логические операторы and, or или not

Условные операторы >, <, >=, <=, == и != могут использоваться в простых условиях вида grade >= 60. Для построения более сложных условий, объединяющих несколько простых условий, используются операторы and, or или not.

Логический оператор and

Чтобы проверить истинность *сразу двух* условий перед выполнением набора управляющей команды, используйте *логический оператор* and. В следующем коде определяются две переменные, после чего проверяется условие, которое истинно только в случае истинности *обоих* простых условий: если хотя бы одно (или оба) простое условие ложно, то все выражение and тоже ложно:

```
In [1]: gender = 'Female'

In [2]: age = 70

In [3]: if gender == 'Female' and age >= 65:
   ...:     print('Senior female')
   ...:
Senior female
```

Команда if содержит два простых условия, gender == 'Female' и age >= 65. Простое условие слева от оператора and вычисляется первым, потому что == имеет более высокий приоритет, чем and. При необходимости следующим вычисляется простое условие справа от and, потому что >= обладает более высоким приоритетом, чем and. (Вскоре мы объясним, почему правая сторона оператора and вычисляется *только* в случае истинности левой стороны.) В совокупности условие команды if истинно в том и только том случае, если *оба* простых условия также истинны. Объединенное условие можно сделать более понятным при помощи дополнительных круглых скобок:

```
(gender == 'Female') and (age >= 65)
```

В табл. 3.3 приведена сводка всех возможных комбинаций значений `True` и `False` для компонентов *выражение1* и *выражение2* — такие таблицы называются *таблицами истинности*:

Таблица 3.3. Сводка комбинаций значений True и False для компонентов «выражение1» и «выражение2»

выражение1	выражение2	выражение1 and выражение2
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

Логический оператор or

Логический оператор `or` проверяет, что одно *или* оба условия истинны. Следующий пример проверяет условие, которое `истинно` в случае `истинности` *хотя бы одного* из простых условий — все условие `ложно` только при `ложности` *обоих* простых условий:

```
In [4]: semester_average = 83

In [5]: final_exam = 95

In [6]: if semester_average >= 90 or final_exam >= 90:
   ...:     print('Student gets an A')
   ...:
Student gets an A
```

Фрагмент [6] также содержит два простых условия, `semester_average >= 90` и `final_exam >= 90`. В табл. 3.4 приведена сводка для логического оператора `or`. Оператор `and` обладает более высоким приоритетом, чем `or`.

Таблица 3.4. Сводка для логического оператора `or`

выражение1	выражение2	выражение1 or выражение2
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`True`

Повышение быстродействия за счет ускоренного вычисления

Python прерывает вычисление выражения and, как только определит, что все условие ложно, а вычисление выражения or — как только определит, что все условие истинно. Это называется *ускоренным вычислением*. Таким образом, обработка условия

```
gender == 'Female' and age >= 65
```

немедленно прерывается, если переменная gender не равна 'Female', потому что все выражение заведомо ложно. Если же переменная gender равна 'Female', то вычисление продолжается, потому что все выражение будет истинным только в случае, если переменная age больше или равна 65.

Аналогичным образом обработка условия

```
semester_average >= 90 or final_exam >= 90
```

немедленно прерывается, если переменная semester_average больше или равна 90, потому что все выражение заведомо истинно. Если же переменная semester_average меньше 90, то выполнение продолжается, потому что выражение будет истинным, если переменная final_exam больше или равна 90.

Выражения, использующие and, следует строить так, чтобы выражение, которое с большей вероятностью будет ложным, располагалось в левой части. В выражениях or в левой части должно располагаться условие, которое с большей вероятностью будет истинным. Эти методы позволяют сократить время выполнения программы.

Логический оператор not

Логический оператор not «инвертирует» смысл условия — True превращается в False, а False превращается в True. Это *унарный оператор*, то есть он имеет только *один* операнд. Поставив оператор not перед условием, вы выберете ветвь выполнения, в которой исходное условие (без оператора not) ложно, как в следующем примере:

```
In [7]: grade = 87

In [8]: if not grade == -1:
```

```
   ...:    print('The next grade is', grade)
   ...:
The next grade is 87
```

Часто можно обойтись без `not`, выражая условие более «естественным» или удобным способом. Например, предыдущая команда `if` может быть записана в следующем виде:

```
In [9]: if grade != -1:
   ...:    print('The next grade is', grade)
   ...:
The next grade is 87
```

Ниже приведена таблица истинности оператора `not` (табл. 3.5).

Таблица 3.5. Таблица истинности оператора not

выражение1	not выражение
`False`	`True`
`True`	`False`

В табл. 3.6 приведены приоритеты и варианты группировки операторов, упоминавшихся до настоящего момента. Операторы перечисляются сверху вниз в порядке убывания приоритета.

Таблица 3.6. Приоритеты и варианты группировки операторов

Операторы	Группировка
`()`	слева направо
`**`	справа налево
`*  /  //  %`	слева направо
`+  -`	слева направо
`<  <=  >  >= == !=`	слева направо
`not`	слева направо
`and`	слева направо
`or`	слева направо

3.14. Введение в data science: параметры, характеризующие положение центра распределения, — математическое ожидание, медиана и мода

Продолжим обсуждение применения статистики для анализа данных с несколькими дополнительными описательными статистическими параметрами, включая:

- **математическое ожидание** — *среднее значение* среди набора значений;
- **медиану** — *среднее значение* при расположении значений *в порядке сортировки*;
- **моду** — *наиболее часто встречающееся значение*.

Существуют и другие *параметры, характеризующие положение центра распределения*, каждый из них генерирует одно значение, представляющее «центральное» значение в множестве значений, то есть значение, которое в определенном смысле типично для множества.

Вычислим математическое ожидание, медиану и моду для списка целых чисел. Следующий сеанс создает список с именем `grades`, использует встроенные функции `sum` и `len` для вычисления математического ожидания «вручную»: `sum` вычисляет сумму оценок (397), а `len` возвращает их количество (5):

```
In [1]: grades = [85, 93, 45, 89, 85]

In [2]: sum(grades) / len(grades)
Out[2]: 79.4
```

В главе 2 упоминались *статистические показатели* количества и суммы — они реализованы в Python в виде встроенных функций `len` и `sum`. Как и функции `min` и `max` (представленные в главе 2), `sum` и `len` являются примерами *свертки* из области программирования в функциональном стиле — они сворачивают коллекцию значений в одно значение, сумму этих значений и количество значений соответственно. В примере вычисления средней оценки из раздела 3.8 можно было бы удалить строки 10–15 из сценария и заменить `average` в строке 16 вычислением из фрагмента [2].

Модуль `statistics` из стандартной библиотеки Python предоставляет функции для вычисления математического ожидания, медианы и моды — все эти характеристики тоже являются свертками. Чтобы использовать эти средства, сначала импортируйте модуль `statistics`:

```
In [3]: import statistics
```

После этого вы сможете обращаться к функциям модуля: укажите префикс `statistics` и имя вызываемой функции. Следующий фрагмент вычисляет математическое ожидание, медиану и моду для списка оценок, используя для этого функции `mean`, `median` и `mode` модуля `statistics`:

```
In [4]: statistics.mean(grades)
Out[4]: 79.4

In [5]: statistics.median(grades)
Out[5]: 85

In [6]: statistics.mode(grades)
Out[6]: 85
```

Аргументом каждой функции должен быть *итерируемый объект* — в данном случае список оценок. Чтобы убедиться в правильности медианы и моды, можно воспользоваться встроенной *функцией* `sorted` для получения списка оценок, упорядоченных по возрастанию:

```
In [7]: sorted(grades)
Out[7]: [45, 85, 85, 89, 93]
```

Список `grades` содержит нечетное количество значений (5), поэтому `median` возвращает *среднее значение* (85). Если список значений в списке имеет четное количество элементов, то `median` возвращает *среднее* для *двух* средних значений. Проверяя отсортированные значения, можно убедиться в том, что мода равна 85, потому что это значение встречается чаще других (дважды). Функция `mode` выдает ошибку `StatisticsError` для списков вида

```
[85, 93, 45, 89, 85, 93]
```

в которых встречаются два и более «самых частых» значения. Такой набор значений называется *бимодальным*. В данном случае каждое из значений 85 и 93 встречается дважды.

3.15. Итоги

В этой главе рассматривались управляющие команды Python, включая `if`, `if...else`, `if...elif...else`, `while`, `for`, `break` и `continue`. Как было показано, команда `for` выполняет повторения, управляемые последовательностью, — она обрабатывает все элементы итерируемого объекта (например, диапазона целых чисел, строки или списка). Встроенная функция `range` использовалась для генерирования последовательностей от 0 до значения своего аргумента, исключая последний, и для определения количества итераций команды `for`.

Мы использовали повторение, управляемое контрольным значением, с командой `while` для создания цикла, который продолжает выполняться, пока не будет обнаружено контрольное значение. Версия встроенной функции `range` с двумя аргументами используется для генерирования последовательностей целых чисел от значения первого аргумента до значения второго аргумента, исключая последний. Также встречалась версия с тремя аргументами, в которой третий аргумент задает приращение между целыми числами диапазона.

Далее был описан тип `Decimal` для точных финансовых вычислений, который был применен для вычисления сложного процента. Форматные строки и различные спецификаторы были использованы для создания отформатированного вывода. Мы представили команды `break` и `continue` для изменения последовательности выполнения в циклах. Также были рассмотрены логические операторы `and`, `or` или `not` для создания условий, объединяющих несколько простых условий.

Наконец, наше обсуждение описательной статистики продолжилось на характеристиках, описывающих положение центра распределения, — математическом ожидании, медиане и моде, — и вычислении их при помощи функций модуля `statistics` стандартной библиотеки Python.

В следующей главе мы перейдем к созданию пользовательских функций и использованию существующих функций из модулей Python `math` и `random`. В ней будут продемонстрированы некоторые заранее определенные свертки и другие средства функционального программирования.

4

Функции

В этой главе...

- Создание пользовательских функций.
- Импортирование и использование модулей стандартной библиотеки Python (таких как `random` и `math`) для повторного использования кода и предотвращения необходимости «изобретения велосипеда».
- Передача данных между функциями.
- Генерирование диапазона случайных чисел.
- Методы моделирования, основанные на генерировании случайных чисел.
- Инициализация генератора случайных чисел для обеспечения воспроизводимости.
- Упаковка значений в кортеж и распаковка значений из кортежа.
- Возвращение нескольких значений из функции в кортеже.
- Область видимости идентификатора и определение того, где он может использоваться.
- Создание функций со значениями параметров по умолчанию.
- Вызов функций с ключевыми аргументами.
- Создание функций, которые могут получать любое количество аргументов.
- Использование методов объекта.
- Написание и использование рекурсивных функций.

4.1. Введение

В этой главе изучение основ Python продолжится в направлении пользовательских функций и сопутствующих тем. Мы воспользуемся модулем `random` стандартной библиотеки Python и генератором случайных чисел для моделирования бросков шестигранного кубика. Пользовательские функции и генерирование случайных чисел будут использованы в сценарии, реализующем азартную игру «крэпс». В этом примере также будет представлен *кортеж* — новый тип последовательности Python; кортежи будут использованы для возвращения нескольких значений из функции. Также в этой главе рассматривается инициализация генератора случайных чисел для обеспечения воспроизводимости.

Мы импортируем модуль `math` стандартной библиотеки Python, а затем используем его для изучения функциональности автозаполнения IPython, ускоряющей процессы программирования и обучения. Мы создадим функции со значениями параметров по умолчанию, исследуем вызов функций с ключевыми аргументами и определение функций с произвольными списками аргументов. Также мы продемонстрируем вызов методов объектов и поговорим о том, как область видимости идентификатора определяет возможности его использования в тех или иных компонентах программы.

Далее процесс импортирования модулей будет исследован более основательно. Вы узнаете, как происходит передача аргументов функциям по ссылке. Затем мы рассмотрим рекурсивную функцию и начнем знакомство с поддержкой программирования в функциональном стиле Python.

В разделе «Введение в data science» продолжится обсуждение описательных статистических характеристик. В нем будут рассмотрены дисперсионные характеристики — дисперсия и стандартное отклонение — и вычисления их при помощи функций из модуля `statistics` стандартной библиотеки Python.

4.2. Определение функций

В предыдущих главах вызывались многие встроенные функции (`int`, `float`, `print`, `input`, `type`, `sum`, `len`, `min` и `max`), а также некоторые функции из модуля statistics (`mean`, `median` и `mode`). Каждая функция выполняла конкретную задачу. В программах часто определяются и вызываются *пользовательские* функции. В следующем сеансе определяется функция `square`, вычисляющая квадрат

своего аргумента. Затем функция вызывается дважды — для значения `7` типа `int` (вызов дает значение `49` типа `int`) и для значения `2.5` типа `float` (вызов дает значение с плавающей точкой `6.25`):

```
In [1]: def square(number):
   ...:     """Вычисление квадрата числа."""
   ...:     return number ** 2
   ...:

In [2]: square(7)
Out[2]: 49

In [3]: square(2.5)
Out[3]: 6.25
```

Команды из определения функции в первом фрагменте написаны один раз, но они могут вызываться «для выполнения своей операции» в разных точках программы, причем это можно делать многократно. Вызов `square` с нечисловым аргументом (например, `'hello'`) вызовет ошибку `TypeError`, потому что оператор возведения в степень (`**`) работает только с числовыми значениями.

Определение пользовательской функции

Определение функции (например, `square` из фрагмента [1]) начинается с *ключевого слова* `def`, за которым следует имя функции `square` в круглых скобках и двоеточие (`:`). Как и идентификаторы переменных, по соглашению имена функций должны начинаться с буквы нижнего регистра, а в именах, состоящих из нескольких слов, составляющие должны разделяться символами подчеркивания.

Требуемые круглые скобки содержат *список параметров* функции — разделенный запятыми список *параметров*, представляющий данные, необходимые функции для выполнения соответствующей операции. Функция `square` имеет только один параметр с именем `number` — значение, возводимое в квадрат. Если круглые скобки пусты, то это означает, что функция не использует параметры, заданные для выполнения операции.

Снабженные отступом строки после двоеточия (`:`) образуют *блок* функции. Он состоит из необязательной doc-строки, за которой следуют команды, выполняющие операцию функции (о различиях между блоком функции и набором управляющей команды см. далее).

Определение doc-строки пользовательской функции

В «*Руководстве по стилю для кода Python*» указано, что первой строкой блока функции должна быть doc-строка, кратко поясняющая назначение функции:

```
"""Вычисление квадрата числа."""
```

Чтобы предоставить более подробную информацию, используйте многострочную doc-строку — «Руководство по стилю» рекомендует начать с краткого пояснения, за которым следует пустая строка и дополнительные подробности.

Возвращение результата на сторону вызова функции

Завершив выполнение, функция возвращает управление в точку вызова, то есть в строку кода, которая вызвала функцию. В блоке `square` команда `return`

```
return number ** 2
```

сначала возводит `число` в квадрат, а затем завершает функцию и возвращает результат на сторону вызова. В нашем примере функция впервые вызывается во фрагменте [2], поэтому IPython выводит результат в `Out[2]`. Второй вызов находится во фрагменте [3], так что IPython выводит результат в `Out[3]`.

Вызовы функций также могут быть встроены в выражения. Следующий код сначала вызывает `square`, а затем выводит результат функцией `print`:

```
In [4]: print('The square of 7 is', square(7))
The square of 7 is 49
```

Функция может вернуть управление еще двумя способами:

- Выполнение команды `return` без выражения завершает функцию и *неявно* возвращает значение `None` на сторону вызова. В документации Python сказано, что `None` означает отсутствие значения. В условиях `None` интерпретируется как `False`.
- Если в функции нет команды `return`, то она *неявно* возвращает значение `None` после выполнения последней команды в блоке функции.

Локальные переменные

Хотя мы не определяем переменные в блоке `square`, это можно сделать. Параметры функции и переменные, определенные в ее блоке, являются *локальными*

переменными — они могут использоваться только внутри функции и существуют только во время ее выполнения. Попытка обратиться к локальной переменной за пределами блока функции приводит к ошибке `NameError`, которая означает, что переменная не определена.

Обращение к doc-строке функции из механизма справки IPython

IPython поможет получить информацию о модулях и функциях, которые вы собираетесь использовать в формируемом коде, а также о самом IPython. Например, для просмотра doc-строки функции (чтобы узнать, как пользоваться ею) введите имя функции, дополненное *вопросительным знаком* (`?`):

```
In [5]: square?
Signature: square(number)
Docstring: Calculate the square of number.
File:      ~/Documents/examples/ch04/<ipython-input-1-7268c8ff93a9>
Type:      function
```

Для функции `square` выводится следующая информация:

- Имя и список параметров функции (ее *сигнатура*).
- Doc-строка функции.
- Имя файла, содержащего определение функции. Для функции в интерактивном сеансе в этой строке выводится информация о фрагменте, определившем функцию, — `1` в `"<ipython-input-1-7268c8ff93a9>"` означает фрагмент [1].
- Тип элемента, для которого вы обратились к механизму справки IPython, — в данном случае это функция.

Если исходный код функции доступен из IPython, например, если функция определяется в текущем сеансе или импортируется в сеанс из файла `.py`, то вы сможете воспользоваться командой `??` для вывода полного определения исходного кода функции:

```
In [6]: square??
Signature: square(number)
Source:
def square(number):
    """Вычисление квадрата числа."""
    return number ** 2
File:      ~/Documents/examples/ch04/<ipython-input-1-7268c8ff93a9>
Type:      function
```

Если исходный код недоступен из IPython, то `??` просто выводит doc-строку.

Если doc-строка помещается в окне, IPython выводит следующее приглашение `In[]`. Если doc-строка имеет слишком большую длину, то IPython сообщает о наличии продолжения, выводя двоеточие (`:`) в нижней части окна: нажмите клавишу «пробел», чтобы вывести следующий экран. Вы можете перемещаться по doc-строке при помощи клавиш ↑ и ↓, соответственно. IPython выводит `(END)` в конце doc-строки. Нажатие *q* (сокращение от «quit») в любом приглашении `:` или `(END)` возвращает вас к следующему приглашению `In []`. Чтобы получить представление о тех или иных возможностях IPython, введите `?` в любом приглашении `In []`, нажмите Enter и прочитайте обзор справочной документации.

4.3. Функции с несколькими параметрами

Определим функцию `maximum`, которая определяет и возвращает наибольшее из трех значений, — в следующем сеансе функция вызывается трижды с целыми числами, числами с плавающей точкой и строками соответственно.

```
In [1]: def maximum(value1, value2, value3):
   ...:     """Возвращает наибольшее из трех значений."""
   ...:     max_value = value1
   ...:     if value2 > max_value:
   ...:         max_value = value2
   ...:     if value3 > max_value:
   ...:         max_value = value3
   ...:     return max_value
   ...:

In [2]: maximum(12, 27, 36)
Out[2]: 36

In [3]: maximum(12.3, 45.6, 9.7)
Out[3]: 45.6

In [4]: maximum('yellow', 'red', 'orange')
Out[4]: 'yellow'
```

Мы не поставили пустые строки до и после команд `if`, потому что нажатие Enter в пустой строке в интерактивном режиме завершает определение функции.

Также `maximum` можно вызывать со смешанными типами — например, `int` и `float`:

```
In [5]: maximum(13.5, -3, 7)
Out[5]: 13.5
```

Вызов `maximum(13.5, 'hello', 7)` приводит к ошибке `TypeError`, потому что строки и числа не могут сравниваться друг с другом оператором «больше» (`>`).

Определение функции maximum

Функция `maximum` задает три параметра в списке, разделенном запятыми. Аргументы фрагмента [2] `12`, `27` и `36` присваиваются параметрам `value1`, `value2` и `value3` соответственно.

Чтобы определить наибольшее значение, обработаем значения по одному:

- Изначально предполагается, что `value1` содержит наибольшее значение, которое присваивается локальной переменной `max_value`. Конечно, может оказаться, что наибольшее значение на самом деле хранится в `value2` или `value3`, поэтому их также необходимо сравнить с `max_value`.
- Затем первая команда `if` проверяет условие `value2 > max_value`; если это условие `истинно`, то `value2` присваивается `max_value`.
- Вторая команда `if` проверяет условие `value3 > max_value`; если это условие `истинно`, то `value3` присваивается `max_value`.

Теперь `max_value` содержит наибольшее значение, поэтому мы возвращаем его. Когда управление возвращается на сторону вызова, параметры `value1`, `value2` и `value3`, а также переменная `max_value` в блоке функции — все они были *локальными переменными* — уже не существуют.

Встроенные функции max и min

Для многих распространенных задач необходимая функциональность уже существует в Python. Например, встроенные функции `max` и `min` «знают», как определить наибольший и наименьший (соответственно) из своих двух и более аргументов:

```
In [6]: max('yellow', 'red', 'orange', 'blue', 'green')
Out[6]: 'yellow'

In [7]: min(15, 9, 27, 14)
Out[7]: 9
```

Каждая из этих функций также может получать в аргументе итерируемый объект, например список или строку. Использование встроенных функций или функций из модулей стандартной библиотеки Python вместо написания собственных реализаций может сократить время разработки и повысить надежность программы, улучшить ее компактность и быстродействие. За списком встроенных функций и модулей Python обращайтесь по адресу https://docs.python.org/3/library/index.html.

4.4. Генератор случайных чисел

Теперь ненадолго отвлечемся на популярную область программных приложений — моделирование и игры. Чтобы добавить в программу *случайный элемент*, воспользуйтесь *модулем* `random` из стандартной библиотеки Python.

Бросок шестигранного кубика

Сгенерируем 10 случайных чисел в диапазоне 1–6 для моделирования броска шестигранного кубика:

```
In [1]: import random

In [2]: for roll in range(10):
   ...:     print(random.randrange(1, 7), end=' ')
   ...:
4 2 5 5 4 6 4 6 1 5
```

Сначала импортируется модуль `random` — это позволит программе использовать функциональность этого модуля. Функция `randrange` генерирует целое число в диапазоне от значения первого аргумента до значения второго аргумента, *не* включая последний. Воспользуйтесь клавишей ↑, чтобы вызвать команду `for`, а затем нажмите Enter, чтобы снова выполнить ее. Обратите внимание: на этот раз выводятся *другие* значения:

```
In [3]: for roll in range(10):
   ...:     print(random.randrange(1, 7), end=' ')
   ...:
4 5 4 5 1 4 1 4 6 5
```

В некоторых ситуациях требуется обеспечить *воспроизводимость* последовательности случайных чисел, например, с целью отладки. В конце этого раздела мы воспользуемся для этой цели функцией `seed` модуля `random`.

6 000 000 бросков шестигранного кубика

Если `randrange` действительно генерирует случайные целые числа, то каждое число в диапазоне имеет одинаковую *вероятность* выпадения при каждом вызове функции. Чтобы продемонстрировать, что грани 1–6 выпадают с одинаковой вероятностью, следующий сценарий моделирует 6 000 000 бросков кубика. При выполнении сценария, представленного в примере, каждая грань встретится приблизительно 1 000 000 раз:

```
 1 # fig04_01.py
 2 """6 000 000 бросков шестигранного кубика."""
 3 import random
 4
 5 # счетчик соответствующей грани
 6 frequency1 = 0
 7 frequency2 = 0
 8 frequency3 = 0
 9 frequency4 = 0
10 frequency5 = 0
11 frequency6 = 0
12
13 # 6,000,000 die rolls
14 for roll in range(6_000_000):  # Обратите внимание на разделители
15     face = random.randrange(1, 7)
16
17     # Увеличение счетчика соответствующей грани
18     if face == 1:
19         frequency1 += 1
20     elif face == 2:
21         frequency2 += 1
22     elif face == 3:
23         frequency3 += 1
24     elif face == 4:
25         frequency4 += 1
26     elif face == 5:
27         frequency5 += 1
28     elif face == 6:
29         frequency6 += 1
30
31 print(f'Face{"Frequency":>13}')
32 print(f'{1:>4}{frequency1:>13}')
33 print(f'{2:>4}{frequency2:>13}')
34 print(f'{3:>4}{frequency3:>13}')
35 print(f'{4:>4}{frequency4:>13}')
36 print(f'{5:>4}{frequency5:>13}')
37 print(f'{6:>4}{frequency6:>13}')
```

```
Face    Frequency
   1       998686
   2      1001481
   3       999900
   4      1000453
   5       999953
   6       999527
```

В сценарии используются *вложенные* управляющие команды (команда `if`... `elif`, вложенная в команду `for`) для определения количества выпадений каждой грани. Команда `for` выполняет цикл 6 000 000 раз. Мы используем разделитель групп разрядов (_), чтобы значение `6000000` лучше читалось. Выражение вида `range(6,000,000)` было бы признано ошибочным: запятые разделяют аргументы в вызовах функций, поэтому Python интерпретирует `range(6,000,000)` как вызов `range` с *тремя* аргументами 6, 0 и 0.

Для каждого броска кубика сценарий увеличивает соответствующую переменную-счетчик на единицу. Запустите программу и проследите за результатом. Возможно, для завершения работы ей понадобится несколько секунд. Как вы увидите, при каждом запуске программа выдает разные результаты. Обратите внимание: в команду `if`...`elif` не включена секция `else`.

Инициализация генератора случайных чисел для воспроизведения результатов

Функция `randrange` на самом деле генерирует *псевдослучайные числа* на основании неких внутренних вычислений, начинающихся с числового значения, называемого *значением инициализации*. Многократные вызовы `randrange` выдают серию чисел, которые *кажутся* случайными, потому что при каждом запуске нового интерактивного сеанса или выполнении сценария, использующего функции модуля `random`, Python во внутренней реализации использует *новое* значение инициализации[1]. Когда вы занимаетесь отладкой логических ошибок в программе, использующей случайно сгенерированные данные, может быть полезно использовать *одну и ту же* серию случайных чисел, пока

[1] Согласно документации, Python выбирает значение инициализации на основании системных часов или источника случайности, зависящего от конкретной операционной системы. В областях, в которых необходимы безопасные случайные числа (например, в криптографии), документация рекомендует использовать модуль `secrets` вместо модуля `random`.

логические ошибки не будут устранены, прежде чем тестировать программу с другими значениями. В этом случае можно воспользоваться функцией `seed` модуля `random`, чтобы *самостоятельно инициализировать генератор случайных чисел*, — это заставит `randrange` начать вычисления своей псевдослучайной числовой последовательности с заданного вами значения. В следующем разделе фрагменты [5] и [8] выдают одинаковые результаты, потому что фрагменты [4] и [7] используют одно значение инициализации (`32`):

```
In [4]: random.seed(32)

In [5]: for roll in range(10):
   ...:     print(random.randrange(1, 7), end=' ')
   ...:
1 2 2 3 6 2 4 1 6 1
In [6]: for roll in range(10):
   ...:     print(random.randrange(1, 7), end=' ')
   ...:
1 3 5 3 1 5 6 4 3 5
In [7]: random.seed(32)

In [8]: for roll in range(10):
   ...:     print(random.randrange(1, 7), end=' ')
   ...:
1 2 2 3 6 2 4 1 6 1
```

Фрагмент [6] генерирует *разные* значения, потому что он просто продолжает последовательность псевдослучайных чисел, начатую во фрагменте [5].

4.5. Практический пример: игра «крэпс»

В этом разделе моделируется популярная азартная игра «крэпс». Формулировка задачи:

> *Игрок бросает два шестигранных кубика, на грани которых нанесены 1, 2, 3, 4, 5 или 6 очков. Когда кубики останавливаются, вычисляется сумма очков на двух верхних гранях. Если при первом броске выпадает 7 или 11, игрок побеждает. Если при первом броске сумма равна 2, 3 или 12 («крэпс»), игрок проигрывает (побеждает «казино»). Если при первом броске сумма равна 4, 5, 6, 8, 9 или 10, то она становится «целью» игрока. Чтобы победить, игрок должен на последующих бросках выбросить то же целевое значение. Если перед этим игрок выбросит 7, он проигрывает.*

Следующий сценарий моделирует игру с несколькими тестовыми запусками: с выигрышем при первом броске, с проигрышем при первом броске, с выигрышем на последующем броске и проигрышем на последующем броске.

```
# fig04_02.py
"""Моделирование игры крэпс."""
import random

def roll_dice():
    """Моделирует бросок двух кубиков и возвращает результат в виде кортежа."""
    die1 = random.randrange(1, 7)
    die2 = random.randrange(1, 7)
    return (die1, die2)  # Два результата упаковываются в кортеж

def display_dice(dice):
    """Выводит результат одного броска двух кубиков."""
    die1, die2 = dice  # Распаковывает кортеж в переменные die1 и die2
    print(f'Player rolled {die1} + {die2} = {sum(dice)}')

die_values = roll_dice()  # first roll
display_dice(die_values)

# Состояние игры и цель определяются на основании первого броска
sum_of_dice = sum(die_values)

if sum_of_dice in (7, 11):  # Выигрыш
    game_status = 'WON'
elif sum_of_dice in (2, 3, 12):  # Проигрыш
    game_status = 'LOST'
else:  # запомнить цель
    game_status = 'CONTINUE'
    my_point = sum_of_dice
    print('Point is', my_point)

# Броски продолжаются до выигрыша или проигрыша
while game_status == 'CONTINUE':
    die_values = roll_dice()
    display_dice(die_values)
    sum_of_dice = sum(die_values)

    if sum_of_dice == my_point:  # Выигрыш по цели
        game_status = 'WON'
    elif sum_of_dice == 7:  # Проигрыш по выпадению 7
        game_status = 'LOST'

# Вывод сообщения "wins" или "loses"
if game_status == 'WON':
    print('Player wins')
```

```
45 else:
46     print('Player loses')
```

```
Player rolled 2 + 5 = 7
Player wins
```

```
Player rolled 1 + 2 = 3
Player loses
```

```
Player rolled 5 + 4 = 9
Point is 9
Player rolled 4 + 4 = 8
Player rolled 2 + 3 = 5
Player rolled 5 + 4 = 9
Player wins
```

```
Player rolled 1 + 5 = 6
Point is 6
Player rolled 1 + 6 = 7
Player loses
```

Функция roll_dice — возвращение нескольких значений в кортеже

Функция `roll_dice` (строки 5–9) моделирует бросок двух кубиков. Функция определяется один раз, а затем вызывается в нескольких точках программы (строки 16 и 33). Пустой список параметров означает, что функции `roll_dice` не требуются аргументы для решения ее задачи.

Встроенные и пользовательские функции, вызываемые в программе, возвращают одно значение. Иногда бывает нужно получить из функции сразу несколько значений, как в функции `roll_dice`, которая возвращает два значения (строка 9) в виде *кортежа* — *неизменяемой* последовательности значений. Чтобы создать кортеж, разделите его значения запятыми, как в строке 9:

```
(die1, die2)
```

Эта операция называется *упаковкой кортежа*. Круглые скобки необязательны, но мы рекомендуем использовать их для ясности. Кортежи будут подробно рассмотрены в следующей главе.

Функция display_dice

Чтобы использовать значения из кортежа, можно присвоить их списку переменных, разделенных запятыми, — при этом происходит *распаковка* кортежа.

Чтобы вывести каждый результат броска, функция `display_dice` (определенная в строках 11–14 и вызываемая в строках 17 и 34) распаковывает полученный аргумент-кортеж (строка 13). Количество переменных слева от `=` должно совпадать с количеством элементов в кортеже; в противном случае происходит ошибка `ValueError`. Строка 14 выводит отформатированную строку, которая содержит как значения на кубиках, так и их сумму. Чтобы вычислить сумму, мы передаем кортеж встроенной функции `sum` — кортеж, как и список, является последовательностью.

Следует заметить, что блок каждой из функций `roll_dice` и `display_dice` начинается с doc-строки, описывающей назначение функции. Кроме того, обе функции содержат локальные переменные `die1` и `die2`. Эти переменные не «конфликтуют», поскольку принадлежат блокам разных функций. Каждая локальная переменная доступна только в том блоке, в котором она была определена.

Первый бросок

В начале выполнения сценария в строках 16–17 моделируется бросок кубиков и выводятся результаты. Строка 20 вычисляет сумму кубиков для использования в строках 22–29. Игрок может выиграть или проиграть как при первом, так и при любом последующем броске. В переменной `game_status` отслеживается статус игры (выигрыш/проигрыш).

Оператор `in` в строке 22

```
sum_of_dice in (7, 11)
```

проверяет, содержит ли кортеж `(7, 11)` значение `sum_of_dice`. Если условие `истинно`, то это означает, что на кубиках выпал результат 7 или 11. В этом случае игрок выиграл при первом броске, поэтому сценарий присваивает переменной `game_status` значение `'WON'`. Правым операндом оператора может быть любой итерируемый объект. Также существует оператор `not in`, который проверяет, что значение *не содержится* в итерируемом объекте. Предыдущее компактное условие эквивалентно сложному условию

```
(sum_of_dice == 7) or (sum_of_dice == 11)
```

Аналогичным образом условие в строке 24

```
sum_of_dice in (2, 3, 12)
```

проверяет, что кортеж (2,3,12) содержит значение sum_of_dice. В этом случае игрок проиграл при первом броске, поэтому сценарий присваивает переменной game_status значение 'LOST'.

Для любой другой суммы на кубиках(4, 5, 6, 8, 9 или 10):

- строка 27 присваивает game_status значение 'CONTINUE', чтобы продолжить броски;
- строка 28 сохраняет сумму кубиков в my_point, чтобы отслеживать нужный для победы результат;
- строка 29 выводит my_point.

Последующие броски

Если переменная game_status равна 'CONTINUE' (строка 32), то игрок не выиграл и не проиграл, поэтому выполняется набор команды while (строки 33–40). Каждая итерация цикла вызывает roll_dice, выводит результаты на кубиках и вычисляет их сумму. Если значение sum_of_dice равно my_point (строка 37) или 7 (строка 39), то сценарий присваивает переменной game_status значение 'WON' или 'LOST' соответственно, и цикл завершается. В противном случае цикл while продолжается моделированием следующего броска.

Вывод окончательных результатов

При завершении цикла сценарий переходит к команде if...else (строки 43–46), которая выводит сообщение 'Player wins', если переменная game_status равна 'WON', или 'Player loses' в противном случае.

4.6. Стандартная библиотека Python

Как правило, при написании программы Python разработчик объединяет функции и классы (то есть пользовательские типы), созданные им, с готовыми функциями и классами, определенными в модулях — например, в стандартной библиотеке Python и других библиотеках. При этом одна из главных целей программирования — обойтись без «изобретения велосипеда».

Модуль представляет собой файл со взаимосвязанными функциями, данными и классами. Тип Decimal из модуля decimal стандартной библиотеки Python

в действительности представляет собой класс. Мы кратко представили классы в главе 1, а их более подробное описание будет приведено в главе 10. Взаимосвязанные модули группируются в *пакетах*. В этой книге мы будем работать со многими готовыми модулями и пакетами, а читатели будет создавать собственные модули — собственно, каждый созданный файл с исходным кодом Python (.py) и является модулем. Создание пакетов выходит за рамки этой книги. Обычно они используются для структурирования функциональности большой библиотеки на меньшие подмножества, более простые в сопровождении, которые можно импортировать по отдельности для удобства. Например, библиотека визуализации `matplotlib`, использованная в разделе 5.17, обладает весьма обширной функциональностью (объем ее документации превышает 2300 страниц), поэтому мы импортируем только те подмножества, которые нужны в наших примерах (`pyplot` и `animation`).

Стандартная библиотека Python входит в базовую поставку Python. Ее пакеты и модули содержат средства для выполнения широкого спектра повседневных задач программирования[1]. Полный список модулей стандартной библиотеки доступен по адресу:

https://docs.python.org/3/library/

Мы уже пользовались функциональностью модулей `decimal`, `statistics` и `random`. В следующем разделе будет использоваться функциональность вычислений из модуля `math`. В примерах этой книги встречаются многие другие модули стандартной библиотеки Python, включая модули из табл. 4.1.

Таблица 4.1. Наиболее часто применяемые модули стандартной библиотеки Python

`collections` — структуры данных помимо списков, кортежей, словарей и множеств. Криптографические модули — шифрование данных для безопасной передачи. `csv` — обработка файлов с данными, разделенными запятыми (по аналогии с Excel). `datetime` — работа с датой и временем (а также модули `time` и `calendar`). `decimal` — вычисления с фиксированной и плавающей точкой, включая денежные вычисления.	`math` — распространенные математические библиотеки и операции. `os` — взаимодействие с операционной системой. `profile`, `pstats`, `timeit` — анализ быстродействия. `random` — псевдослучайные числа. `re` — регулярные выражения для поиска по тексту. `sqlite3` — работа с реляционными базами данных SQLite.

[1] В учебном курсе Python этот подход называется «батарейки входят в комплект».

`doctest` — внедрение проверочных тестов и ожидаемых результатов в doc-строки для простого модульного тестирования. `gettext` и `locale` — модули интернационализации и локализации. `json` — обработка формата JSON (JavaScript Object Notation), используемого при работе с веб-сервисами и документными базами данных NoSQL. `statistics` — функции математической статистики (такие как `mean`, `median`, `mode` и `variance`).	`string` — работа со строками. `sys` — обработка аргументов командной строки; стандартные потоки ввода, вывода и ошибок. `tkinter` — графические интерфейсы пользователя (GUI) и рисование на холсте. `turtle` — графика Turtle. `webbrowser` — удобное отображение веб-страниц в приложениях Python

4.7. Функции модуля math

В *модуле* `math` определяются функции для выполнения различных распространенных математических вычислений. Вспомните, о чем говорилось в предыдущей главе: команда `import` следующего вида позволяет использовать определения из внешнего модуля, для чего следует указать имя модуля и точку (`.`):

```
In [1]: import math
```

Например, следующий фрагмент вычисляет квадратный корень из 900 вызовом *функции* `sqrt` модуля `math`, возвращающей свой результат в виде значения `float`:

```
In [2]: math.sqrt(900)
Out[2]: 30.0
```

Аналогичным образом следующий фрагмент вычисляет абсолютное значение `-10` вызовом *функции* `fabs` модуля `math`, возвращающей свой результат в виде значения `float`:

```
In [3]: math.fabs(-10)
Out[3]: 10.0
```

В табл. 4.2 перечислены некоторые функции модуля `math` — полный список можно просмотреть по адресу:

https://docs.python.org/3/library/math.html

Таблица 4.2. Некоторые функции модуля math

Функция	Описание	Пример
`ceil(x)`	Округляет x до наименьшего целого, не меньшего x	`ceil(9.2)` равно `10.0` `ceil(-9.8)` равно `-9.0`
`floor(x)`	Округляет x до наименьшего целого, не большего x	`floor(9.2)` равно `9.0` `floor(-9.8)` равно `-10.0`
`sin(x)`	Синус x (x задается в радианах)	`sin(0.0)` равно `0.0`
`cos(x)`	Косинус x (x задается в радианах)	`cos(0.0)` равно `1.0`
`tan(x)`	Тангенс x (x задается в радианах)	`tan(0.0)` равно `0.0`
`exp(x)`	Экспонента e^x	`exp(1.0)` равно `2.718282` `exp(2.0)` равно `7.389056`
`log(x)`	Натуральный логарифм x (по основанию e)	`log(2.718282)` равно `1.0` `log(7.389056)` равно `2.0`
`log10(x)`	Логарифм x (по основанию 10)	`log10(10.0)` равно `1.0` `log10(100.0)` равно `2.0`
`pow(x, y)`	x в степени y (x^y)	`pow(2.0, 7.0)` равно `128.0` `pow(9.0, .5)` равно `3.0`
`sqrt(x)`	Квадратный корень из x	`sqrt(900.0)` равно `30.0` `sqrt(9.0)` равно `3.0`
`fabs(x)`	Абсолютное значение x — всегда возвращает `float`. В Python также существует встроенная функция `abs`, которая возвращает `int` или `float` в зависимости от ее аргумента	`fabs(5.1)` равно `5.1` `fabs(-5.1)` равно `5.1`
`fmod(x, y)`	Остаток от деления x/y в виде `float`	`fmod(9.8, 4.0)` равно `1.8`

4.8. Использование автозаполнения IPython

Документацию модуля можно просмотреть в интерактивном режиме IPython при помощи *автозаполнения* — возможности, ускоряющей процессы программирования и изучения языка. Если ввести часть идентификатора и нажать клавишу *Tab*, то IPython завершает идентификатор за вас или предоставляет список идентификаторов, начинающихся с введенных символов. Состав списка зависит от платформы ОС и того, что именно было импортировано в текущий сеанс IPython:

```
In [1]: import math
In [2]: ma<Tab>
           map          %macro       %%markdown
           math         %magic       %matplotlib
           max()        %man
```

Список идентификаторов прокручивается клавишами ↑ и ↓. При этом IPython автоматически выделяет идентификатор и выводит его справа от префикса `In[]`.

Просмотр идентификаторов в модуле

Чтобы просмотреть список идентификаторов, определенных в модуле, введите имя модуля и точку (`.`), а затем нажмите *Tab*:

```
In [3]: math.<Tab>
         acos()      atan()      copysign()  e           expm1()
         acosh()     atan2()     cos()       erf()       fabs()
         asin()      atanh()     cosh()      erfc()      factorial() >
         asinh()     ceil()      degrees()   exp()       floor()
```

Если подходящих идентификаторов больше, чем отображается в настоящий момент, то IPython выводит символ `>` (на некоторых платформах) у правого края (в данном случае справа от `factorial()`). Вы можете воспользоваться клавишами ↑ и ↓ для прокрутки списка. В списке идентификаторов:

- Идентификаторы, за которыми стоят круглые скобки, являются именами функций (или методов, см. далее).
- Идентификаторы из одного слова (скажем, `Employee`), начинающиеся с буквы верхнего регистра, и идентификаторы из нескольких слов, в которых каждое слово начинается с буквы верхнего регистра (например, `CommissionEmployee`), представляют имена классов (в приведенном выше списке их нет). Эта схема формирования имен, рекомендованная в «*Руководстве по стилю для кода Python*», называется «*верблюжьим регистром*», потому что буквы верхнего регистра выделяются, словно горбы верблюда.
- Идентификаторы нижнего регистра без круглых скобок, такие как `pi` (не входит в приведенный список) и `e`, являются именами переменных. Идентификатор `pi` представляет число `3.141592653589793`, а идентификатор `e` — число `2.718281828459045`. В модуле `math` идентификаторы `pi` и `e` представляют математические константы π и e соответственно.

В Python нет *констант*, хотя многие объекты в Python неизменяемы. Таким образом, хотя `pi` и `e` в реальном мире являются константами, *им не следует присваивать новые значения* — это приведет к изменению их значений. Чтобы разработчику было проще отличить константы от других переменных, «Руководство по стилю» рекомендует записывать имена констант, определенных в вашем коде, в верхнем регистре.

Использование выделенной функции

Чтобы в процессе перебора идентификаторов использовать функцию, выделенную в настоящий момент, просто начните вводить ее аргументы в круглых скобках. После этого IPython скрывает список автозаполнения. Если потребуется больше информации о текущем выделенном элементе, то вы сможете просмотреть его doc-строку — введите после имени вопросительный знак (`?`) и нажмите Enter, чтобы просмотреть справочную документацию. Ниже приведена doc-строка функции `fabs`:

```
In [4]: math.fabs?
Docstring:
fabs(x)

Return the absolute value of the float x.
Type:      builtin_function_or_method
```

Текст `builtin_function_or_method` показывает, что `fabs` является частью модуля стандартной библиотеки Python. Такие модули считаются встроенными в Python. В данном случае `fabs` является встроенной функцией из модуля `math`.

4.9. Значения параметров по умолчанию

При определении функции можно указать, что *параметр имеет значение по умолчанию*. Если при вызове функции отсутствует аргумент для параметра со значением по умолчанию, то для этого параметра автоматически передается значение по умолчанию. Определим функцию `rectangle_area` со значениями параметров по умолчанию:

```
In [1]: def rectangle_area(length=2, width=3):
   ...:     """Возвращает площадь прямоугольника."""
   ...:     return length * width
   ...:
```

Чтобы задать значение параметра по умолчанию, поставьте после имени параметра знак = и укажите значение — в данном случае значениями по умолчанию являются 2 и 3 для параметров length и width соответственно. Любые параметры со значениями по умолчанию должны находиться в списке *справа* от параметров, не имеющих значений по умолчанию.

Следующий вызов rectangle_area не имеет аргументов, поэтому IPython использует оба значения параметров по умолчанию так, как если бы функция была вызвана в виде rectangle_area(2, 3):

```
In [2]: rectangle_area()
Out[2]: 6
```

Следующий вызов rectangle_area имеет только один аргумент. Аргументы сопоставляются с параметрами слева направо, поэтому для length используется значение 10. Интерпретатор передает для width значение параметра по умолчанию 3, как если бы функция была вызвана в виде rectangle_area(10, 3):

```
In [3]: rectangle_area(10)
Out[3]: 30
```

Новый вызов rectangle_area получает аргументы для length и width, и IPython игнорирует значения параметров по умолчанию:

```
In [4]: rectangle_area(10, 5)
Out[4]: 50
```

4.10. Ключевые аргументы

При вызове функций можно использовать *ключевые* (именованные) *аргументы* для передачи аргументов в *любом* порядке. Чтобы продемонстрировать, как работают ключевые аргументы, переопределим функцию rectangle_area — на этот раз без значений параметров по умолчанию:

```
In [1]: def rectangle_area(length, width):
   ...:     """Возвращает площадь прямоугольника."""
   ...:     return length * width
   ...:
```

Каждый ключевой *аргумент при вызове* задается в форме *имя параметра = значение*. Следующий вызов показывает, что порядок ключевых аргумен-

тов роли не играет — он не обязан соответствовать порядку этих параметров в определении функции:

```
In [2]: rectangle_area(width=5, length=10)
Out[3]: 50
```

В каждом вызове функции ключевые аргументы должны размещаться *после* позиционных аргументов, то есть аргументов, для которых имя параметра не указано. Такие аргументы присваиваются параметрам функции слева направо в соответствии с позицией аргумента в списке. Ключевые аргументы делают более понятными вызовы функций, особенно для функций с несколькими аргументами.

4.11. Произвольные списки аргументов

Функции с *произвольными списками аргументов* (например, встроенные функции `min` и `max`) могут получать *любое* количество аргументов. Возьмем следующий вызов `min`:

```
min(88, 75, 96, 55, 83)
```

В документации функции указано, что `min` имеет два *обязательных* параметра (с именами `arg1` и `arg2`) и необязательный третий параметр в форме `*args`, который означает, что функция может получать любое количество дополнительных аргументов. Оператор `*` перед именем параметра предписывает Python упаковать остальные аргументы в кортеж, который передается параметру `args`. В приведенном вызове параметр `arg1` получает значение `88`, параметр `arg2` — значение `75`, а параметр `args` — кортеж `(96, 55, 83)`.

Определение функции с произвольным списком аргументов

Определим функцию `average`, которая может получать произвольное количество аргументов:

```
In [1]: def average(*args):
   ...:     return sum(args) / len(args)
   ...:
```

По общепринятому соглашению используется имя параметра `args`, но вы можете использовать любой идентификатор. Если функция имеет несколько параметров, то параметр `*args` должен стоять *на последнем месте*.

Теперь вызовем average несколько раз с произвольными списками аргументов разной длины:

```
In [2]: average(5, 10)
Out[2]: 7.5

In [3]: average(5, 10, 15)
Out[3]: 10.0

In [4]: average(5, 10, 15, 20)
Out[4]: 12.5
```

Чтобы вычислить среднее значение, разделим сумму элементов кортежа `args` (возвращаемую встроенной функцией `sum`) на количество элементов в кортеже (возвращаемое встроенной функцией `len`). В нашем определении `average` обратите внимание на то, что при нулевой длине `args` происходит ошибка `ZeroDivisionError`. В следующей главе поясняется, как обратиться к элементам кортежа без их распаковки.

Передача отдельных элементов итерируемого объекта в аргументах функций

Элементы кортежа, списка или другого итерируемого объекта можно распаковать, чтобы передать их в отдельных аргументах функции. *Оператор* *, применяемый к итерируемому объекту из аргумента при вызове функции, распаковывает свои элементы. Следующий код создает список оценок из пяти элементов, после чего использует выражение `*grades` для распаковки своих элементов в аргументы `average`:

```
In [5]: grades = [88, 75, 96, 55, 83]

In [6]: average(*grades)
Out[6]: 79.4
```

Этот вызов эквивалентен вызову `average(88, 75, 96, 55, 83)`.

4.12. Методы: функции, принадлежащие объектам

Метод представляет собой функцию, вызываемую для конкретного объекта в форме

имя_объекта.имя_метода(аргументы)

Например, в следующем сеансе создается строковая переменная `s`, которой присваивается строковый объект `'Hello'`. Затем сеанс вызывает для объекта методы `lower` и `upper`, которые возвращают *новые* строки с версией исходной строки, преобразованной, соответственно, к нижнему или верхнему регистру; переменная `s` при этом остается без изменений:

```
In [1]: s = 'Hello'

In [2]: s.lower()  # вызов s в нижнем регистре
Out[2]: 'hello'

In [3]: s.upper()
Out[3]: 'HELLO'

In [4]: s
Out[4]: 'Hello'
```

Стандартная библиотека Python доступна по адресу:

https://docs.python.org/3/library/index.html

В документации описаны методы встроенных типов и типов из стандартной библиотеки Python. В главе 10 мы займемся созданием *собственных* типов, называемых классами, и определением методов, которые могут вызываться для объектов этих классов.

4.13. Правила области видимости

Каждый идентификатор обладает *областью видимости*, определяющей, в каких местах программы он может использоваться. В этой части программы данный идентификатор находится в области видимости.

Локальная область видимости

Идентификатор локальной переменной обладает *локальной областью видимости*. Он находится в области видимости только от своего определения до конца блока функции. Когда функция возвращает управление на сторону вызова, идентификатор «выходит из области видимости». Таким образом, локальная переменная может использоваться только внутри той функции, в которой она определена.

Глобальная область видимости

Идентификаторы, определенные за пределами любой функции (или класса), обладают *глобальной областью видимости* — к их числу могут принадлежать функции, переменные и классы. Переменные с глобальной областью видимости называются *глобальными переменными*. Идентификаторы с глобальной областью видимости могут использоваться в файлах .py или интерактивных сеансах в любой точке после определения.

Обращение к глобальной переменной из функции

К значению глобальной переменной можно обратиться из функции:

```
In [1]: x = 7

In [2]: def access_global():
   ...:     print('x printed from access_global:', x)
   ...:

In [3]: access_global()
x printed from access_global: 7
```

Однако по умолчанию глобальную переменную не удастся *изменить* внутри функции — когда вы присваиваете значение переменной в блоке функции, Python создает *новую* локальную переменную:

```
In [4]: def try_to_modify_global():
   ...:     x = 3.5
   ...:     print('x printed from try_to_modify_global:', x)
   ...:

In [5]: try_to_modify_global()
x printed from try_to_modify_global: 3.5

In [6]: x
Out[6]: 7
```

В блоке функции `try_to_modify_global` локальная переменная `x` замещает глобальную переменную `x`, в результате чего последняя становится недоступной в области видимости блока функции. Фрагмент [6] показывает, что глобальная переменная `x` продолжает существовать и сохраняет свое исходное значение (`7`) после выполнения функции `try_to_modify_global`.

Чтобы изменить глобальную переменную в блоке функции, используем команду `global` для объявления того, что переменная определена в глобальной области видимости:

```
In [7]: def modify_global():
   ...:     global x
   ...:     x = 'hello'
   ...:     print('x printed from modify_global:', x)
   ...:

In [8]: modify_global()
x printed from modify_global: hello

In [9]: x
Out[9]: 'hello'
```

Блоки и наборы

Ранее мы определяли *блоки* функций и *наборы* управляющих команд. При создании переменной в блоке эта переменная становится *локальной* по отношению к блоку. Если переменная создается в наборе управляющего блока, то область видимости переменной зависит от того, где именно определяется управляющая команда:

- Если управляющая команда находится в глобальной области видимости, то любые переменные, определяемые в управляющей команде, имеют глобальную область видимости.
- Если управляющая команда находится в блоке функции, то любые переменные, определяемые в управляющей команде, имеют локальную область видимости.

Мы вернемся к теме областей видимости в главе 10, когда займемся написанием пользовательских классов.

Замещение функций

В предыдущих главах при суммировании значений сумма сохранялась в переменной с именем `total`. А почему не `sum`? Дело в том, что `sum` является встроенной функцией. Если вы определите переменную с именем `sum`, то переменная *заместит* встроенную функцию, и та станет недоступной для вашего кода. При выполнении следующей команды Python связывает идентификатор `sum` с объектом `int`, содержащим `15`. На этой стадии идентификатор `sum` уже не ссылается на встроенную функцию. Таким образом, при попытке использования `sum` как функции произойдет ошибка `TypeError`:

```
In [10]: sum = 10 + 5

In [11]: sum
Out[11]: 15

In [12]: sum([10, 5])
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-1237d97a65fb> in <module>()
----> 1 sum([10, 5])

TypeError: 'int' object is not callable
```

Команды в глобальной области видимости

В сценариях, которые приводились ранее, мы написали несколько команд, которые находились в глобальной области видимости вне функций, и несколько команд внутри блоков функций. Команды сценария с глобальной областью видимости выполняются сразу же, как только они встретятся интерпретатору, тогда как команды в блоках выполняются только при вызове функции.

4.14. Подробнее об импортировании

Для импортирования модулей (таких как `math` и `random`) мы ранее использовали команды следующего вида:

```
import имя_модуля
```

а затем обращались к нужной функциональности с указанием имени модуля через точку (`.`). Кроме того, из модуля также можно импортировать конкретный идентификатор (например, тип `Decimal` из модуля `decimal`) командой следующего вида:

```
from имя_модуля import идентификатор
```

а затем использовать его без префикса из имени модуля с точкой (`.`).

Импортирование нескольких идентификаторов из модуля

Использование команды `from...import` позволяет импортировать разделенный запятыми список идентификаторов из модуля, а затем использовать их без указания имени модуля и точки (`.`):

```
In [1]: from math import ceil, floor

In [2]: ceil(10.3)
Out[2]: 11

In [3]: floor(10.7)
Out[3]: 10
```

Попытка использования неимпортированной функции приводит к ошибке `NameError`; она указывает на то, что имя не определено.

Не используйте импортирование с *

Все идентификаторы, определенные в модуле, можно импортировать массовой формой команды

```
from имя_модуля import *
```

Все идентификаторы модуля становятся доступными для использования в вашем коде. Импортирование идентификаторов модуля с * может привести к коварным ошибкам — такая практика считается опасной, и ее следует избегать. Возьмем следующие фрагменты:

```
In [4]: e = 'hello'

In [5]: from math import *

In [6]: e
Out[6]: 2.718281828459045
```

Изначально строка `'hello'` присваивается переменной с именем `e`. Однако после выполнения фрагмента [5] переменная `e` заменяется (вероятно, непреднамеренно) константой `e` из модуля `math`, представляющей математическую константу с плавающей точкой *e*.

Определение синонимов для модулей и идентификаторов модулей

Иногда бывает полезно импортировать модуль и назначить ему сокращение для упрощения кода. Секция `as` команды `import` позволяет задать имя, которое будет использоваться для обращения к идентификаторам модуля. Например, в разделе 3.14 можно было импортировать модуль `statistics` и обратиться к его функции `mean` следующим образом:

```
In [7]: import statistics as stats

In [8]: grades = [85, 93, 45, 87, 93]

In [9]: stats.mean(grades)
Out[9]: 80.6
```

В последующих главах конструкция `import...as` часто используется для импортирования библиотек Python с удобными сокращениями — например, `stats` для модуля `statistics`. Или, например, мы будем использовать модуль `numpy`, который обычно импортируется командой

```
import numpy as np
```

В документации к библиотекам указаны популярные сокращенные имена.

Как правило, при импортировании модуля следует использовать команды `import` или `import...as`, а затем обращаться к функциональности модуля по имени модуля или по сокращению, следующему за ключевым словом `as` соответственно. Тем самым предотвращается случайное импортирование идентификатора, конфликтующего с идентификатором в вашем коде.

4.15. Подробнее о передаче аргументов функциям

Давайте подробнее разберемся в том, как аргументы передаются функциям. Во многих языках программирования существуют два механизма передачи аргументов — *передача по значению* и *передача по ссылке*.

- При передаче по значению вызываемая функция получает *копию значения* аргумента и работает исключительно с этой копией. Изменения в копии, принадлежащей функции, *не* затрагивают значение исходной переменной на стороне вызова.
- При передаче по ссылке вызываемая функция может обратиться к значению аргумента на стороне вызова напрямую и изменить это значение, если оно не является неизменяемым.

Аргументы в Python всегда передаются по ссылке. Некоторые разработчики называют этот механизм «*передачей по ссылке на объект*», потому что «все в Python является объектом»[1]. Когда при вызове функции передается аргумент,

[1] Даже функции, которые вы определяли в этой главе, и классы (пользовательские типы), которые будут определяться в следующих главах, в Python являются объектами.

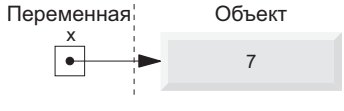
Python копирует *ссылку* на объект аргумента (не сам объект!) в соответствующий параметр. Это важно для быстродействия программы. Функции часто работают с большими объектами — постоянное копирование таких объектов ведет к большим затратам компьютерной памяти и существенному замедлению работы программы.

Адреса памяти, ссылки и «указатели»

Все взаимодействие с объектом выполняется по ссылке, которая во внутренней реализации представляет адрес объекта (его местоположение в памяти компьютера) — в других языках иногда используется термин «указатель». После присваивания вида

```
x = 7
```

переменная `x` на самом деле не содержит значение `7`. Вместо этого она содержит ссылку на *объект*, содержащий значение `7`, хранящийся *где-то* в памяти. Можно сказать, что `x` «указывает» на объект, содержащий `7`, как на следующей диаграмме:



Встроенная функция id и идентичность объектов

Теперь посмотрим, как происходит передача аргументов функциям. Начнем с создания целочисленной переменной `x`, упоминавшейся выше, — вскоре `x` будет использоваться как аргумент функции:

```
In [1]: x = 7
```

Теперь `x` *ссылается* (или *указывает*) на объект, представляющий целое число со значением `7`. Два объекта ни при каких условиях не могут размещаться в памяти по одному адресу, поэтому каждый объект в памяти обладает *уникальным адресом*.

Хотя фактический адрес объекта остается неизвестным, вы можете воспользоваться встроенной *функцией* `id` для получения *уникального* значения `int`,

идентифицирующего этот объект, пока он остается в памяти (скорее всего, при выполнении этого фрагмента на своем компьютере вы получите другое значение):

```
In [2]: id(x)
Out[2]: 4350477840
```

Целочисленный результат вызова `id` называется *идентичностью* объекта[1]. Два объекта в памяти не могут обладать одинаковой *идентичностью*. Мы будем использовать идентичность объектов для демонстрации того, что объекты передаются по ссылке.

Передача объекта функции

Определим функцию `cube`, которая выводит идентичность своего параметра, а затем возвращает значение параметра, возведенное в куб:

```
In [3]: def cube(number):
   ...:     print('id(number):', id(number))
   ...:     return number ** 3
   ...:
```

Теперь вызовем `cube` с аргументом `x`, который указывает на объект, содержащий значение `7`:

```
In [4]: cube(x)
id(number): 4350477840
Out[4]: 343
```

Идентичность, выводимая для параметра `cube` — `4350477840`, — *совпадает* с той, которая ранее выводилась для `x`. Поскольку каждый объект обладает уникальной идентичностью, *аргумент* `x` и *параметр* `number` ссылаются на *один и тот же* объект во время выполнения `cube`. Таким образом, когда функция `cube` использует свой параметр `number` в вычислениях, она получает значение `number` из исходного объекта на стороне вызова.

---

[1] Согласно документации Python, в зависимости от используемой реализации идентичность объекта может совпадать с фактическим адресом объекта в памяти, что, впрочем, не обязательно.

Проверка идентичности объекта оператором is

Чтобы доказать, что аргумент и параметр ссылаются на один и тот же объект, можно воспользоваться *оператором* Python `is`. Этот оператор возвращает `True`, если идентичности двух операндов *совпадают*:

```
In [5]: def cube(number):
   ...:     print('number is x:', number is x)  # x - глобальная переменная
   ...:     return number ** 3
   ...:

In [6]: cube(x)
number is x: True
Out[6]: 343
```

Неизменяемые объекты как аргументы

Когда функция получает в аргументе ссылку на *неизменяемый* объект (например, `int`, `float`, `string` или `tuple`), даже при том, что вы можете напрямую обратиться к исходному объекту на стороне вызова, вам не удастся изменить значение исходного неизменяемого объекта. Чтобы убедиться в этом, изменим функцию `cube`, чтобы она выводила `id(number)` до и после присваивания нового объекта параметру `number` с использованием расширенного присваивания:

```
In [7]: def cube(number):
   ...:     print('id(number) before modifying number:', id(number))
   ...:     number **= 3
   ...:     print('id(number) after modifying number:', id(number))
   ...:     return number
   ...:

In [8]: cube(x)
id(number) before modifying number: 4350477840
id(number) after modifying number: 4396653744
Out[8]: 343
```

При вызове `cube(x)` первая команда `print` показывает, что значение `id(number)` изначально совпадало с `id(x)` из фрагмента [2]. Числовые значения неизменяемы, поэтому команда

```
number **= 3
```

на самом деле создает *новый объект* со значением, возведенным в куб, а затем присваивает ссылку на этот объект параметру `number`. Вспомните, что

при отсутствии ссылок на *исходный* объект он будет уничтожен *уборщиком мусора*. Вторая команда `print` функции `cube` выводит идентичность *нового* объекта. Идентичности объектов должны быть уникальными. Следовательно, переменная `number` содержит ссылку на *другой* объект. Чтобы показать, что значение `x` не изменилось, снова выведем значение и идентичность переменной:

```
In [9]: print(f'x = {x}; id(x) = {id(x)}')
x = 7; id(x) = 4350477840
```

Изменяемые объекты в аргументах

В следующей главе мы покажем, что при передаче функции ссылки на *изменяемый* объект (например, список) функция *может* изменить исходный объект на стороне вызова.

4.16. Рекурсия

Напишем программу для выполнения известных математических вычислений. *Факториал* положительного целого числа n записывается в виде $n!$ (читается «n факториал».) Он вычисляется как произведение

$$n \times (n - 1) \times (n - 2) \times ... \times 1.$$

При этом и 1!, и 0! определяются равными 1. Но, например, 5! вычисляется как произведение $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, то есть его значение равно 120.

Итеративное вычисление факториала

Значение 5! можно вычислить *итеративным* методом с использованием цикла `for`:

```
In [1]: factorial = 1

In [2]: for number in range(5, 0, -1):
   ...:     factorial *= number
   ...:

In [3]: factorial
Out[3]: 120
```

Рекурсивное решение задачи

Методы рекурсивного решения задач имеют несколько общих элементов. Когда вы вызываете рекурсивную функцию для решения задачи, она в действительности умеет решать только *простейший*, или *базовый*, *случай(-и)* задачи. Если вызвать функцию для *базового случая*, то она немедленно вернет результат. Если же функция вызывается для более сложной задачи, то эта задача обычно делится на две части: одну из них функция умеет решать, другую — нет. Чтобы рекурсия была приемлемой, вторая часть должна быть слегка упрощенной или сокращенной версией исходной задачи. Так как новая задача напоминает исходную задачу, функция вызывает новую *копию* самой себя для работы над меньшей задачей — этот *вызов* называется *рекурсивным* (также используется термин «*шаг рекурсии*»). Концепция разбиения задачи на две меньшие части является разновидностью известного принципа «*разделяй и властвуй*».

На время выполнения шага рекурсии исходный вызов функции остается активным (то есть его выполнение еще не завершилось). Это может привести к появлению множества новых рекурсивных вызовов, так как функция делит каждую новую подзадачу на две концептуальные части.

Чтобы рекурсия рано или поздно завершилась, каждый раз, когда функция вызывает сама себя с упрощенной версией исходной задачи, последовательность уменьшающихся задач должна *сходиться к базовому случаю*. Когда функция распознает базовый случай, она возвращает результат предыдущей копии функции. Последовательность возвратов продолжается до тех пор, пока исходный вызов функции не вернет окончательный результат на сторону вызова.

Рекурсивное вычисление факториала

До перехода к рекурсивному представлению задачи вычисления факториала заметим, что $n!$ можно записать и в виде:

$$n! = n \cdot (n - 1)!$$

Например, выражение 5! эквивалентно $5 \cdot 4!$:

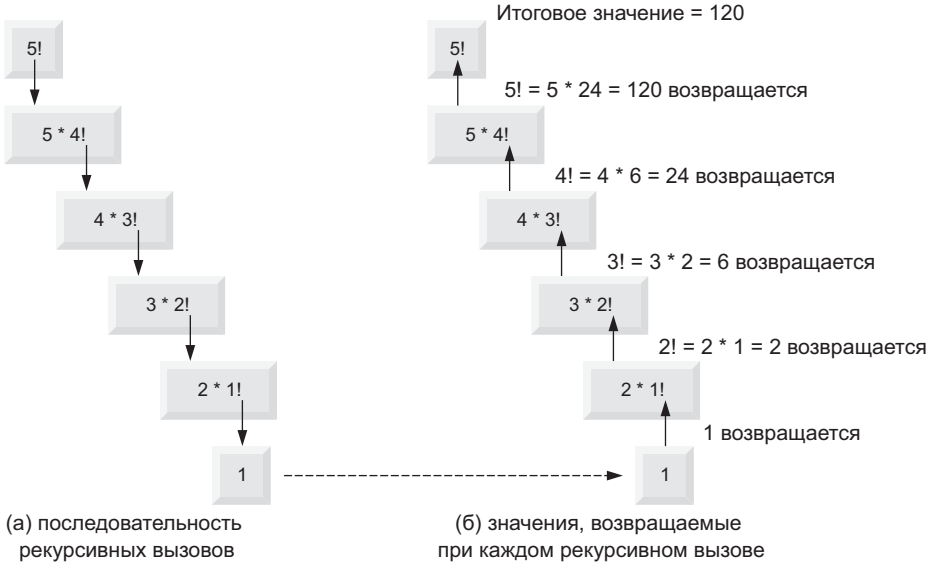
$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

$$5! = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1)$$

$$5! = 5 \cdot (4!)$$

Наглядное представление рекурсии

Процесс вычисления 5! продемонстрирован на следующей схеме. Левый столбец показывает, как проходит последовательность рекурсивных вызовов до того момента, как для 1! (базовый случай) возвращается 1, что завершает рекурсию. В правом столбце показаны (снизу вверх) значения, возвращаемые на каждом шаге рекурсии на сторону вызова, пока не будет вычислено и возвращено итоговое значение.



Реализация рекурсивной функции вычисления факториала

Следующий сеанс использует рекурсию для вычисления и вывода факториалов целых чисел от 0 до 10:

```
In [4]: def factorial(number):
   ...:     """Возвращает факториал числа."""
   ...:     if number <= 1:
   ...:         return 1
   ...:     return number * factorial(number - 1)  # Рекурсивный вызов
   ...:
In [5]: for i in range(11):
   ...:     print(f'{i}! = {factorial(i)}')
   ...:
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

Рекурсивная функция `factorial` из фрагмента [4] сначала проверяет, `истинно` ли *условие завершения* `number <= 1`. Если условие `истинно` (базовый случай), то `factorial` возвращает 1, и дальнейшая рекурсия не нужна. Если `number` больше 1, то вторая команда `return` выражает задачу в форме произведения `number` и *рекурсивного вызова* `factorial`, который вычисляет `factorial(number - 1)`. Эта задача немного проще исходного вычисления `factorial(number)`. Стоит напомнить, что функция `factorial` должна получать *неотрицательный* аргумент. В нашем примере это условие не проверяется.

Цикл в фрагменте [5] вызывает функцию `factorial` для значений от 0 до 10. Из вывода видно, что значения `factorial` стремительно растут. *В отличие от многих других языков программирования, Python не ограничивает размер целого числа.*

Косвенная рекурсия

Рекурсивная функция может вызвать другую функцию, которая, в свою очередь, может снова вызвать рекурсивную функцию. Такая ситуация называется *косвенным рекурсивным вызовом*, или *косвенной рекурсией*. Например, функция `A` вызывает функцию `B`, которая снова вызывает функцию `A`. Ситуация по-прежнему остается рекурсией, потому что второй вызов функции `A` совершается в то время, когда первый вызов функции `A` остается активным. Иначе говоря, первый вызов функции `A` еще не завершил выполнение (потому что он ожидает, пока функция `B` вернет ему результат) и еще не вернул управление на исходную сторону вызова функции `A`.

Переполнение стека и бесконечная рекурсия

Конечно, объем памяти на компьютере конечен, поэтому память для хранения записей активизации стека вызовов ограниченна. Если количество рекурсив-

ных вызовов превысит емкость стека по хранению записей активизации, то происходит фатальная ошибка, называемая *переполнением стека*. Обычно она является результатом *бесконечной рекурсии*, возникающей из-за того, что разработчик пропустил базовый случай или неправильно записал шаг рекурсии, вследствие чего последовательность не сходится к базовому случаю. Эта ошибка аналогична проблеме *бесконечного цикла* в *итеративном* (не рекурсивном) решении.

4.17. Программирование в функциональном стиле

Python, как и другие популярные языки (такие как Java и C#), не является чисто функциональным языком, поскольку предоставляет средства «в функциональном стиле», помогающие писать код более компактный, более простой для чтения, отладки и изменения и с меньшей вероятностью содержащий ошибки. Программы в функциональном стиле также проще параллелизуются, чтобы добиться более высокого быстродействия на современных многоядерных процессорах. Ниже (табл. 4.3) перечислены основные средства программирования в функциональном стиле в языке Python (в круглых скобках указаны главы, в которых впервые рассматриваются многие из них).

Таблица 4.3. Программирование в функциональном стиле

Темы программирования в функциональном стиле	
внутренние итерации (4)	отложенное вычисление (5)
выражения-генераторы (5)	предотвращение побочных эффектов (4)
декларативное программирование (4)	свертки (3, 5)
декораторы (10)	трансформации множеств (6)
замыкания	трансформации словарей (6)
итераторы (3)	трансформации списков (5)
лямбда-выражение (5)	функции высшего порядка (5)
модуль `itertools` (16)	функции-генераторы
модуль `functools`	функция `range` (3, 4)
модуль `operator` (5, 11, 16)	чистые функции (4)
неизменяемость (4)	`filter/map/reduce` (5)

Многие из этих возможностей рассматриваются в книге — одни с примерами кода, другие с точки зрения обучения. Мы уже использовали списки, строки и встроенную функцию `range` с командой `for`, а также некоторые *свертки* (функции `sum`, `len`, `min` и `max`). Ниже рассматриваются темы декларативного программирования, неизменяемости и внутренних итераций.

«Что» и «как»

Со временем задачи, которые вы решаете, станут более сложными, а ваш код будет сложнее читать, отлаживать и модифицировать, — соответственно, и ошибки в нем станут более вероятными, и сложнее будет определить, *как* должен работать код.

Программирование в функциональном стиле позволяет просто указать, *что* вы хотите сделать. Оно скрывает многие подробности того, *как* должна выполняться та или иная задача. Как правило, библиотечный код позволяет устранить многие ошибки при минимальном вмешательстве программиста в этот процесс.

Возьмем команду `for` во многих языках программирования. Обычно все подробности итераций, управляемых счетчиком, должны быть заданы разработчиком: управляющая переменная, ее исходное значение, ее приращение и условие продолжения цикла, которое использует управляющее значение для определения того, следует ли продолжать итерации. Такой механизм называется *внешними итерациями* и сопряжен с высоким риском ошибок. Например, вы можете предоставить неправильный инициализатор, приращение или условие продолжения цикла. Внешние итерации *изменяют* управляющую переменную, и в наборе команды `for` часто изменяются другие переменные. Любое изменение переменных может создать ошибку в программе. Программирование в функциональном стиле уделяет особое внимание *неизменяемости*. Иначе говоря, оно избегает операций, которые изменяют значения переменных (подробнее об этом в следующей главе).

Команда `for` и функция `range` в Python *скрывают* большинство подробностей итераций, управляемых счетчиком. Предположим, вы указываете, *какие* значения должна генерировать функция `range`, и переменную, которой должно последовательно присваиваться каждое сгенерированное значение. Функция `range` знает, *как* генерировать эти значения. Аналогичным образом команда `for` *знает*, *как* получить каждое значение из `range` и *как* остановить итерации,

если значений больше нет. Указание того, *что* нужно сделать, но не *как* это должно делаться, является важным аспектом *внутренних итераций* — ключевой концепции программирования в функциональном стиле.

Каждая из встроенных функций Python `sum`, `min` и `max` использует внутренние итерации. Чтобы просуммировать элементы списка `grades`, вы просто объявляете, *что* именно намерены сделать, то есть включаете вызов `sum(grades)`. Функция `sum` *знает, как* перебрать элементы списка и прибавить каждый элемент к накапливаемой сумме. Парадигма, в которой вы определяете, *что* должно быть сделано, но не программируете выполнение операции во всех подробностях, называется *декларативным программированием*.

Чистые функции

В чистых языках функционального программирования разработчик концентрируется на написании чистых функций. Результат *чистой функции* зависит только от передаваемых ей аргументов. Кроме того, для конкретных аргументов чистая функция всегда выдает один и тот же результат. Например, возвращаемое значение встроенной функции `sum` зависит только от итерируемого объекта, который ей передается. Для заданного списка `[1, 2, 3]` функция `sum` *всегда* возвращает 6, сколько бы раз она ни была вызвана. Кроме того, чистая функция не имеет *побочных эффектов*. Например, при передаче чистой функции *изменяемого* списка последний будет содержать одинаковые значения до и после вызова функции. При вызове чистой функции `sum` она не изменяет свой аргумент.

```
In [1]: values = [1, 2, 3]

In [2]: sum(values)
Out[2]: 6

In [3]: sum(values)  # Одинаковые вызовы всегда возвращают одинаковые результаты
Out[3]: 6

In [4]: values
Out[5]: [1, 2, 3]
```

В следующей главе мы продолжим пользоваться концепциями программирования в функциональном стиле. Кроме того, вы увидите, что *функции являются объектами*, которые могут передаваться другим функциям как данные.

4.18. Введение в data science: дисперсионные характеристики

В нашем обсуждении описательной статистики были рассмотрены характеристики, описывающие положение центра распределения, — математическое ожидание, медиана и мода. Эти характеристики помогают разделить типичные значения на группы — например, средний рост ваших однокурсников (математическое ожидание) или наиболее часто приобретаемая марка автомобиля для заданной страны (мода).

Когда речь заходит о группе, ее обычно называют *генеральной совокупностью* (population). Иногда генеральная совокупность оказывается достаточно большой — например, численность жителей США, которые с большой вероятностью будут голосовать на следующих президентских выборах, превышает 100 000 000 человек. По практическим соображениям организации по исследованию общественного мнения, пытающиеся предсказать, кто станет следующим президентом, работают с тщательно отобранными небольшими подмножествами генеральной совокупности, которые называются *выборками* (samples). Во многих опросах на выборах 2016 года размер выборки составлял около 1000 человек.

Продолжим обсуждение базовых описательных статистических характеристик. Коснемся теперь *дисперсионных характеристик* (также называемых *мерами изменчивости*), которые помогают понять, насколько силен разброс значений. Например, в учебной группе у большинства студентов рост может быть близок к среднему, а незначительная часть студентов может быть существенно ниже или выше остальных.

Каждая дисперсионная метрика будет вычисляться как вручную, так и с использованием функций из модуля `statistics` для следующей генеральной совокупности 10 бросков шестигранного кубика:

```
1, 3, 4, 2, 6, 5, 3, 4, 5, 2
```

Дисперсия

Чтобы вычислить *дисперсию*[1], начнем с вычисления математического ожидания этих значений — `3.5`. Этот результат получается делением суммы выпав-

[1] Для простоты мы вычисляем *дисперсию генеральной совокупности*. Существуют тонкие различия между *дисперсией генеральной совокупности* и *дисперсией выборки*. Вместо

ших значений 35 на количество бросков 10. Затем математическое ожидание вычитается из каждого выпавшего результата (при этом некоторые результаты окажутся отрицательными):

```
-2.5, -0.5, 0.5, -1.5, 2.5, 1.5, -0.5, 0.5, 1.5, -1.5
```

Затем каждый из результатов возводится в квадрат (при этом будут получены только положительные значения):

```
6.25, 0.25, 0.25, 2.25, 6.25, 2.25, 0.25, 0.25, 2.25, 2.25
```

Наконец, мы вычисляем среднее арифметическое этих квадратов, равное `2.25` (22.5 / 10). Полученный результат и является *дисперсией генеральной совокупности*. Возведение в квадрат разности каждого значения и математического ожидания всех значений усиливает влияние *выбросов* — значений, расположенных дальше всего от математического ожидания. При более глубоком анализе данных иногда приходится обращать более пристальное внимание на выбросы, хотя порой их лучше игнорировать. В следующем коде функция `pvariance` модуля `statistics` используется для подтверждения результата, полученного вручную:

```
In [1]: import statistics

In [2]: statistics.pvariance([1, 3, 4, 2, 6, 5, 3, 4, 5, 2])
Out[2]: 2.25
```

Стандартное отклонение

Стандартное (*среднеквадратическое*) *отклонение* равно квадратному корню из дисперсии (в данном случае `1.5`); эта метрика несколько компенсирует эффект выбросов. Чем меньше дисперсия и стандартное отклонение, тем ближе значения данных к математическому ожиданию и тем меньше общий *разброс* между значениями и математическим ожиданием. Следующий фрагмент вычисляет *стандартное отклонение генеральной совокупности* с использованием

деления на n (количество бросков кубиков в нашем примере) в выборочной дисперсии используется деление на $n - 1$. Это различие играет важную роль для малых выборок и становится несущественным при возрастании размера выборки. Модуль `statistics` предоставляет функции `pvariance` и `variance` для вычисления дисперсии генеральной совокупности и выборочной дисперсии соответственно. Аналогичным образом модуль `statistics` предоставляет функции `pstdev` и `stdev` для вычисления стандартного отклонения генеральной совокупности и выборочного стандартного отклонения соответственно.

функции `pstdev` модуля `statistics`, подтверждая вычисленный вручную результат:

```
In [3]: statistics.pstdev([1, 3, 4, 2, 6, 5, 3, 4, 5, 2])
Out[3]: 1.5
```

Передача результата функции `pvariance` функции `sqrt` модуля `math` подтверждает результат 1.5:

```
In [4]: import math

In [5]: math.sqrt(statistics.pvariance([1, 3, 4, 2, 6, 5, 3, 4, 5, 2]))
Out[5]: 1.5
```

Преимущество стандартного отклонения генеральной совокупности перед дисперсией генеральной совокупности

Предположим, вы записали средние значения температуры в своем городе за месяц, например: 19, 32, 28 и 35. Эти значения измеряются в градусах. Когда вы возводите температуры в квадрат для вычисления дисперсии генеральной совокупности, дисперсия генеральной совокупности измеряется в градусах в квадрате. Но когда вычисляется квадратный корень из дисперсии генеральной совокупности, единицей измерения снова становится градус — *та же* единица, в которой сохранялись результаты измерений.

4.19. Итоги

В этой главе мы занимались созданием пользовательских функций. При этом импортировалась функциональность из модулей `random` и `math`. Мы познакомились с генератором случайных чисел и использовали его для моделирования бросков шестигранного кубика. Чтобы из функции можно было вернуть сразу несколько значений, эти значения были упакованы в кортеж. Позднее кортеж был распакован для обращения к его значениям. Мы обсудили использование модулей стандартной библиотеки Python, для того чтобы избежать «изобретения велосипеда».

Далее рассматривалось создание функций со значениями параметров по умолчанию и вызовы функций с ключевыми аргументами, а также определение функций с произвольными списками аргументов и вызовы методов объектов. Вы узнали, как область видимости идентификатора определяет, в каких точках программы может использоваться этот идентификатор.

Затем было приведено более подробное описание импортирования модулей. Вы узнали, что аргументы передаются функциям по ссылке и то, как стек вызовов функций и кадры стека поддерживают механизм вызова/возвращения управления из функций. Также мы представили рекурсивные функции и начали знакомить вас со средствами программирования в функциональном стиле в языке Python. В двух последних главах мы представили базовые средства для работы со списками и кортежами, а в следующей главе эти структуры данных будут рассмотрены более подробно.

В рамках обсуждения описательной статистики мы коснулись дисперсионных характеристик — дисперсии и стандартного отклонения и их вычисления с использованием функций модуля `statistics` из стандартной библиотеки Python.

В некоторых типах задач бывает полезно, чтобы функция вызывала сама себя. *Рекурсивные функции* могут вызывать себя как напрямую, так и косвенно через другую функцию.

5

Последовательности: списки и кортежи

В этой главе...

- Создание и инициализация списков и кортежей.
- Обращение к элементам списков, кортежей и строк.
- Сортировка и поиск в списках, поиск в кортежах.
- Передача списков и кортежей функциям и методам.
- Использование методов списков для выполнения стандартных операций со строками: поиска элементов, сортировки списка, вставки и удаления элементов.
- Использование дополнительных средств Python в стиле функционального программирования, включая лямбда-выражения и операции программирования в функциональном стиле — фильтрацию, отображение и свертку.
- Использование трансформаций списков для быстрого и удобного создания списков; использование выражений-генераторов для генерирования значений по требованию.
- Двумерные списки.
- Расширение возможностей анализа и представления информации при помощи библиотек визуализации Seaborn и Matplotlib.

5.1. Введение

В двух последних главах мы кратко представили списки и кортежи — типы последовательностей для представления упорядоченных коллекций элементов. *Коллекции* представляют собой структуры данных, состоящие из взаимосвязанных элементов данных. Примерами коллекций могут служить подборки ваших любимых песен на смартфоне, списки контактов, библиотечных книг, ваши карты в карточной игре, игроки вашей любимой спортивной команды, акции в инвестиционном портфеле, списки покупок и т. д. Встроенные коллекции Python позволяют хранить данные и обращаться к ним удобно и эффективно. В этой главе списки и кортежи будут рассмотрены более подробно.

Мы продемонстрируем основные операции со списками и кортежами. Вы увидите, что списки (которые являются изменяемыми) и кортежи (которые изменяться не могут) обладают многими общими возможностями. И в списках, и в кортежах могут храниться элементы одного или разных типов. Списки (одномерные и двумерные) могут *динамически изменять свои размеры* по мере надобности, увеличиваясь и уменьшаясь во время выполнения.

В предыдущей главе мы продемонстрировали генерирование случайных чисел и моделирование серии бросков шестигранного кубика. Эта глава завершается следующим разделом «Введение в data science», в котором библиотеки визуализации Seaborn и Matplotlib используются для интерактивного построения статических гистограмм с частотами выпадения разных значений на кубиках. В разделе «Введение в data science» следующей главы будет представлена анимированная визуализация, в которой вид гистограммы изменяется *динамически* с ростом количества бросков, — таким образом вы сможете понаблюдать за законом больших чисел в действии.

5.2. Списки

Теперь рассмотрим списки более подробно и объясним, как обратиться к конкретному *элементу* списка. Многие средства, представленные в этом разделе, применимы ко всем типам последовательностей.

Создание списка

В списках обычно хранятся *однородные данные*, то есть значения *одного* типа данных. Возьмем список с, содержащий пять целочисленных элементов:

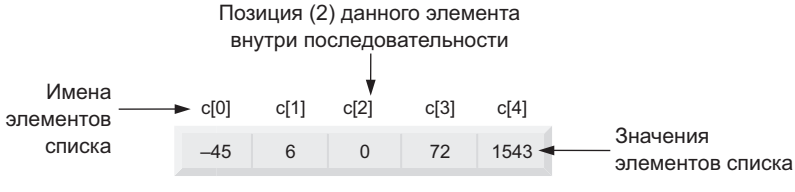
```
In [1]: c = [-45, 6, 0, 72, 1543]

In [2]: c
Out[2]: [-45, 6, 0, 72, 1543]
```

Однако в списках также могут храниться *разнородные данные*, то есть данные разных типов. Например, в следующем списке хранится имя студента (строка), его фамилия (строка), его средняя оценка (`float`) и год окончания студентом учебного заведения (`int`):

```
['Mary', 'Smith', 3.57, 2022]
```

Обращение к элементам списка

Чтобы обратиться к элементу списка, укажите имя списка, за которым следует *индекс* элемента (то есть *номер* его *позиции*) в квадратных скобках (`[]`); эта конструкция называется *оператором индексирования*. На следующей диаграмме изображен список `c` с именами его элементов:



Первый элемент в списке имеет индекс 0. Таким образом, в списке из пяти элементов `c` первому элементу соответствует имя `c[0]`, а последнему `c[4]`:

```
In [3]: c[0]
Out[3]: -45

In [4]: c[4]
Out[4]: 1543
```

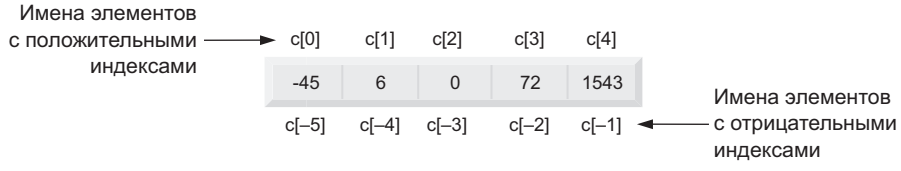
Определение длины списка

Для определения длины списка используется встроенная *функция* `len`:

```
In [5]: len(c)
Out[5]: 5
```

Обращение к элементам от конца списка

К элементам списка также можно обращаться, указывая их смещение от конца списка при помощи *отрицательных индексов*:



Таким образом, к последнему элементу списка `c` (`c[4]`) можно обратиться в виде `c[-1]`, а к первому — в виде `c[-5]`:

```
In [6]: c[-1]
Out[6]: 1543

In [7]: c[-5]
Out[7]: -45
```

Индексы должны быть целыми числами или целочисленными выражениями

Индекс должен быть целым числом или выражением, дающим целочисленный результат (или *сегментом*, как вскоре будет показано):

```
In [8]: a = 1

In [9]: b = 2

In [10]: c[a + b]
Out[10]: 72
```

При попытке использования нецелочисленного индекса происходит ошибка `TypeError`.

Изменяемость списков

Списки являются изменяемыми, то есть их элементы можно модифицировать:

```
In [11]: c[4] = 17

In [12]: c
Out[12]: [-45, 6, 0, 72, 17]
```

Вскоре вы увидите, что со списками также можно выполнять операции вставки и удаления, приводящие к изменению длины списка.

Некоторые последовательности неизменяемы

Такие последовательности Python, как строки и кортежи, являются неизменяемыми — это означает, что их нельзя модифицировать в программе. Вы можете прочитать отдельные символы в строке, но при попытке присвоить новое значение одному из символов происходит ошибка `TypeError`:

```
In [13]: s = 'hello'

In [14]: s[0]
Out[14]: 'h'

In [15]: s[0] = 'H'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-812ef2514689> in <module>()
----> 1 s[0] = 'H'

TypeError: 'str' object does not support item assignment
```

Попытка обращения к несуществующему элементу

При использовании индекса, выходящего за границы диапазона, кортежа или строки, происходит ошибка `IndexError`:

```
In [16]: c[100]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-16-9a31ea1e1a13> in <module>()
----> 1 c[100]

IndexError: list index out of range
```

Использование элементов списков в выражениях

Элементы списков могут использоваться как переменные в выражениях:

```
In [17]: c[0] + c[1] + c[2]
Out[17]: -39
```

Присоединение элементов к списку оператором +=

Начнем с *пустого* списка `list []`, а затем в цикле `for` присоединим к нему значения от 1 до 5 — список динамически расширяется, чтобы вместить новые элементы:

```
In [18]: a_list = []

In [19]: for number in range(1, 6):
    ...:     a_list += [number]
    ...:

In [20]: a_list
Out[20]: [1, 2, 3, 4, 5]
```

Когда левый операнд `+=` является списком, правый операнд должен быть *итерируемым объектом*; в противном случае происходит ошибка `TypeError`. В наборе фрагмента [19] квадратные скобки вокруг числа создают список из одного элемента, который присоединяется к списку. Если правый операнд содержит несколько элементов, то операнд `+=` присоединяет их все. В следующем фрагменте символы слова `'Python'` присоединяются к списку `letters`:

```
In [21]: letters = []

In [22]: letters += 'Python'

In [23]: letters
Out[23]: ['P', 'y', 't', 'h', 'o', 'n']
```

Если правым операндом `+=` является кортеж, то его элементы также присоединяются к списку. Позднее в этой главе мы используем метод списков `append` для добавления элементов в список.

Конкатенация списков оператором +

Оператор `+` может использоваться для *конкатенации* (слияния) двух списков, двух кортежей или двух строк. Результатом является *новая* последовательность того же типа, содержащая элементы левого операнда, за которыми следуют элементы правого операнда. Исходные последовательности остаются неизменными:

```
In [24]: list1 = [10, 20, 30]

In [25]: list2 = [40, 50]

In [26]: concatenated_list = list1 + list2

In [27]: concatenated_list
Out[27]: [10, 20, 30, 40, 50]
```

Если операнды оператора + относятся к разным типам последовательностей, например происходит ошибка `TypeError`, то и конкатенация списка и кортежа является ошибкой.

Использование for и range для обращения к индексам и значениям списков

К элементам списков также можно обращаться по их индексам с использованием оператора индексирования (`[]`):

```
In [28]: for i in range(len(concatenated_list)):
    ...:     print(f'{i}: {concatenated_list[i]}')
    ...:
0: 10
1: 20
2: 30
3: 40
4: 50
```

Вызов функции `range(len(concatenated_list))` выдает последовательность целых чисел, представляющих индексы `concatenated_list` (в данном случае от 0 до 4). Перебирая элементы таким способом, необходимо следить за тем, чтобы индексы не вышли за пределы диапазона. Вскоре мы покажем более безопасный способ обращения к индексам и значениям элементов с использованием встроенной функции `enumerate`.

Операторы сравнения

Операторы сравнения также могут использоваться для поэлементного сравнения целых списков:

```
In [29]: a = [1, 2, 3]

In [30]: b = [1, 2, 3]

In [31]: c = [1, 2, 3, 4]
```

```
In [32]: a == b  # True: соответствующие элементы обоих списков равны
Out[32]: True

In [33]: a == c  # False: у a и c различаются элементы и длины
Out[33]: False

In [34]: a < c  # True: a содержит меньше элементов, чем c
Out[34]: True

In [35]: c >= b  # True: элементы 0-2 равны, но в c больше элементов
Out[35]: True
```

5.3. Кортежи

Кортежи неизменяемы, в них часто хранятся разнородные данные, но данные также могут быть однородными. Длина кортежа определяется количеством элементов в кортеже и не может изменяться во время выполнения программы.

Создание кортежей

Чтобы создать пустой кортеж, используйте пустые круглые скобки:

```
In [1]: student_tuple = ()

In [2]: student_tuple
Out[2]: ()

In [3]: len(student_tuple)
Out[3]: 0
```

Напомним, для упаковки элементов в кортеж можно перечислить их, разделяя запятыми:

```
In [4]: student_tuple = 'John', 'Green', 3.3

In [5]: student_tuple
Out[5]: ('John', 'Green', 3.3)

In [6]: len(student_tuple)
Out[6]: 3
```

Когда вы выводите кортеж, Python всегда отображает его содержимое в круглых скобках. Список значений кортежа, разделенных запятыми, также можно заключить в круглые скобки (хотя это и не обязательно):

```
In [7]: another_student_tuple = ('Mary', 'Red', 3.3)

In [8]: another_student_tuple
Out[8]: ('Mary', 'Red', 3.3)
```

Следующий фрагмент создает кортеж из одного элемента:

```
In [9]: a_singleton_tuple = ('red',)  # Обратите внимание на запятую

In [10]: a_singleton_tuple
Out[10]: ('red',)
```

Запятая `(,)` после строки `'red'` идентифицирует `a_singleton_tuple` как кортеж — круглые скобки необязательны. Если бы запятой не было, то круглые скобки стали бы избыточными, а имя `a_singleton_tuple` обозначало бы строку `'red'` вместо кортежа.

Обращение к элементам кортежа

Элементы кортежа, хотя и логически связаны друг с другом, часто относятся к разным типам. Обычно в программе вы не перебираете их, а обращаетесь к каждому элементу по отдельности. Как и индексы списков, индексы кортежей начинаются с 0. Следующий код создает кортеж `time_tuple`, представляющий часы, минуты и секунды, выводит кортеж, а затем использует его элементы для вычисления количества секунд от полуночи; обратите внимание на то, что с *разными* значениями в кортеже выполняются и разные операции:

```
In [11]: time_tuple = (9, 16, 1)

In [12]: time_tuple
Out[12]: (9, 16, 1)

In [13]: time_tuple[0] * 3600 + time_tuple[1] * 60 + time_tuple[2]
Out[13]: 33361
```

Присваивание значения элементу кортежа приводит к ошибке `TypeError`.

Добавление элементов в строку или кортеж

Как и в случае со списками, расширенное присваивание `+=` может использоваться со строками и кортежами, несмотря на их *неизменяемость*. В следующем коде после двух присваиваний `tuple1` и `tuple2` ссылаются на *один и тот же* объект кортежа:

```
In [14]: tuple1 = (10, 20, 30)

In [15]: tuple2 = tuple1

In [16]: tuple2
Out[16]: (10, 20, 30)
```

При конкатенации кортежа `(40, 50)` с `tuple1` создается *новый* кортеж, ссылка на который присваивается переменной `tuple1`, тогда как `tuple2` все еще ссылается на исходный кортеж:

```
In [17]: tuple1 += (40, 50)

In [18]: tuple1
Out[18]: (10, 20, 30, 40, 50)

In [19]: tuple2
Out[19]: (10, 20, 30)
```

Для строки или кортежа справа от `+=` должна находиться строка или кортеж, соответственно, смешение типов приводит к ошибке `TypeError`.

Присоединение кортежей к спискам

Конструкция `+=` также может использоваться для присоединения кортежа к списку:

```
In [20]: numbers = [1, 2, 3, 4, 5]

In [21]: numbers += (6, 7)

In [22]: numbers
Out[22]: [1, 2, 3, 4, 5, 6, 7]
```

Кортежи могут содержать изменяемые объекты

Создадим кортеж `student_tuple` с именем, фамилией и списком оценок:

```
In [23]: student_tuple = ('Amanda', 'Blue', [98, 75, 87])
```

И хотя сам кортеж неизменяем, его элемент-список может изменяться:

```
In [24]: student_tuple[2][1] = 85

In [25]: student_tuple
Out[25]: ('Amanda', 'Blue', [98, 85, 87])
```

В *имени* `student_tuple[2][1]` *с двойным индексированием* Python рассматривает `student_tuple[2]` как элемент кортежа, содержащий список `[98, 75, 87]`, а затем использует `[1]` для обращения к элементу списка `75`. Присваивание в фрагменте [24] заменяет это значение на `85`.

5.4. Распаковка последовательностей

В предыдущей главе был представлен механизм распаковки кортежей. Элементы любой последовательности можно распаковать, присваивая эту последовательность списку переменных, разделенных запятыми. Ошибка ValueError возникает, если число переменных слева от символа присваивания не совпадает с количеством элементов в последовательности справа:

```
In [1]: student_tuple = ('Amanda', [98, 85, 87])

In [2]: first_name, grades = student_tuple

In [3]: first_name
Out[3]: 'Amanda'

In [4]: grades
Out[4]: [98, 85, 87]
```

Следующий код распаковывает строку, список и последовательность, сгенерированную `range`:

```
In [5]: first, second = 'hi'

In [6]: print(f'{first}  {second}')
h  i

In [7]: number1, number2, number3 = [2, 3, 5]

In [8]: print(f'{number1}  {number2}  {number3}')
2  3  5

In [9]: number1, number2, number3 = range(10, 40, 10)

In [10]: print(f'{number1}  {number2}  {number3}')
10  20  30
```

Перестановка значений

Вы можете поменять местами значения двух переменных, используя механизмы упаковки и распаковки последовательностей:

```
In [11]: number1 = 99

In [12]: number2 = 22

In [13]: number1, number2 = (number2, number1)

In [14]: print(f'number1 = {number1}; number2 = {number2}')
number1 = 22; number2 = 99
```

Безопасное обращение к индексам и значениям при помощи встроенной функции enumerate

Ранее мы вызывали функцию `range` для генерирования последовательности значений индексов, а затем обращались к элементам списков в цикле `for` при помощи значений индексов и оператора индексирования (`[]`). Этот способ повышает риск ошибок, потому что `range` могут быть переданы неправильные аргументы. Если любое значение, сгенерированное `range`, окажется индексом, выходящим за пределы диапазона, то его использование в качестве индекса вызовет ошибку `IndexError`.

Для обращения к индексу *и* значению элемента рекомендуется использовать встроенную функцию `enumerate`. Эта функция получает итерируемый объект и создает итератор, который для каждого элемента возвращает кортеж с индексом и значением этого элемента. В следующем коде встроенная функция `list` используется для создания списка, содержащего результаты `enumerate`:

```
In [15]: colors = ['red', 'orange', 'yellow']

In [16]: list(enumerate(colors))
Out[16]: [(0, 'red'), (1, 'orange'), (2, 'yellow')]
```

Аналогичным образом встроенная функция `tuple` создает кортеж из последовательности:

```
In [17]: tuple(enumerate(colors))
Out[17]: ((0, 'red'), (1, 'orange'), (2, 'yellow'))
```

Следующий цикл `for` распаковывает каждый кортеж, возвращенный `enumerate`, в переменные `index` и `value`, и выводит их:

```
In [18]: for index, value in enumerate(colors):
    ...:     print(f'{index}: {value}')
    ...:
0: red
1: orange
2: yellow
```

Построение примитивной гистограммы

Следующий сценарий создает примитивную *гистограмму*, у которой строки строятся из звездочек (*) в количестве, пропорциональном значению соответствующего элемента списка. Мы используем функцию `enumerate` для безопасного обращения к индексам и значениям списка. Чтобы запустить этот пример, перейдите в папку примеров главы `ch05` и введите следующую команду:

```
ipython fig05_01.py
```

или, если вы уже запустили IPython, введите команду:

```
run fig05_01.py
```

```
1 # fig05_01.py
2 """Построение гистограммы"""
3 numbers = [19, 3, 15, 7, 11]
4
5 print('\nCreating a bar chart from numbers:')
6 print(f'Index{"Value":>8}   Bar')
7
8 for index, value in enumerate(numbers):
9     print(f'{index:>5}{value:>8}   {"*" * value}')
```

```
Creating a bar chart from numbers:
Index   Value   Bar
    0      19   *******************
    1       3   ***
    2      15   ***************
    3       7   *******
    4      11   ***********
```

Команда `for` использует `enumerate` для получения индекса и значения каждого элемента, а затем выводит отформатированную строку с индексом, значением элемента и серией звездочек соответствующей длины. Выражение

```
"*" * value
```

создает строку, состоящую из `value` звездочек. При использовании с последовательностью оператор умножения (*) *повторяет* последовательность — в данном случае строку `"*"` — `value` раз. Позднее в этой главе библиотеки с открытым кодом Seaborn и Matplotlib используются для построения гистограммы типографского качества.

5.5. Сегментация последовательностей

Вы можете создавать *сегменты* последовательностей, чтобы сформировать новые последовательности того же типа, содержащие *подмножества* исходных элементов. Операции сегментации могут модифицировать изменяемые последовательности, а те, которые *не* изменяют последовательность, работают одинаково для списков, кортежей и строк.

Определение сегмента по начальному и конечному индексу

Создадим сегмент, состоящий из элементов списка с индексами с 2 по 5:

```
In [1]: numbers = [2, 3, 5, 7, 11, 13, 17, 19]

In [2]: numbers[2:6]
Out[2]: [5, 7, 11, 13]
```

Операция сегментации *копирует* элементы от *начального индекса* слева от двоеточия (`2`) до *конечного индекса* справа от двоеточия (`6`), не включая последний. Исходный список не изменяется.

Определение сегмента только по конечному индексу

Если начальный индекс не указан, то предполагается значение 0. Таким образом, обозначение сегмента `numbers[:6]` эквивалентно записи `numbers[0:6]`:

```
In [3]: numbers[:6]
Out[3]: [2, 3, 5, 7, 11, 13]

In [4]: numbers[0:6]
Out[4]: [2, 3, 5, 7, 11, 13]
```

Определение сегмента только по начальному индексу

Если не указан конечный индекс, то Python предполагает, что он равен длине последовательности (`8` в данном случае), поэтому сегмент из фрагмента [5] содержит элементы `numbers` с индексами 6 и 7:

```
In [5]: numbers[6:]
Out[5]: [17, 19]

In [6]: numbers[6:len(numbers)]
Out[6]: [17, 19]
```

Определение сегмента без индексов

Если не указаны ни начальный, ни конечный индексы, то копируется вся последовательность:

```
In [7]: numbers[:]
Out[7]: [2, 3, 5, 7, 11, 13, 17, 19]
```

Хотя при сегментации создаются новые объекты, при этом выполняется *поверхностное копирование*, иначе говоря, операция сегментации копирует ссылки на элементы, но не на объекты, на которые они указывают. Таким образом, в приведенном фрагменте элементы нового списка указывают на *те же* объекты, что и элементы исходного списка, а не на отдельные копии. В главе 7 объясняется процесс *глубокого* копирования, при котором копируются объекты, на которые указывают ссылки, и мы объясним, почему глубокое копирование является предпочтительным.

Сегментация с шагом

В следующем коде используется *шаг* `2` для создания сегмента, содержащего каждый второй элемент `numbers`:

```
In [8]: numbers[::2]
Out[8]: [2, 5, 11, 17]
```

Начальный и конечный индекс не указаны, поэтому предполагаются значения `0` и `len(numbers)` соответственно.

Сегментация с отрицательными индексами и шагом

Отрицательное значение шага используется для сегментации в *обратном* порядке. Следующий код компактно создает новый список в обратном порядке:

```
In [9]: numbers[::-1]
Out[9]: [19, 17, 13, 11, 7, 5, 3, 2]
```

Эта запись эквивалентна следующей:

```
In [10]: numbers[-1:-9:-1]
Out[10]: [19, 17, 13, 11, 7, 5, 3, 2]
```

Изменение списков через сегменты

Вы можете изменить список, выполнив присваивание его сегменту, остальная часть списка остается неизменной. В следующем коде заменяются первые три элемента `numbers`, а остальные элементы остаются без изменений:

```
In [11]: numbers[0:3] = ['two', 'three', 'five']

In [12]: numbers
Out[12]: ['two', 'three', 'five', 7, 11, 13, 17, 19]
```

А в этом примере удаляются только первые три элемента `numbers`, для чего *пустой* список присваивается сегменту из трех элементов:

```
In [13]: numbers[0:3] = []

In [14]: numbers
Out[14]: [7, 11, 13, 17, 19]
```

В этом примере новые значения присваиваются через один элемент списка:

```
In [15]: numbers = [2, 3, 5, 7, 11, 13, 17, 19]

In [16]: numbers[::2] = [100, 100, 100, 100]

In [17]: numbers
Out[17]: [100, 3, 100, 7, 100, 13, 100, 19]

In [18]: id(numbers)
Out[18]: 4434456648
```

А здесь удаляются все элементы `numbers`, в результате чего *существующий* список остается пустым:

```
In [19]: numbers[:] = []

In [20]: numbers
Out[20]: []

In [21]: id(numbers)
Out[21]: 4434456648
```

Удаление содержимого `numbers` (фрагмент [19]) отличается от присваивания `numbers` *нового* пустого списка `[]` (фрагмент [22]). Чтобы доказать это, выведем

идентичность numbers после каждой операции. Идентичности не совпадают, а значит, они представляют разные объекты в памяти:

```
In [22]: numbers = []

In [23]: numbers
Out[23]: []

In [24]: id(numbers)
Out[24]: 4406030920
```

Когда вы присваиваете новый объект переменной (как во фрагменте [21]), исходный объект будет уничтожен уборщиком мусора, если на него не ссылаются другие объекты.

5.6. Команда del

Команда del также может использоваться для удаления элементов из списка и переменных из интерактивного сеанса. Вы можете удалить элемент с любым действительным индексом или элемент(-ы) любого действительного сегмента.

Удаление элемента списка с заданным индексом

Создадим список, а затем воспользуемся del для удаления его последнего элемента:

```
In [1]: numbers = list(range(0, 10))

In [2]: numbers
Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [3]: del numbers[-1]

In [4]: numbers
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Удаление сегмента из списка

Следующий пример удаляет из списка первые два элемента:

```
In [5]: del numbers[0:2]

In [6]: numbers
Out[6]: [2, 3, 4, 5, 6, 7, 8]
```

А здесь шаг сегментации применяется для удаления элементов через один во всем списке:

```
In [7]: del numbers[::2]

In [8]: numbers
Out[8]: [3, 5, 7]
```

Удаление сегмента, представляющего весь список

В этом примере из списка удаляются все элементы:

```
In [9]: del numbers[:]

In [10]: numbers
Out[10]: []
```

Удаление переменной из текущего сеанса

Команда `del` может удалить любую переменную. Если удалить `numbers` из интерактивного сеанса, а затем попытаться вывести значение переменной, то произойдет ошибка `NameError`:

```
In [11]: del numbers

In [12]: numbers
---------------------------------------------------------------------_-------
NameError                                 Traceback (most recent call last)
<ipython-input-12-426f8401232b> in <module>()
----> 1 numbers

NameError: name 'numbers' is not defined
```

5.7. Передача списков функциям

В предыдущей главе мы упоминали о том, что все объекты передаются по ссылке, и продемонстрировали передачу неизменяемого объекта в аргументе функции. Здесь мы продолжим обсуждение ссылок и посмотрим, что происходит, когда программа передает функции изменяемый объект списка.

Передача функции всего списка

Рассмотрим функцию `modify_elements`, которая получает ссылку на список и умножает значение каждого элемента на 2:

```
In [1]: def modify_elements(items):
   ...:     """Умножает значения всех элементов items на 2."""
   ...:     for i in range(len(items)):
   ...:         items[i] *= 2
   ...:

In [2]: numbers = [10, 3, 7, 1, 9]

In [3]: modify_elements(numbers)

In [4]: numbers
Out[4]: [20, 6, 14, 2, 18]
```

Параметр `items` функции `modify_elements` получает ссылку на *исходный* список, поэтому команда в наборе цикла изменяет каждый элемент в исходном объекте списка.

Передача кортежа функции

Если функции передается кортеж, то попытка изменить неизменяемые элементы кортежа приводит к ошибке `TypeError`:

```
In [5]: numbers_tuple = (10, 20, 30)

In [6]: numbers_tuple
Out[6]: (10, 20, 30)

In [7]: modify_elements(numbers_tuple)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-9339741cd595> in <module>()
----> 1 modify_elements(numbers_tuple)

<ipython-input-1-27acb8f8f44c> in modify_elements(items)
      2     """Умножает значения всех элементов items на 2."""
      3     for i in range(len(items)):
----> 4         items[i] *= 2
      5
      6

TypeError: 'tuple' object does not support item assignment
```

Вспомните, что кортежи могут содержать изменяемые объекты, например списки. Такие объекты могут быть изменены при передаче кортежа функции.

Примечание по поводу трассировок

В приведенной трассировке показаны *два* фрагмента, приведшие к ошибке `TypeError`. Первый — вызов функции из фрагмента [7]. Второй — определение функции из фрагмента [1]. Коду каждого фрагмента предшествуют номера строк. Мы демонстрировали в основном однострочные фрагменты. Когда в таком фрагменте происходит исключение, ему всегда предшествует префикс `----> 1`, означающий, что исключение вызвала строка 1 (единственная строка фрагмента). Для многострочных фрагментов, таких как определение `modify_elements`, приводятся последовательные номера строк начиная с 1. Запись `----> 4` в приведенном примере показывает, что исключение произошло в строке 4 функции `modify_elements`. Какой бы длинной ни была трассировка, причиной исключения стала последняя строка кода с `---->`.

5.8. Сортировка списков

Сортировка позволяет упорядочить данные по возрастанию или по убыванию.

Сортировка списка по возрастанию

Метод списков `sort` *изменяет* список и размещает элементы по возрастанию:

```
In [1]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [2]: numbers.sort()

In [3]: numbers
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Сортировка списка по убыванию

Чтобы отсортировать список по убыванию, вызовите метод списка `sort` с необязательным ключевым аргументом `reverse`, равным `True` (по умолчанию используется значение `False`):

```
In [4]: numbers.sort(reverse=True)

In [5]: numbers
Out[5]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Встроенная функция sorted

Встроенная функция `sorted` *возвращает новый список*, содержащий отсортированные элементы своей *последовательности*-аргумента — исходная последовательность при этом *не изменяется*. Следующий код демонстрирует использование функции `sorted` со списками, строками и кортежами:

```
In [6]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [7]: ascending_numbers = sorted(numbers)

In [8]: ascending_numbers
Out[8]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [9]: numbers
Out[9]: [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [10]: letters = 'fadgchjebi'

In [11]: ascending_letters = sorted(letters)

In [12]: ascending_letters
Out[12]: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

In [13]: letters
Out[13]: 'fadgchjebi'

In [14]: colors = ('red', 'orange', 'yellow', 'green', 'blue')

In [15]: ascending_colors = sorted(colors)

In [16]: ascending_colors
Out[16]: ['blue', 'green', 'orange', 'red', 'yellow']

In [17]: colors
Out[17]: ('red', 'orange', 'yellow', 'green', 'blue')
```

Необязательный ключевой аргумент `reverse` со значением `True` заставляет функцию отсортировать элементы по убыванию.

5.9. Поиск в последовательностях

Часто бывает нужно определить, содержит ли последовательность (список, кортеж или строка) значение, соответствующее заданному (*ключу поиска*). *Поиск* представляет собой процесс нахождения ключа.

Метод index

У списков имеется метод `index`, в аргументе которого передается ключ поиска. Метод начинает поиск с индекса 0 и возвращает индекс *первого* элемента, который равен ключу поиска:

```
In [1]: numbers = [3, 7, 1, 4, 2, 8, 5, 6]

In [2]: numbers.index(5)
Out[2]: 6
```

Если искомое значение отсутствует в списке, то происходит ошибка `ValueError`.

Определение начального индекса поиска

С необязательными аргументами метода `index` можно ограничить поиск подмножеством элементов списка. Оператор `*=` может использоваться для *умножения последовательности*, то есть присоединения последовательности к самой себе несколько раз. После выполнения следующего фрагмента `numbers` содержит две копии содержимого исходного списка:

```
In [3]: numbers *= 2

In [4]: numbers
Out[4]: [3, 7, 1, 4, 2, 8, 5, 6, 3, 7, 1, 4, 2, 8, 5, 6]
```

А этот код ищет в обновленном списке значение `5`, начиная с индекса `7` и до конца списка:

```
In [5]: numbers.index(5, 7)
Out[5]: 14
```

Определение начального и конечного индексов для поиска

Если определить начальный и конечный индексы, то `index` начинает поиск от начального до конечного индекса, исключая последний. Вызов `index` в фрагменте [5]:

```
numbers.index(5, 7)
```

предполагает, что третьим необязательным аргументом является длина `numbers`; этот вызов эквивалентен следующему:

```
numbers.index(5, 7, len(numbers))
```

Следующий код ищет значение 7 в диапазоне элементов с индексами от 0 до 3:

```
In [6]: numbers.index(7, 0, 4)
Out[6]: 1
```

Операторы in и not in

Оператор in проверяет, содержит ли итерируемый объект, заданный правым операндом, значение левого операнда:

```
In [7]: 1000 in numbers
Out[7]: False

In [8]: 5 in numbers
Out[8]: True
```

Аналогичным образом оператор not in проверяет, что итерируемый объект, заданный правым операндом, *не содержит* значение левого операнда:

```
In [9]: 1000 not in numbers
Out[9]: True

In [10]: 5 not in numbers
Out[10]: False
```

Использование оператора in для предотвращения ошибок ValueError

Оператор in поможет убедиться в том, что вызов метода index не приведет к ошибке ValueError из-за ключей поиска, не входящих в соответствующую последовательность:

```
In [11]: key = 1000

In [12]: if key in numbers:
    ...:     print(f'found {key} at index {numbers.index(search_key)}')
    ...: else:
    ...:     print(f'{key} not found')
    ...:
1000 not found
```

Встроенные функции any и all

Иногда требуется проверить, равен ли True *хотя бы один* элемент итерируемого объекта или равны ли True *все* элементы. Встроенная функция any воз-

вращает `True`, если хотя бы один элемент итерируемого объекта из аргумента равен `True`. Встроенная функция `all` возвращает `True`, если все элементы итерируемого объекта из аргумента равны `True`. Напомним, что ненулевые значения интерпретируются как `True`, а `0` интерпретируется как `False`. Непустые итерируемые объекты также интерпретируются как `True`, тогда как любой пустой итерируемый объект интерпретируется как `False`. Функции `any` и `all` также являются примерами внутренних итераций при программировании в функциональном стиле.

5.10. Другие методы списков

У списков также имеются методы для добавления и удаления элементов. Возьмем список `color_names`:

```
In [1]: color_names = ['orange', 'yellow', 'green']
```

Вставка элемента в заданную позицию списка

Метод `insert` вставляет новый элемент в позицию с заданным индексом. Следующий код вставляет `'red'` в позицию с индексом `0`:

```
In [2]: color_names.insert(0, 'red')
In [3]: color_names
Out[3]: ['red', 'orange', 'yellow', 'green']
```

Добавление элемента в конец списка

Метод `append` используется для добавления нового элемента в конец списка:

```
In [4]: color_names.append('blue')

In [5]: color_names
Out[5]: ['red', 'orange', 'yellow', 'green', 'blue']
```

Добавление всех элементов последовательности в конец списка

Метод `extend` списков используется для добавления всех элементов другой последовательности в конец списка:

```
In [6]: color_names.extend(['indigo', 'violet'])

In [7]: color_names
Out[7]: ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```

Результат эквивалентен использованию +=. Следующий код добавляет в список все символы строки, а затем все элементы кортежа:

```
In [8]: sample_list = []

In [9]: s = 'abc'

In [10]: sample_list.extend(s)

In [11]: sample_list
Out[11]: ['a', 'b', 'c']

In [12]: t = (1, 2, 3)

In [13]: sample_list.extend(t)

In [14]: sample_list
Out[14]: ['a', 'b', 'c', 1, 2, 3]
```

Вместо того чтобы создавать временную переменную (такую как `t`) для хранения кортежа перед присоединением его к списку, возможно, вы захотите передать кортеж методу `extend` напрямую. В этом случае круглые скобки кортежа становятся обязательными, потому что `extend` ожидает получить один аргумент с итерируемым объектом:

```
In [15]: sample_list.extend((4, 5, 6))  # Обратите внимание на вторые круглые
                                        # скобки

In [16]: sample_list
Out[16]: ['a', 'b', 'c', 1, 2, 3, 4, 5, 6]
```

При отсутствии необходимых круглых скобок происходит ошибка `TypeError`.

Удаление первого вхождения элемента в списке

Метод `remove` удаляет первый элемент с заданным значением — если аргумент `remove` отсутствует в списке, то происходит ошибка `ValueError`:

```
In [17]: color_names.remove('green')


In [18]: color_names
Out[18]: ['red', 'orange', 'yellow', 'blue', 'indigo', 'violet']
```

Очистка списка

Чтобы удалить все элементы из списка, вызовите метод `clear`:

```
In [19]: color_names.clear()

In [20]: color_names
Out[20]: []
```

Эта команда эквивалентна приведенному ранее сегментному присваиванию

```
color_names[:] = []
```

Подсчет количества вхождений элемента

Метод `count` ищет в списке заданный аргумент и возвращает количество его найденных вхождений:

```
In [21]: responses = [1, 2, 5, 4, 3, 5, 2, 1, 3, 3,
    ...:              1, 4, 3, 3, 3, 2, 3, 3, 2, 2]
    ...:

In [22]: for i in range(1, 6):
   ...:     print(f'{i} appears {responses.count(i)} times in responses')
   ...:
1 appears 3 times in responses
2 appears 5 times in responses
3 appears 8 times in responses
4 appears 2 times in responses
5 appears 2 times in responses
```

Перестановка элементов списка в обратном порядке

Метод списков `reverse` переставляет содержимое списка в обратном порядке (вместо создания копии с переставленными элементами, как это делалось ранее с сегментом):

```
In [23]: color_names = ['red', 'orange', 'yellow', 'green', 'blue']

In [24]: color_names.reverse()

In [25]: color_names
Out[25]: ['blue', 'green', 'yellow', 'orange', 'red']
```

Копирование списка

Метод списков `copy` возвращает *новый* список, содержащий *поверхностную* копию исходного списка:

```
In [26]: copied_list = color_names.copy()

In [27]: copied_list
Out[27]: ['blue', 'green', 'yellow', 'orange', 'red']
```

Эта команда эквивалентна приведенной ранее сегментной операции:

```
copied_list = color_names[:]
```

5.11. Моделирование стека на базе списка

В главе 4 был представлен стек вызовов функций. В Python нет встроенного типа стека, но стек можно рассматривать как ограниченную версию списка. Для *занесения* элементов в стек можно использовать метод `append`, который добавляет новый элемент в *конец* списка. *Извлечение* элементов может моделироваться методом списков `pop` без аргументов — этот метод удаляет и возвращает элемент, находящийся в *конце* списка.

Создадим пустой список с именем `stack`, занесем в него две строки (при помощи `append`), а затем извлечем (`pop`) строки, чтобы убедиться в том, что они извлекаются в порядке LIFO (Last In First Out, то есть «последним пришел, первым вышел»):

```
In [1]: stack = []

In [2]: stack.append('red')

In [3]: stack
Out[3]: ['red']

In [4]: stack.append('green')

In [5]: stack
Out[5]: ['red', 'green']

In [6]: stack.pop()
Out[6]: 'green'

In [7]: stack
```

```
Out[7]: ['red']

In [8]: stack.pop()
Out[8]: 'red'

In [9]: stack
Out[9]: []

In [10]: stack.pop()
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-10-50ea7ec13fbe> in <module>()
----> 1 stack.pop()

IndexError: pop from empty list
```

В каждом фрагменте с `pop` выводится значение, которое удаляет и возвращает `pop`. При попытке извлечения из пустого стека происходит ошибка `IndexError` по аналогии с обращением к несуществующему элементу списка с `[]`. Чтобы предотвратить ошибку `IndexError`, перед вызовом `pop` убедитесь в том, что `len(stack)` больше 0. Если элементы будут заноситься в стек быстрее, чем извлекаться, то это может привести к исчерпанию свободной памяти.

Список также может использоваться для моделирования другой популярной разновидности коллекций — *очереди*, у которой элементы вставляются в конце, а извлекаются сначала. Элементы извлекаются из очередей *в порядке FIFO («первым пришел, первым вышел»)*.

5.12. Трансформации списков

В этом разделе рассматриваются такие средства программирования в *функциональном стиле*, как *трансформации списков* (list comprehensions), — компактная и удобная запись для создания новых списков. Трансформации списков могут заменить многие команды `for`, в которых выполняется перебор существующих последовательностей и создание новых списков:

```
In [1]: list1 = []

In [2]: for item in range(1, 6):
   ...:     list1.append(item)
   ...:

In [3]: list1
Out[3]: [1, 2, 3, 4, 5]
```

Использование трансформации списка для создания списка целых чисел

С трансформацией списка ту же операцию можно выполнить всего в одной строке кода:

```
In [4]: list2 = [item for item in range(1, 6)]

In [5]: list2
Out[5]: [1, 2, 3, 4, 5]
```

Как и команда `for` из фрагмента [2], *секция* `for` трансформации списка

```
for item in range(1, 6)
```

перебирает последовательность, сгенерированную вызовом `range(1, 6)`. Для каждого элемента трансформация списка вычисляет выражение слева от `for` и помещает значение выражения (в данном случае сам элемент `item`) в новый список. Конкретную трансформацию из фрагмента [4] можно было бы более компактно выразить при помощи функции `list`:

```
list2 = list(range(1, 6))
```

Отображение: выполнение операций в выражениях трансформации списков

Выражение трансформации списка может выполнять разные операции (например, вычисления), *отображающие* элементы на новые значения (в том числе, возможно, и других типов). Отображение (mapping) — стандартная операция программирования в функциональном стиле, которая выдает результат с *таким же* количеством элементов, как в исходных отображаемых данных. Следующая трансформация отображает каждое значение на его куб, для чего используется выражение `item ** 3`:

```
In [6]: list3 = [item ** 3 for item in range(1, 6)]

In [7]: list3
Out[7]: [1, 8, 27, 64, 125]
```

Фильтрация: трансформации списков с if

Другая распространенная операция программирования в функциональном стиле — *фильтрация* элементов и отбор только тех элементов, которые удовлетворяют заданному условию. Как правило, при этом строится список с *мень-*

шим количеством элементов, чем в фильтруемых данных. Чтобы выполнить эту операцию с использованием трансформации списка, используйте *секцию* `if`. Следующий пример включает в `list4` только четные значения, сгенерированные в секции `for`:

```
In [8]: list4 = [item for item in range(1, 11) if item % 2 == 0]

In [9]: list4
Out[9]: [2, 4, 6, 8, 10]
```

Трансформация списка, которая обрабатывает элементы другого списка

Секция `for` может обрабатывать любые итерируемые объекты. Создадим список строк в нижнем регистре и используем трансформацию списка для создания нового списка, содержащего их версии в верхнем регистре:

```
In [10]: colors = ['red', 'orange', 'yellow', 'green', 'blue']

In [11]: colors2 = [item.upper() for item in colors]

In [12]: colors2
Out[12]: ['RED', 'ORANGE', 'YELLOW', 'GREEN', 'BLUE']

In [13]: colors
Out[13]: ['red', 'orange', 'yellow', 'green', 'blue']
```

5.13. Выражения-генераторы

Выражение-генератор отчасти напоминает трансформацию списка, но оно создает итерируемый *объект-генератор*, производящий значения *по требованию*. Этот механизм называется *отложенным вычислением*. В трансформациях списков используется *быстрое вычисление*, позволяющее создавать списки в момент выполнения. При большом количестве элементов создание списка может потребовать значительных затрат памяти и времени. Таким образом, выражения-генераторы могут сократить потребление памяти программой и повысить быстродействие, если не все содержимое списка понадобится одновременно.

Выражения-генераторы обладают теми же возможностями, что и трансформации списков, но они определяются в круглых скобках вместо квадратных. Выражение-генератор в фрагменте [2] возводит в квадрат и возвращает только нечетные числа из `numbers`:

```
In [1]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [2]: for value in (x ** 2 for x in numbers if x % 2 != 0):
   ...:     print(value, end='  ')
   ...:
9  49  1  81  25
```

Чтобы показать, что выражение-генератор не создает список, присвоим выражение-генератор из предыдущего фрагмента переменной и выведем значение этой переменной:

```
In [3]: squares_of_odds = (x ** 2 for x in numbers if x % 2 != 0)

In [3]: squares_of_odds
Out[3]: <generator object <genexpr> at 0x1085e84c0>
```

Текст "`generator object <genexpr>`" сообщает, что `square_of_odds` является объектом-генератором, который был создан на базе выражения-генератора (`genexpr`).

5.14. Фильтрация, отображение и свертка

В предыдущем разделе были представлены некоторые средства программирования в функциональном стиле — трансформации списков, фильтрация и отображение. В этом разделе мы продемонстрируем применение встроенных функций `filter` и `map` для фильтрации и отображения соответственно. Мы продолжим обсуждение операции свертки, которая преобразует коллекцию элементов в *одно* значение (примерами служат операции получения количества элементов, суммирования, произведения, усреднения, минимума и максимума).

Фильтрация значений последовательности встроенной функцией filter

Воспользуемся встроенной функцией `filter` для получения нечетных значений из `numbers`:

```
In [1]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [2]: def is_odd(x):
   ...:     """Возвращает True только для нечетных x."""
   ...:     return x % 2 != 0
   ...:
```

```
In [3]: list(filter(is_odd, numbers))
Out[3]: [3, 7, 1, 9, 5]
```

Функции Python, как и данные, представляют собой объекты, которые можно присваивать переменным, передавать другим функциям и возвращать из функций. Функции, получающие другие функции в аргументах, относятся к инструментарию функционального программирования и называются *функциями высшего порядка*. Например, первым аргументом `filter` должна быть функция, которая получает один аргумент и возвращает `True`, если значение должно включаться в результат. Функция `is_odd` возвращает `True`, если ее аргумент является нечетным. Функция `filter` вызывает `is_odd` по одному разу для каждого значения в итерируемом объекте из второго аргумента (`numbers`). Функции высшего порядка также могут возвращать функцию как результат.

Функция `filter` возвращает итератор, так что для получения результатов `filter` нужно будет выполнить их перебор. Это еще один пример отложенного вычисления. Во фрагменте [3] функция `list` перебирает результаты и создает список, в котором они содержатся. Те же результаты можно получить с использованием трансформации списка с секцией `if`:

```
In [4]: [item for item in numbers if is_odd(item)]
Out[4]: [3, 7, 1, 9, 5]
```

Использование лямбда-выражения вместо функции

Для простых функций (таких как `is_odd`), возвращающих только *значение одного выражения*, можно использовать *лямбда-выражение* для определения функции во «встроенном» виде в том месте, где она нужна, — обычно при передаче другой функции:

```
In [5]: list(filter(lambda x: x % 2 != 0, numbers))
Out[5]: [3, 7, 1, 9, 5]
```

Возвращаемое значение `filter` (итератор) передается функции `list` для преобразования результатов в список и их вывода.

Лямбда-выражение является *анонимной функцией*, то есть функцией, не имеющей имени. В вызове `filter`

```
filter(lambda x: x % 2 != 0, numbers)
```

первым аргументом является лямбда-выражение

```
lambda x: x % 2 != 0
```

Лямбда-выражение начинается с ключевого слова `lambda`, за которым следует разделенный запятыми список параметров, двоеточие (`:`) и выражение. В данном случае список параметров состоит из одного параметра с именем `x`. Лямбда-выражение *неявно* возвращает значение своего выражения. Таким образом, любая простая функция в форме

```
def имя_функции(список_параметров):
    return выражение
```

может быть выражена в более компактной форме посредством лямбда-выражения

```
lambda список_параметров: выражение
```

Отображение значений последовательности на новые значения

Воспользуемся встроенной функцией `map` с лямбда-выражением для возведения в квадрат каждого значения из `numbers`:

```
In [6]: numbers
Out[6]: [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [7]: list(map(lambda x: x ** 2, numbers))
Out[7]: [100, 9, 49, 1, 81, 16, 4, 64, 25, 36]
```

Первым аргументом функции `map` является функция, которая получает одно значение и возвращает новое значение — в данном случае лямбда-выражение, которое возводит свой аргумент в квадрат. Вторым аргументом является итерируемый объект с отображаемыми значениями. Функция `map` использует отложенное вычисление, поэтому возвращаемый `map` итератор передается функции `list`. Это позволит перебрать и создать список отображенных значений. Эквивалентная трансформация списка выглядит так:

```
In [8]: [item ** 2 for item in numbers]
Out[8]: [100, 9, 49, 1, 81, 16, 4, 64, 25, 36]
```

Объединение filter и map

Предшествующие операции `filter` и `map` можно объединить следующим образом:

```
In [9]: list(map(lambda x: x ** 2,
    ...:          filter(lambda x: x % 2 != 0, numbers)))
    ...:
Out[9]: [9, 49, 1, 81, 25]
```

Во фрагменте [9] происходит достаточно много всего, поэтому к нему стоит присмотреться повнимательнее. Сначала `filter` возвращает итерируемый объект, представляющий только нечетные значения из `numbers`. Затем `map` возвращает итерируемый объект, представляющий квадраты отфильтрованных значений. Наконец, `list` использует итерируемый объект, возвращенный `map`, для создания списка. Возможно, вы предпочтете использовать следующую трансформацию списка вместо предыдущего фрагмента:

```
In [10]: [x ** 2 for x in numbers if x % 2 != 0]
Out[10]: [9, 49, 1, 81, 25]
```

Для каждого значения `x` из `numbers` выражение `x**2` выполняется только в том случае, если условие `x%2!=0` дает `истинный результат`.

Свертка: суммирование элементов последовательности функцией sum

Как вам известно, свертки обрабатывают элементы последовательности с вычислением одного значения. В частности, свертки выполняются встроенными функциями `len`, `sum`, `min` и `max`. Также можно создавать собственные свертки при помощи функции `reduce` модуля `functools`. За примером кода обращайтесь по адресу https://docs.python.org/3/library/functools.html. Когда мы займемся изучением больших данных и Hadoop в главе 16, то продемонстрируем и программирование MapReduce, основанное на операциях фильтрации, отображения и свертки из программирования в функциональном стиле.

5.15. Другие функции обработки последовательностей

Python предоставляет ряд других встроенных функций для работы с последовательностями.

Нахождение минимального и максимального значения функцией key

Ранее были продемонстрированы встроенные функции свертки `min` и `max` с аргументами (такими как `int` или списки `int`). Иногда требуется найти минимум или максимум в наборе более сложных объектов, например строк. Возьмем следующее сравнение:

```
In [1]: 'Red' < 'orange'
Out[1]: True
```

Буква `'R'` следует «после» буквы `'o'` в алфавите, поэтому можно было бы ожидать, что строка `'Red'` окажется меньше `'orange'`, а условие вернет `False`. Однако строки сравниваются по *числовым кодам* входящих в них символов, а буквы в нижнем регистре имеют *более высокие* значения кодов, чем буквы верхнего регистра. В этом нетрудно убедиться при помощи встроенной функции `ord`, возвращающей числовое значение символа:

```
In [2]: ord('R')
Out[2]: 82

In [3]: ord('o')
Out[3]: 111
```

Возьмем список `colors`, содержащий строки с символами верхнего и нижнего регистра:

```
In [4]: colors = ['Red', 'orange', 'Yellow', 'green', 'Blue']
```

Допустим, вы хотите определить наименьшую и наибольшую строку в *алфавитном* порядке, а не в *числовом* (лексикографическом). При расположении цветов в алфавитном порядке:

```
'Blue', 'green', 'orange', 'Red', 'Yellow'
```

строка `'Blue'` является наименьшей (то есть ближайшей к началу алфавита), а строка `'Yellow'` — наибольшей (то есть ближайшей к концу алфавита).

Так как Python сравнивает строки по числовым критериям, сначала необходимо преобразовать каждую строку к нижнему (или верхнему) регистру. Тогда числовые и значения символов будут соответствовать их *алфавитному* порядку. В следующих фрагментах для определения наименьшей и наибольшей строки в алфавитном порядке используются функции `min` и `max`:

```
In [5]: min(colors, key=lambda s: s.lower())
Out[5]: 'Blue'

In [6]: max(colors, key=lambda s: s.lower())
Out[6]: 'Yellow'
```

Ключевой аргумент `key` должен содержать функцию с одним параметром, который возвращает значение. В данном случае это лямбда-выражение, вы-

зывающее метод строк `lower` для получения версии строки в нижнем регистре. Функции `min` и `max` вызывают функцию из аргумента `key` для каждого элемента и используют результаты для сравнения элементов.

Обратный перебор элементов последовательности

Встроенная функция `reversed` возвращает итератор, позволяющий перебрать значения последовательности в обратном порядке. Следующая трансформация списка создает новый список, содержащий квадраты значений `numbers` в обратном порядке:

```
In [7]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

In [7]: reversed_numbers = [item for item in reversed(numbers)]

In [8]: reversed_numbers
Out[8]: [36, 25, 64, 4, 16, 81, 1, 49, 9, 100]
```

Объединение итерируемых объектов в кортежи с соответствующими элементами

Встроенная функция `zip` перебирает *несколько* итерируемых объектов данных *одновременно*. Функция получает в аргументах произвольное количество итерируемых объектов и возвращает итератор, который строит кортежи, содержащий элементы с одинаковыми индексами из каждого итерируемого объекта. Например, вызов `zip` из фрагмента [11] генерирует кортежи `('Bob', 3.5)`, `('Sue', 4.0)` и `('Amanda', 3.75)`, содержащие элементы с индексами 0, 1 и 2 каждого списка соответственно:

```
In [9]: names = ['Bob', 'Sue', 'Amanda']

In [10]: grade_point_averages = [3.5, 4.0, 3.75]

In [11]: for name, gpa in zip(names, grade_point_averages):
    ...:     print(f'Name={name}; GPA={gpa}')
    ...:
Name=Bob; GPA=3.5
Name=Sue; GPA=4.0
Name=Amanda; GPA=3.75
```

Каждый кортеж распаковывается на значения имени и среднего балла, которые затем выводятся. Количество сгенерированных кортежей определяется самым коротким аргументом функции `zip`. В данном случае оба аргумента имеют одинаковую длину.

5.16. Двумерные списки

Элементами списков также могут быть другие списки. Типичный пример использования таких вложенных (или многомерных) списков — представление *таблиц* с информацией, упорядоченной по *строкам* и *столбцам*. Чтобы задать конкретный элемент таблицы, необходимо указать *два* индекса — по общепринятой схеме первый определяет строку элемента, а второй определяет столбец.

Списки, требующие индексов для идентификации элемента, называются *двумерными списками* и рассматриваются в текущем разделе. Многомерные списки могут иметь более двух индексов.

Создание двумерного списка

Рассмотрим двумерный список из трех строк и четырех столбцов (то есть список 3×4), который может представлять оценки трех студентов, каждый из которых сдал четыре экзамена:

```
In [1]: a = [[77, 68, 86, 73], [96, 87, 89, 81], [70, 90, 86, 81]]
```

Если записать список в нижеследующем виде, то его структура с делением на строки и столбцы станет более понятной:

```
a = [[77, 68, 86, 73],  # Оценки первого студента
     [96, 87, 89, 81],  # Оценки второго студента
     [70, 90, 86, 81]]  # Оценки третьего студента
```

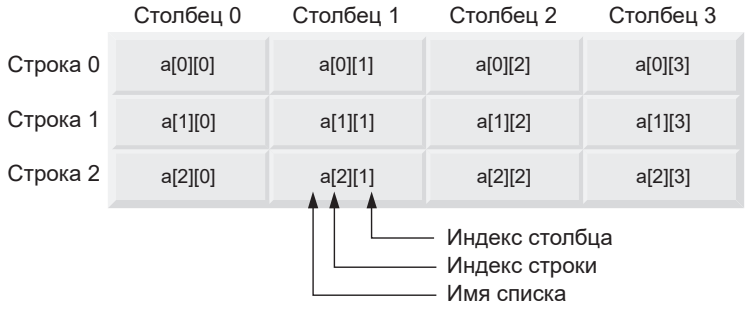
Работа с двумерным списком

На следующей диаграмме изображен список a, в строках и столбцах которого располагаются оценки на экзаменах:

	Столбец 0	Столбец 1	Столбец 2	Столбец 3
Строка 0	77	68	86	73
Строка 1	96	87	89	81
Строка 2	70	90	86	81

Идентификация элементов в двумерном списке

На следующей диаграмме представлены имена элементов списка a:



Каждый элемент идентифицируется именем вида `a[i][j]`; здесь `a` — имя списка, а `i` и `j` — индексы, однозначно определяющие строку и столбец каждого элемента соответственно. У всех элементов строки 0 первый индекс равен 0. У всех элементов столбца 3 второй индекс равен 3.

В двумерном списке `a`:

- значениями `77`, `68`, `86` и `73` инициализируются элементы `a[0][0]`, `a[0][1]`, `a[0][2]` и `a[0][3]` соответственно;
- значениями `96`, `87`, `89` и `81` инициализируются элементы `a[1][0]`, `a[1][1]`, `a[1][2]` и `a[1][3]` соответственно, а также
- значениями `70`, `90`, `86` и `81` инициализируются элементы `a[2][0]`, `a[2][1]`, `a[2][2]` и `a[2][3]` соответственно.

Список из m строк и n столбцов называется списком $m \times n$ и содержит $m \times n$ элементов. Следующая вложенная команда `for` выводит элементы предыдущего двумерного списка по строкам:

```
In [2]: for row in a:
   ...:     for item in row:
   ...:         print(item, end=' ')
   ...:     print()
   ...:
77 68 86 73
96 87 89 81
70 90 86 81
```

Как выполняются вложенные циклы

Изменим вложенный цикл так, чтобы он выводил имя списка, индексы строки и столбца, а также значение каждого элемента:

```
In [3]: for i, row in enumerate(a):
   ...:     for j, item in enumerate(row):
   ...:         print(f'a[{i}][{j}]={item} ', end=' ')
   ...:     print()
   ...:
a[0][0]=77  a[0][1]=68  a[0][2]=86  a[0][3]=73
a[1][0]=96  a[1][1]=87  a[1][2]=89  a[1][3]=81
a[2][0]=70  a[2][1]=90  a[2][2]=86  a[2][3]=81
```

Внешняя команда `for` последовательно перебирает строки двумерного списка. При каждой итерации внешнего цикла `for` внутренний цикл `for` перебирает *все* столбцы текущей строки. Таким образом, для первой итерации внешнего цикла `строка` 0 имеет вид

```
[77, 68, 86, 73]
```

а вложенный цикл перебирает четыре элемента этого списка `a[0][0]=77`, `a[0][1]=68`, `a[0][2]=86` и `a[0][3]=73`.

При второй итерации внешнего цикла `строка` 1 состоит из элементов

```
[96, 87, 89, 81]
```

а вложенный цикл перебирает четыре элемента `a[1][0]=96`, `a[1][1]=87`, `a[1][2]=89` и `a[1][3]=81`.

При третьей итерации внешнего цикла `строка` 2 состоит из элементов

```
[70, 90, 86, 81]
```

а вложенный цикл перебирает четыре элемента `a[2][0]=70`, `a[2][1]=90`, `a[2][2]=86` и `a[2][3]=81`.

В главе 7 рассматривается коллекция `ndarray` из библиотеки NumPy и коллекция `DataFrame` из библиотеки Pandas. Они позволяют работать с многомерными коллекциями более компактно и удобно по сравнению с операциями с двумерным списком, продемонстрированными в этом разделе.

5.17. Введение в data science: моделирование и статические визуализации

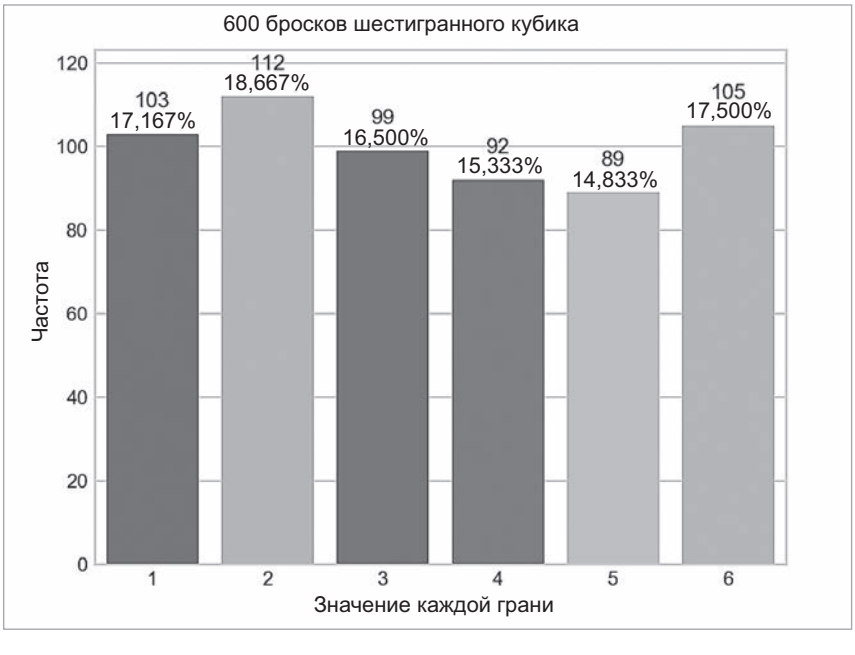
В разделах «Введение в data science» последних глав рассматривались основные характеристики описательной статистики. В этом разделе мы сосредото-

чимся на визуализациях, помогающих «разобраться в сути» данных. Визуализации предоставляют мощный механизм понимания данных, выходящий за границы простого просмотра данных.

Мы используем две библиотеки визуализации с открытым кодом — Seaborn и Matplotlib — для построения *статических* гистограмм, отображающих результаты моделирования бросков шестигранных кубиков. *Библиотека визуализации Seaborn* построена на основе *библиотеки визуализации Matplotlib* и упрощает многие операции Matplotlib. Мы будем пользоваться аспектами обеих библиотек, потому что некоторые операции Seaborn возвращают объекты из библиотеки Matplotlib. В разделе «Введение в data science» следующей главы мы «оживим» гистограмму при помощи *динамических визуализаций*.

5.17.1. Примеры диаграмм для 600, 60 000 и 6 000 000 бросков

На следующем снимке изображена вертикальная гистограмма, которая для 600 бросков отображает сводную информацию о частотах выпадения каждой из шести граней и их процента от общего количества бросков.



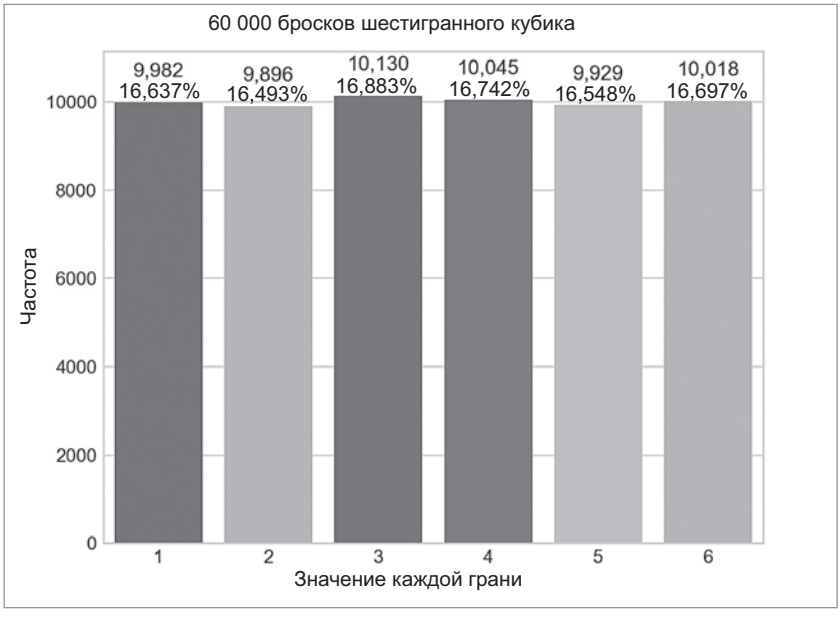
Каждая грань должна выпасть приблизительно 100 раз. Тем не менее при таком малом количестве бросков ни одна из частот не равна 100 (хотя некоторые достаточно близки), а большинство процентов отклоняется от 16,6667% (около 1/6). Если же запустить моделирование для 60 000 бросков, то размеры столбцов гистограммы будут намного более равномерными. При 6 000 000 бросков размеры столбцов будут практически одинаковыми. В этом проявляется закон больших чисел. В следующей главе будет показано, как построить гистограмму с динамическим изменением размеров столбцов.

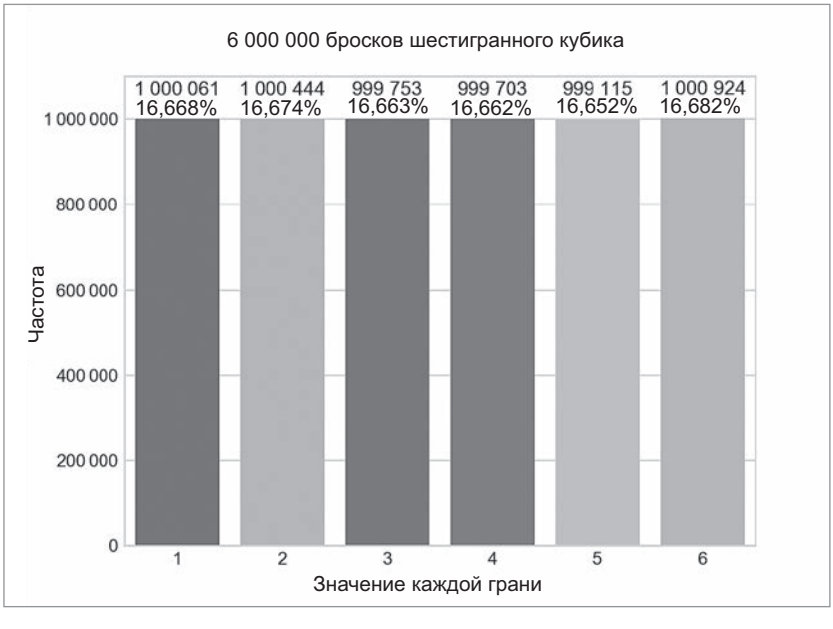
Вы узнаете, как управлять внешним видом и содержанием диаграммы, в частности:

- заголовком диаграммы в окне (**`Rolling a Six-Sided Die 600 Times`**);
- пояснительными метками: **`Die Value`** для оси *x* и **`Frequency`** для оси *y*;
- текстом, отображаемым над каждым столбцом, с *частотами* и *процентом* от общего количества бросков;
- цветом столбцов.

Мы воспользуемся некоторыми настройками по умолчанию Seaborn. Например, Seaborn определяет текстовые метки по оси *x* для результатов бросков 1–6, а текстовые метки по оси *y* — для фактических частот выпадения. Во внутренней реализации Matplotlib определяет позиции и размеры столбцов на основании размера окна и диапазона значений, представленных столбцами. Библиотека, кроме того, расставляет числовые метки оси **`Frequency`** на основании фактических частот результатов, представленных столбцами. Существует и много других параметров, которые можно настраивать по своему вкусу.

На первом скриншоте показаны результаты для 60 000 бросков — только представьте, что вам пришлось бы делать это вручную. В этом случае мы ожидаем, что каждая грань выпадет около 10 000 раз. На втором скриншоте показаны результаты для 6 000 000 бросков — безусловно, вручную вы бы с этим просто не справились! На этот раз каждая грань должна выпасть около 1 000 000 раз, и столбцы частот имеют почти одинаковую длину (они близки, но точного совпадения все же нет). При большем количестве бросков проценты намного ближе к ожидаемым 16,667%.


60 000 бросков шестигранного кубика
9,982
16,637%
9,896
16,493%
10,130
16,883%
10,045
16,742%
9,929
16,548%
10,018
16,697%
10000
8000
6000
4000
2000
0
Частота
1
2
3
4
5
6
Значение каждой грани


6 000 000 бросков шестигранного кубика
1 000 061
16,668%
1 000 444
16,674%
999 753
16,663%
999 703
16,662%
999 115
16,652%
1 000 924
16,682%
1 000 000
800 000
600 000
400 000
200 000
0
Частота
1
2
3
4
5
6
Значение каждой грани

5.17.2. Визуализация частот и процентов

Займемся теперь интерактивной разработкой гистограмм.

Запуск IPython для интерактивной разработки Matplotlib

В IPython предусмотрена встроенная поддержка интерактивной разработки диаграмм Matplotlib, которая также понадобится для разработки диаграмм Seaborn. Просто запустите IPython следующей командой:

```
ipython --matplotlib
```

Импортирование библиотек

Начнем с импортирования библиотек, которые будут использоваться для работы с диаграммами:

```
In [1]: import matplotlib.pyplot as plt

In [2]: import numpy as np

In [3]: import random

In [4]: import seaborn as sns
```

1. Модуль `matplotlib.pyplot` содержит графическую поддержку библиотеки Matplotlib, которой мы будем пользоваться. Обычно этот модуль импортируется под именем `plt`.
2. Библиотека NumPy (Numerical Python) включает функцию `unique`, которая понадобится для обобщения результатов бросков. *Модуль* `numpy` обычно импортируется под именем `np`.
3. Модуль `random` содержит функции генерирования случайных чисел Python.
4. *Модуль* `seaborn` содержит графические средства библиотеки Seaborn. Этот модуль обычно импортируется под именем `sns` (если вам интересно, поищите информацию о том, почему было выбрано именно это сокращение).

Моделирование бросков и вычисление частот

Воспользуемся *трансформацией списка* для создания списка из 600 случайных результатов, а затем при помощи функции `unique` библиотеки NumPy опре-

делим уникальные значения (скорее всего, это будут все шесть возможных результатов) и их частоты:

```
In [5]: rolls = [random.randrange(1, 7) for i in range(600)]

In [6]: values, frequencies = np.unique(rolls, return_counts=True)
```

Библиотека NumPy предоставляет высокопроизводительную коллекцию `ndarray`, которая обычно работает намного быстрее списков[1]. Хотя `ndarray` не используется здесь напрямую, функция `unique` из NumPy ожидает получить аргумент `ndarray` и возвращает `ndarray`. Если передать ей список (например, `rolls`), NumPy преобразует его в `ndarray` для повышения быстродействия. Коллекция `ndarray`, возвращаемая `unique`, просто присваивается переменной, которая должна использоваться функцией построения диаграмм Seaborn.

Ключевой аргумент `return_counts=True` приказывает `unique` подсчитать количество вхождений каждого уникального значения. В данном случае `unique` возвращает кортеж из двух одномерных коллекций `ndarray`, содержащих отсортированные уникальные значения и их частоты соответственно. Коллекции `ndarray` из кортежа распаковываются в переменные `values` и `frequencies`. Если аргумент `return_counts` равен `False`, то возвращается только список уникальных значений.

Построение исходной гистограммы

Создадим заголовок гистограммы, назначим ей стиль, а затем нанесем на гистограмму результаты бросков и частоты:

```
In [7]: title = f'Rolling a Six-Sided Die {len(rolls):,} Times'

In [8]: sns.set_style('whitegrid')

In [9]: axes = sns.barplot(x=values, y=frequencies, palette='bright')
```

Форматная строка из фрагмента [7] включает количество бросков кубика в заголовок гистограммы. Спецификатор «`,`» (запятая) в

```
{len(rolls):,}
```

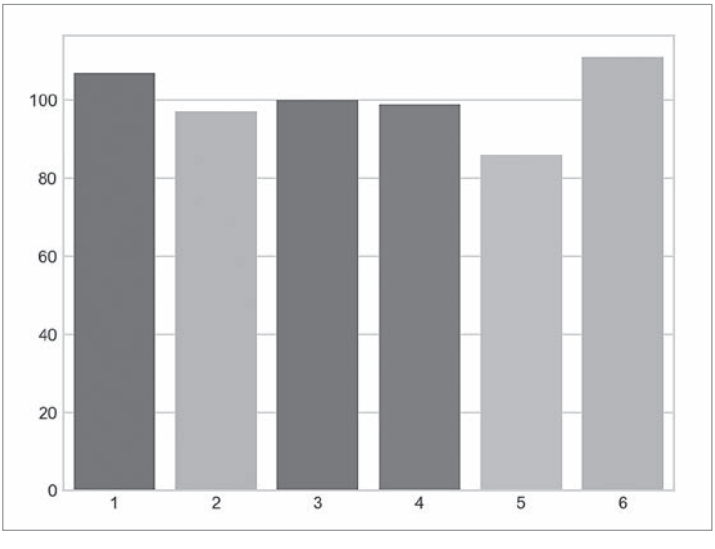
выводит число с *разделителями групп разрядов* — таким образом, значение `60000` будет выведено в виде `60,000`.

---

[1] Мы проведем сравнительный анализ быстродействия в главе 7, когда будем подробно рассматривать `ndarray`.

По умолчанию Seaborn выводит диаграммы на простом белом фоне, но библиотека предоставляет вам на выбор несколько стилей оформления (`'darkgrid'`, `'whitegrid'`, `'dark'`, `'white'` и `'ticks'`). Фрагмент [8] задает стиль `'whitegrid'`, который выводит светло-серые горизонтальные линии на вертикальной гистограмме. Линии помогают понять соответствие между высотой каждого столбца и числовыми метками частот в левой части диаграммы.

Фрагмент [9] строит диаграмму частот при помощи *функции* Seaborn `barplot`. Когда вы выполняете этот фрагмент, на экране появляется следующее окно (потому что вы запустили IPython с параметром `--matplotlib`):



Seaborn взаимодействует с Matplotlib для вывода столбцов гистограммы; для этого библиотека создает объект Matplotlib `Axes`, управляющий контентом, находящимся в окне. Во внутренней реализации Seaborn использует объект Matplotlib `Figure` для управления окном, в котором отображается `Axes`. Первые два аргумента функции `barplot` содержат коллекции `ndarray`, содержащие значения осей *x* и *y* соответственно. Необязательный ключевой аргумент `palette` используется для выбора заранее определенной цветовой палитры Seaborn `'bright'`. За информацией о настройке палитры обращайтесь по адресу:

https://seaborn.pydata.org/tutorial/color_palettes.html

Функция `barplot` возвращает настроенный ей объект `Axes`. Он присваивается переменной `axes`, чтобы его можно было использовать для настройки других

аспектов итоговой диаграммы. Все изменения, внесенные в гистограмму после этой точки, *немедленно* отображаются при выполнении соответствующего фрагмента.

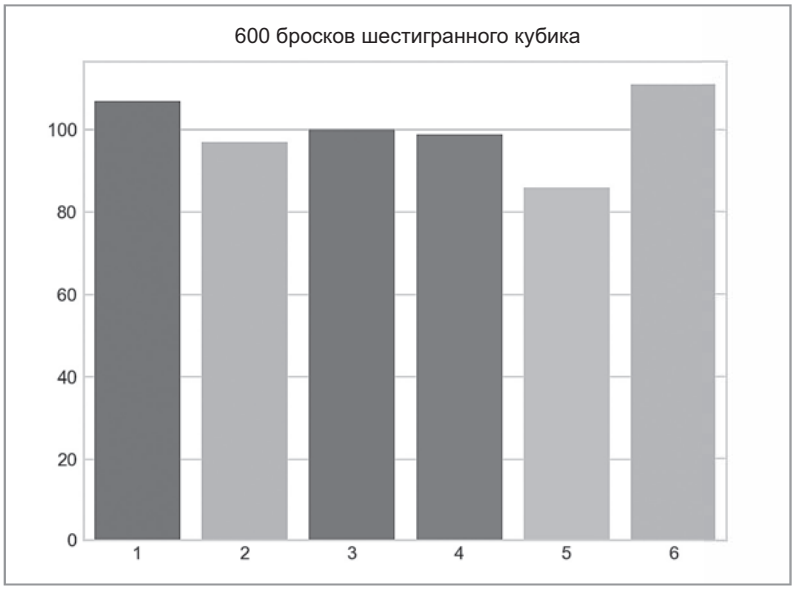
Назначение заголовка окна и пометка осей *x* и *y*

Следующие два фрагмента добавляют описательный текст на гистограмму:

```
In [10]: axes.set_title(title)
Out[10]: Text(0.5,1,'Rolling a Six-Sided Die 600 Times')

In [11]: axes.set(xlabel='Die Value', ylabel='Frequency')
Out[11]: [Text(92.6667,0.5,'Frequency'), Text(0.5,58.7667,'Die Value')]
```

Фрагмент [10] использует метод `set_title` объекта `axes` для вывода строки `title`, выровненной по центру гистограммы. Метод возвращает объект `Text` с заголовком и его *позицией* в окне, который IPython просто отображает для подтверждения. Вы можете игнорировать строки `Out[]` в приведенных выше фрагментах.

Фрагмент [11] добавляет метки на оси. Метод `set` получает ключевые аргументы для устанавливаемых свойств объекта `Axes`. Метод выводит текст `xlabel` вдоль оси *x*, текст `ylabel` — вдоль оси *y* и возвращает список объектов `Text`, содержащий метки и их позиции. Гистограмма теперь выглядит так:



Завершение гистограммы

Следующие два фрагмента завершают гистограмму — они выделяют место для текста над каждым столбцом, а затем выводят его:

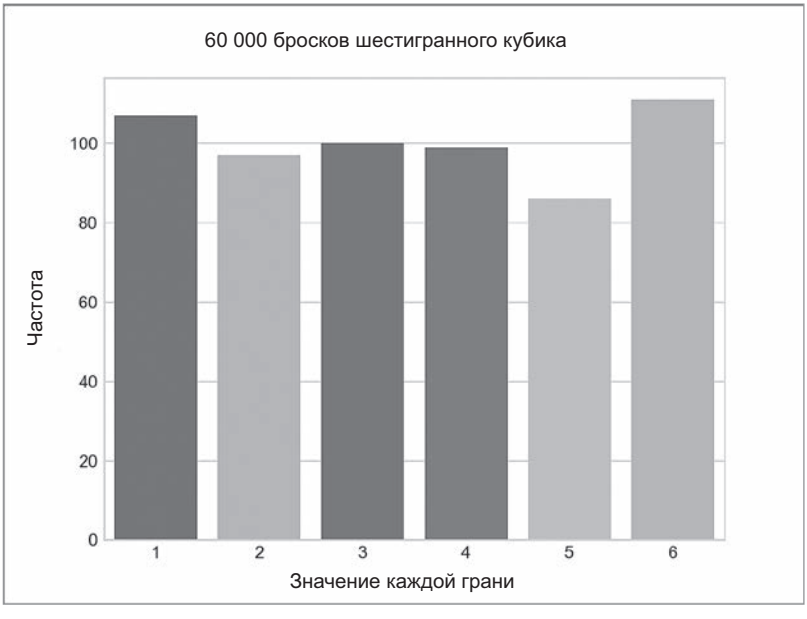
```
In [12]: axes.set_ylim(top=max(frequencies) * 1.10)
Out[12]: (0.0, 122.10000000000001)

In [13]: for bar, frequency in zip(axes.patches, frequencies):
    ...:     text_x = bar.get_x() + bar.get_width() / 2.0
    ...:     text_y = bar.get_height()
    ...:     text = f'{frequency:,}\n{frequency / len(rolls):.3%}'
    ...:     axes.text(text_x, text_y, text,
    ...:               fontsize=11, ha='center', va='bottom')
    ...:
```

Чтобы выделить место для текста над столбцами, фрагмент [12] масштабирует ось *y* на 10%. Это значение было выбрано на основе экспериментов. Метод `set_ylim` объекта `Axes` поддерживает много необязательных ключевых аргументов. В нашем примере используется только аргумент `top` для изменения максимального значения, представляемого осью *y*. Наибольшая частота умножается на `1.10`, чтобы гарантировать, что ось *y* на 10% выше самого высокого столбца.

Наконец, фрагмент [13] выводит значение частоты каждого столбца и процент от общего количества бросков. Коллекция `patches` объекта `axes` содержит двумерные цветные фигуры, представляющие столбцы гистограммы. Команда `for` использует функцию `zip` для перебора элементов `patches` и соответствующих им значений `frequency`. Каждая итерация распаковывает в `bar` и `frequency` один из кортежей, возвращенных `zip`. Набор команды `for` работает следующим образом:

- Первая команда вычисляет координату *x* центра области для вывода текста. Она вычисляется как сумма координаты *x* левого края столбца (`bar.get_x()`) и половины ширины столбца (`bar.get_width() / 2.0`).
- Вторая команда получает координату *y* области для вывода текста — значение `bar.get_y()` представляет верх столбца.
- Третья команда создает двустрочную строку, содержащую частоту этого столбца и соответствующий процент от общего количества бросков.
- Последняя команда вызывает метод `text` объекта `Axes` для вывода текста над столбцом. Первые два аргумента метода задают позицию текста *x*-*y*, а третий аргумент — выводимый текст. Ключевой аргумент `ha` задает *горизонтальное выравнивание* — мы выполнили горизонтальное выравнива-

ние текста по центру относительно координаты *x*. Ключевой аргумент `va` задает *вертикальное выравнивание* — нижний край текста выравнивается по координате *y*. Итоговый вид гистограммы показан на следующем рисунке.



Повторный бросок и обновление гистограммы — магические команды IPython

Итак, гистограмма готова, и скорее всего, вы захотите поэкспериментировать с другим количеством бросков. Для начала очистите существующий график вызовом функции `cla` библиотеки Matplotlib:

```
In [14]: plt.cla()
```

IPython предоставляет специальные команды, называемые *магическими командами* (magics), для удобного выполнения различных операций. Воспользуйтесь *магической командой* `%recall` для получения фрагмента [5], который создал список `rolls`, и включите код в следующее приглашение `In []`:

```
In [15]: %recall 5

In [16]: rolls = [random.randrange(1, 7) for i in range(600)]
```

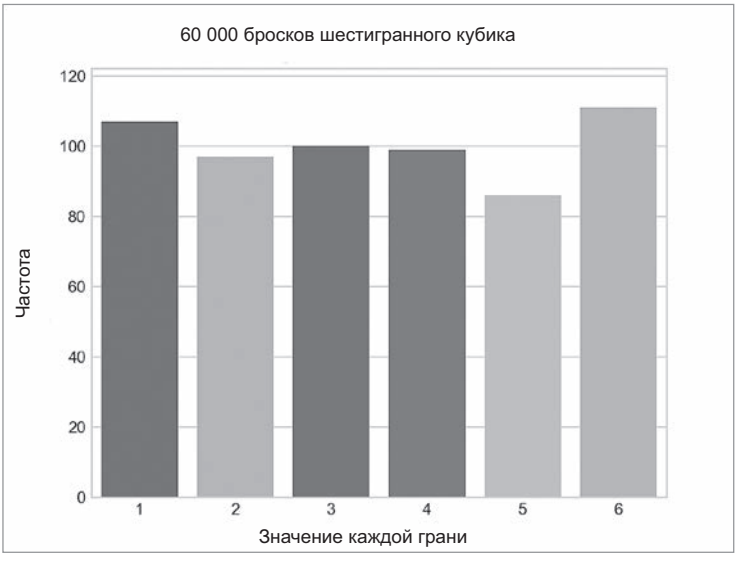
Отредактируйте фрагмент и измените количество бросков на `60000`, а затем нажмите Enter для создания нового списка:

```
In [16]: rolls = [random.randrange(1, 7) for i in range(60000)]
```

Теперь загрузите фрагменты с [6] по [13]. При этом все фрагменты выводятся в заданном диапазоне в следующем приглашении `In []`. Нажмите Enter, чтобы снова выполнить эти фрагменты:

```
In [17]: %recall 6-13

In [18]: values, frequencies = np.unique(rolls, return_counts=True)
    ...: title = f'Rolling a Six-Sided Die {len(rolls):,} Times'
    ...: sns.set_style('whitegrid')
    ...: axes = sns.barplot(x=values, y=frequencies, palette='bright')
    ...: axes.set_title(title)
    ...: axes.set(xlabel='Die Value', ylabel='Frequency')
    ...: axes.set_ylim(top=max(frequencies) * 1.10)
    ...: for bar, frequency in zip(axes.patches, frequencies):
    ...:     text_x = bar.get_x() + bar.get_width() / 2.0
    ...:     text_y = bar.get_height()
    ...:     text = f'{frequency:,}\n{frequency / len(rolls):.3%}'
    ...:     axes.text(text_x, text_y, text,
    ...:               fontsize=11, ha='center', va='bottom')
    ...:
```

Обновленная гистограмма показана ниже:



Сохранение фрагментов в файле при помощи магической команды %save

После того как диаграмма будет построена в интерактивном режиме, возможно, вы захотите сохранить код в файле, чтобы превратить его в сценарий и запустить его в будущем. Воспользуемся *магической командой* `%save` для сохранения фрагментов с 1-го по 13-й в файле с именем RollDie.py. IPython обозначает файл, в который были записаны данные, а затем выводит сохраненные строки:

```
In [19]: %save RollDie.py 1-13
The following commands were written to file `RollDie.py`:
import matplotlib.pyplot as plt
import numpy as np
import random
import seaborn as sns
rolls = [random.randrange(1, 7) for i in range(600)]
values, frequencies = np.unique(rolls, return_counts=True)
title = f'Rolling a Six-Sided Die {len(rolls):,} Times'
sns.set_style("whitegrid")
axes = sns.barplot(values, frequencies, palette='bright')
axes.set_title(title)
axes.set(xlabel='Die Value', ylabel='Frequency')
axes.set_ylim(top=max(frequencies) * 1.10)
for bar, frequency in zip(axes.patches, frequencies):
    text_x = bar.get_x() + bar.get_width() / 2.0
    text_y = bar.get_height()
    text = f'{frequency:,}\n{frequency / len(rolls):.3%}'
    axes.text(text_x, text_y, text,
              fontsize=11, ha='center', va='bottom')
```

Аргументы командной строки; вывод диаграммы из сценария

В примерах этой главы содержится отредактированная версия файла RollDie.py, сохраненного в предыдущем разделе. Мы добавили комментарии и два небольших изменения, чтобы сценарий можно было запустить с аргументом, определяющим количество бросков, например:

```
ipython RollDie.py 600
```

Модуль `sys` стандартной библиотеки Python дает возможность сценариям получать *аргументы командной строки*, переданные программе. К их числу относится имя сценария и любые значения, указанные справа от него при запуске сценария. Аргументы содержатся в списке `argv` модуля `sys`. В приведенной выше команде `argv[0]` содержит *строку* `'RollDie.py'`, а `argv[1]` — *строку* `'600'`.

Чтобы управлять количеством бросков при помощи аргумента командной строки, мы изменили команду создания списка `rolls`; теперь она выглядит так:

```
rolls = [random.randrange(1, 7) for i in range(int(sys.argv[1]))]
```

Обратите внимание на преобразование строки `argv[1]` в `int`.

Библиотеки Matplotlib и Seaborn не выводят автоматически диаграмму, созданную в сценарии. По этой причине в конец сценария добавляется вызов функции `show` библиотеки Matplotlib, которая выводит на экран окно с диаграммой:

```
plt.show()
```

5.18. Итоги

В этой главе приведена более подробная информация о списках и кортежах. Были рассмотрены операции создания списков, обращения к их элементам и определения длины. Вы узнали, что списки являются изменяемыми; это означает, что вы можете изменять их содержимое, в частности увеличивать и уменьшать размер списка во время выполнения программы. При обращении к несуществующему элементу происходит ошибка `IndexError`. Для перебора элементов списка используются команды `for`.

Далее были рассмотрены кортежи — они, как и списки, относятся к последовательностям, но являются неизменяемыми. Мы распаковывали элементы кортежей на отдельные переменные и использовали `enumerate` для создания итерируемого объекта, содержащего кортежи (каждый кортеж состоит из индекса списка и значения соответствующего элемента).

Вы узнали, что все последовательности поддерживают операцию сегментации — создания новых последовательностей из подмножеств исходных элементов. Команда `del` использовалась для удаления элементов из списков и для удаления переменных из интерактивных сеансов. Мы передавали функциям списки, элементы списков и сегменты списков. Вы научились проводить поиск и сортировать списки, а также выполнять поиск в кортежах. Методы списков использовались для вставки, присоединения и удаления элементов, а также для перестановки элементов списков в обратном порядке и копирования списков.

Затем было показано, как моделировать стеки на базе списков. Вы узнали, как использовать компактную запись трансформаций списков для создания

новых списков. При помощи дополнительных встроенных методов выполнялись такие операции, как суммирование элементов списков, перебор списка в обратном порядке, нахождение наименьшего и наибольшего значения, фильтрация значений и отображение значений на новые значения. Мы показали, как использовать встроенные списки для представления двумерных таблиц, в которых данные упорядочены по строкам и столбцам, и как обрабатывать двумерные списки во вложенных циклах `for`.

Эта глава завершается разделом «Введение в data science», в котором были представлены примеры с моделированием бросков кубиков и статическими визуализациями. В приведенном примере кода библиотеки визуализации Seaborn и Matplotlib использовались для построения *статической* гистограммы на основе результатов моделирования. В следующем разделе «Введение в data science» моделирование бросков кубиков будет использоваться для построения *динамической* визуализации, чтобы гистограмма «ожила» на экране.

В следующей главе продолжится изучение встроенных коллекций Python. Словари предназначены для хранения неупорядоченных коллекций пар «ключ-значение», связывающих неизменяемые ключи со значениями (по аналогии с тем, как «обычные» словари связывают слова с определениями). Множества используются для хранения неупорядоченных коллекций уникальных элементов.

В главе 7 коллекция `ndarray` библиотеки NumPy рассматривается более подробно. Вы узнаете, что списки хорошо подходят для малых объемов данных, но неэффективны для больших объемов, встречающихся в области аналитики больших данных. В таких случаях следует применять высокооптимизированную коллекцию `ndarray` библиотеки NumPy. Коллекция `ndarray` (то есть «*n*-мерный массив») может заметно превосходить списки по скорости. Чтобы узнать, насколько быстрее они работают, воспользуемся профилирующими тестами Python. NumPy также включает много средств для удобных и эффективных операций с *многомерными* массивами. В области аналитики больших данных потребности в вычислительных мощностях могут быть колоссальными, поэтому все усилия, направленные на повышение быстродействия, приводят к значительным последствиям. В главе 16 будет использоваться одна из самых высокопроизводительных баз данных для работы с большими данными — MongoDB[1].

[1] Название этой базы данных произошло от сокращенного слова *humongous*, что значит «огромный».

6

Словари и множества

В этой главе...

- Использование словарей для представления неупорядоченных коллекций пар «ключ-значение».
- Использование множеств для представления неупорядоченных коллекций уникальных значений.
- Создание, инициализация и обращение к элементам словарей и множеств.
- Перебор ключей, значений и пар «ключ-значение» в словарях.
- Добавление, удаление и обновление пар «ключ-значение» в словарях.
- Операторы сравнения словарей и множеств.
- Объединение множеств операторами и методами множеств.
- Проверка наличия ключа в словаре или значения во множестве операторами `in` и `not in`.
- Использование операций изменяемых множеств для изменения содержимого множеств.
- Использование трансформаций для быстрого и удобного создания словарей и множеств.
- Построение динамических визуализаций.
- Более глубокое понимание изменяемости и неизменяемости.

6.1. Введение

В предыдущей главе были рассмотрены три встроенные коллекции, относящиеся к категории последовательностей, — строки, списки и кортежи. Эта глава посвящена встроенным коллекциям, которые не являются последовательностями, — словарям и множествам. *Словарь* (dictionary) представляет собой *неупорядоченную* коллекцию для хранения *пар «ключ-значение»*, связывающих неизменяемые ключи со значениями (по аналогии с тем, как в традиционных словарях слова связываются с определениями). *Множество* (set) представляет собой неупорядоченную коллекцию *уникальных* неизменяемых элементов.

6.2. Словари

Словарь *связывает* ключи со значениями. Каждому ключу *соответствует* конкретное значение. В табл. 6.1 представлены примеры словарей с ключами, типами ключей, значениями и типами значений:

Таблица 6.1. Примеры словарей с ключами, типами ключей, значениями и типами значений

Ключи	Типы ключей	Значения	Типы значений
Названия стран	`str`	Коды стран в интернете	`str`
Целые числа	`int`	Римские числа	`str`
Штаты	`str`	Сельскохозяйственные продукты	список `str`
Пациенты в больнице	`srt`	Физиологические параметры	кортеж `int` и `float`
Игроки в бейсбольной команде	`str`	Средняя результативность	`float`
Единицы измерения	`str`	Сокращения	`str`
Коды складского учета	`str`	Запасы на складе	`int`

Уникальность ключей

Ключи словаря должны быть *неизменяемыми* (например, строки, числа и кортежи) и *уникальными* (то есть дубликаты запрещены). Разным ключам могут

соответствовать одинаковые значения — например, двум кодам складского учета могут соответствовать одинаковые размеры складских запасов.

6.2.1. Создание словаря

Чтобы создать словарь, заключите в фигурные скобки `{}` список пар «ключ-значение», разделенных запятыми, в форме *ключ*`:` *значение*. Пустой словарь создается в форме `{}`.

Создадим словарь с ключами, содержащими названия стран (`'Finland'`, `'South Africa'` и `'Nepal'`), и значениями, представляющими коды стран в интернете: `'fi'`, `'za'` и `'np'`:

```
In [1]: country_codes = {'Finland': 'fi', 'South Africa': 'za',
   ...:                  'Nepal': 'np'}
   ...:

In [2]: country_codes
Out[2]: {'Finland': 'fi', 'South Africa': 'za', 'Nepal': 'np'}
```

При выводе словаря его список пар «ключ-значение», разделенных запятыми, всегда заключается в фигурные скобки. Поскольку словари относятся к *неупорядоченным* коллекциям, порядок вывода может отличаться от порядка добавления пар «ключ-значение» в словарь. Во фрагменте [2] пары «ключ-значение» выводятся в порядке вставки, но ваш код не должен зависеть от какого-то определенного порядка пар «ключ-значение».

Проверка пустого словаря

Встроенная функция `len` возвращает количество пар «ключ-значение» в словаре:

```
In [3]: len(country_codes)
Out[3]: 3
```

Словарь может использоваться в качестве условия для проверки того, содержит ли он хотя бы одну пару, — непустой словарь интерпретируется как `True`:

```
In [4]: if country_codes:
   ...:     print('country_codes is not empty')
   ...: else:
   ...:     print('country_codes is empty')
   ...:
country_codes is not empty
```

Пустой словарь интерпретируется как `False`. Чтобы продемонстрировать этот факт, в следующем коде мы вызываем метод `clear` для удаления всех пар «ключ-значение», а затем во фрагменте [6] возвращаем и снова выполняем фрагмент [4]:

```
In [5]: country_codes.clear()

In [6]: if country_codes:
   ...:     print('country_codes is not empty')
   ...: else:
   ...:     print('country_codes is empty')
   ...:
country_codes is empty
```

6.2.2. Перебор по словарю

Следующий словарь связывает строки названий месяцев со значениями `int`, представляющими количество дней в соответствующем месяце. Обратите внимание: *разным* ключам может соответствовать *одно* значение:

```
In [1]: days_per_month = {'January': 31, 'February': 28, 'March': 31}

In [2]: days_per_month
Out[2]: {'January': 31, 'February': 28, 'March': 31}
```

И снова в строковом представлении словаря пары «ключ-значение» выводятся в порядке вставки, но этот порядок не гарантирован, потому что словари *не упорядочиваются*. Позднее в этой главе мы покажем, как организовать обработку ключей в порядке *сортировки*.

Следующая команда `for` перебирает пары «ключ-значение» словаря `days_per_month`. Метод словарей `items` возвращает каждую пару «ключ-значение» в виде кортежа, который распаковывается по переменным `month` и `days`:

```
In [3]: for month, days in days_per_month.items():
   ...:     print(f'{month} has {days} days')
   ...:
January has 31 days
February has 28 days
March has 31 days
```

6.2.3. Основные операции со словарями

В этом разделе мы начнем с создания и отображения словаря `roman_numerals`. Для ключа `'X'` намеренно задается неправильное значение `100`, которое мы вскоре исправим:

```
In [1]: roman_numerals = {'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 100}

In [2]: roman_numerals
Out[2]: {'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 100}
```

Обращение к значению, связанному с ключом

Получим значение, связанное с ключом `'V'`:

```
In [3]: roman_numerals['V']
Out[3]: 5
```

Обновление значения в существующей паре «ключ-значение»

Вы можете обновить значение, связанное с ключом, при помощи команды присваивания. В следующем примере это делается для замены неправильного значения, связанного с ключом `'X'`:

```
In [4]: roman_numerals['X'] = 10

In [5]: roman_numerals
Out[5]: {'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 10}
```

Добавление новых пар «ключ-значение»

При попытке присваивания значения несуществующему ключу в словарь вставляется пара «ключ-значение»:

```
In [6]: roman_numerals['L'] = 50

In [7]: roman_numerals
Out[7]: {'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 10, 'L': 50}
```

В строковых ключах учитывается регистр символов. Присваивание несуществующему ключу приводит к вставке новой пары «ключ-значение». Возможно, именно это вам и требуется, а может быть, это логическая ошибка в программе.

Удаление пары «ключ-значение»

Пары «ключ-значение» удаляются из словаря командой `del`:

```
In [8]: del roman_numerals['III']

In [9]: roman_numerals
Out[9]: {'I': 1, 'II': 2, 'V': 5, 'X': 10, 'L': 50}
```

Пары «ключ-значение» также могут удаляться методом словарей pop, который возвращает значение удаленного ключа:

```
In [10]: roman_numerals.pop('X')
Out[10]: 10

In [11]: roman_numerals
Out[11]: {'I': 1, 'II': 2, 'V': 5, 'L': 50}
```

Попытки обращения к несуществующему ключу

Попытка обращения к несуществующему ключу приводит к ошибке KeyError:

```
In [12]: roman_numerals['III']
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-12-ccd50c7f0c8b> in <module>()
----> 1 roman_numerals['III']

KeyError: 'III'
```

Для предотвращения этой ошибки можно воспользоваться методом словарей get, который обычно возвращает значение, соответствующее переданному аргументу. Если ключ не найден, то get возвращает None. При возвращении None в фрагменте [13] вывод отсутствует. Если передать при вызове второй аргумент, то при отсутствии ключа выдается значение:

```
In [13]: roman_numerals.get('III')

In [14]: roman_numerals.get('III', 'III not in dictionary')
Out[14]: 'III not in dictionary'

In [15]: roman_numerals.get('V')
Out[15]: 5
```

Проверка наличия заданного ключа в словаре

Чтобы определить, содержит ли словарь заданный ключ, можно воспользоваться операторами in и not in:

```
In [16]: 'V' in roman_numerals
Out[16]: True

In [17]: 'III' in roman_numerals
Out[17]: False

In [18]: 'III' not in roman_numerals
Out[18]: True
```

6.2.4. Методы keys и values

Ранее метод `items` словарей использовался для перебора кортежей пар «ключ-значение», содержащихся в словаре. Похожие методы `keys` и `values` могут использоваться для перебора только ключей или значений соответственно:

```
In [1]: months = {'January': 1, 'February': 2, 'March': 3}

In [2]: for month_name in months.keys():
   ...:     print(month_name, end='  ')
   ...:
January  February  March
In [3]: for month_number in months.values():
   ...:     print(month_number, end='  ')
   ...:
1  2  3
```

Представления словарей

Каждый из методов `items`, `keys` и `values`, поддерживаемых словарями, возвращает представление данных словаря. При переборе представление «видит» текущее содержимое словаря — собственной копии данных у него *нет*.

Чтобы показать, что представления *не* поддерживают собственную копию данных словаря, сначала сохраним представление, возвращаемое `keys`, в переменной `months_view`, а затем переберем ее содержимое:

```
In [4]: months_view = months.keys()

In [5]: for key in months_view:
   ...:     print(key, end='  ')
   ...:
January  February  March
```

Затем добавим в `month` новую пару «ключ-значение» и выведем обновленный словарь:

```
In [6]: months['December'] = 12

In [7]: months
Out[7]: {'January': 1, 'February': 2, 'March': 3, 'December': 12}
```

Теперь снова переберем содержимое `months_view`. Добавленный ключ действительно выводится в числе прочих:

```
In [8]: for key in months_view:
   ...:     print(key, end='  ')
   ...:
January  February  March  December
```

Не изменяйте словарь во время перебора представления. Как указано в разделе 4.10.1 документации стандартной библиотеки Python[1], в этом случае либо произойдет ошибка `RuntimeError`, либо цикл не обработает все значения из представления.

Преобразование ключей, значений и пар «ключ-значение» в списки

Возможно, в каких-то ситуациях вам понадобится получить *список*, содержащий ключи, значения или пары «ключ-значение» из словаря. Чтобы получить такой список, передайте представление, возвращенное `keys`, `values` или `items`, встроенной функции `list`. Модификация таких списков *не приведет* к изменению соответствующего словаря:

```
In [9]: list(months.keys())
Out[9]: ['January', 'February', 'March', 'December']

In [10]: list(months.values())
Out[10]: [1, 2, 3, 12]

In [11]: list(months.items())
Out[11]: [('January', 1), ('February', 2), ('March', 3), ('December', 12)]
```

Обработка ключей в порядке сортировки

Чтобы обработать ключи в порядке *сортировки*, используйте встроенную функцию `sorted`:

```
In [12]: for month_name in sorted(months.keys()):
    ...:     print(month_name, end='  ')
    ...:
February  December  January  March
```

[1] https://docs.python.org/3/library/stdtypes.html#dictionary-view-objects.

6.2.5. Сравнения словарей

Операторы сравнения == и != могут использоваться для проверки того, имеют ли два словаря идентичное (или разное) содержимое. Проверка равенства (==) дает результат True, если оба словаря содержат одинаковые пары «ключ-значение» *независимо от того*, в каком порядке эти пары добавлялись в каждый словарь:

```
In [1]: country_capitals1 = {'Belgium': 'Brussels',
   ...:                      'Haiti': 'Port-au-Prince'}
   ...:

In [2]: country_capitals2 = {'Nepal': 'Kathmandu',
   ...:                      'Uruguay': 'Montevideo'}
   ...:

In [3]: country_capitals3 = {'Haiti': 'Port-au-Prince',
   ...:                      'Belgium': 'Brussels'}
   ...:

In [4]: country_capitals1 == country_capitals2
Out[4]: False

In [5]: country_capitals1 == country_capitals3
Out[5]: True

In [6]: country_capitals1 != country_capitals2
Out[6]: True
```

6.2.6. Пример: словарь с оценками студентов

Следующий сценарий представляет преподавательский журнал в виде словаря, связывающего имя каждого студента (строка) со списком целых чисел, представляющим оценки этого студента на трех экзаменах. При каждой итерации цикла вывода данных (строки 13–17) пара «ключ-значение» распаковывается в переменные name и grades, содержащие имя одного студента и соответствующий список из трех оценок. В строке 14 встроенная функция sum суммирует оценки, а строка 15 вычисляет среднюю оценку этого студента делением суммы total на количество оценок (len(grades)) и выводит ее. В строках 16–17 подсчитывается сумма оценок всех четырех студентов и количество оценок для всех студентов соответственно. В строке 19 выводится средняя оценка для класса, вычисленная усреднением оценок по всем экзаменам.

```
 1 # fig06_01.py
 2 """Хранение данных преподавательского журнала в словаре."""
 3 grade_book = {
 4     'Susan': [92, 85, 100],
 5     'Eduardo': [83, 95, 79],
 6     'Azizi': [91, 89, 82],
 7     'Pantipa': [97, 91, 92]
 8 }
 9
10 all_grades_total = 0
11 all_grades_count = 0
12
13 for name, grades in grade_book.items():
14     total = sum(grades)
15     print(f'Average for {name} is {total/len(grades):.2f}')
16     all_grades_total += total
17     all_grades_count += len(grades)
18
19 print(f"Class's average is: {all_grades_total / all_grades_count:.2f}")
```

```
Average for Susan is 92.33
Average for Eduardo is 85.67
Average for Azizi is 87.33
Average for Pantipa is 93.33
Class's average is: 89.67
```

6.2.7. Пример: подсчет слов[1]

Следующий сценарий строит словарь для подсчета количества вхождений каждого слова в строке. В строках 4–5 создается строка `text`, которую мы собираемся разбить на слова, — этот процесс называется *разбиением строки на лексемы*. Python автоматически выполняет конкатенацию строк, разделенных пропусками, в круглых скобках. Строка 7 создает пустой словарь. Ключами словаря будут уникальные слова, а значениями — целочисленные счетчики вхождений каждого слова в `тексте`.

[1] Такие методы, как вычисление частот вхождения слов, часто используются для анализа публикаций. Например, некоторые специалисты полагают, что работы Уильяма Шекспира на самом деле были написаны сэром Фрэнсисом Бэконом, Кристофером Марлоу или кем-нибудь еще. Сравнение частот слов в этих работах с частотами слов в работах Шекспира может выявить сходство в стиле письма. Другие методы анализа документов будут рассмотрены в главе 11.

```
 1 # fig06_02.py
 2 """Разбиение строк на лексемы и подсчет уникальных слов."""
 3
 4 text = ('this is sample text with several words '
 5         'this is more sample text with some different words')
 6
 7 word_counts = {}
 8
 9 # Подсчет вхождений уникальных слов
10 for word in text.split():
11     if word in word_counts:
12         word_counts[word] += 1  # Обновление существующей пары "ключ-значение"
13     else:
14         word_counts[word] = 1  # Вставка новой пары "ключ-значение"
15
16 print(f'{"WORD":<12}COUNT')
17
18 for word, count in sorted(word_counts.items()):
19     print(f'{word:<12}{count}')
20
21 print('\nNumber of unique words:', len(word_counts))
```

```
WORD        COUNT
different   1
is          2
more        1
sample      2
several     1
some        1
text        2
this        2
with        2
words       2
Number of unique words: 10
```

Строка 10 разбивает текст на лексемы вызовом метода `split` строк; при этом слова разделяются по строковому аргументу метода `delimiter`. Если аргумент не задан, то `split` использует пробел. Метод возвращает список лексем (то есть слов в тексте). В строках 10–14 перебирается список слов. Для каждого слова строка 11 определяет, присутствует ли это слово (ключ) в словаре. Если оно присутствует, то строка 12 увеличивает счетчик для этого слова; в противном случае строка 14 вставляет новую пару «ключ-значение» для этого слова с исходным значением счетчика 1.

В строках 16–21 приведена сводка результатов в виде таблицы из двух столбцов, содержащих каждое слово и соответствующий счетчик. Команда `for` в строках 18 и 19 перебирает пары «ключ-значение» в словаре. Каждый ключ

и значение распаковываются в переменные `word` и `count`, после чего выводятся в два столбца. Строка 21 выводит количество уникальных слов.

Модуль стандартной библиотеки Python collections

Стандартная библиотека Python уже содержит функциональность подсчета, реализованную с использованием словаря и цикла в строках 10–14. Модуль `collections` содержит тип `Counter`, который получает итерируемый объект и генерирует сводную информацию на основании его элементов. Реализуем предыдущий сценарий в нескольких строках кода с использованием `Counter`:

```
In [1]: from collections import Counter

In [2]: text = ('this is sample text with several words '
   ...:         'this is more sample text with some different words')
   ...:

In [3]: counter = Counter(text.split())

In [4]: for word, count in sorted(counter.items()):
   ...:     print(f'{word:<12}{count}')
   ...:
different   1
is          2
more        1
sample      2
several     1
some        1
text        2
this        2
with        2
words       2

In [5]: print('Number of unique keys:', len(counter.keys()))
Number of unique keys: 10
```

Фрагмент [3] создает объект `Counter`, который генерирует сводную информацию для списка строк, возвращаемых вызовом `text.split()`. Во фрагменте [4] метод `items` объекта `Counter` возвращает каждую строку и связанный с ней счетчик в форме кортежа. Встроенная функция `sorted` используется для получения списка этих кортежей, упорядоченного по возрастанию. По умолчанию `sorted` упорядочивает кортежи по первым элементам. Если они идентичны, то функция проверяет второй элемент, и т. д. Команда `for` перебирает полученный отсортированный список, выводя каждое слово и счетчик в два столбца.

6.2.8. Метод update

Для вставки пар «ключ-значение» можно использовать метод `update` словарей. Начнем с создания пустого словаря `country_codes`:

```
In [1]: country_codes = {}
```

Следующий вызов `update` получает словарь пар «ключ-значение» для вставки или обновления:

```
In [2]: country_codes.update({'South Africa': 'za'})

In [3]: country_codes
Out[3]: {'South Africa': 'za'}
```

Метод `update` может преобразовать ключевые аргументы в пары «ключ-значение» для вставки. Следующий вызов автоматически преобразует имя параметра `Australia` в строковый ключ `'Australia'` и связывает с этим ключом значение `'ar'`:

```
In [4]: country_codes.update(Australia='ar')

In [5]: country_codes
Out[5]: {'South Africa': 'za', 'Australia': 'ar'}
```

Во фрагменте [4] предоставляется неверный код для `Australia`. Исправим проблему, передав другой ключевой аргумент для обновления значения, связанного со строкой `'Australia'`:

```
In [6]: country_codes.update(Australia='au')

In [7]: country_codes
Out[7]: {'South Africa': 'za', 'Australia': 'au'}
```

Метод `update` также может получать итерируемый объект с парами «ключ-значение», например список кортежей, состоящих из двух элементов.

6.2.9. Трансформации словарей

Трансформации словарей предоставляют удобную запись для быстрого генерирования словарей, часто посредством отображения одного словаря на другой. Например, в словаре с *уникальными* значениями можно поменять местами пары «ключ-значение»:

```
In [1]: months = {'January': 1, 'February': 2, 'March': 3}

In [2]: months2 = {number: name for name, number in months.items()}

In [3]: months2
Out[3]: {1: 'January', 2: 'February', 3: 'March'}
```

Фигурные скобки ограничивают *трансформацию словаря*, а выражение слева от `for` задает пару «ключ-значение» в форме *ключ*`:` *значение*. Трансформация перебирает `months.items()`, распаковывая кортеж каждой пары «ключ-значение» в переменные `name` и `number`. Выражение `number: name` меняет местами ключи и значения, в результате новый словарь отображает номера месяцев на их названия.

При наличии в `months` *повторяющихся* значений последние становятся ключами в `months2`. Попытка вставки *повторяющегося* ключа приведет к обновлению существующего ключа. Таким образом, если строки `'February'` и `'March'` изначально отображались в `2`, то этот код воспроизвел бы словарь следующего вида

```
{1: 'January', 2: 'March'}
```

Трансформация словаря также может связывать значения словаря с новыми значениями. Следующая трансформация преобразует словарь с именами и списками оценок в словарь с именами и средними оценками. Переменные `k` и `v` обычно представляют *ключи* (key) и *значения* (value):

```
In [4]: grades = {'Sue': [98, 87, 94], 'Bob': [84, 95, 91]}

In [5]: grades2 = {k: sum(v) / len(v) for k, v in grades.items()}

In [6]: grades2
Out[6]: {'Sue': 93.0, 'Bob': 90.0}
```

Трансформация распаковывает каждый кортеж, возвращенный `grades.items()`, в `k` (имя) и `v` (список оценок). Таким образом, трансформация создает новую пару «ключ-значение» с ключом `k` и значением `sum(v) / len(v)`, усредняющим элементы списка.

6.3. Множества

Множество представляет собой неупорядоченную коллекцию *уникальных* значений. Множества могут содержать только неизменяемые объекты: строки,

`int`, `float` и кортежи, содержащие исключительно неизменяемые элементы. Хотя множества являются итерируемыми объектами, они не относятся к последовательностям и не поддерживают индексирование и сегментацию с квадратными скобками `[]`. Словари также не поддерживают сегментацию.

Создание множества в фигурных скобках

Следующий код создает множество строк с именем `colors`:

```
In [1]: colors = {'red', 'orange', 'yellow', 'green', 'red', 'blue'}

In [2]: colors
Out[2]: {'blue', 'green', 'orange', 'red', 'yellow'}
```

Обратите внимание: повторяющаяся строка `'red'` была проигнорирована (без возникновения ошибки). Одно из важных областей применения множеств — *удаление дубликатов*, выполняемое автоматически при создании множества. Кроме того, значения полученного множества *не* выводятся в порядке их перечисления во фрагменте [1]. Хотя названия цветов отображаются в порядке сортировки, множества *не упорядочены*. Ваш код не должен зависеть от порядка элементов.

Определение длины множества

Количество элементов в множестве можно определить при помощи встроенной функции `len`:

```
In [3]: len(colors)
Out[3]: 5
```

Проверка наличия значения во множестве

Чтобы проверить, содержит ли множество конкретное значение, можно воспользоваться операторами `in` и `not in`:

```
In [4]: 'red' in colors
Out[4]: True

In [5]: 'purple' in colors
Out[5]: False

In [6]: 'purple' not in colors
Out[6]: True
```

Перебор элементов множества

Множества являются итерируемыми объектами. Для перебора их элементов можно воспользоваться циклом `for`:

```
In [7]: for color in colors:
   ...:     print(color.upper(), end=' ')
   ...:
RED GREEN YELLOW BLUE ORANGE
```

Поскольку множества *не упорядочены*, порядок перебора роли не играет.

Создание множества встроенной функцией set

Множество также можно создать на базе другой коллекции значений, используя встроенную функцию `set`, — здесь мы создаем список, содержащий несколько дубликатов целочисленных значений, и передаем этот список в аргументе `set`:

```
In [8]: numbers = list(range(10)) + list(range(5))

In [9]: numbers
Out[9]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4]

In [10]: set(numbers)
Out[10]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Если вам понадобится создать пустое множество, то необходимо использовать функцию `set` с пустыми круглыми скобками вместо фигурных скобок `{}`, которые представляют пустой словарь:

```
In [11]: set()
Out[11]: set()
```

Python выводит пустое множество в виде `set()`, чтобы избежать путаницы со строковым представлением Python пустого словаря (`{}`).

Фиксированное множество: неизменяемый тип множества

Множества *изменяемы* — вы можете добавлять и удалять элементы, но при этом *элементы множеств* должны оставаться *неизменяемыми*. Таким образом, элементами множеств не могут быть другие множества. *Фиксированное множество* (frozenset) *неизменяемо* — его невозможно изменить после создания,

поэтому элементами множеств могут быть фиксированные множества. Встроенная функция `frozenset` создает фиксированное множество на базе любого итерируемого объекта.

6.3.1. Сравнение множеств

Для сравнения множеств могут использоваться различные операторы и методы. Следующие множества содержат одинаковые значения, поэтому `==` возвращает `True`, а `!=` возвращает `False`.

```
In [1]: {1, 3, 5} == {3, 5, 1}
Out[1]: True

In [2]: {1, 3, 5} != {3, 5, 1}
Out[2]: False
```

Оператор `<` проверяет, является ли множество в левой части подмножества *строгим подмножеством* множества в правой части, то есть все элементы левого операнда присутствуют в правом операнде и множества не равны:

```
In [3]: {1, 3, 5} < {3, 5, 1}
Out[3]: False

In [4]: {1, 3, 5} < {7, 3, 5, 1}
Out[4]: True
```

Оператор `<=` проверяет, является ли множество в левой части *нестрогим подмножеством* множества в правой части, то есть все элементы левого операнда присутствуют в правом операнде и эти множества могут быть равны:

```
In [5]: {1, 3, 5} <= {3, 5, 1}
Out[5]: True

In [6]: {1, 3} <= {3, 5, 1}
Out[6]: True
```

Для проверки нестрогого подмножества также можно воспользоваться методом `issubset`:

```
In [7]: {1, 3, 5}.issubset({3, 5, 1})
Out[7]: True

In [8]: {1, 2}.issubset({3, 5, 1})
Out[8]: False
```

Оператор > проверяет, является ли множество в левой части подмножества *строгим надмножеством* множества в правой части, то есть все элементы правого операнда присутствуют в левом операнде и левый операнд содержит больше элементов:

```
In [9]: {1, 3, 5} > {3, 5, 1}
Out[9]: False

In [10]: {1, 3, 5, 7} > {3, 5, 1}
Out[10]: True
```

Оператор >= проверяет, является ли множество в левой части *нестрогим надмножеством* множества в правой части, то есть все элементы правого операнда присутствуют в левом операнде и эти множества могут быть равны:

```
In [11]: {1, 3, 5} >= {3, 5, 1}
Out[11]: True

In [12]: {1, 3, 5} >= {3, 1}
Out[12]: True

In [13]: {1, 3} >= {3, 1, 7}
Out[13]: False
```

Для проверки нестрогого надмножества также можно воспользоваться методом `issuperset`:

```
In [14]: {1, 3, 5}.issuperset({3, 5, 1})
Out[14]: True

In [15]: {1, 3, 5}.issuperset({3, 2})
Out[15]: False
```

Аргументом `issubset` или `issuperset` может быть *любой* итерируемый объект. Когда любой из этих методов получает итерируемый аргумент, не являющийся множеством, он сначала преобразует итерируемый объект в множество, а затем выполняет операцию.

6.3.2. Математические операции с множествами

В этом разделе представлены основные математические операции множеств |, &, - и ^, а также соответствующие методы.

Объединение

Объединением двух множеств называется множество, состоящее из всех уникальных элементов обоих множеств. Для вычисления объединения можно воспользоваться *оператором* | или методом `union` типа множества:

```
In [1]: {1, 3, 5} | {2, 3, 4}
Out[1]: {1, 2, 3, 4, 5}

In [2]: {1, 3, 5}.union([20, 20, 3, 40, 40])
Out[2]: {1, 3, 5, 20, 40}
```

Оба операнда бинарных операторов множеств (таких как |) должны быть множествами. Соответствующие методы множеств могут получать в аргументе любой итерируемый объект — например, мы передавали список. Если математический метод множества получает аргумент — итерируемый объект, который не является множеством, то он сначала преобразует итерируемый объект в множество, а затем применяет математическую операцию. Еще раз напомним: хотя в строковых представлениях новых множеств значения следуют по возрастанию, ваш код не должен зависеть от этого порядка.

Пересечение

Пересечением двух множеств является множество, состоящее из всех уникальных элементов, входящих в оба множества. Пересечение можно вычислить *оператором* & или методом `intersection` типа множества:

```
In [3]: {1, 3, 5} & {2, 3, 4}
Out[3]: {3}

In [4]: {1, 3, 5}.intersection([1, 2, 2, 3, 3, 4, 4])
Out[4]: {1, 3}
```

Разность

Разностью двух множеств является множество, состоящее из элементов левого операнда, не входящих в правый операнд. Разность можно вычислить *оператором* или методом `difference` типа множества:

```
In [5]: {1, 3, 5} - {2, 3, 4}
Out[5]: {1, 5}

In [6]: {1, 3, 5, 7}.difference([2, 2, 3, 3, 4, 4])
Out[6]: {1, 5, 7}
```

Симметрическая разность

Симметрической разностью двух множеств является множество, состоящее из элементов каждого множества, не входящих в другое множество. Симметрическая разность вычисляется *оператором* ^ или методом `symmetric_difference` типа множества:

```
In [7]: {1, 3, 5} ^ {2, 3, 4}
Out[7]: {1, 2, 4, 5}

In [8]: {1, 3, 5, 7}.symmetric_difference([2, 2, 3, 3, 4, 4])
Out[8]: {1, 2, 4, 5, 7}
```

Непересекающиеся множества

Два множества называются *непересекающимися*, если они не содержат общих элементов. Проверка непересекающихся множеств может быть выполнена методом `isdisjoint` типа множества:

```
In [9]: {1, 3, 5}.isdisjoint({2, 4, 6})
Out[9]: True

In [10]: {1, 3, 5}.isdisjoint({4, 6, 1})
Out[10]: False
```

6.3.3. Операторы и методы изменяемых множеств

Операторы и методы, представленные в предыдущем разделе, создают *новое* множество. В этом разделе речь пойдет об операторах и методах, которые изменяют *существующее* множество.

Математические операции с изменяемыми множествами

Расширенное присваивание с объединением |= выполняет операцию объединения множеств, как и оператор |, но |= изменяет свой левый операнд:

```
In [1]: numbers = {1, 3, 5}

In [2]: numbers |= {2, 3, 4}

In [3]: numbers
Out[3]: {1, 2, 3, 4, 5}
```

Аналогичным образом метод `update` типа множества выполняет операцию объединения множеств с множеством, для которого он был вызван, причем аргументом может быть любой итерируемый объект:

```
In [4]: numbers.update(range(10))

In [5]: numbers
Out[5]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Другие методы изменяемых множеств:

- расширенное присваивание с пересечением `&=`
- расширенное присваивание с разностью `-=`
- расширенное присваивание с симметрической разностью `^=`

и соответствующие им методы с аргументами — итерируемыми объектами:

- `intersection_update`
- `difference_update`
- `symmetric_difference_update`

Методы добавления и удаления элементов

Метод `add` множества вставляет свой аргумент, если он еще *не* входит в множество; в противном случае множество остается без изменений:

```
In [6]: numbers.add(17)

In [7]: numbers.add(3)

In [8]: numbers
Out[8]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 17}
```

Метод `remove` множества удаляет свой аргумент из множества. Если значение отсутствует в множестве, то происходит ошибка `KeyError`:

```
In [9]: numbers.remove(3)

In [10]: numbers
Out[10]: {0, 1, 2, 4, 5, 6, 7, 8, 9, 17}
```

Метод `discard` также удаляет аргумент из множества, но если значение отсутствует в множестве, то исключения не происходит.

Также возможно удалить *произвольный* элемент множества и вернуть его вызовом pop, но множества не упорядочены, поэтому вы не знаете, какой именно элемент будет возвращен:

```
In [11]: numbers.pop()
Out[11]: 0

In [12]: numbers
Out[12]: {1, 2, 4, 5, 6, 7, 8, 9, 17}
```

Если pop вызывается для пустого множества, то происходит ошибка `KeyError`.

Наконец, метод `clear` очищает множество, для которого он был вызван:

```
In [13]: numbers.clear()

In [14]: numbers
Out[14]: set()
```

6.3.4. Трансформации множеств

Трансформации множеств, как и трансформации словарей, определяются в фигурных скобках. Создадим новое множество, которое содержит только уникальные четные значения из списка `numbers`:

```
In [1]: numbers = [1, 2, 2, 3, 4, 5, 6, 6, 7, 8, 9, 10, 10]

In [2]: evens = {item for item in numbers if item % 2 == 0}

In [3]: evens
Out[3]: {2, 4, 6, 8, 10}
```

6.4. Введение в data science: динамические визуализации

В разделе «Введение в data science» предыдущей главы мы описали основы визуализации. Мы моделировали броски шестигранного кубика и использовали библиотеки визуализации Seaborn и Matplotlib для построения *статических* гистограмм типографского качества, на которых приведены частоты и проценты каждого результата. В этом разделе вы узнаете, как «оживить» гистограмму с использованием *динамических визуализаций*.

Закон больших чисел

При описании генерирования случайных чисел мы упоминали о том, что если функция `randrange` модуля `random` действительно генерирует случайные целые числа, то все числа в заданном диапазоне обладают одинаковой вероятностью выпадения при каждом вызове функции. Для шестигранного кубика каждое из значений от 1 до 6 должно происходить в одном из шести случаев, так что вероятность выпадения любого из этих значений равна 1/6, или около 16,667%.

В следующем разделе мы создадим и выполним сценарий *динамического* (то есть *анимированного*) моделирования бросков кубиков. В общем случае чем больше бросков вы выполните, тем ближе процент каждого результата от общего количества бросков, при этом высоты столбцов постепенно выравниваются (о подобном проявлении *закона больших чисел* см. также ранее).

6.4.1. Как работает динамическая визуализация

Графики, построенные с использованием библиотек Seaborn и Matplotlib в разделе «Введение в data science» предыдущей главы, позволяют проанализировать результаты для фиксированного количества бросков кубиков *после* завершения моделирования. В этом разделе код построения гистограммы дополняется функцией `FuncAnimation` модуля `animation` библиотеки Matplotlib, которая обновляет гистограмму *динамически*. Столбцы, частоты и проценты «оживают» на экране, *непрерывно* обновляясь в процессе моделирования бросков.

Кадры анимации

Функция `FuncAnimation` осуществляет *покадровую анимацию*. Каждый *кадр анимации* определяет все, что должно измениться во время одного обновления. При частом выполнении таких обновлений во времени возникает эффект анимации. Вы решаете, что должен отображать каждый кадр, при помощи функции, которую вы определяете и передаете `FuncAnimation`.

Каждый кадр анимации:

- моделирует броски кубика заданное количество раз (от одного до произвольного на ваш выбор), обновляя частоты выпадения результатов с каждым броском;
- очищает текущую диаграмму;

- создает новый набор столбцов, представляющих обновленные частоты; и
- создает новые тексты частот и процентов для каждого столбца.

В общем случае вывод большего количества кадров в секунду делает анимацию более плавной. Например, видеоигры с быстро движущимися элементами стараются отображать *не менее* 30 кадров в секунду, а часто и более. И хотя вы задаете промежуток в миллисекундах между кадрами анимации, реальное количество кадров в секунду может зависеть от объема работы, выполняемой между кадрами, и производительности процессора вашего компьютера. В нашем примере кадр анимации выводится каждые 33 миллисекунды, что соответствует приблизительно 30 (1000/33) кадрам в секунду. Попробуйте задать большие или меньшие значения, чтобы понять, как они влияют на анимацию. Эксперименты играют важную роль в разработке наиболее удачных визуализаций.

Выполнение RollDieDynamic.py

В разделе «Введение в data science» предыдущей главы статическая визуализация разрабатывалась в *интерактивном* режиме, чтобы вы видели, как код обновляет гистограмму при выполнении каждой команды. Гистограмма с окончательными значениями частот и процентов была выведена всего один раз.

В этой динамической визуализации результаты обновляются настолько часто, что вы видите анимацию. Многие аспекты изображения изменяются непрерывно — длины столбцов, частоты и проценты над столбцами, расстояние и надписи на осях, а также общее количество бросков кубиков в заголовке диаграммы. Из-за этого мы представляем эту визуализацию в виде сценария, вместо того чтобы строить ее в интерактивном режиме.

Сценарий получает два аргумента командной строки:

- `number_of_frames` — количество кадров анимации. Это значение определяет общее количество обновлений гистограммы функцией `FuncAnimation`. Для каждого кадра анимации `FuncAnimation` вызывает функцию, которую вы определяете (в данном примере `update`), для того чтобы указать, как должна изменяться диаграмма;
- `rolls_per_frame` — количество бросков кубика в каждом кадре анимации. Цикл используется для моделирования нужного количества бросков, обобщения результатов и обновления диаграммы новыми столбцами и текстом.

Чтобы понять, как использовать эти два значения, рассмотрим следующую команду:

```
ipython RollDieDynamic.py 6000 1
```

В данном случае `FuncAnimation` вызывает функцию `update` 6000 раз, моделируя один бросок на каждый кадр (итого 6000 бросков). Это позволяет вам видеть, как столбцы, частоты и проценты обновляются при каждом броске. В нашей системе эта анимация заняла в целом около 3,33 минуты (6000 кадров / 30 кадров в секунду / 60 секунд в минуте) для 6000 бросков.

Вывод кадров анимации на экран — относительно медленная операция *ввода/вывода* по сравнению с бросками кубиков, выполняемыми на сверхбыстрых скоростях современных процессоров. Если бросать только один кубик на кадр анимации, нам не удастся смоделировать большое количество бросков за разумный промежуток времени. Кроме того, при малом количестве бросков вы вряд ли увидите, как проценты сходятся к ожидаемым 16,667% от общего количества бросков.

Чтобы понаблюдать за законом больших чисел в действии, можно увеличить скорость выполнения, чтобы за один кадр анимации обрабатывалось большее количество бросков кубика. Рассмотрим следующую команду:

```
ipython RollDieDynamic.py 10000 600
```

В этом случае `FuncAnimation` вызовет функцию `update` 10 000 раз, выполняя 600 бросков на кадр (всего — 6 000 000 бросков). В нашей системе выполнение программы заняло около 5,55 минуты (10 000 кадров / 30 кадров в секунду / 60 секунд в минуту), но в секунду выводилось приблизительно 18 000 бросков (30 кадров в секунду * 600 бросков на кадр), так что вы быстро увидите, как частоты и проценты сходятся к ожидаемым значениям: около 1 000 000 бросков и 16,667% на каждую грань.

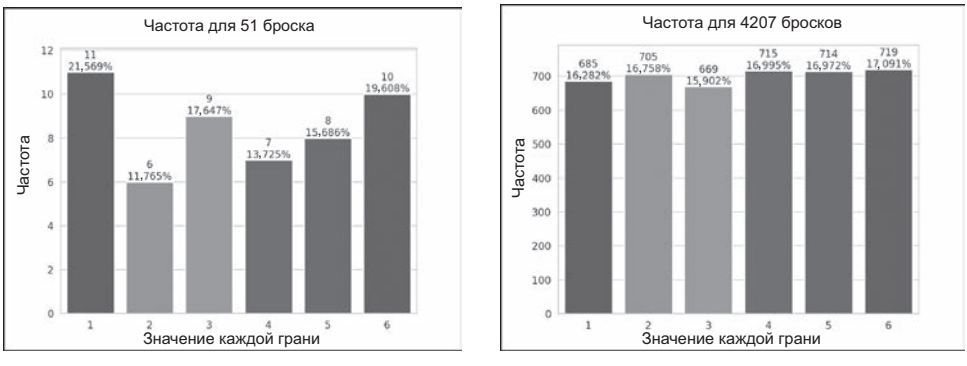
Поэкспериментируйте с количествами бросков и кадров, пока не почувствуете, что программа помогает наиболее эффективно представить результаты в наглядной форме. Понаблюдайте за выполнением программы и настраивайте параметры, пока вас не удовлетворит качество анимации; это довольно увлекательно и поучительно.

Примеры выполнения

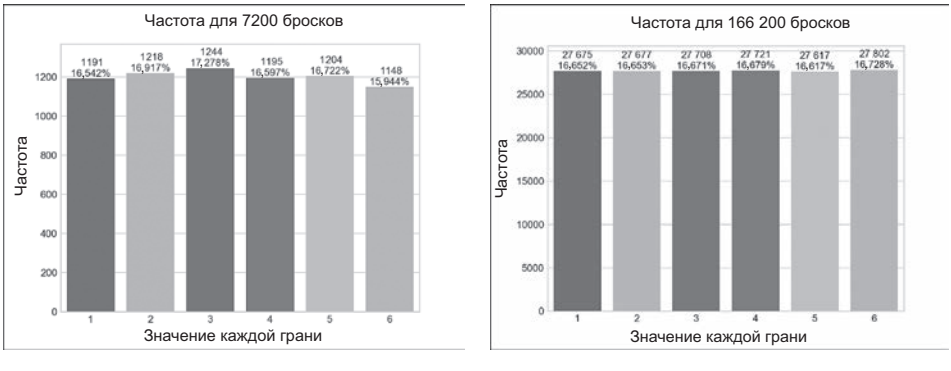
Следующие четыре снимка экрана были сделаны при двух пробных запусках. На первом снимке показана диаграмма после всего 64 бросков кубика,

а затем после 604 из 6000 бросков. Запустите сценарий и понаблюдайте за динамическим обновлением столбцов. При втором запуске на снимках показано состояние диаграммы после 7200 бросков, а потом после 166 200 из 6 000 000 бросков. При большем количестве бросков вы увидите, как проценты сходятся к ожидаемому значению 16,667%, предсказанному законом больших чисел.

Вывод 6000 кадров анимации с броском одного кубика на кадр:



Вывод 10 000 кадров анимации с броском 600 кубиков на кадр:



6.4.2. Реализация динамической визуализации

Сценарий, представленный в этом разделе, использует средства библиотек Seaborn и Matplotlib, представленные в предыдущем разделе «Введение в data science». Мы слегка изменили структуру кода для использования средств *анимации* Matplotlib.

Импортирование модуля animation библиотеки Matplotlib

В описании мы сосредоточимся прежде всего на новых возможностях, использованных в этом примере. Строка 3 импортирует модуль `animation` библиотеки Matplotlib.

```
1 # RollDieDynamic.py
2 """Динамическое построение гистограммы частот бросков кубиков."""
3 from matplotlib import animation
4 import matplotlib.pyplot as plt
5 import random
6 import seaborn as sns
7 import sys
8
```

Функция update

В строках 9–27 определяется функция `update`, которую `FuncAnimation` вызывает по одному разу для каждого кадра анимации. Эта функция должна получать по крайней мере один аргумент. В строках 9–10 показано начало функции. Параметры:

- `frame_number` — следующее значение из аргумента `frames` функции `FuncAnimation`, который вскоре будет рассмотрен более подробно. И хотя `FuncAnimation` требует, чтобы функция `update` имела этот параметр, в данной функции `update` он не используется;
- `rolls` — количество бросков кубиков на кадр анимации;
- `faces` — обозначения граней, используемые как метки на оси *x* диаграммы;
- `frequencies` — список, в котором обобщаются частоты выпадения результатов.

Остальной код тела функции рассматривается в нескольких ближайших подразделах.

```
 9 def update(frame_number, rolls, faces, frequencies):
10     """Настраивает содержимое диаграммы для каждого кадра анимации."""
```

Функция update: бросок кубика и обновление списка frequencies

В строках 12–13 моделируются `rolls` бросков кубика, и соответствующий элемент `frequencies` увеличивается для каждого броска. Обратите внимание на вычитание 1 из результата на грани кубика (от 1 до 6) перед увеличением

соответствующего элемента `frequencies` — как вы вскоре увидите, `frequencies` является списком из шести элементов (см. определение в строке 36), поэтому его индексы лежат в диапазоне от 0 до 5:

```
11      # Бросок кубика и обновление частот
12      for i in range(rolls):
13          frequencies[random.randrange(1, 7) - 1] += 1
14
```

Функция update: настройка гистограммы и текста

Строка 16 функции `update` вызывает функцию `cla` модуля `matplotlib.pyplot` для удаления существующих элементов гистограммы, перед тем как рисовать новые элементы для текущего кадра анимации. Код в строках 17–27 обсуждался в разделе «Введение в data science» предыдущего раздела. Строки 17–20 создают столбцы, назначают заголовок диаграммы, назначают метки осей *x* и *y* и масштабируют график, чтобы освободить место для текста со значениями и частотами над каждым столбцом. В строках 23–27 выводится текст с частотами и процентами.

```
15      # Настройка диаграммы для обновленных частот
16      plt.cla()  # Очистка старого содержимого текущей диаграммы
17      axes = sns.barplot(faces, frequencies, palette='bright')  # Новые столбцы
18      axes.set_title(f'Die Frequencies for {sum(frequencies):,} Rolls')
19      axes.set(xlabel='Die Value', ylabel='Frequency')
20      axes.set_ylim(top=max(frequencies) * 1.10)  # Масштабирование оси y на 10%
21
22      # Вывод частоты и процента над каждым столбцом
23      for bar, frequency in zip(axes.patches, frequencies):
24          text_x = bar.get_x() + bar.get_width() / 2.0
25          text_y = bar.get_height()
26          text = f'{frequency:,}\n{frequency / sum(frequencies):.3%}'
27          axes.text(text_x, text_y, text, ha='center', va='bottom')
28
```

Переменные для настройки диаграммы и хранения состояния

В строках 30 и 31 список `argv` модуля `sys` используется для получения аргументов командной строки. В строке 33 задается стиль Seaborn `'whitegrid'`. В строке 34 вызывается функция `figure` модуля `matplotlib.pyplot` для получения объекта `Figure`, на котором `FuncAnimation` отображает анимацию. Аргумент функции содержит заголовок окна. Как вы вскоре увидите, это один из обязательных аргументов `FuncAnimation`. В строке 35 создается список, содержащий обозначения граней 1–6 для отображения на оси *x* диаграммы.

Строка 36 создает список `frequencies` из шести элементов, инициализированных нулями, — счетчики из этого списка обновляются при каждом броске.

```
29 # Получение аргументов командной строки для количества кадров и бросков на кадр
30 number_of_frames = int(sys.argv[1])
31 rolls_per_frame = int(sys.argv[2])
32
33 sns.set_style('whitegrid')  # Белый фон с серыми линиями
34 figure = plt.figure('Rolling a Six-Sided Die')  # Рисунок для анимации
35 values = list(range(1, 7))  # Обозначения граней для вывода на оси x
36 frequencies = [0] * 6  # Список частот из шести элементов
37
```

Вызов функции FuncAnimation модуля animation

В строках 39–41 функция `FuncAnimation` модуля `animation` библиотеки Matplotlib вызывается для динамического обновления гистограммы. Функция возвращает объект, представляющий анимацию. Хотя мы не используем этот объект явно, ссылку на анимацию *необходимо* сохранить; в противном случае Python немедленно завершит анимацию и вернет ее память в систему.

```
38 # Настройка и запуск анимации, вызывающей функцию update
39 die_animation = animation.FuncAnimation(
40     figure, update, repeat=False, frames=number_of_frames, interval=33,
41     fargs=(rolls_per_frame, values, frequencies))
42
43 plt.show()  # Вывод окна
```

`FuncAnimation` имеет два обязательных аргумента:

- `figure` — объект `Figure` для отображения анимации;
- `update` — функция, вызываемая по одному разу для каждого кадра анимации.

В данном примере также передаются необязательные ключевые аргументы:

- `repeat` — `False` завершает анимацию после заданного количества кадров. Если аргумент равен `True` (по умолчанию), то при завершении анимация начинается заново;
- `frames` — общее количество кадров анимации, управляющее количеством вызовов `update` из `FuncAnimation`. Передача целого числа эквивалентна передаче `range` — например, 600 означает `range(600)`. `FuncAnimation` передает одно значение из диапазона в первом аргументе каждого вызова `update`;

- `interval` — промежуток в миллисекундах между кадрами анимации (`33` в данном случае); значение по умолчанию равно 200. После каждого вызова `update` функция `FuncAnimation` ожидает 33 миллисекунды перед следующим вызовом;
- `fargs` (сокращение от «function arguments», то есть «аргументы функции») — кортеж других аргументов для передачи функции, заданной вторым аргументом `FuncAnimation`. Аргументы, заданные в кортеже `fargs`, соответствуют параметрам `rolls`, `faces` и `frequencies` функции `update` (строка 9).

За информацией о других необязательных аргументах `FuncAnimation` обращайтесь по адресу:

https://matplotlib.org/api/_as_gen/matplotlib.animation.FuncAnimation.html

Наконец, строка 43 выводит окно.

6.5. Итоги

В этой главе рассматривались коллекции Python: словари и множества. Мы рассказали, что собой представляют словари, и рассмотрели несколько примеров. Мы продемонстрировали синтаксис пар «ключ-значение» и показали, как использовать их для создания словарей, содержащих списки разделенных запятыми пар «ключ-значение» в фигурных скобках `{}`. Также мы создавали словари при помощи трансформаций словарей.

Квадратные скобки `[]` используются для получения значения, соответствующего ключу, а также для вставки и обновления пар «ключ-значение». Также мы использовали метод словарей `update` для изменения значения, связанного с ключом, и перебирали ключи, значения и элементы словаря.

Мы создавали множества, содержащие уникальные неизменяемые значения, сравнивали множества операторами сравнения, объединяли множества операторами и методами множеств, изменяли значения в множествах операциями изменяемых множеств и создавали множества с использованием трансформаций множеств. Вы узнали, что множества являются изменяемыми. Фиксированные множества неизменяемы, поэтому они могут использоваться как элементы множеств и фиксированных множеств.

В разделе «Введение в data science» продолжилось ваше знакомство с визуализациями. Мы рассмотрели моделирование бросков кубиков с построением

динамической гистограммы, наглядно демонстрирующей закон больших чисел. В дополнение к средствам Seaborn и Matplotlib, представленным в предыдущей главе «Введение в data science», мы воспользовались функцией `FuncAnimation` библиотеки Matplotlib для управления покадровой анимацией. Функция `FuncAnimation` вызывала определенную нами функцию, которая указывала, что нужно вывести в каждом кадре анимации.

Темой следующей главы станет программирование, ориентированное на массивы, на базе популярной библиотеки NumPy. Как вы увидите, коллекции `ndarray` библиотеки NumPy могут работать на два порядка быстрее, чем при выполнении тех же операций со встроенными списками Python. Эта производительность будет чрезвычайно полезной для современных приложений, работающих с большими данными.

7

NumPy и программирование, ориентированное на массивы

В этой главе...

- Отличия между массивами и списками.
- Использование высокопроизводительных коллекций `ndarray` модуля `numpy`.
- Сравнение производительности списков и `ndarray` с использованием IPython.
- Магическая команда `%timeit`.
- Использование коллекций `ndarray` для эффективного хранения и загрузки данных.
- Создание и инициализация коллекций `ndarray`.
- Обращение к отдельным элементам `ndarray`.
- Перебор `ndarray`.
- Создание многомерных коллекций `ndarray` и работа с ними.
- Выполнение стандартных операций с `ndarray`.
- Создание и выполнение операций с одномерными коллекциями `Series` и двумерными коллекциями `DataFrame`.
- Настройка индексов `Series` и `DataFrame`.
- Вычисление базовых характеристик описательной статистики для данных в `Series` и `DataFrame`.
- Настройка точности чисел с плавающей точкой при форматировании вывода `pandas`.

7.1. Введение

Библиотека *NumPy* (*Numerical Python*), впервые появившаяся в 2006 году, считается основной реализацией массивов Python. Она предоставляет высокопроизводительный, полнофункциональный тип *n*-мерного массива, который называется `ndarray` (в дальнейшем мы будем называть его синонимом `array`). NumPy — одна из многих библиотек с открытым кодом, устанавливаемых в дистрибутиве Anaconda Python. Операции с `array` выполняются на два порядка быстрее, чем операции со списками. В мире больших данных, в которых приложениям приходится выполнять серьезную обработку огромных объемов данных на базе массивов, этот прирост быстродействия может оказаться критичным. Согласно `libraries.io`, свыше 450 библиотек Python зависят от NumPy. Многие популярные библиотеки data science, такие как pandas, SciPy (Scientific Python) и Keras (глубокое обучение), построены на базе NumPy или зависят от последней.

В этой главе исследуются базовые возможности `array`. Как известно, списки могут быть многомерными. Обычно многомерные списки обрабатываются во вложенных циклах или с использованием трансформаций списков с несколькими секциями `for`. Сильной стороной NumPy является «программирование, ориентированное на массивы», использующее программирование в функциональном стиле с *внутренними* итерациями; работа с массивами становится компактной и прямолинейной, а код избавляется от ошибок, которые могут возникнуть во *внешних* итерациях или в явно запрограммированных циклах.

В разделе «Введение в data science» этой главы начнется ваше знакомство с библиотекой *pandas*, которая будет использоваться во многих практических примерах при изучении data science. В приложениях больших данных часто возникает необходимость в коллекциях более гибких, чем массивы NumPy, — коллекциях с поддержкой смешанных типов данных, нестандартного индексирования, отсутствующих данных, данных с нарушенной структурой и данных, которые должны быть приведены к форме, более подходящей для баз данных и пакетов анализа данных, с которыми вы работаете. Мы также представим структуры данных pandas, сходные с массивами, — одномерные коллекции `Series` и двумерные `DataFrame`, а затем продемонстрируем некоторые из их выдающихся возможностей. После прочтения этой главы в вашем арсенале будут уже четыре коллекции, сходные с массивами, — списки, `array`, `Series` и `DataFrame`. О пятой разновидности — тензорах — речь пойдет в главе 15.

7.2. Создание массивов на основе существующих данных

Документация NumPy рекомендует импортировать *модуль* `numpy` под именем `np`, чтобы к его компонентам можно было обращаться с префиксом `"np."`:

```
In [1]: import numpy as np
```

Модуль `numpy` предоставляет различные функции для создания массивов. В данном случае будет использоваться функция `array`, которая получает в аргументе массив или другую коллекцию и возвращает новый массив с элементами своего аргумента. Передадим при вызове список:

```
In [2]: numbers = np.array([2, 3, 5, 7, 11])
```

Функция `array` копирует содержимое своего аргумента в массив. Проверим тип объекта, возвращенного функцией `array`, и выведем его содержимое:

```
In [3]: type(numbers)
Out[3]: numpy.ndarray

In [4]: numbers
Out[4]: array([ 2,  3,  5,  7, 11])
```

Заметим, что для объекта указан *тип* `numpy.ndarray`, но при выводе массива используется обозначение «`array`». При выводе `array` NumPy отделяет каждое значение от следующего запятой и пробелом и выравнивает все значения *по правому краю* поля постоянной ширины. Ширина поля определяется на основании значения, занимающего *наибольшее* количество знаков при выводе. В данном случае значение 11 занимает два знака, поэтому все значения форматируются по полям из двух символов (поэтому символы `[` и `2` отделены друг от друга начальным пробелом).

Многомерные аргументы

Функция `array` копирует размерности своего аргумента. Создадим объект `array` на основе списка из двух строк и трех столбцов:

```
In [5]: np.array([[1, 2, 3], [4, 5, 6]])
Out[5]:
array([[1, 2, 3],
       [4, 5, 6]])
```

NumPy автоматически форматирует `array` на основании количества их измерений и выравнивает столбцы в каждой строке.

7.3. Атрибуты array

Объект `array` предоставляет *атрибуты* для получения информации об их структуре и содержимом. В этом разделе будут использоваться следующие объекты `array`:

```
In [1]: import numpy as np

In [2]: integers = np.array([[1, 2, 3], [4, 5, 6]])

In [3]: integers
Out[3]:
array([[1, 2, 3],
       [4, 5, 6]])

In [4]: floats = np.array([0.0, 0.1, 0.2, 0.3, 0.4])

In [5]: floats
Out[5]: array([ 0. ,  0.1,  0.2,  0.3,  0.4])
```

NumPy не выводит завершающие нули в дробной части в значениях с плавающей точкой.

Определение типа элементов array

Функция `array` определяет тип элемента `array` на основании элементов ее аргументов. Для проверки типа элементов можно воспользоваться атрибутом `dtype` типа `array`:

```
In [6]: integers.dtype
Out[6]: dtype('int64')  # int32 on some platforms

In [7]: floats.dtype
Out[7]: dtype('float64')
```

Как будет показано в следующем разделе, различные функции создания `array` получают ключевой аргумент `dtype` для определения типа элементов `array`.

Ради высокого быстродействия библиотека NumPy написана на языке программирования C и в ней используются типы данных C. По умолчанию NumPy сохраняет целые числа в виде значений типа `int64` библиотеки NumPy, соот-

ветствующих 64-разрядным (8-байтовым) целым числам C, а числа с плавающей точкой — в виде значений типа `float64` библиотеки NumPy. В наших примерах чаще всего будут встречаться типы `int64`, `float64`, `bool` (логический тип) и `object` для нечисловых данных (например, строк). Полный список поддерживаемых типов приведен по адресу https://docs.scipy.org/doc/numpy/user/basics.types.html.

Определение размерности array

Атрибут `ndim` содержит количество измерений `array`, а атрибут `shape` содержит *кортеж*, определяющий размерность `array`:

```
In [8]: integers.ndim
Out[8]: 2

In [9]: floats.ndim
Out[9]: 1

In [10]: integers.shape
Out[10]: (2, 3)

In [11]: floats.shape
Out[11]: (5,)
```

Здесь `integers` состоит из двух строк и трех столбцов (6 элементов), а структура данных `floats` является одномерной, поэтому фрагмент [11] выводит кортеж из одного элемента (на что указывает запятая) с количеством элементов `floats` (5).

Определение количества элементов и размера элементов

Общее количество элементов в `array` можно получить из атрибута `size`, а количество байтов, необходимое для хранения каждого элемента, — из атрибута `itemsize`:

```
In [12]: integers.size
Out[12]: 6

In [13]: integers.itemsize  # 4 для компиляторов C с 32-разрядными int
Out[13]: 8

In [14]: floats.size
Out[14]: 5

In [15]: floats.itemsize
Out[15]: 8
```

Обратите внимание: размер `integers` равен произведению значений из кортежа `shape` — две строки по три элемента, итого шесть элементов. В каждом случае значение `itemsize` равно 8, потому что `integers` содержит значения `int64`, а `floats` содержит значения `float64`; каждое занимает 8 байт.

Перебор элементов многомерной коллекции array

Обычно для работы с `array` используются компактные конструкции программирования в функциональном стиле. Так как `array` являются *итерируемыми объектами*, при желании можно использовать внешние итерации:

```
In [16]: for row in integers:
    ...:     for column in row:
    ...:         print(column, end='  ')
    ...:     print()
    ...:
1  2  3
4  5  6
```

Чтобы перебрать элементы многомерной коллекции `array` так, как если бы она была одномерной, используйте атрибут `flat`:

```
In [17]: for i in integers.flat:
    ...:     print(i, end='  ')
    ...:
1  2  3  4  5  6
```

7.4. Заполнение array конкретными значениями

NumPy предоставляет функции `zeros`, `ones` и `full` для создания коллекций `array`, содержащих 0, 1 или заданное значение соответственно. По умолчанию функции `zeros` и `ones` создают коллекции `array`, содержащие значения `float64`. Вскоре мы покажем, как настроить тип элементов. Первым аргументом этих функций должно быть целое число или кортеж целых чисел, определяющий нужные размеры. Для целого числа каждая функция возвращает одномерную коллекцию `array` с заданным количеством элементов:

```
In [1]: import numpy as np

In [2]: np.zeros(5)
Out[2]: array([ 0.,  0.,  0.,  0.,  0.])
```

Для кортежа целых чисел эти функции возвращают многомерную коллекцию `array` с заданными размерами. При вызове функций `zeros` и `ones` можно задать тип элементов при помощи ключевого аргумента `dtype`:

```
In [3]: np.ones((2, 4), dtype=int)
Out[3]:
array([[1, 1, 1, 1],
       [1, 1, 1, 1]])
```

Коллекция `array`, возвращаемая `full`, содержит элементы со значением и типом второго аргумента:

```
In [4]: np.full((3, 5), 13)
Out[4]:
array([[13, 13, 13, 13, 13],
       [13, 13, 13, 13, 13],
       [13, 13, 13, 13, 13]])
```

7.5. Создание коллекций array по диапазонам

NumPy предоставляет оптимизированные функции для создания коллекций `array` на базе диапазонов. Мы ограничимся простыми равномерными диапазонами целых чисел и чисел с плавающей точкой, помня, впрочем, что NumPy поддерживает и нелинейные диапазоны[1].

Создание диапазонов функцией arange

Воспользуемся функцией `arange` библиотеки NumPy для создания целочисленных диапазонов — по аналогии со встроенной функцией `range`. Функция `arange` сначала определяет количество элементов в полученной коллекции `array`, выделяет память, а затем сохраняет заданный диапазон значений в `array`:

```
In [1]: import numpy as np

In [2]: np.arange(5)
Out[2]: array([0, 1, 2, 3, 4])

In [3]: np.arange(5, 10)
Out[3]: array([5, 6, 7, 8, 9])

In [4]: np.arange(10, 1, -2)
Out[4]: array([10,  8,  6,  4,  2])
```

[1] https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html.

И хотя при создании `array` можно передавать в аргументах `range`, всегда используйте функцию `arange`, так как она оптимизирована для `array`. Скоро мы покажем, как определить время выполнения различных операций, чтобы сравнить их быстродействие.

Создание диапазонов чисел с плавающей точкой функцией linspace

Для создания равномерно распределенных диапазонов чисел с плавающей точкой можно воспользоваться функцией `linspace` библиотеки NumPy. Первые два аргумента функции определяют начальное и конечное значение диапазона, при этом конечное значение *включается* в `array`. Необязательный ключевой аргумент `num` задает количество равномерно распределенных генерируемых значений (по умолчанию используется значение 50):

```
In [5]: np.linspace(0.0, 1.0, num=5)
Out[5]: array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

Изменение размерности array

Коллекцию `array` также можно создать на базе диапазона элементов, а затем воспользоваться методом `reshape` для преобразования одномерной коллекции `array` в многомерную. Создадим коллекцию `array` со значениями от 1 до 20, а затем преобразуем ее к двумерной структуре из четырех строк и пяти столбцов:

```
In [6]: np.arange(1, 21).reshape(4, 5)
Out[6]:
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
```

Обратите внимание на *сцепленные вызовы методов* в приведенном фрагменте. Сначала `arange` создает коллекцию `array` со значениями 1–20. После этого вызов `reshape` для полученной коллекции `array` создает коллекцию 4 × 5, показанную выше.

Размерность можно изменить для любой коллекции `array` — при условии что по количеству элементов новая версия не отличается от оригинала. Таким образом, одномерная коллекция `array` из шести элементов может превратиться в коллекцию 3 × 2 или 2 × 3, и наоборот, но попытка преобразовать коллекцию `array` из 15 элементов в коллекцию 4 × 4 (16 элементов) приводит к ошибке `ValueError`.

Вывод больших коллекций array

При выводе коллекции `array`, содержащей 1000 и более элементов, NumPy исключает из вывода строки и/или столбцы в середине. Следующие фрагменты генерируют 100 000 элементов. В первом примере показаны все четыре строки, каждая из 25 000 столбцов (знак многоточия ... представляет отсутствующие данные).

```
In [7]: np.arange(1, 100001).reshape(4, 25000)
Out[7]:
array([[     1,      2,      3, ...,  24998,  24999,  25000],
       [ 25001,  25002,  25003, ...,  49998,  49999,  50000],
       [ 50001,  50002,  50003, ...,  74998,  74999,  75000],
       [ 75001,  75002,  75003, ...,  99998,  99999, 100000]])
```

Во втором примере приведены три первых и три последних строки (из 100 строк), причем каждая из шести насчитывает 1000 столбцов:

```
In [8]: np.arange(1, 100001).reshape(100, 1000)
Out[8]:
array([[     1,      2,      3, ...,    998,    999,   1000],
       [  1001,   1002,   1003, ...,   1998,   1999,   2000],
       [  2001,   2002,   2003, ...,   2998,   2999,   3000],
       ...,
       [ 97001,  97002,  97003, ...,  97998,  97999,  98000],
       [ 98001,  98002,  98003, ...,  98998,  98999,  99000],
       [ 99001,  99002,  99003, ...,  99998,  99999, 100000]])
```

7.6. Сравнение быстродействия списков и array

Многие операции с коллекциями `array` выполняются *намного* быстрее, чем соответствующие операции со списками. Для демонстрации мы воспользуемся *магической* командой IPython `%timeit`, измеряющей *среднюю* продолжительность операций. Учтите, что время в вашей системе может отличаться от наших результатов.

Хронометраж создания списка с результатами 6 000 000 бросков кубика

Ранее мы показали, как смоделировать 6 000 000 бросков шестигранного кубика. Теперь воспользуемся функцией `randrange` модуля `random` с трансформацией списка, чтобы создать список с результатами 6 000 000 бросков и провести хронометраж операции командой `%timeit`. Обратите внимание на

использование символа продолжения строки (\) для разбиения команды из фрагмента [2] на две строки:

```
In [1]: import random

In [2]: %timeit rolls_list = \
   ...:    [random.randrange(1, 7) for i in range(0, 6_000_000)]
6.29 s ± 119 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

По умолчанию `%timeit` выполняет команду в цикле, который прогоняется *семь* раз. Если количество итераций не задано, то `%timeit` выбирает подходящее значение. В нашем тестировании операции, занимавшие более 500 миллисекунд, выполнялись всего один раз, тогда как операции, занимавшие менее 500 миллисекунд, повторялись 10 раз и более.

После выполнения команды `%timeit` выводит *среднее* время ее выполнения, а также стандартное отклонение по всем выполнениям. В среднем `%timeit` указывает, что создание списка заняло 6,29 секунды со стандартным отклонением 119 миллисекунд (мс). В сумме семикратное выполнение фрагмента заняло около 44 секунд.

Хронометраж создания коллекции array с результатами 6 000 000 бросков

Теперь воспользуемся *функцией* `randint` из *модуля* `numpy.random` для создания коллекции `array` с 6 000 000 бросков:

```
In [3]: import numpy as np

In [4]: %timeit rolls_array = np.random.randint(1, 7, 6_000_000)
72.4 ms ± 635 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

В среднем `%timeit` показывает, что создание `array` заняло всего 72,4 *миллисекунды* со стандартным отклонением 635 микросекунд (мкс). В сумме выполнение предыдущего фрагмента заняло на нашем компьютере менее половины секунды — около 1/100 от времени выполнения фрагмента [2]. Таким образом, с `array` операция действительно выполняется *на два порядка быстрее*!

60 000 000 и 600 000 000 бросков

Теперь создадим `array` с результатами 60 000 000 бросков:

```
In [5]: %timeit rolls_array = np.random.randint(1, 7, 60_000_000)
873 ms ± 29.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

В среднем создание `array` заняло всего 873 миллисекунды.

Теперь проведем моделирование 600 000 000 бросков:

```
In [6]: %timeit rolls_array = np.random.randint(1, 7, 600_000_000)
10.1 s ± 232 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Создание 600 000 000 элементов средствами NumPy заняло лишь около 10 секунд — сравните с 6 секундами, потраченными на создание всего 6 000 000 элементов с трансформацией списка.

Эти данные наглядно показывают, почему для операций, связанных с интенсивными вычислениями, обычно отдается предпочтение коллекциям `array` перед списками. В своих практических примерах data science мы войдем в мир больших данных и искусственного интеллекта с его высокими требованиями к производительности. Вы узнаете, как сочетание современного оборудования, программных продуктов, коммуникаций и структур алгоритмов позволяет справиться с колоссальными требованиями современных приложений к вычислительным мощностям.

Настройка итераций %timeit

Количество итераций в каждом цикле `%timeit` и количество циклов настраиваются при помощи параметров `-n` и `-r`. В следующем примере команда из фрагмента [4] выполняется три раза в каждом цикле, а сам цикл выполняется дважды[1]:

```
In [7]: %timeit -n3 -r2 rolls_array = np.random.randint(1, 7, 6_000_000)
85.5 ms ± 5.32 ms per loop (mean ± std. dev. of 2 runs, 3 loops each)
```

Другие магические команды IPython

IPython предоставляет десятки магических команд для самых разнообразных задач — за полным списком обращайтесь к документации IPython[2]. Ниже перечислены самые полезные команды:

- `%load` — загружает в IPython код из локального файла или по URL-адресу;
- `%save` — сохраняет фрагменты в файле;
- `%run` — выполняет файл .py из IPython;

---

[1] Для большинства читателей настроек `%timeit` по умолчанию будет достаточно.

[2] http://ipython.readthedocs.io/en/stable/interactive/magics.html.

- `%precision` — изменяет точность чисел с плавающей точкой по умолчанию для вывода IPython;
- `%cd` — позволяет изменить текущий каталог без выхода из IPython;
- `%edit` — запускает внешний редактор; может пригодиться для редактирования более сложных фрагментов;
- `%history` — выводит список всех фрагментов и команд, выполненных в текущем сеансе IPython.

7.7. Операторы array

NumPy предоставляет обширный набор операторов, позволяющих писать простые выражения для выполнения операций с целыми коллекциями `array`. В этом разделе продемонстрированы математические операции между коллекциями `array` и числовыми значениями, а также между коллекциями `array` одинакового размера.

Арифметические операции с array и числовыми значениями

Начнем с *поэлементных арифметических операций* с `array` и числовыми значениями, использующих арифметические операторы и расширенное присваивание. Поэлементные операции применяются к каждому элементу, так что фрагмент [4] умножает каждый элемент на `2`, а фрагмент [5] возводит каждый элемент в куб. В каждом случае возвращается *новая* коллекция `array`, содержащая результат:

```
In [1]: import numpy as np

In [2]: numbers = np.arange(1, 6)

In [3]: numbers
Out[3]: array([1, 2, 3, 4, 5])

In [4]: numbers * 2
Out[4]: array([ 2,  4,  6,  8, 10])

In [5]: numbers ** 3
Out[5]: array([  1,   8,  27,  64, 125])

In [6]: numbers  # numbers не изменяется арифметическими операторами
Out[6]: array([1, 2, 3, 4, 5])
```

Фрагмент [6] показывает, что арифметические операторы не изменили `numbers`. Операторы `+` и `*` *коммутативны*, поэтому фрагмент [4] также можно было записать в виде `2 * numbers`.

Расширенные присваивания *изменяют* каждый элемент левого операнда.

```
In [7]: numbers += 10

In [8]: numbers
Out[8]: array([11, 12, 13, 14, 15])
```

Распространение

Обычно операндами арифметических операций должны быть две коллекции `array` *с одинаковыми размерами*. Если один операнд представляет собой отдельное значение (называемое *скалярным значением*), то NumPy выполняет поэлементные вычисления так, словно скалярное значение является коллекцией `array` с тем же размером, что и у другого операнда, но содержащей скалярное значение во всех элементах. Это называется *распространением* (broadcasting). Данная возможность используется во фрагментах [4], [5] и [7]. Например, фрагмент [4] эквивалентен следующей команде:

```
numbers * [2, 2, 2, 2, 2]
```

Распространение также может применяться между коллекциями `array` разных размеров, что позволяет выполнять мощные операции в компактном виде. Другие примеры распространения будут приведены позднее, когда мы будем рассматривать универсальные функции NumPy.

Арифметические операции между коллекциями array

С коллекциями `array`, имеющими *одинаковые* размеры, можно выполнять арифметические операции и расширенные присваивания. Перемножим одномерные коллекции `array` `numbers` и `numbers2` (созданные ниже), каждая из которых содержит пять элементов:

```
In [9]: numbers2 = np.linspace(1.1, 5.5, 5)

In [10]: numbers2
Out[10]: array([ 1.1,  2.2,  3.3,  4.4,  5.5])

In [11]: numbers * numbers2
Out[11]: array([ 12.1,  26.4,  42.9,  61.6,  82.5])
```

Результат представляет собой новый объект `array`, полученный *поэлементным перемножением* обоих операндов — `11 * 1.1`, `12 * 2.2`, `13 * 3.3` и т. д. Результаты выполнения арифметических операций между коллекциями `array` с целыми числами и числами с плавающей точкой — коллекция чисел с плавающей точкой.

Сравнение коллекций array

Коллекции `array` можно сравнивать как с отдельными значениями, так и с другими коллекциями `array`. Сравнения выполняются *поэлементно*. В результате таких сравнений создаются коллекции `array` с логическими значениями, каждое из которых обозначает результат сравнения соответствующих элементов:

```
In [12]: numbers
Out[12]: array([11, 12, 13, 14, 15])

In [13]: numbers >= 13
Out[13]: array([False, False,  True,  True,  True])

In [14]: numbers2
Out[14]: array([ 1.1,  2.2,  3.3,  4.4,  5.5])

In [15]: numbers2 < numbers
Out[15]: array([ True,  True,  True,  True,  True])

In [16]: numbers == numbers2
Out[16]: array([False, False, False, False, False])

In [17]: numbers == numbers
Out[17]: array([ True,  True,  True,  True,  True])
```

Фрагмент [13] использует распространение для проверки того, что каждый элемент `numbers` больше или равен 13. Остальные фрагменты сравнивают соответствующие элементы своих операндов `array`.

7.8. Вычислительные методы NumPy

Коллекция `array` содержит различные методы для выполнения вычислений. По умолчанию эти методы игнорируют размеры `array` и используют в вычислениях *все* элементы. Например, при вычислении среднего значения для `array` суммируются все элементы независимо от размера, после чего сумма делится на общее количество элементов. Эти вычисления можно выполнять

и с отдельными измерениями. Например, в двумерной коллекции `array` можно вычислить среднее значение по каждой строке и по каждому столбцу.

Возьмем коллекцию `array`, представляющую оценки четырех студентов на трех экзаменах:

```
In [1]: import numpy as np

In [2]: grades = np.array([[87, 96, 70], [100, 87, 90],
   ...:                    [94, 77, 90], [100, 81, 82]])
   ...:

In [3]: grades
Out[3]:
array([[ 87,  96,  70],
       [100,  87,  90],
       [ 94,  77,  90],
       [100,  81,  82]])
```

При помощи различных методов можно вычислить сумму (`sum`), наименьшее (`min`) и наибольшее (`max`) значение, математическое ожидание (`mean`), стандартное отклонение (`std`) и дисперсию (`var`) — каждая подобная операция в контексте программирования в функциональном стиле является *сверткой*:

```
In [4]: grades.sum()
Out[4]: 1054

In [5]: grades.min()
Out[5]: 70

In [6]: grades.max()
Out[6]: 100

In [7]: grades.mean()
Out[7]: 87.83333333333333

In [8]: grades.std()
Out[8]: 8.792357792739987

In [9]: grades.var()
Out[9]: 77.30555555555556
```

Вычисления по строкам или по столбцам

Многие вычислительные методы также могут применяться к конкретным размерностям `array` (они называются *осями* `array`). Такие методы получают ключевой аргумент `axis`, определяющий размерность, используемую в вы-

числениях; это позволяет быстро проводить вычисления по строкам или по столбцам в двумерной коллекции `array`.

Допустим, вы хотите вычислить среднюю оценку по каждому *экзамену*, представленному одним из столбцов. Аргумент `axis=0` выполняет вычисления по всем значениям *строк* внутри каждого столбца:

```
In [10]: grades.mean(axis=0)
Out[10]: array([95.25, 85.25, 83.  ])
```

Значение `95.25` представляет собой среднее значение для оценок первого столбца (`87`, `100`, `94` и `100`), `85.25` — среднее значение для оценок второго столбца (`96`, `87`, `77` и `81`), а `83` — среднее значение для оценок третьего столбца (`70`, `90`, `90` и `82`). Как и прежде, NumPy *не* отображает завершающие нули в дробной части: `'83.'`. Также стоит заметить, что все значения элементов выводятся в полях постоянной ширины, именно поэтому за `'83.'` следуют два пробела.

С аргументом `axis=1` вычисления будут выполняться со всеми значениями *столбца* внутри каждой отдельной строки. Так, для вычисления средней оценки каждого студента по всем экзаменам можно использовать следующую команду:

```
In [11]: grades.mean(axis=1)
Out[11]: array([84.33333333, 92.33333333, 87.        , 87.66666667])
```

Этот фрагмент вычисляет четыре средних значения — по одному для каждой строки. Таким образом, `84.33333333` представляет собой среднее значение для оценок строки 0 (`87`, `96` и `70`), и так далее для остальных строк.

Коллекции `array` библиотеки NumPy поддерживают много других вычислительных методов. За полным списком обращайтесь по адресу:

https://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html

7.9. Универсальные функции

NumPy предоставляет десятки автономных *универсальных функций* для выполнения различных поэлементных операций. Каждая функция выполняет свою задачу с использованием одного или двух аргументов, которыми могут быть коллекции `array` или аналогичные структуры (например, списки). Некоторые из этих функций вызываются при применении к `array` таких операторов, как `+` и `*`. Каждая функция возвращает новую коллекцию `array` с результатами.

Создадим коллекцию `array` и вычислим квадратный корень всех ее значений при помощи *универсальной функции* `sqrt`:

```
In [1]: import numpy as np

In [2]: numbers = np.array([1, 4, 9, 16, 25, 36])

In [3]: np.sqrt(numbers)
Out[3]: array([1., 2., 3., 4., 5., 6.])
```

А теперь просуммируем две коллекции `array` одинакового размера при помощи *универсальной функции* `add`:

```
In [4]: numbers2 = np.arange(1, 7) * 10

In [5]: numbers2
Out[5]: array([10, 20, 30, 40, 50, 60])

In [6]: np.add(numbers, numbers2)
Out[6]: array([11, 24, 39, 56, 75, 96])
```

Выражение `np.add(numbers, numbers2)` эквивалентно выражению

```
numbers + numbers2
```

Распространение с универсальными функциями

Воспользуемся *универсальной функцией* `multiply` для умножения каждого элемента `numbers2` на скалярное значение `5`:

```
In [7]: np.multiply(numbers2, 5)
Out[7]: array([ 50, 100, 150, 200, 250, 300])
```

Выражение `np.multiply(numbers2, 5)` эквивалентно выражению

```
numbers2 * 5
```

Преобразуем `numbers2` в коллекцию `array` 2×3, а затем умножим ее значения на одномерную коллекцию `array` из трех элементов:

```
In [8]: numbers3 = numbers2.reshape(2, 3)

In [9]: numbers3
Out[9]:
array([[10, 20, 30],
       [40, 50, 60]])
```

```
In [10]: numbers4 = np.array([2, 4, 6])

In [11]: np.multiply(numbers3, numbers4)
Out[11]:
array([[ 20,  80, 180],
       [ 80, 200, 360]])
```

Это решение работает, потому что длина `numbers4` равна длине каждой строки `numbers3`, что позволяет NumPy применить операцию умножения; `numbers4` интерпретируется так, как если бы это была следующая коллекция `array`:

```
array([[2, 4, 6],
       [2, 4, 6]])
```

Если универсальная функция получает две коллекции `array` разного размера, которые не поддерживают распространения, то происходит ошибка `ValueError`. С правилами распространения можно ознакомиться по адресу:

https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

Другие универсальные функции

В документации NumPy универсальные функции разделены на пять категорий — математические, тригонометрические, поразрядные, функции сравнения и функции с плавающей точкой. В табл. 7.1 перечислены некоторые функции из каждой категории. Полный список вместе с описаниями и более подробной информацией об универсальных функциях доступен по адресу:

https://docs.scipy.org/doc/numpy/reference/ufuncs.html

Таблица 7.1. Универсальные функции NumPy

Математические — `add`, `subtract`, `multiply`, `divide`, `remainder`, `exp`, `log`, `sqrt`, `power` и т. д.
Тригонометрические — `sin`, `cos`, `tan`, `hypot`, `arcsin`, `arccos`, `arctan` и т. д.
Поразрядные — `bitwise_and`, `bitwise_or`, `bitwise_xor`, `invert`, `left_shift` и `right_shift`
Функции сравнения — `greater`, `greater_equal`, `less`, `less_equal`, `equal`, `not_equal`, `logical_and`, `logical_or`, `logical_xor`, `logical_not`, `minimum`, `maximum` и т. д.
Функции с плавающей точкой — `floor`, `ceil`, `isinf`, `isnan`, `fabs`, `trunc` и т. д.

7.10. Индексирование и сегментация

К одномерным коллекциям `array` могут применяться операции индексирования и сегментации; при этом используются синтаксис и приемы, продемонстрированные в главе 5. В этом разделе мы сосредоточимся на средствах индексирования и сегментации, специфичных для `array`.

Индексирование с двумерными коллекциями array

Чтобы выбрать элемент двумерной коллекции `array`, укажите кортеж с индексами строки и столбца элемента в квадратных скобках (как во фрагменте [4]):

```
In [1]: import numpy as np

In [2]: grades = np.array([[87, 96, 70], [100, 87, 90],
   ...:                    [94, 77, 90], [100, 81, 82]])
   ...:

In [3]: grades
Out[3]:
array([[ 87,  96,  70],
       [100,  87,  90],
       [ 94,  77,  90],
       [100,  81,  82]])

In [4]: grades[0, 1]  # строка 0, столбец 1
Out[4]: 96
```

Выбор подмножества строк двумерной коллекции array

Чтобы выбрать одну строку, укажите только один индекс в квадратных скобках:

```
In [5]: grades[1]
Out[5]: array([100,  87,  90])
```

Для выбора нескольких смежных строк используется синтаксис сегмента:

```
In [6]: grades[0:2]
Out[6]:
array([[ 87,  96,  70],
       [100,  87,  90]])
```

Для выбора нескольких несмежных строк используется список индексов строк:

```
In [7]: grades[[1, 3]]
Out[7]:
array([[100,  87,  90],
       [100,  81,  82]])
```

Выбор подмножества столбцов двумерной коллекции array

Чтобы выбрать подмножество столбцов, укажите кортеж, который определяет выбираемые строки и столбцы. Каждым элементом может быть конкретный индекс, сегмент или список. Выберем элементы только первого столбца:

```
In [8]: grades[:, 0]
Out[8]: array([ 87, 100,  94, 100])
```

Ноль после запятой указывает, что выбирается только столбец 0. Двоеточие (`:`) перед запятой указывает, какие строки в этом столбце должны выбираться. В данном случае `:` является *сегментом*, представляющим *все* строки. Здесь также может использоваться номер конкретной строки, сегмент, представляющий подмножество строк, или список индексов конкретных строк, как во фрагментах [5]–[7].

Для выбора нескольких смежных столбцов используется синтаксис сегмента:

```
In [9]: grades[:, 1:3]
Out[9]:
array([[96, 70],
       [87, 90],
       [77, 90],
       [81, 82]])
```

Для выбора конкретных столбцов используется *список* индексов этих столбцов:

```
In [10]: grades[:, [0, 2]]
Out[10]:
array([[ 87,  70],
       [100,  90],
       [ 94,  90],
       [100,  82]])
```

7.11. Представления: поверхностное копирование

В предыдущей главе рассматривались *объекты представлений*, то есть объекты, которые «видят» данные в других объектах, но не располагают собственными

копиями этих данных. Таким образом, представления являются поверхностными копиями. Различные методы массивов и операции сегментации создают представления данных массива.

Метод `view` коллекций `array` возвращает *новый* объект `array`, содержащий *представление* данных исходной коллекции. Создадим коллекцию `array` и ее представление:

```
In [1]: import numpy as np

In [2]: numbers = np.arange(1, 6)

In [3]: numbers
Out[3]: array([1, 2, 3, 4, 5])

In [4]: numbers2 = numbers.view()

In [5]: numbers2
Out[5]: array([1, 2, 3, 4, 5])
```

При помощи встроенной функции `id` можно убедиться в том, что `numbers` и `numbers2` являются *разными* объектами:

```
In [6]: id(numbers)
Out[6]: 4462958592

In [7]: id(numbers2)
Out[7]: 4590846240
```

Чтобы убедиться в том, что `numbers2` представляет *те же* данные, что и `numbers`, изменим элемент в `numbers`, а затем выведем обе коллекции `array`:

```
In [8]: numbers[1] *= 10

In [9]: numbers2
Out[9]: array([ 1, 20,  3,  4,  5])

In [10]: numbers
Out[10]: array([ 1, 20,  3,  4,  5])
```

Аналогичным образом изменение значения в представлении также приводит к изменению этого значения в исходном массиве:

```
In [11]: numbers2[1] /= 10

In [12]: numbers
```

```
Out[12]: array([1, 2, 3, 4, 5])

In [13]: numbers2
Out[13]: array([1, 2, 3, 4, 5])
```

Представления сегментов

Сегменты также создают представления. Преобразуем `numbers2` в сегмент, который видит только первые три элемента `numbers`:

```
In [14]: numbers2 = numbers[0:3]

In [15]: numbers2
Out[15]: array([1, 2, 3])
```

И снова функция `id` поможет убедиться в том, что `numbers` и `numbers2` являются разными объектами:

```
In [16]: id(numbers)
Out[16]: 4462958592

In [17]: id(numbers2)
Out[17]: 4590848000
```

Чтобы убедиться в том, что `numbers2` является представлением *только* первых *трех* элементов, можно попытаться обратиться к элементу `numbers2[3]`; при этом происходит ошибка `IndexError`:

```
In [18]: numbers2[3]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-18-582053f52daa> in <module>()
----> 1 numbers2[3]

IndexError: index 3 is out of bounds for axis 0 with size 3
```

Изменим элемент, общий для обоих массивов, а затем выведем их. Результат подтверждает, что `numbers2` является представлением `numbers`:

```
In [19]: numbers[1] *= 20

In [20]: numbers
Out[20]: array([1, 2, 3, 4, 5])

In [21]: numbers2
Out[21]: array([ 1, 40,  3])
```

7.12. Глубокое копирование

Хотя представления являются отдельными объектами `array`, они экономят память за счет совместного использования данных из других коллекций `array`. Однако при общем доступе к изменяемым значениям иногда бывает необходимо создать *глубокую копию* с *независимыми* экземплярами исходных данных. Это особенно важно при многоядерном программировании, в котором разные части вашей программы могут попытаться одновременно изменить данные, что может привести к их повреждению.

Метод `copy` коллекций `array` возвращает новый объект `array` с *глубокой копией* данных объекта исходной коллекции `array`. Начнем с создания коллекции `array` и ее глубокого копирования:

```
In [1]: import numpy as np

In [2]: numbers = np.arange(1, 6)

In [3]: numbers
Out[3]: array([1, 2, 3, 4, 5])

In [4]: numbers2 = numbers.copy()

In [5]: numbers2
Out[5]: array([1, 2, 3, 4, 5])
```

Чтобы доказать, что `numbers2` содержит отдельную копию данных в `numbers`, изменим элемент в `numbers`, после чего выведем оба массива:

```
In [6]: numbers[1] *= 10

In [7]: numbers
Out[7]: array([ 1, 20,  3,  4,  5])

In [8]: numbers2
Out[8]: array([ 1, 2,  3,  4,  5])
```

Как видим, изменения отражаются только в `numbers`.

Модуль copy — поверхностное и глубокое копирование для других типов объектов Python

В предыдущих главах рассматривалось *поверхностное копирование*. В этой главе мы покажем, как выполнить *глубокое копирование* объектов `array` методом `copy`. Если вам понадобятся глубокие копии других типов объектов Python, передайте их функции `deepcopy` модуля `copy`.

7.13. Изменение размеров и транспонирование

Мы использовали метод `reshape` для получения двумерных коллекций `array` по одномерным диапазонам. NumPy предоставляет различные способы изменения размера массивов.

Методы reshape и resize

Методы `reshape` и `resize` коллекций `array` позволяют изменить размеры коллекции. Метод `reshape` возвращает *представление* (поверхностную копию) исходной коллекции `array` с новыми размерами. Исходная коллекция `array` при этом *не* изменяется:

```
In [1]: import numpy as np

In [2]: grades = np.array([[87, 96, 70], [100, 87, 90]])

In [3]: grades
Out[3]:
array([[ 87,  96,  70],
       [100,  87,  90]])

In [4]: grades.reshape(1, 6)
Out[4]: array([[ 87,  96,  70, 100,  87,  90]])

In [5]: grades
Out[5]:
array([[ 87,  96,  70],
       [100,  87,  90]])
```

Метод `resize` *изменяет размер исходной коллекции* `array`:

```
In [6]: grades.resize(1, 6)

In [7]: grades
Out[7]: array([[ 87,  96,  70, 100,  87,  90]])
```

Методы flatten и ravel

Вы можете взять многомерную коллекцию и деструктурировать ее в одно измерение методами `flatten` и `ravel`. Метод `flatten` выполняет *глубокое копирование* данных исходной коллекции:

```
In [8]: grades = np.array([[87, 96, 70], [100, 87, 90]])

In [9]: grades
```

```
Out[9]:
array([[ 87,  96,  70],
       [100,  87,  90]])

In [10]: flattened = grades.flatten()

In [11]: flattened
Out[11]: array([ 87,  96,  70, 100,  87,  90])

In [12]: grades
Out[12]:
array([[ 87,  96,  70],
       [100,  87,  90]])
```

Чтобы подтвердить, что `grades` и `flattened` *не* используют общие данные, изменим элемент `flattened`, а затем выведем обе коллекции `array`:

```
In [13]: flattened[0] = 100

In [14]: flattened
Out[14]: array([100,  96,  70, 100,  87,  90])

In [15]: grades
Out[15]:
array([[ 87,  96,  70],
       [100,  87,  90]])
```

Метод `ravel` создает *представление* исходной коллекции `array`, которое *использует* данные, общие с `grades`:

```
In [16]: raveled = grades.ravel()


In [17]: raveled
Out[17]: array([ 87,  96,  70, 100,  87,  90])

In [18]: grades
Out[18]:
array([[ 87,  96,  70],
       [100,  87,  90]])
```

Чтобы убедиться в том, что `grades` и `raveled` *используют* общие данные, изменим элемент `raveled`, а затем выведем оба массива:

```
In [19]: raveled[0] = 100

In [20]: raveled
Out[20]: array([100,  96,  70, 100,  87,  90])
```

```
In [21]: grades
Out[21]:
array([[100,  96,  70],
       [100,  87,  90]])
```

Транспонирование строк и столбцов

Вы можете быстро *транспонировать* строки и столбцы массива, то есть сделать так, чтобы строки стали столбцами, а столбцы — строками. *Атрибут* `T` возвращает транспонированное *представление* (поверхностную копию) коллекции `array`. Исходный массив `grades` представляет оценки двух студентов (строки) на трех экзаменах (столбцы). Транспонируем строки и столбцы, чтобы просмотреть данные оценок на трех экзаменах (строки) для двух студентов (столбцы):

```
In [22]: grades.T
Out[22]:
array([[100, 100],
       [ 96,  87],
       [ 70,  90]])
```

Транспонирование *не* изменяет исходную коллекцию `array`:

```
In [23]: grades
Out[23]:
array([[100,  96,  70],
       [100,  87,  90]])
```

Горизонтальное и вертикальное дополнение

Коллекции `array` можно объединять, добавляя новые столбцы или новые строки, — эти два механизма называются *горизонтальным* или *вертикальным дополнением*. Создадим еще одну коллекцию `array` с размерами 2 × 3:

```
In [24]: grades2 = np.array([[94, 77, 90], [100, 81, 82]])
```

Допустим, содержимое `grades2` представляет результаты еще трех экзаменов для двух студентов из коллекции `grades`. Мы можем объединить `grades` и `grades2` *функцией* `hstack` из библиотеки NumPy; для этого функции передается кортеж с объединяемыми коллекциями. Дополнительные круглые скобки необходимы из-за того, что функция `hstack` ожидает получить один аргумент:

```
In [25]: np.hstack((grades, grades2))
Out[25]:
array([[100,  96,  70,  94,  77,  90],
       [100,  87,  90, 100,  81,  82]])
```

Теперь предположим, что `grades2` представляет оценки двух других студентов на трех экзаменах. В этом случае `grades` и `grades2` можно объединить *функцией* `vstack` библиотеки NumPy:

```
In [26]: np.vstack((grades, grades2))
Out[26]:
array([[100,  96,  70],
       [100,  87,  90],
       [ 94,  77,  90],
       [100,  81,  82]])
```

7.14. Введение в data science: коллекции Series и DataFrame библиотеки pandas

Массив NumPy оптимизирован для однородных числовых данных, при обращении к которым используются целочисленные индексы. Область data science создает уникальные требования, для которых необходимы более специализированные структуры данных. Приложения больших данных должны поддерживать смешанные типы данных, нестандартное индексирование, отсутствующие данные, данные с нарушенной структурой и данные, которые должны быть приведены к форме, более подходящей для баз данных и пакетов анализа данных, с которыми вы работаете.

Pandas — самая популярная библиотека для работы с такими данными. Она предоставляет две ключевые коллекции, которые мы будем использовать в нескольких разделах «Введение в data science» и в практических примерах data science — `Series` для одномерных коллекций и `DataFrame` для двумерных коллекций. Коллекция `MultiIndex` библиотеки pandas может использоваться для работы с многомерными данными в контексте `Series` и `DataFrame`.

Уэс Маккинни (Wes McKinney) создал pandas в 2008 году, пока он работал в отрасли. Название `pandas` происходит от термина «panel data», то есть данные измерений по времени (например, биржевые котировки или исторические температурные данные). Маккинни понадобилась библиотека, в которой одни и те же структуры данных могли бы обрабатывать как временные, так и не временные данные, с поддержкой выравнивания данных, отсутствующих данных, стандартных операций в стиле баз данных и т. д.[1]

---

[1] McKinney, Wes. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, с. 123–165. Sebastopol, CA: OReilly Media, 2018.

Между NumPy и Pandas существует тесная связь. Коллекции `Series` и `DataFrame` используют `array` в своей внутренней реализации. `Series` и `DataFrame` являются допустимыми аргументами для многих операций NumPy. С другой стороны, коллекции `array` являются допустимыми аргументами для многих операций `Series` и `DataFrame`.

Тема pandas чрезвычайно обширна — документация в формате PDF занимает более 2000 страниц[1]. В разделах «Введение в data science» этой и следующей глав приведен краткий вводный курс pandas. Мы рассмотрим коллекции `Series` и `DataFrame` и используем их для подготовки данных. Вы увидите, что коллекции `Series` и `DataFrame` упрощают выполнение таких распространенных задач, как выбор элементов, операции фильтрации/отображения/свертки (занимающие центральное место в области программирования в функциональном стиле и больших данных), математические операции, визуализация и т. д.

7.14.1. Коллекция Series

`Series` представляет собой расширенную одномерную версию `array`. Если `array` использует только целочисленные индексы, начинающиеся с нуля, то коллекция `Series` поддерживает нестандартное индексирование, включая даже использование нецелочисленных индексов (например, строк). Коллекция `Series` также предоставляет дополнительные возможности, которые делают ее более удобной для многих задач, ориентированных на область data science. Например, коллекция `Series` может содержать отсутствующие данные, которые по умолчанию игнорируются многими операциями `Series`.

Создание Series с индексами по умолчанию

По умолчанию `Series` использует целочисленные индексы с последовательной нумерацией от 0. Следующий фрагмент создает коллекцию `Series` на базе списка целых чисел:

```
In [1]: import pandas as pd

In [2]: grades = pd.Series([87, 100, 94])
```

[1] Новейшая документация pandas доступна по адресу http://pandas.pydata.org/pandas-docs/stable/.

Инициализатором также может быть кортеж, словарь, `array`, другая коллекция `Series` или одиночное значение (последний случай продемонстрирован ниже).

Вывод коллекции Series

Pandas выводит `Series` в формате из двух столбцов: индексы выравниваются *по левому краю* в левом столбце, а значения — *по правому краю* в правом столбце. После элементов `Series` pandas выводит тип данных (`dtype`) элементов используемой коллекции `array`:

```
In [3]: grades
Out[3]:
0     87
1    100
2     94
dtype: int64
```

Обратите внимание, насколько просто выводится `Series` в этом формате по сравнению с соответствующим кодом вывода списка в двухстолбцовом формате.

Создание коллекции Series с одинаковыми значениями

Вы можете создать коллекцию `Series`, все элементы которой имеют одинаковые значения:

```
In [4]: pd.Series(98.6, range(3))
Out[4]:
0    98.6
1    98.6
2    98.6
dtype: float64
```

Второй аргумент представляет собой одномерный итерируемый объект (такой как список, массив или диапазон) с индексами `Series`. Количество индексов определяет количество элементов.

Обращение к элементам Series

Чтобы обратиться к элементу `Series`, укажите его индекс в квадратных скобках:

```
In [5]: grades[0]
Out[5]: 87
```

Вычисление описательных статистик для Series

Коллекция `Series` предоставляет многочисленные методы для выполнения часто встречающихся операций, включая получение различных характеристик описательной статистики. В этом разделе продемонстрированы методы `count`, `mean`, `min`, `max` и `std` (стандартное отклонение):

```
In [6]: grades.count()
Out[6]: 3

In [7]: grades.mean()
Out[7]: 93.66666666666667

In [8]: grades.min()
Out[8]: 87

In [9]: grades.max()
Out[9]: 100

In [10]: grades.std()
Out[10]: 6.506407098647712
```

Каждая из этих характеристик является сверткой в стиле функционального программирования. Вызов методов `Series` создает не только названные характеристики, но и многие другие:

```
In [11]: grades.describe()
Out[11]:
count      3.000000
mean      93.666667
std        6.506407
min       87.000000
25%       90.500000
50%       94.000000
75%       97.000000
max      100.000000
dtype: float64
```

Строки `25%`, `50%` и `75%` содержат *квартили*:

- `50%` — медиана отсортированных значений;
- `25%` — медиана первой половины отсортированных значений;M
- `75%` — медиана второй половины отсортированных значений.

Если в квартиле существуют два средних элемента, то медианой этого квартиля становится их среднее значение. Наша коллекция `Series` состоит из трех значений, так что 25-процентным квартилем становится среднее значение `87` и `94`, а 75-процентным квартилем — среднее значение `94` и `100`. Еще одной дисперсионной метрикой (наряду со стандартным отклонением и дисперсией) является *интерквартильный диапазон* — разность между 75-процентным квартилем и 25-процентным квартилем. Конечно, квартили и интерквартильный диапазон приносят больше пользы в больших наборах данных.

Создание коллекции Series с нестандартными индексами

Для назначения *нестандартных* индексов используется ключевой аргумент `index`:

```
In [12]: grades = pd.Series([87, 100, 94], index=['Wally', 'Eva', 'Sam'])

In [13]: grades
Out[13]:
Wally     87
Eva      100
Sam       94
dtype: int64
```

В данном случае используются строковые индексы, но можно использовать и другие неизменяемые типы, включая целые числа, не начинающиеся с 0, а также непоследовательные целые числа. Стоит при этом заметить, как удобно и компактно pandas форматирует коллекции `Series` для вывода.

Словари как инициализаторы

Если коллекция `Series` инициализируется словарем, то ключи становятся индексами `Series`, а значения — значениями элементов `Series`:

```
In [14]: grades = pd.Series({'Wally': 87, 'Eva': 100, 'Sam': 94})

In [15]: grades
Out[15]:
Wally     87
Eva      100
Sam       94
dtype: int64
```

Обращение к элементам Series с использованием нестандартных индексов

Чтобы обратиться к отдельному элементу в коллекции `Series` с нестандартными индексами, укажите значение нестандартного индекса в квадратных скобках:

```
In [16]: grades['Eva']
Out[16]: 100
```

Если нестандартными индексами являются строки, которые могут представлять допустимые идентификаторы Python, то pandas автоматически добавляет их в `Series` как атрибуты, к которым можно обращаться через точку (`.`):

```
In [17]: grades.Wally
Out[17]: 87
```

Коллекция `Series` также содержит *встроенные* атрибуты. Например, *атрибут* `dtype` возвращает тип элемента базовой коллекции `array`:

```
In [18]: grades.dtype
Out[18]: dtype('int64')
```

Атрибут `values` возвращает базовую коллекцию `array`:

```
In [19]: grades.values
Out[19]: array([ 87, 100,  94])
```

Создание коллекции Series со строковыми элементами

Если коллекция `Series` содержит строки, то можно воспользоваться ее *атрибутом* `str` для вызова методов строк элементов. Сначала создадим коллекцию `Series` со строками:

```
In [20]: hardware = pd.Series(['Hammer', 'Saw', 'Wrench'])

In [21]: hardware
Out[21]:
0    Hammer
1       Saw
2    Wrench
dtype: object
```

Обратите внимание: pandas также выравнивает *по правому краю* строковые значения элементов, а типом данных (`dtype`) для строк является `object`.

Теперь вызовем метод `contains` для каждого элемента, чтобы определить, содержит ли значение каждого элемента букву `'a'` нижнего регистра:

```
In [22]: hardware.str.contains('a')
Out[22]:
0     True
1     True
2    False
dtype: bool
```

Pandas возвращает коллекцию `Series` с логическими значениями, обозначающими результат вызова `contains` для каждого элемента, — элемент с индексом 2 (`'Wrench'`) не содержит `'a'`, поэтому элемент полученной коллекции `Series` равен `False`. Обратите внимание: pandas реализует итерации за вас — и это еще один наглядный пример программирования в функциональном стиле. Атрибут `str` предоставляет много методов обработки строк, сходных с методами строкового типа Python. Полный список доступен по адресу:

https://pandas.pydata.org/pandas-docs/stable/api.html#string-handling.

В следующем примере метод `upper` используется для создания *новой* коллекции `Series`, содержащей элементы `hardware` в верхнем регистре:

```
In [23]: hardware.str.upper()
Out[23]:
0    HAMMER
1       SAW
2    WRENCH
dtype: object
```

7.14.2. DataFrame

`DataFrame` — расширенная двумерная версия `array`. Как и `Series`, `DataFrame` может иметь нестандартные индексы строк и столбцов и предоставляет дополнительные операции и средства, с которыми удобнее использовать коллекцию `DataFrame` для многих задач, ориентированных на специфику data science. `DataFrame` также поддерживает пропущенные данные. Каждый столбец `DataFrame` является коллекцией `Series`. Коллекция `Series`, представляющая каждый столбец, может содержать элементы разных типов, как будет вскоре показано при рассмотрении загрузки наборов данных в `DataFrame`.

Создание DataFrame на базе словаря

Создадим коллекцию `DataFrame` на базе словаря, представляющего оценки студентов на трех экзаменах:

```
In [1]: import pandas as pd

In [2]: grades_dict = {'Wally': [87, 96, 70], 'Eva': [100, 87, 90],
   ...:                'Sam': [94, 77, 90], 'Katie': [100, 81, 82],
   ...:                'Bob': [83, 65, 85]}
   ...:

In [3]: grades = pd.DataFrame(grades_dict)

In [4]: grades
Out[4]:
   Wally  Eva  Sam  Katie  Bob
0     87  100   94    100   83
1     96   87   77     81   65
2     70   90   90     82   85
```

Pandas выводит содержимое `DataFrame` в табличном формате с индексами, выровненными *по левому краю* в столбце индексов, а значения остальных столбцов выравниваются *по правому краю*. *Ключи* словаря становятся именами столбцов, а *значения*, связанные с каждым ключом, становятся значениями элементов соответствующего столбца. Вскоре мы покажем, как поменять местами строки и столбцы. По умолчанию индексы строк задаются автоматически сгенерированными целыми числами, начиная с 0.

Настройка индексов DataFrame с использованием атрибута index

При создании `DataFrame` можно задать нестандартные индексы при помощи ключевого аргумента `index`:

```
pd.DataFrame(grades_dict, index=['Test1', 'Test2', 'Test3'])
```

Воспользуемся *атрибутом* `index` для преобразования индексов `DataFrame` из последовательных целых чисел в текстовые метки:

```
In [5]: grades.index = ['Test1', 'Test2', 'Test3']

In [6]: grades
Out[6]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test2     96   87   77     81   65
Test3     70   90   90     82   85
```

При задании индексов необходимо передать одномерную коллекцию, количество элементов в которой совпадает с количеством *строк* данных в `DataFrame`; в противном случае происходит ошибка `ValueError`. `Series` также предоставляет *атрибут* `index` для изменения существующих индексов `Series`.

Обращение к столбцам DataFrame

Одна из сильных сторон pandas — возможность быстро и удобно просматривать данные многими разными способами, включая отдельные части данных. Начнем с получения оценок студента `Eva` по имени и вывода столбца в формате `Series`:

```
In [7]: grades['Eva']
Out[7]:
Test1    100
Test2     87
Test3     90
Name: Eva, dtype: int64
```

Если строки с именами столбцов `DataFrame` являются допустимыми идентификаторами Python, они могут использоваться как атрибуты. Получим оценки студента `Sam` при помощи *атрибута* `Sam`:

```
In [8]: grades.Sam
Out[8]:
Test1    94
Test2    77
Test3    90
Name: Sam, dtype: int64
```

Выбор строк с использованием атрибутов loc и iloc

Хотя коллекции `DataFrame` поддерживают возможность индексирования в синтаксисе `[]`, документация pandas рекомендует использовать атрибуты `loc`, `iloc`, `at` и `iat`, которые оптимизированы для обращения к `DataFrame` и предоставляют дополнительные средства, выходящие за рамки того, что можно сделать исключительно с `[]`. В документации указано, что при индексировании `[]` *часто* создается копия данных, что может привести к логической ошибке, если вы попытаетесь присвоить новые значения `DataFrame` результату операции `[]`.

Чтобы обратиться к строке по ее текстовой метке, воспользуйтесь *атрибутом* `loc` коллекции `DataFrame`. Ниже выводятся все оценки из строки `'Test1'`:

```
In [9]: grades.loc['Test1']
Out[9]:
Wally     87
Eva      100
Sam       94
Katie    100
Bob       83
Name: Test1, dtype: int64
```

К строкам также можно обращаться по целочисленным индексам, начинающимся с 0, при помощи *атрибута* `iloc` (буква `i` в `iloc` означает, что атрибут используется с целочисленными индексами). Следующий пример выводит все оценки из второй строки:

```
In [10]: grades.iloc[1]
Out[10]:
Wally    96
Eva      87
Sam      77
Katie    81
Bob      65
Name: Test2, dtype: int64
```

Выбор строк с использованием атрибутов loc и iloc

Индексом может быть и *сегмент*. При использовании с атрибутом `loc` сегментов, содержащих метки, заданный диапазон *включает* верхний индекс (`'Test3'`):

```
In [11]: grades.loc['Test1':'Test3']
Out[11]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test2     96   87   77     81   65
Test3     70   90   90     82   85
```

При использовании с атрибутом `iloc` сегментов, содержащих целочисленные индексы, заданный диапазон *не включает* верхний индекс (`2`):

```
In [12]: grades.iloc[0:2]
Out[12]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test2     96   87   77     81   65
```

Чтобы выбрать *конкретные строки*, используйте синтаксис *списка* вместо синтаксиса сегмента в сочетании с `loc` или `iloc`:

```
In [13]: grades.loc[['Test1', 'Test3']]
Out[13]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test3     70   90   90     82   85

In [14]: grades.iloc[[0, 2]]
Out[14]:
       Wally  Eva  Sam  Katie  Bob
Test1     87  100   94    100   83
Test3     70   90   90     82   85
```

Выбор подмножеств строк и столбцов

До настоящего момента мы выбирали только *целые* строки. Вы также можете ограничиться меньшими подмножествами `DataFrame` и выбирать строки *и* столбцы с использованием двух сегментов, двух списков или сочетания сегментов и списков.

Предположим, вы хотите просмотреть только оценки студентов `Eva` и `Katie` для экзаменов `Test1` и `Test2`. Для этого можно воспользоваться атрибутом `loc` с сегментом для двух последовательных строк и списком для двух непоследовательных столбцов:

```
In [15]: grades.loc['Test1':'Test2', ['Eva', 'Katie']]
Out[15]:
       Eva  Katie
Test1  100    100
Test2   87     81
```

Сегмент `'Test1':'Test2'` выбирает строки для `Test1` и `Test2`. Список `['Eva', 'Katie']` выбирает только соответствующие оценки из этих двух столбцов.

Воспользуемся `iloc` со списком и сегментом, чтобы выбрать первый и третий экзамены и первые три столбца для этих экзаменов:

```
In [16]: grades.iloc[[0, 2], 0:3]
Out[16]:
       Wally  Eva  Sam
Test1     87  100   94
Test3     70   90   90
```

Логическое индексирование

Одним из самых мощных средств выбора в pandas является *логическое индексирование*. Для примера выберем все оценки, большие или равные 90:

```
In [17]: grades[grades >= 90]
Out[17]:
       Wally    Eva   Sam  Katie  Bob
Test1    NaN  100.0  94.0  100.0  NaN
Test2   96.0    NaN   NaN    NaN  NaN
Test3    NaN   90.0  90.0    NaN  NaN
```

Pandas проверяет каждую оценку и, если она больше или равна 90, включает ее в новую коллекцию `DataFrame`. Оценки, для которых условие равно `False`, представлены в новой коллекции `DataFrame` значением `NaN` («*Not A Number*», то есть «*не число*».) `NaN` используется pandas для обозначения отсутствующих значений.

Выберем все оценки в диапазоне 80–89:

```
In [18]: grades[(grades >= 80) & (grades < 90)]
Out[18]:
       Wally   Eva  Sam  Katie   Bob
Test1   87.0   NaN  NaN    NaN  83.0
Test2    NaN  87.0  NaN   81.0   NaN
Test3    NaN   NaN  NaN   82.0  85.0
```

Логические индексы Pandas объединяют несколько условий оператором Python `&` (поразрядная операция И) — *не* путайте с логическим оператором `and`. Для условий `or` используется оператор `|` (поразрядная операция ИЛИ). NumPy также поддерживает логическое индексирование для массивов, но всегда возвращает одномерный массив, содержащий только те значения, для которых выполняется условие.

Обращение к конкретной ячейке DataFrame по строке и столбцу

Атрибуты `at` и `iat` коллекции `DataFrame` могут использоваться для получения отдельного значения из `DataFrame`. Как и `loc` и `iloc`, атрибут `at` использует метки, а `iat` — целочисленные индексы. В любом случае индексы строки и столбца должны быть разделены запятой. Выберем оценки студента `Eva` на экзамене `Test2` (`87`) и студента `Wally` на экзамене `Test3` (`70`):

```
In [19]: grades.at['Test2', 'Eva']
Out[19]: 87

In [20]: grades.iat[2, 0]
Out[20]: 70
```

Также можно присвоить новые значения конкретным элементам. Заменим оценку `Eva` на экзамене `Test2` на 100 при помощи атрибута `at`, а затем снова вернем ей значение `87` при помощи `iat`:

```
In [21]: grades.at['Test2', 'Eva'] = 100

In [22]: grades.at['Test2', 'Eva']
Out[22]: 100

In [23]: grades.iat[1, 2] = 87

In [24]: grades.iat[1, 2]
Out[24]: 87.0
```

Описательная статистика

Коллекции `Series` и `DataFrame` содержат *метод* `describe`, который вычисляет основные характеристики описательной статистики для данных и возвращает их в форме `DataFrame`. В `DataFrame` статистики вычисляются по столбцам (еще раз: вскоре мы покажем, как поменять местами строки и столбцы):

```
In [25]: grades.describe()
Out[25]:
           Wally         Eva        Sam       Katie        Bob
count   3.000000    3.000000   3.000000    3.000000   3.000000
mean   84.333333   92.333333  87.000000   87.666667  77.666667
std    13.203535    6.806859   8.888194   10.692677  11.015141
min    70.000000   87.000000  77.000000   81.000000  65.000000
25%    78.500000   88.500000  83.500000   81.500000  74.000000
50%    87.000000   90.000000  90.000000   82.000000  83.000000
75%    91.500000   95.000000  92.000000   91.000000  84.000000
max    96.000000  100.000000  94.000000  100.000000  85.000000
```

Как видно из вывода, метод `describe` позволяет быстро получить сводную картину данных. Он неплохо демонстрирует мощь программирования, ориентированного на массивы, на примере понятного, компактного вызова в функциональном стиле. Pandas берет на себя все подробности вычисления этих статистик для каждого столбца. Возможно, вам захочется получить аналогичную статистику для экзаменов, чтобы уяснить, какие результаты

показали студенты на экзаменах Test1, Test2 и Test3, — вскоре мы покажем, как это делается.

По умолчанию pandas вычисляет характеристики описательной статистики в формате чисел с плавающей точкой и выводит их с шестью знаками точности. Для управления точностью и другими настройками по умолчанию может использоваться *функция* pandas `set_option`:

```
In [26]: pd.set_option('precision', 2)

In [27]: grades.describe()
Out[27]:
       Wally     Eva    Sam   Katie    Bob
count   3.00    3.00   3.00    3.00   3.00
mean   84.33   92.33  87.00   87.67  77.67
std    13.20    6.81   8.89   10.69  11.02
min    70.00   87.00  77.00   81.00  65.00
25%    78.50   88.50  83.50   81.50  74.00
50%    87.00   90.00  90.00   82.00  83.00
75%    91.50   95.00  92.00   91.00  84.00
max    96.00  100.00  94.00  100.00  85.00
```

Вероятно, для оценок самой важной характеристикой является математическое ожидание. Чтобы вычислить его для каждого студента, достаточно вызвать `mean` для коллекции `DataFrame`:

```
In [28]: grades.mean()
Out[28]:
Wally    84.33
Eva      92.33
Sam      87.00
Katie    87.67
Bob      77.67
dtype: float64
```

Вскоре мы покажем, как вычислить среднюю оценку всех студентов по каждому экзамену всего в одной строке кода.

Транспонирование DataFrame с использованием атрибута T

Чтобы быстро *транспонировать* `DataFrame`, то есть поменять местами строки и столбцы, можно воспользоваться *атрибутом* `T`:

```
In [29]: grades.T
Out[29]:
       Test1  Test2  Test3
```

```
Wally     87     96     70
Eva      100     87     90
Sam       94     77     90
Katie    100     81     82
Bob       83     65     85
```

`T` возвращает транспонированное *представление* (не копию) коллекции `DataFrame`.

Допустим, вместо того чтобы вычислить сводную статистику по студентам, вы хотите получить ее по экзаменам. Для этого достаточно вызвать `describe` для `grades.T`:

```
In [30]: grades.T.describe()
Out[30]:
        Test1  Test2  Test3
count    5.00   5.00   5.00
mean    92.80  81.20  83.40
std      7.66  11.54   8.23
min     83.00  65.00  70.00
25%     87.00  77.00  82.00
50%     94.00  81.00  85.00
75%    100.00  87.00  90.00
max    100.00  96.00  90.00
```

Чтобы просмотреть средние оценки всех студентов на каждом экзамене, вызовите `mean` для атрибута `T`:

```
In [31]: grades.T.mean()
Out[31]:
Test1    92.8
Test2    81.2
Test3    83.4
dtype: float64
```

Сортировка строк по индексам

Данные часто сортируются для удобства чтения. `DataFrame` можно сортировать по строкам и по столбцам как по индексам, так и по значениям. Отсортируем строки *по убыванию индексов* при помощи функции `sort_index`, которой передается ключевой аргумент `ascending=False` (по умолчанию сортировка выполняется *по возрастанию*). Функция возвращает новую коллекцию `DataFrame` с отсортированными данными:

```
In [32]: grades.sort_index(ascending=False)
Out[32]:
       Wally  Eva  Sam  Katie  Bob
```

```
Test3     70   90   90     82   85
Test2     96   87   77     81   65
Test1     87  100   94    100   83
```

Сортировка по индексам столбцов

Теперь отсортируем столбцы по возрастанию (слева направо) по именам столбцов. *Ключевой аргумент* `axis=1` означает, что сортировка должна выполняться по индексам *столбцов*, а не по индексам строк — аргумент `axis=0` (по умолчанию) сортирует индексы *строк*:

```
In [33]: grades.sort_index(axis=1)
Out[33]:
       Bob  Eva  Katie  Sam  Wally
Test1   83  100    100   94     87
Test2   65   87     81   77     96
Test3   85   90     82   90     70
```

Сортировка по значениям столбцов

Предположим, вы хотите просмотреть оценки `Test1` по убыванию, чтобы имена студентов следовали от наибольшей оценки к наименьшей. Вызов метода `sort_values` выглядит так:

```
In [34]: grades.sort_values(by='Test1', axis=1, ascending=False)
Out[34]:
       Eva  Katie  Sam  Wally  Bob
Test1  100    100   94     87   83
Test2   87     81   77     96   65
Test3   90     82   90     70   85
```

Ключевые аргументы `by` и `axis` совместно работают для определения того, какие значения будут сортироваться. В данном случае сортировка осуществляется по значениям столбцов (`axis=1`) для `Test1`.

Возможно, имена студентов и оценки станет удобнее читать при выводе в столбец, так что отсортировать можно и транспонированную коллекцию `DataFrame`. В следующем примере указывать ключевой аргумент `axis` не нужно, потому что `sort_values` по умолчанию сортирует данные в заданном столбце:

```
In [35]: grades.T.sort_values(by='Test1', ascending=False)
Out[35]:
       Test1  Test2  Test3
Eva      100     87     90
```

```
Katie    100     81     82
Sam       94     77     90
Wally     87     96     70
Bob       83     65     85
```

Наконец, поскольку сортируются только оценки экзамена `Test1`, не исключено, что другие экзамены выводить вообще не нужно. Объединим операцию выбора с сортировкой:

```
In [36]: grades.loc['Test1'].sort_values(ascending=False)
Out[36]:
Katie    100
Eva      100
Sam       94
Wally     87
Bob       83
Name: Test1, dtype: int64
```

Копирование и сортировка на месте

По умолчанию `sort_index` и `sort_values` возвращают *копию* исходной коллекции `DataFrame`, что может потребовать значительных затрат памяти в приложениях больших данных. `DataFrame` также можно отсортировать *на месте*, чтобы обойтись без *копирования* данных. Для этого передайте ключевой аргумент `inplace=True` функции `sort_index` или `sort_values`.

Мы продемонстрировали многие возможности коллекций `Series` и `DataFrame` библиотеки pandas. В разделе «Введение в data science» следующей главы коллекции `Series` и `DataFrame` будут использованы для *первичной обработки данных* — очистки и подготовки данных для использования в базах данных или аналитических пакетах.

7.15. Итоги

В этой главе рассматривалось применение высокопроизводительных коллекций `ndarray` библиотеки NumPy для хранения и загрузки данных, а также для выполнения стандартной обработки данных в более компактной форме и с пониженным риском ошибок за счет использования программирования в функциональном стиле. В тексте главы `ndarray` обозначается синонимом `array`.

Примеры главы показывают, как создать, инициализировать и обращаться к отдельным элементам одно- и двумерных коллекций `array`. Атрибуты использовались для определения размеров коллекции и типа элементов. Мы представили функции, которые создают массивы, заполненные нулями, единицами, конкретными значениями или диапазонами значений. Мы сравнили быстродействие списков и `array` при помощи магических команд IPython `%timeit` и убедились в том, что `array` работает на два порядка быстрее.

Операторы `array` и универсальные функции NumPy используются для выполнения поэлементных вычислений со всеми элементами коллекций `array`, имеющих одинаковые размеры. Вы также узнали, что NumPy применяет распространение для выполнения поэлементных операций между `array` и скалярными значениями, а также между коллекциями `array` разных размеров. Мы представили различные встроенные методы `array` для выполнения вычислений со всеми элементами массива и показали, как выполнять эти вычисления по строкам или столбцам. Были продемонстрированы и различные средства сегментации и индексирования массивов — более мощные, чем у встроенных коллекций Python. Кроме того, мы представили различные способы изменения размеров `array` и обсудили нюансы поверхностного и глубокого копирования `array` и других объектов Python.

В разделе «Введение в data science» началось ваше знакомство с популярной библиотекой pandas, которая будет использоваться во многих практических примерах data science. Вы узнали, что многим приложениям больших данных необходимы более гибкие коллекции, чем массивы NumPy, — коллекции с поддержкой смешанных типов данных, нестандартного индексирования, отсутствующих данных, данных с нарушенной структурой и данных, которые должны быть приведены к форме, более подходящей для баз данных и пакетов анализа данных, с которыми вы работаете.

Вы узнали, как создавать и обрабатывать одномерные коллекции `Series` и двумерные коллекции `DataFrame` и как настраивать индексы `Series` и `DataFrame`. Вы увидели, как выглядит отформатированный вывод pandas, и научились настраивать точность вывода значений с плавающей точкой. Были продемонстрированы различные способы обращения к данным и их выбора из коллекций `Series` и `DataFrame`. Мы использовали метод `describe` для получения основных описательных статистик для `Series` и `DataFrame`. Вы узнали, как транспонировать строки и столбцы `DataFrame` с помощью атрибута `T`. Мы рассмотрели различные способы сортировки `DataFrame` по значениям индексов, по именам столбцов, по данным строк и данным столбцов. После прочтения этой главы в вашем арсенале появились уже четыре коллекции,

сходные с массивами, — списки, `array`, `Series` и `DataFrame`. Пятая разновидность — тензоры — добавится в главе 15.

В следующей главе более глубоко рассматриваются строки, форматирование строк и методы строк. Также будут представлены регулярные выражения, используемые для поиска по шаблону в тексте. Средства, которые в ней описаны, подготовят вас к знакомству с главой 11 и другими ключевыми главами, посвященными data science. В разделе «Введение в data science» следующей главы будет представлен процесс *первичной обработки данных* — подготовки данных для использования в базах данных или аналитических пакетах. В дальнейшем мы воспользуемся pandas для анализа временных рядов, а также представим средства визуализации pandas.

8

Подробнее о строках

В этой главе...

- Обработка текста.
- Методы строк.
- Содержимое форматных строк.
- Конкатенация и повторение строк.
- Удаление пропусков в конце строк.
- Преобразование символов нижнего регистра к верхнему и наоборот.
- Сравнение строк операторами сравнения.
- Поиск и замена подстрок.
- Разбиение строк на лексемы.
- Конкатенация строк с использованием заданного разделителя.
- Создание и использование регулярных выражений для поиска по шаблону в строках, замены подстрок и проверки данных.
- Использование в регулярных выражениях метасимволов, квантификаторов, символьных классов и группировки.
- Роль строковых операций в обработке естественного языка.
- Терминология data science: первичная обработка данных, очистка данных и использование регулярных выражений для приведения данных к нужному формату.

8.1. Введение

Ранее мы представили строки, базовое форматирование строк и некоторые строковые операции и методы, показав, в частности, что строки поддерживают многие операции последовательностей, как и списки и кортежи, и что строки, как и кортежи, являются неизменяемыми. В этой главе мы поглубже разберемся в строках и представим регулярные выражения и модуль `re`, используемый для поиска по шаблону в тексте[1]. Регулярные выражения особенно важны в современных приложениях, для которых характерна интенсивная обработка данных. Возможности, представленные в тексте, подготовят читателей к знакомству с главой 15 и другими фрагментами содержания, посвященными data science (в главе 15 рассматриваются различные способы обработки и даже «понимания» данных компьютерами). В табл. 8.1 перечислены различные области применения строковых операций и NLP (обработки естественного языка). В разделе «Введение в data science» мы кратко расскажем об очистке данных/предварительной обработке данных в коллекциях `Series` и `DataFrame` библиотеки pandas.

Таблица 8.1. Области применения строковых операций и NLP

Автоматизированная оценка письменных работ	Переводы с иностранных языков
Автоматизированные системы обучения	Подготовка юридических документов
Авторство книг Шекспира	Поисковые системы
Анаграммы	Пометка частей речи
Анализ мнений	Понимание естественных языков
Анализ эмоций	Преобразователи речи в текст
Бесплатные книги проекта «Гутенберг»	Преобразователи текста в речь
Веб-агрегаторы	Проверка грамматики
Выявление мошенничества	Проверка правописания
Игры со словами	Система пополосной верстки
Классификация документов	Стеганография
Классификация спама	Текстовые редакторы
Классификация статей	Уплотнение документов
Компиляторы и интерпретаторы	Формирование медицинских диагнозов на основании рентгеновских снимков, томографии, анализов крови
Криптография	Чат-боты
Литературное творчество	Чтение книг, статей и документов с извлечением информации
Облака тегов	Электронные системы чтения
Определение сходства документов	
Палиндромы	

[1] В главах с практическими примерами data science будет показано, что поиск по шаблону в тексте является важнейшим аспектом машинного обучения.

8.2. Форматирование строк

Правильное форматирование текста упрощает чтение и понимание данных. В этом разделе будут описаны многие средства форматирования текста.

8.2.1. Типы представлений

Мы продемонстрировали базовые средства форматирования текста с использованием форматных строк. Когда вы включаете заполнитель для значения в форматной строке, Python предполагает, что значение будет отображаться в строковом виде, если только вы явно не укажете другой тип. В некоторых случаях тип обязателен. Для примера отформатируем значение с плавающей точкой `17.489`, округленное до двух знаков в дробной части:

```
In [1]: f'{17.489:.2f}'
Out[1]: '17.49'
```

Python поддерживает настройку точности *только* для значений *с плавающей точкой* и `Decimal`. Форматирование *зависит от типа* — если вы попытаетесь применить спецификатор `.2f` для форматирования такой строки, как `'hello'`, то произойдет ошибка `ValueError`. По этой причине *тип представления* `f` в *форматном спецификаторе* `.2f` обязателен. Он показывает, какой тип форматируется, чтобы Python мог определить, разрешена ли другая информация форматирования для этого типа. Ниже перечислены некоторые распространенные типы представления. Полный список доступен по адресу:

https://docs.python.org/3/library/string.html#formatspec

Целые числа

Тип представления `d` форматирует целочисленные значения как строки:

```
In [2]: f'{10:d}'
Out[2]: '10'
```

Существуют и другие типы представления (`b`, `o`, `x` и `X`), форматирующие целые числа в двоичной, восьмеричной и шестнадцатеричной системах счисления[1].

[1] За информацией о двоичной, восьмеричной и шестнадцатеричной системах счисления обращайтесь к электронному приложению «Системы счисления».

Символы

Тип представления `c` форматирует целочисленный код символа в виде соответствующего символа:

```
In [3]: f'{65:c} {97:c}'
Out[3]: 'A a'
```

Строки

Тип представления `s` используется по умолчанию. Если тип `s` указан явно, то форматируемое значение должно быть переменной, которая ссылается на строку, выражением, результатом которого является строка или строковый литерал, как в первом заполнителе из следующего примера. Если тип представления не указан, как во втором заполнителе из следующего примера, то нестроковые значения (такие как целое число `7`) преобразуются в строки:

```
In [4]: f'{"hello":s} {7}'
Out[4]: 'hello 7'
```

Здесь строка `"hello"` заключена в двойные кавычки. Напомним, одинарные кавычки не могут содержаться в строках, заключенных в одинарные кавычки.

Значения с плавающей точкой и Decimal

Мы использовали тип представления `f` для форматирования значений с плавающей точкой и значений `Decimal`. В случае очень больших или очень малых значений данного типа можно использовать экспоненциальную (научную) запись для более компактного форматирования этих значений. Продемонстрируем различия между `f` и `e` для больших значений, каждое из которых выводится с тремя знаками в дробной части:

```
In [5]: from decimal import Decimal

In [6]: f'{Decimal("10000000000000000000000000.0"):.3f}'
Out[6]: '10000000000000000000000000.000'

In [7]: f'{Decimal("10000000000000000000000000.0"):.3e}'
Out[7]: '1.000e+25'
```

Для *типа представления* `e` во фрагменте [5] отформатированное значение `1.000e+25` эквивалентно

1.000×10^{25}.

Если вы предпочитаете видеть букву `E` в экспоненциальной записи, то используйте *тип представления* `E` вместо `e`.

8.2.2. Ширины полей и выравнивание

Ранее мы использовали *ширину поля* для форматирования текста по заданному количеству позиций символов. По умолчанию Python использует *выравнивание 0 по правому краю* для чисел и *выравнивание по левому краю* для других значений (например, строк) — мы заключили результаты в приведенных ниже примерах в квадратные скобки (`[]`), чтобы вы видели, как значения выравниваются внутри поля:

```
In [1]: f'[{27:10d}]'
Out[1]: '[        27]'

In [2]: f'[{3.5:10f}]'
Out[2]: '[  3.500000]'

In [3]: f'[{"hello":10}]'
Out[3]: '[hello     ]'
```

Фрагмент [2] показывает, что Python по умолчанию форматирует значения с плавающей точкой с шестью цифрами в дробной части. Для значений, количество символов в которых меньше ширины поля, остальные позиции заполняются нулями. Для значений, количество символов в которых больше ширины поля, используется количество позиций, необходимое для вывода.

Явное назначение выравнивания в поле

Тип выравнивания (по левому или по правому краю) можно назначить при помощи символов `<` и `>`:

```
In [4]: f'[{27:<15d}]'
Out[4]: '[27             ]'

In [5]: f'[{3.5:<15f}]'
Out[5]: '[3.500000       ]'

In [6]: f'[{"hello":>15}]'
Out[6]: '[          hello]'
```

Выравнивание значения по центру поля

Предусмотрена и возможность выравнивания значений *по центру поля*:

```
In [7]: f'[{27:^7d}]'
Out[7]: '[  27   ]'

In [8]: f'[{3.5:^7.1f}]'
Out[8]: '[  3.5  ]'

In [9]: f'[{"hello":^7}]'
Out[9]: '[ hello ]'
```

При выравнивании по центру Python старается равномерно распределить свободные позиции символов слева и справа от отформатированного значения. Если оставшееся количество позиций нечетное, то Python добавляет дополнительный пробел справа.

8.2.3. Форматирование чисел

Python предоставляет обширные возможности форматирования чисел.

Форматирование положительных чисел со знаком

Иногда требуется принудительно выводить знак рядом с положительным числом:

```
In [1]: f'[{27:+10d}]'
Out[1]: '[       +27]'
```

+ перед полем означает, что перед положительным числом должен стоять знак +. Перед отрицательным числом всегда стоит знак -. Чтобы остальные позиции поля заполнялись нулями (вместо пробелов), поставьте 0 перед шириной поля (и *после* знака +, если он присутствует):

```
In [1]: f'[{27:+010d}]'
Out[1]: '[+000000027]'
```

Использование пробела в позиции знака + в положительном числе

Пробел означает, что в позиции знака в положительном числе должен выводиться пробел. Это может быть полезно для выравнивания положительных и отрицательных чисел при выводе:

```
In [3]: print(f'{27:d}\n{27: d}\n{-27: d}')
27
 27
-27
```

Обратите внимание: числа с пробелами в их форматных спецификаторах выровнены. При заданной ширине поля пробел должен *предшествовать* ширине поля.

Группировка цифр

Для форматирования числа с *разделением групп разрядов* используем *запятую* (`,`):

```
In [4]: f'{12345678:,d}'
Out[4]: '12,345,678'

In [5]: f'{123456.78:,.2f}'
Out[5]: '123,456.78'
```

8.2.4. Метод format

Форматные строки Python были добавлены в язык в версии 3.6. До этого форматирование строк выполнялось методом строк `format`. Собственно, функциональность форматных строк основана на средствах метода `format`. Мы описываем здесь метод `format`, потому что он встречается в коде, написанном до выхода Python 3.6. Метод `format` часто встречается в документации Python и во многих книгах и статьях Python, написанных до появления форматных строк. Тем не менее мы рекомендуем использовать в ваших программах и новые возможности форматных строк.

Метод `format` вызывается для *форматной строки*, содержащей *заполнители* в фигурных скобках (`{}`), — возможно, с форматными спецификаторами. Методу передаются форматируемые значения. Отформатируем значение с плавающей точкой `17.489` до двух знаков в дробной части:

```
In [1]: '{:.2f}'.format(17.489)
Out[1]: '17.49'
```

Если в заполнителе присутствует форматный спецификатор, перед ним ставится двоеточие (`:`), как и в форматных строках. Результатом вызова является новая строка, содержащая отформатированные результаты.

Несколько заполнителей

Форматная строка может содержать сразу несколько заполнителей; в этом случае аргументы метода `format` сопоставляются с заполнителями слева направо:

```
In [2]: '{} {}'.format('Amanda', 'Cyan')
Out[2]: 'Amanda Cyan'
```

Ссылка на аргументы по позиции

Форматная строка может ссылаться на аргументы по их позиции в списке аргументов метода `format`; первому аргументу соответствует позиция 0:

```
In [3]: '{0} {0} {1}'.format('Happy', 'Birthday')
Out[3]: 'Happy Happy Birthday'
```

Обратите внимание: позиция аргумента 0 (`'Happy'`) используется дважды — вы можете обращаться к любым аргументам сколько угодно раз и в любом порядке.

Ссылка на ключевые аргументы

К ключевым аргументам можно обращаться в заполнителях по их ключам:

```
In [4]: '{first} {last}'.format(first='Amanda', last='Gray')
Out[4]: 'Amanda Gray'

In [5]: '{last} {first}'.format(first='Amanda', last='Gray')
Out[5]: 'Gray Amanda'
```

8.3. Конкатенация и повторение строк

В предыдущих главах оператор `+` использовался для конкатенации строк, а оператор `*` — для их повторения. Эти операции также могут выполняться с расширенным присваиванием. Строки неизменяемы, поэтому каждая операция присваивает переменной новый объект строки:

```
In [1]: s1 = 'happy'

In [2]: s2 = 'birthday'

In [3]: s1 += ' ' + s2

In [4]: s1
```

```
Out[4]: 'happy birthday'

In [5]: symbol = '>'

In [6]: symbol *= 5

In [7]: symbol
Out[7]: '>>>>>'
```

8.4. Удаление пропусков из строк

Строки поддерживают несколько методов, предназначенных для удаления пропусков в конце строки. Каждый метод возвращает новую строку, а исходная строка остается без изменений. Строки неизменяемы, поэтому каждый метод, который на первый взгляд изменяет строку, в действительности возвращает новую строку.

Удаление начальных и конечных пропусков

Метод `strip` используется для удаления начальных и конечных пропусков из строк:

```
In [1]: sentence = '\t  \n  This is a test string. \t\t \n'

In [2]: sentence.strip()
Out[2]: 'This is a test string.'
```

Удаление начальных пропусков

Метод `lstrip` удаляет только начальные пропуски:

```
In [3]: sentence.lstrip()
Out[3]: 'This is a test string. \t\t \n'
```

Удаление конечных пропусков

Метод `rstrip` удаляет только конечные пропуски:

```
In [4]: sentence.rstrip()
Out[4]: '\t  \n  This is a test string.'
```

Как видно из вывода, методы удаляют все разновидности пропусков, включая пробелы, символы новой строки и табуляции.

8.5. Изменение регистра символов

В предыдущих главах методы строк `lower` и `upper` использовались для преобразования строк, после чего они содержали только символы нижнего (верхнего) регистра. Можно изменить регистр символов и методами `capitalize` и `title`.

Изменение регистра первого символа строки

Метод `capitalize` копирует исходную строку и возвращает новую строку, в которой преобразован к верхнему регистру только первый символ:

```
In [1]: 'happy birthday'.capitalize()
Out[1]: 'Happy birthday'
```

Изменение регистра первого символа каждого слова в строке

Метод `title` копирует исходную строку и возвращает новую строку, в которой преобразован к верхнему регистру первый символ каждого слова:

```
In [2]: 'strings: a deeper look'.title()
Out[2]: 'Strings: A Deeper Look'
```

8.6. Операторы сравнения для строк

Строки можно сравнивать при помощи операторов сравнения. Напомним, при сравнении строк учитываются используемые ими целочисленные значения. Таким образом, при сравнении буквы верхнего регистра «меньше» букв нижнего регистра, потому что буквам верхнего регистра соответствуют меньшие числовые значения. Например, букве `'A'` соответствует числовой код `65`, а букве `'a'` — код `97`. Вы уже знаете, что код символа можно проверить функцией `ord`:

```
In [1]: print(f'A: {ord("A")}; a: {ord("a")}')
A: 65; a: 97
```

Сравним строки `'Orange'` и `'orange'` операторами сравнения:

```
In [2]: 'Orange' == 'orange'
Out[2]: False

In [3]: 'Orange' != 'orange'
Out[3]: True
```

```
In [4]: 'Orange' < 'orange'
Out[4]: True

In [5]: 'Orange' <= 'orange'
Out[5]: True

In [6]: 'Orange' > 'orange'
Out[6]: False

In [7]: 'Orange' >= 'orange'
Out[7]: False
```

8.7. Поиск подстрок

Методы строк позволяют провести поиск в строке одного или нескольких смежных символов (то есть *подстроки*). Вы можете подсчитать вхождения подстроки, определить, содержит ли строка заданную подстроку, или определить индекс, с которого подстрока входит в строку. Каждый метод, представленный в этом разделе, осуществляет лексикографическое сравнение символов по их числовым значениям.

Подсчет вхождений

Метод `count` возвращает количество вхождений аргумента в строке, для которой вызывается метод:

```
In [1]: sentence = 'to be or not to be that is the question'

In [2]: sentence.count('to')
Out[2]: 2
```

Если указать во втором аргументе *начальный_индекс*, то поиск ограничивается сегментом *строка*`[`*начальный_индекс*`:]`, то есть от *начального_индекса* до конца строки:

```
In [3]: sentence.count('to', 12)
Out[3]: 1
```

Если указаны второй и третий аргументы, то поиск ограничивается сегментом *строка*`[`*начальный_индекс*`:`*конечный_индекс*`]`, то есть от *начального_индекса* до *конечного_индекса*, исключая последний:

```
In [4]: sentence.count('that', 12, 25)
Out[4]: 1
```

Как и в случае с `count`, все методы строк, представленные в этом разделе, могут получать аргументы *начальный_индекс* и *конечный_индекс* для ограничения поиска сегментом исходной строки.

Поиск подстроки в строке

Метод `index` ищет подстроку в строке и возвращает первый индекс, по которому она была найдена; если поиск не принес результатов, то происходит ошибка `ValueError`:

```
In [5]: sentence.index('be')
Out[5]: 3
```

Метод `rindex` выполняет ту же операцию, что и `index`, но проводит поиск от конца строки и возвращает *последний* индекс, по которому была найдена подстрока; если поиск не принес результатов, происходит ошибка `ValueError`:

```
In [6]: sentence.rindex('be')
Out[6]: 16
```

Методы строк `find` и `rfind` выполняют те же задачи, что и `index` с `rindex`, но если подстрока не найдена, методы возвращают –1 вместо ошибки `ValueError`.

Проверка вхождения подстроки в строку

Чтобы выяснить, содержит строка заданную подстроку или нет, воспользуйтесь оператором `in` или `not in`:

```
In [7]: 'that' in sentence
Out[7]: True

In [8]: 'THAT' in sentence
Out[8]: False

In [9]: 'THAT' not in sentence
Out[9]: True
```

Поиск подстроки в начале или в конце строки

Методы `startswith` и `endswith` возвращают `True`, если строка начинается или заканчивается заданной подстрокой:

```
In [10]: sentence.startswith('to')
Out[10]: True
```

```
In [11]: sentence.startswith('be')
Out[11]: False

In [12]: sentence.endswith('question')
Out[12]: True

In [13]: sentence.endswith('quest')
Out[13]: False
```

8.8. Замена подстрок

Поиск подстроки с ее заменой входит в число стандартных операций с текстом. Метод `replace` получает две подстроки. Он ищет в строке подстроку, заданную первым аргументом, и заменяет *каждое* ее вхождение подстрокой, заданной вторым аргументом. Метод возвращает новую строку с результатами. Заменим символы табуляции в строке запятыми:

```
In [1]: values = '1\t2\t3\t4\t5'

In [2]: values.replace('\t', ',')
Out[2]: '1,2,3,4,5'
```

Метод `replace` может получать необязательный третий аргумент, определяющий максимальное количество выполняемых замен.

8.9. Разбиение и объединение строк

Когда вы читаете предложение, ваш мозг разбивает его на отдельные слова, или *лексемы*; каждая лексема передает значение. Интерпретаторы, такие как IPython, тоже разбивают предложения на лексемы, то есть на отдельные компоненты: ключевые слова, идентификаторы, операторы и др. Лексемы обычно разделяются символами-пропусками (пробелы, табуляции, символы новой строки), хотя вместо них могут использоваться другие символы, называемые *ограничителями* (*delimiters*).

Разбиение строк

Метод `split` *без* аргументов разбивает строку на подстроки по символам-пропускам, возвращая список лексем. Чтобы выполнить разбиение строки по нестандартному ограничителю (например, по парам «запятая и пробел»),

укажите строку-ограничитель (например, `', '`), которая должна использоваться при разбиении строки:

```
In [1]: letters = 'A, B, C, D'

In [2]: letters.split(', ')
Out[2]: ['A', 'B', 'C', 'D']
```

Если во втором аргументе передается целое число, то оно задает максимальное количество разбиений. Последней лексемой становится оставшаяся часть строки после максимального количества разбиений:

```
In [3]: letters.split(', ', 2)
Out[3]: ['A', 'B', 'C, D']
```

Также существует метод `rsplit`, который делает то же самое, что `split`, но максимальное количество разбиений отсчитывается от конца строки к началу.

Объединение строк

Метод `join` выполняет конкатенацию строк в своем аргументе, в котором должен передаваться итерируемый объект, содержащий только строковые значения; в противном случае происходит ошибка `TypeError`. Разделителем между объединяемыми строками становится строка, для которой вызывается `join`. В следующем коде создаются строки со списками значений, разделенных запятыми:

```
In [4]: letters_list = ['A', 'B', 'C', 'D']

In [5]: ','.join(letters_list)
Out[5]: 'A,B,C,D'
```

Следующий фрагмент объединяет результаты трансформации списка, который создает список строк:

```
In [6]: ','.join([str(i) for i in range(10)])
Out[6]: '0,1,2,3,4,5,6,7,8,9'
```

В главе 9 вы научитесь работать с файлами, содержащими значения, разделенные запятыми. Такие файлы, называемые *CSV-файлами* (сокращение от «Comma Separated Values»), являются стандартным форматом хранения данных электронных таблиц (вроде Microsoft Excel или Google Sheets). В главах с практическими примерами data science многие ключевые библиотеки (например, NumPy, Pandas и Seaborn) предоставляют встроенные средства для работы с CSV-данными.

Методы partition и rpartition

Метод `partition` разбивает строку на кортеж из трех строк по *разделителю*, передаваемому в аргументе. Кортеж содержит следующие строки:

- часть исходной строки перед разделителем;
- сам разделитель;
- часть строки после разделителя.

Эта возможность может пригодиться при разбиении более сложных строк. Возьмем строку с именем и оценками студента:

```
'Amanda: 89, 97, 92'
```

Разобьем исходную строку на имя студента, разделитель `': '` и строку, представляющую список оценок:

```
In [7]: 'Amanda: 89, 97, 92'.partition(': ')
Out[7]: ('Amanda', ': ', '89, 97, 92')
```

Чтобы поиск разделителя проводился от конца строки, используйте метод `rpartition`. Для примера возьмем следующую строку с URL-адресом:

```
'http://www.deitel.com/books/PyCDS/table_of_contents.html'
```

Воспользуемся методом `rpartition`, чтобы отделить часть `'table_of_contents.html'` от остатка URL-адреса:

```
In [8]: url = 'http://www.deitel.com/books/PyCDS/table_of_contents.html'

In [9]: rest_of_url, separator, document = url.rpartition('/')

In [10]: document
Out[10]: 'table_of_contents.html'

In [11]: rest_of_url
Out[11]: 'http://www.deitel.com/books/PyCDS'
```

Метод splitlines

В главе 9 мы займемся чтением текста из файла. Возможно, при чтении больших объемов текста в строку вы захотите разбить данные на список по символам новой строки. Метод `splitlines` возвращает список новых строк, представляющих внутренние строки текста, разбитого по символам новой строки в исходном тексте. Вспомните, что Python хранит многострочный

текст с внутренними символами \n, представляющими разрывы строк, как показано во фрагменте [13]:

```
In [12]: lines = """This is line 1
    ...: This is line2
    ...: This is line3"""

In [13]: lines
Out[13]: 'This is line 1\nThis is line2\nThis is line3'

In [14]: lines.splitlines()
Out[14]: ['This is line 1', 'This is line2', 'This is line3']
```

Если при вызове `splitlines` передается аргумент `True`, то в конце каждого элемента списка остается символ новой строки:

```
In [15]: lines.splitlines(True)
Out[15]: ['This is line 1\n', 'This is line2\n', 'This is line3']
```

8.10. Символы и методы проверки символов

Во многих языках программирования символы и строки представлены разными типами. В Python символ представляет собой строку из одного символа.

Python предоставляет методы строк для проверки того, обладает ли строка некоторыми характеристиками. Так, метод `isdigit` возвращает `True`, если строка, для которой вызван метод, состоит только из цифровых символов (`0`–`9`). С помощью этого метода можно проверить, что пользовательский ввод состоит из одних цифр:

```
In [1]: '-27'.isdigit()
Out[1]: False

In [2]: '27'.isdigit()
Out[2]: True
```

Метод `isalnum` возвращает `True`, если строка, для которой вызывается метод, является алфавитно-цифровой, то есть состоит только из цифр и букв:

```
In [3]: 'A9876'.isalnum()
Out[3]: True

In [4]: '123 Main Street'.isalnum()
Out[4]: False
```

В табл. 8.2 перечислены многие методы проверки символов. Каждый метод возвращает `False`, если описанное условие не выполняется.

Таблица 8.2. Методы проверки символов

Метод строк	Описание
`isalnum()`	Возвращает `True`, если строка состоит только из *алфавитно-цифровых* символов (то есть только из букв и цифр)
`isalpha()`	Возвращает `True`, если строка состоит только из *алфавитных* символов (то есть только из букв)
`isdecimal()`	Возвращает `True`, если строка состоит только из *десятичных* цифр (то есть цифр десятичной системы счисления) и не содержит знаков + или -
`isdigit()`	Возвращает `True`, если строка состоит только из цифр (например, '0', '1', '2')
`isidentifier()`	Возвращает `True`, если строка представляет действительный *идентификатор*
`islower()`	Возвращает `True`, если все алфавитные символы в строке относятся к *нижнему регистру* (например, `'a'`, `'b'`, `'c'`)
`isnumeric()`	Возвращает `True`, если символы в строке представляют *числовое значение* без знака + или - и без точки-разделителя дробной части
`isspace()`	Возвращает `True`, если строка содержит только символы-*пропуски*
`istitle()`	Возвращает `True`, если символами *верхнего регистра* в строке являются только первые символы каждого слова
`isupper()`	Возвращает `True`, если все алфавитные символы в строке относятся к *верхнему регистру* (например, `'A'`, `'B'`, `'C'`)

8.11. Необработанные строки

Напомним, символы \ (обратный слеш) в строках служат началом *служебных последовательностей* — например, `\n` для символа новой строки или `\t` для табуляции. Для включения символа \ в строку требуется использовать два символа `\\`. Это усложняет чтение некоторых строк. Например, в Microsoft Windows символ \ используется для разделения имен каталогов при определении местоположения файла. Чтобы указать путь к файлу в Windows, используйте запись вида:

```
In [1]: file_path = 'C:\\MyFolder\\MySubFolder\\MyFile.txt'

In [2]: file_path
Out[2]: 'C:\\MyFolder\\MySubFolder\\MyFile.txt'
```

В таких случаях удобнее использовать *необработанные строки* (raw strings), начинающиеся с префикса `r`. В таких строках каждый символ `\` рассматривается как обычный символ, а не как начало служебной последовательности:

```
In [3]: file_path = r'C:\MyFolder\MySubFolder\MyFile.txt'

In [4]: file_path
Out[4]: 'C:\\MyFolder\\MySubFolder\\MyFile.txt'
```

Как показывает последний фрагмент, Python преобразует необработанные строки в обычные строки, которые используют удвоенные символы `\` в своем внутреннем представлении. Необработанные строки упрощают чтение вашего кода, особенно при использовании регулярных выражений, о которых речь пойдет в следующем разделе. Регулярные выражения часто содержат много символов `\`.

8.12. Знакомство с регулярными выражениями

Иногда возникает необходимость в поиске в тексте *определенных структур* — телефонных номеров, адресов электронной почты, почтовых кодов, адресов веб-страниц, номеров социального страхования и т. д. Строка с *регулярным выражением* описывает *шаблон для поиска совпадений* в других строках.

Регулярные выражения помогают извлекать данные из неструктурированного текста, например сообщений в социальных сетях. Они также помогают удостовериться в правильности формата данных перед их обработкой[1].

Проверка данных

Прежде чем работать с текстовыми данными, вы будете часто применять регулярные выражения для их *проверки*. Так, при помощи последних можно проверить, что:

[1] Тема регулярных выражений может показаться более сложной, чем многие другие средства Python, которые мы использовали ранее. После того как вы освоите эту тему, вы сможете писать более компактный код, чем с традиционными средствами обработки строк, а это ускорит процесс разработки. Также ваш код будет обрабатывать «граничные» случаи, которые вы могли упустить, — и возможно, это позволит избежать коварных ошибок.

- почтовый код США состоит из пяти цифр (например, `02215`) или пяти цифр, за которыми следует дефис и еще четыре цифры (например, `02215-4775`);
- фамилия состоит только из букв, пробелов, апострофов и дефисов;
- адрес электронной почты состоит только из допустимых символов в допустимом порядке;
- номер социального страхования в США состоит из трех цифр, дефиса, двух цифр, дефиса и четырех цифр и соответствует другим правилам относительно цифр, которые могут входить в каждую группу цифр.

Вам не придется создавать ваши регулярные выражения для таких стандартных конструкций. На веб-сайтах:

- https://regex101.com
- http://www.regexlib.com
- https://www.regular-expressions.info

и других сайтах имеются репозитории готовых регулярных выражений, которые вы можете копировать и использовать для своих целей. Многие сайты также предоставляют интерфейсы, в которых вы сможете протестировать регулярные выражения и проверить, соответствуют ли они вашим потребностям.

Другие применения регулярных выражений

Кроме проверки данных, регулярные выражения часто используются для следующих целей:

- извлечение данных из текста, например поиск всех URL-адресов на веб-странице (возможно, для решения этой задачи вы предпочтете такие инструменты, как BeautifulSoup, XPath и lxml);
- очистка данных, например удаление необязательных данных, удаление дубликатов, обработка неполных данных, исправление опечаток, проверка целостности форматов данных, обработка выбросов и т. д.;
- преобразование данных в другие форматы, например переформатирование данных, собранных в формате значений, разделенных табуляциями (пробелами), в формат CSV для приложения, которому требуются данные в формате CSV.

8.12.1. Модуль re и функция fullmatch

Чтобы использовать регулярные выражения, импортируйте модуль `re` стандартной библиотеки Python:

```
In [1]: import re
```

Одна из простейших функций регулярных выражений `fullmatch` проверяет, совпадает ли шаблон, заданный первым аргументом, со *всей* строкой, заданной вторым аргументом.

Проверка совпадения для литеральных символов

Начнем с проверки совпадений для *литеральных символов*, то есть символов, которые совпадают сами с собой:

```
In [2]: pattern = '02215'

In [3]: 'Match' if re.fullmatch(pattern, '02215') else 'No match'
Out[3]: 'Match'

In [4]: 'Match' if re.fullmatch(pattern, '51220') else 'No match'
Out[4]: 'No match'
```

Первым аргументом функции является регулярное выражение — шаблон, для которого проверяется совпадение в строке. Любая строка может быть регулярным выражением. Значение переменной `pattern '02215'` состоит из *цифровых литералов*, которые совпадают только *сами с собой* в заданном порядке. Во втором аргументе передается строка, с которой должен полностью совпасть шаблон.

Если шаблон из первого аргумента совпадает со строкой из второго аргумента, `fullmatch` возвращает объект с текстом совпадения, который интерпретируется как `True`. Позднее мы более подробно расскажем об этом объекте. Во фрагменте [4] второй аргумент содержит те же цифры, но эти цифры следуют в *другом* порядке. Таким образом, совпадения нет, а `fullmatch` возвращает `None`, что интерпретируется как `False`.

Метасимволы, символьные классы и квантификаторы

Регулярные выражения обычно содержат различные специальные символы, которые называются *метасимволами*:

```
[] {} () \ * + ^ $ ? . |
```

С *метасимвола* \ начинается каждый из предварительно определенных *символьных классов*, каждый из которых совпадает с символом из конкретного набора. Проверим, что почтовый код состоит из пяти цифр:

```
In [5]: 'Valid' if re.fullmatch(r'\d{5}', '02215') else 'Invalid'
Out[5]: 'Valid'

In [6]: 'Valid' if re.fullmatch(r'\d{5}', '9876') else 'Invalid'
Out[6]: 'Invalid'
```

В регулярном выражении `\d{5}` `\d` является символьным классом, представляющим цифру (0–9). Символьный класс — *служебная последовательность в регулярном выражении*, совпадающая с *одним* символом. Чтобы совпадение могло состоять из нескольких символов, за символьным классом следует указать *квантификатор*. Квантификатор `{5}` повторяет `\d` пять раз, как если бы мы использовали запись `\d\d\d\d\d` для совпадения с пятью последовательными цифрами. Во фрагменте [6] `fullmatch` возвращает `None`, потому что `'9876'` совпадает только с четырьмя последовательными цифровыми символами.

Другие предопределенные символьные классы

В табл. 8.3 перечислены некоторые предопределенные символьные классы и группы символов, с которыми они совпадают. Чтобы любой метасимвол совпадал со своим *литеральным* значением, поставьте перед ним символ \ (обратный слеш). Например, `\\` совпадает с обратным слешем (\), а `\$` совпадает со знаком `$`.

Таблица 8.3. Некоторые предопределенные символьные классы и группы символов, с которыми они совпадают

Символьный класс	Совпадение
`\d`	Любая цифра (0–9)
`\D`	Любой символ, *кроме* цифр
`\s`	Любой символ-пропуск (пробелы, табуляции, новые строки)
`\S`	Любой символ, *кроме* пропусков
`\w`	Любой символ слова (также называемый *алфавитно-цифровым символом*) — то есть любая буква верхнего или нижнего регистра, любая цифра или символ подчеркивания
`\W`	Любой символ, *кроме* символов слов

Пользовательские символьные классы

Квадратные скобки `[]` определяют *пользовательский символьный класс*, совпадающий с *одним* символом. Так, `[aeiou]` совпадает с гласной буквой нижнего регистра, `[A-Z]` — с буквой верхнего регистра, `[a-z]` — с буквой нижнего регистра и `[a-zA-Z]` — с любой буквой нижнего (верхнего) регистра.

Выполним простую проверку имени — последовательности букв без пробелов или знаков препинания. Проверим, что последовательность начинается с буквы верхнего регистра (`A–Z`), а за ней следует произвольное количество букв нижнего регистра (`a–z`):

```
In [7]: 'Valid' if re.fullmatch('[A-Z][a-z]*', 'Wally') else 'Invalid'
Out[7]: 'Valid'

In [8]: 'Valid' if re.fullmatch('[A-Z][a-z]*', 'eva') else 'Invalid'
Out[8]: 'Invalid'
```

Имя может содержать неизвестное заранее количество букв. *Квантификатор* `*` совпадает с *нулем и более вхождениями* подвыражения, находящегося слева (в данном случае `[a-z]`). Таким образом, `[A-Z][a-z]*` совпадает с буквой верхнего регистра, за которой следует *нуль и более* букв нижнего регистра (например, `'Amanda'`, `'Bo'` и даже `'E'`).

Если пользовательский символьный класс начинается с символа `^` (крышка), то класс совпадает с любым символом, который *не* подходит под определение из класса. Таким образом, `[^a-z]` совпадает с любым символом, который не является буквой нижнего регистра:

```
In [9]: 'Match' if re.fullmatch('[^a-z]', 'A') else 'No match'
Out[9]: 'Match'

In [10]: 'Match' if re.fullmatch('[^a-z]', 'a') else 'No match'
Out[10]: 'No match'
```

Метасимволы в пользовательском символьном классе интерпретируются как литеральные символы, то есть как сами символы, не имеющие специального смысла. Таким образом, символьный класс `[*+$]` совпадает с *одним* из символов `*`, `+` или `$`:

```
In [11]: 'Match' if re.fullmatch('[*+$]', '*') else 'No match'
Out[11]: 'Match'

In [12]: 'Match' if re.fullmatch('[*+$]', '!') else 'No match'
Out[12]: 'No match'
```

Квантификаторы * и +

Для того чтобы имя содержало *хотя бы одну* букву нижнего регистра, квантификатор * во фрагменте [7] можно заменить знаком +, который совпадает *по крайней мере с одним вхождением* подвыражения:

```
In [13]: 'Valid' if re.fullmatch('[A-Z][a-z]+', 'Wally') else 'Invalid'
Out[13]: 'Valid'

In [14]: 'Valid' if re.fullmatch('[A-Z][a-z]+', 'E') else 'Invalid'
Out[14]: 'Invalid'
```

Квантификаторы * и + являются *максимальными* («жадными») — они совпадают с максимально возможным количеством символов. Таким образом, регулярные выражения `[A-Z][a-z]+` совпадают с именами `'Al'`, `'Eva'`, `'Samantha'`, `'Benjamin'` и любыми другими словами, начинающимися с буквы верхнего регистра, за которой следует хотя бы одна буква нижнего регистра.

Другие квантификаторы

Квантификатор `?` совпадает с *нулем или одним вхождением* подвыражения:

```
In [15]: 'Match' if re.fullmatch('labell?ed', 'labelled') else 'No match'
Out[15]: 'Match'

In [16]: 'Match' if re.fullmatch('labell?ed', 'labeled') else 'No match'
Out[16]: 'Match'

In [17]: 'Match' if re.fullmatch('labell?ed', 'labellled') else 'No
match'
Out[17]: 'No match'
```

Регулярное выражение `labell?ed` совпадает со словами `labelled` и `labeled`, но не с ошибочно написанным словом `labellled`. В каждом из приведенных выше фрагментов первые пять литеральных символов регулярного выражения (`label`) совпадают с первыми пятью символами второго аргумента. Часть `l?` означает, что оставшимся литеральным символам `ed` может предшествовать *нуль или один* символ `l`.

Квантификатор {n,} совпадает *не менее чем с n* вхождениями подвыражения. Следующее регулярное выражение совпадает со строками, содержащими *не менее* трех цифр:

```
In [18]: 'Match' if re.fullmatch(r'\d{3,}', '123') else 'No match'
Out[18]: 'Match'
```

```
In [19]: 'Match' if re.fullmatch(r'\d{3,}', '1234567890') else 'No match'
Out[19]: 'Match'

In [20]: 'Match' if re.fullmatch(r'\d{3,}', '12') else 'No match'
Out[20]: 'No match'
```

Чтобы совпадение включало *от n до m* (*включительно*) вхождений, используйте *квантификатор {n,m}*. Следующее регулярное выражение совпадает со строками, содержащими от 3 до 6 цифр:

```
In [21]: 'Match' if re.fullmatch(r'\d{3,6}', '123') else 'No match'
Out[21]: 'Match'

In [22]: 'Match' if re.fullmatch(r'\d{3,6}', '123456') else 'No match'
Out[22]: 'Match'

In [23]: 'Match' if re.fullmatch(r'\d{3,6}', '1234567') else 'No match'
Out[23]: 'No match'

In [24]: 'Match' if re.fullmatch(r'\d{3,6}', '12') else 'No match'
Out[24]: 'No match'
```

8.12.2. Замена подстрок и разбиение строк

Модуль `re` предоставляет функцию `sub` для замены совпадений шаблона в строке, а также функцию `split` для разбиения строки на фрагменты на основании шаблонов.

Функция sub — замена совпадений шаблона

По умолчанию *функция* `sub` модуля `re` заменяет *все* вхождения шаблона заданным текстом. Преобразуем строку, разделенную табуляциями, в формат с разделением запятыми:

```
In [1]: import re

In [2]: re.sub(r'\t', ', ', '1\t2\t3\t4')
Out[2]: '1, 2, 3, 4'
```

Функция `sub` получает три обязательных аргумента:

- *шаблон для поиска* (символ табуляции `'\t'`);
- *текст замены* (`', '`);
- *строка, в которой ведется поиск* (`'1\t2\t3\t4'`),

и возвращает новую строку. Ключевой аргумент `count` может использоваться для определения максимального количества замен:

```
In [3]: re.sub(r'\t', ', ', '1\t2\t3\t4', count=2)
Out[3]: '1, 2, 3\t4'
```

Функция split

Функция `split` *разбивает* строку на лексемы, используя регулярное выражение для определения *ограничителя*, и возвращает список строк. Разобьем строку по запятым, за которыми следует 0 или более пропусков — для обозначения пропусков используется символьный класс `\s`, а * обозначает *0 и более* вхождений предшествующего подвыражения:

```
In [4]: re.split(r',\s*', '1,  2,  3,4,    5,6,7,8')
Out[4]: ['1', '2', '3', '4', '5', '6', '7', '8']
```

Ключевой аргумент `maxsplit` задает максимальное количество разбиений:

```
In [5]: re.split(r',\s*', '1,  2,  3,4,    5,6,7,8', maxsplit=3)
Out[5]: ['1', '2', '3', '4,    5,6,7,8']
```

В данном случае после трех разбиений четвертая строка содержит остаток исходной строки.

8.12.3. Другие функции поиска, обращение к совпадениям

Ранее мы использовали функцию `fullmatch` для определения того, совпала ли *вся* строка с регулярным выражением. Но существует и ряд других функций поиска совпадений. В этом разделе мы рассмотрим функции `search`, `match`, `findall` и `finditer` и покажем, как обратиться к подстрокам совпадений.

Функция search — поиск первого совпадения в строке

Функция `search` ищет в строке *первое* вхождение подстроки, совпадающей с регулярным выражением, и возвращает *объект совпадения* (типа `SRE_Match`), содержащий подстроку с совпадением. Метод `group` объекта совпадения возвращает эту подстроку:

```
In [1]: import re

In [2]: result = re.search('Python', 'Python is fun')
```

```
In [3]: result.group() if result else 'not found'
Out[3]: 'Python'
```

Если совпадение шаблона в строке *не* найдено, то функция `search` возвращает `None`:

```
In [4]: result2 = re.search('fun!', 'Python is fun')

In [5]: result2.group() if result2 else 'not found'
Out[5]: 'not found'
```

Функция `match` ищет совпадение только *от начала* строки.

Игнорирование регистра символов при помощи необязательного ключевого аргумента flags

Многие функции модуля `re` получают необязательный ключевой аргумент `flags`, который изменяет способ поиска совпадений для регулярного выражения. Например, по умолчанию при поиске учитывается *регистр символов*, но при помощи константы `IGNORECASE` модуля `re` можно провести поиск *без учета регистра*:

```
In [6]: result3 = re.search('Sam', 'SAM WHITE', flags=re.IGNORECASE)

In [7]: result3.group() if result3 else 'not found'
Out[7]: 'SAM'
```

Подстрока `'SAM'` совпадает с шаблоном `'Sam'`, потому что обе строки состоят из одинаковых букв, хотя `'SAM'` содержит буквы только верхнего регистра.

Метасимволы, ограничивающие совпадение началом или концом строки

Метасимвол `^` в начале регулярного выражения (и не в квадратных скобках) — якорь, указывающий, что выражение совпадает только *от начала* строки:

```
In [8]: result = re.search('^Python', 'Python is fun')

In [9]: result.group() if result else 'not found'
Out[9]: 'Python'

In [10]: result = re.search('^fun', 'Python is fun')

In [11]: result.group() if result else 'not found'
Out[11]: 'not found'
```

Аналогичным образом *символ* `$` в конце регулярного выражения является якорем, указывающим, что выражение совпадает только *в конце* строки:

```
In [12]: result = re.search('Python$', 'Python is fun')

In [13]: result.group() if result else 'not found'
Out[13]: 'not found'

In [14]: result = re.search('fun$', 'Python is fun')

In [15]: result.group() if result else 'not found'
Out[15]: 'fun'
```

Функции findall и finditer — поиск всех совпадений в строке

Функция `findall` находит *все* совпадающие подстроки и возвращает список совпадений. Для примера извлечем все телефонные номера в строке, полагая, что телефонные номера в США записываются в форме `###-###-####`:

```
In [16]: contact = 'Wally White, Home: 555-555-1234, Work: 555-555-4321'

In [17]: re.findall(r'\d{3}-\d{3}-\d{4}', contact)
Out[17]: ['555-555-1234', '555-555-4321']
```

Функция `finditer` работает аналогично `findall`, но возвращает *итерируемый* объект, содержащий объекты совпадений, с отложенным вычислением. При большом количестве совпадений использование `finditer` позволит сэкономить память, потому что она возвращает по одному совпадению, тогда как `findall` возвращает все совпадения сразу:

```
In [18]: for phone in re.finditer(r'\d{3}-\d{3}-\d{4}', contact):
    ...:     print(phone.group())
    ...:
555-555-1234
555-555-4321
```

Сохранение подстрок в совпадениях

Метасимволы `(` и `)` (*круглые скобки*) используются для сохранения подстрок в совпадениях. Для примера сохраним отдельно имя и адрес электронной почты в тексте строки:

```
In [19]: text = 'Charlie Cyan, e-mail: demo1@deitel.com'

In [20]: pattern = r'([A-Z][a-z]+ [A-Z][a-z]+), e-mail: (\w+@\w+\.\w{3})'

In [21]: result = re.search(pattern, text)
```

Регулярное выражение задает две сохраняемые подстроки, заключенные в метасимволы (и). Эти метасимволы *не* влияют на то, в каком месте текста строки будет найдено совпадение шаблона, — функция `match` возвращает объект совпадения *только* в том случае, если совпадение *всего* шаблона будет найдено в тексте строки.

Рассмотрим регулярное выражение по частям:

- `'([A-Z][a-z]+ [A-Z][a-z]+)'` совпадает с двумя словами, разделенными пробелом. Каждое слово должно начинаться с буквы верхнего регистра.
- `', e-mail: '` содержит литеральные символы, которые совпадают сами с собой.
- `(\w+@\w+\.\w{3})` совпадает с *простым* адресом электронной почты, состоящим из одного или нескольких алфавитно-цифровых символов (`\w+`), символа `@`, одного или нескольких алфавитно-цифровых символов (`\w+`), точки (`\.`) и трех алфавитно-цифровых символов (`\w{3}`). Перед точкой ставится символ `\`, потому что точка (`.`) в регулярных выражениях является метасимволом, совпадающим с одним символом.

Метод `groups` объекта совпадения возвращает кортеж совпавших подстрок:

```
In [22]: result.groups()
Out[22]: ('Charlie Cyan', 'demo1@deitel.com')
```

Метод `group` объекта совпадения возвращает *все* совпадения в виде одной строки:

```
In [23]: result.group()
Out[23]: 'Charlie Cyan, e-mail: demo1@deitel.com'
```

Вы можете обратиться к каждой сохраненной строке, передав целое число методу `group`. *Нумерация* сохраненных подстрок *начинается с 1* (в отличие от индексов списков, которые начинаются с 0):

```
In [24]: result.group(1)
Out[24]: 'Charlie Cyan'

In [25]: result.group(2)
Out[25]: 'demo1@deitel.com'
```

8.13. Введение в data science: pandas, регулярные выражения и первичная обработка данных

Данные не всегда поступают в форме, готовой для анализа. Например, они могут иметь неправильный формат, быть ошибочными, а то и вовсе отсутствовать. Опыт показывает, что специалисты по data science тратят до 75% своего времени на подготовку данных перед началом анализа. Подготовка данных называется *первичной обработкой*, два важнейших шага которой заключаются в *очистке данных* и последующем *преобразовании данных* в форматы, оптимальные для ваших систем баз данных и аналитических программ. Несколько типичных примеров очистки данных:

- удаление наблюдений с отсутствующими значениями;
- замена отсутствующих значений подходящими значениями;
- удаление наблюдений с некорректными значениями;
- замена некорректных значений подходящими значениями;
- исключение выбросов (хотя в некоторых случаях их лучше оставить);
- устранение дубликатов (хотя некоторые дубликаты содержат действительную информацию);
- обработка с данными, целостность которых была нарушена

  и так далее.

Вероятно, вы уже думаете, что очистка данных — сложный и хлопотный процесс, в котором легко можно принять ошибочные решения, отрицательно сказывающиеся на ваших результатах. Да, все правильно. Когда мы дойдем до практических примеров data science в следующих главах, вы увидите, что data science в большей степени является *дисциплиной эмпирической*, как медицина, и в меньшей степени — дисциплиной теоретической, как теоретическая физика. В эмпирических дисциплинах выводы делаются на основании наблюдений и практического опыта. Например, многие лекарства, эффективно решающие современные медицинские проблемы, были получены на основе наблюдения за воздействием ранних версий этих лекарств на лабораторных животных, а в конечном итоге и на людей и постепенного уточнения ингредиентов и дозировок. Действия специалистов data science могут изменяться в зависимости от проекта, могут зависеть от качества

и природы данных, а также от развивающихся организационных и профессиональных стандартов.

Примеры типичных преобразований данных:

- удаление необязательных данных и *признаков* (о признаках будет более подробно рассказано в практических примерах data science);
- объединение взаимосвязанных признаков;
- формирование выборок данных для получения репрезентативного подмножества (в практических примерах data science вы увидите, что *случайная выборка* особенно эффективна в этом отношении; мы также объясним почему);
- стандартизация форматов данных;
- группировка данных

  и так далее.

Всегда целесообразно сохранить исходные данные.Некоторые примеры очистки и преобразования данных будут рассмотрены в контексте коллекций `Series` и `DataFrame` библиотеки pandas.

Очистка данных

Некорректные и отсутствующие значения в данных могут оказать значительное влияние на анализ данных. Некоторые специалисты data science протестуют против любых попыток вставки «разумных значений». Вместо этого они рекомендуют четко пометить отсутствующие данные и предоставить решение проблемы пакету аналитики данных. Другие советуют действовать более осторожно[1].

[1] Эта сноска написана на основе комментария, полученного нами 20 июля 2018 года от одного из рецензентов этой книги, доктора Элисон Санчес (Alison Sanchez) факультета экономики и предпринимательства Университета Сан-Диего. Она заметила: «При подстановке “разумных значений” на место отсутствующих или некорректных данных необходимо действовать очень осторожно. “Подставлять” значения, повышающие статистическую значимость или обеспечивающие более “разумные” или “лучшие” результаты, недопустимо. “Подстановка” данных не должна превратиться в “подтасовку” данных. Первое, чему должен научиться читатель, — не исключать и не изменять значения, противоречащие гипотезам. “Подстановка” разумных значений не означает, что читатель может изменять данные, чтобы получить нужный результат».

Представьте, что в больнице пациентам измеряют температуру (и возможно, другие показатели состояния организма) четыре раза в день. Данные состоят из имени и четырех значений с плавающей точкой:

```
['Brown, Sue', 98.6, 98.4, 98.7, 0.0]
```

Для этого пациента были зарегистрированы только три значения температуры (по шкале Фаренгейта) — `99.7`, `98.4` и `98.7` (возможно, из-за сбоя датчика.) Среднее первых трех значений равно 98,57, что близко к нормальной температуре. Но если при вычислении средней температуры *будет учтено* отсутствующее значение, замененное на `0.0`, то среднее будет равно всего `73.93` — весьма сомнительный результат. Конечно, врач не должен срочно отправлять такого пациента в реанимацию — очень важно привести данные «к разумному виду».

Один из распространенных способов очистки данных заключается в подстановке *разумного* значения на место отсутствующей температуры, например среднего значения других показаний температуры пациента. Если бы это было сделано в рассмотренном примере, то средняя температура пациента оставалась бы равной `98.57` — намного более вероятная средняя температура на основании других показаний.

Проверка данных

Начнем с создания коллекции `Series` почтовых кодов, состоящих из пяти цифр, на базе словаря пар «название-города/почтовый-код-из-5-цифр». Мы намеренно указали ошибочный индекс для Майами:

```
In [1]: import pandas as pd

In [2]: zips = pd.Series({'Boston': '02215', 'Miami': '3310'})

In [3]: zips
Out[3]:
Boston     02215
Miami       3310
dtype: object
```

Хотя `zips` может показаться двумерным массивом, в действительности это одномерный массив. «Второй столбец» представляет *значения* почтовых кодов из коллекции `Series` (из значений словаря), а «первый столбец» — их *индексы* (из ключей словаря).

Для проверки данных можно воспользоваться регулярными выражениями с pandas. *Атрибут* `str` коллекции `Series` предоставляет средства обработки строк и различные методы регулярных выражений. Чтобы проверить правильность каждого отдельного почтового кода, воспользуемся *методом* `match` атрибута `str`:

```
In [4]: zips.str.match(r'\d{5}')
Out[4]:
Boston      True
Miami      False
dtype: bool
```

Метод `match` применяет регулярное выражение `\d{5}` к *каждому* элементу `Series`, чтобы убедиться в том, что элемент состоит ровно из пяти цифр. Явно перебирать все почтовые коды в цикле не нужно — `match` сделает это за вас. Это еще один пример программирования в функциональном стиле с внутренними итерациями (в отличие от внешних). Метод возвращает новую коллекцию `Series`, содержащую значение `True` для каждого действительного элемента. В данном случае почтовый код Майами проверку *не* прошел, поэтому его элемент равен `False`.

С недопустимыми данными есть несколько вариантов действий. Первый — обнаружить их источник и взаимодействовать с источником, чтобы исправить ошибку. Это возможно не всегда. Например, данные могут поступить с высокоскоростных датчиков в «интернете вещей». В этом случае исправить ошибку на уровне источника не удастся, поэтому можно применить методы очистки данных. Например, в случае недействительного кода Майами 3310 можно поискать почтовые коды Майами, начинающиеся с 3310. Поиск возвращает два таких кода — `33101` и `33109`; можно выбрать один из них.

Иногда вместо того, чтобы проверять на совпадение шаблона *всю* строку, требуется узнать, содержит ли значение *подстроку*, совпадающую с шаблоном. В этом случае следует использовать метод `contains` вместо `match`. Создадим коллекцию `Series` строк, каждая из которых содержит название города в США, штата и почтовый код, а затем определим, содержит ли каждая строку подстроку, совпадающую с шаблоном `' [A-Z]{2} '` (пробел, за которым следуют две буквы верхнего регистра, и еще один пробел):

```
In [5]: cities = pd.Series(['Boston, MA 02215', 'Miami, FL 33101'])

In [6]: cities
Out[6]:
```

```
0    Boston, MA 02215
1     Miami, FL 33101
dtype: object

In [7]: cities.str.contains(r' [A-Z]{2} ')
Out[7]:
0    True
1    True
dtype: bool

In [8]: cities.str.match(r' [A-Z]{2} ')
Out[8]:
0    False
1    False
dtype: bool
```

Значения индексов не указаны, поэтому коллекция `Series` по умолчанию использует индексы, начинающиеся с 0 (фрагмент [6]). Фрагмент [7] использует функцию `contains`, чтобы показать, что оба элемента `Series` содержат подстроки, совпадающие с `' [A-Z]{2} '`. Фрагмент [8] использует `match` для проверки того, что значение элемента не совпадает с шаблоном полностью, потому что в полном значении каждого элемента присутствуют другие символы.

Переформатирование данных

От очистки данных перейдем к первичной обработке данных в другой формат. Возьмем простой пример: допустим, приложение работает с телефонными номерами США в формате `###-###-####`, с разделением групп цифр дефисами. При этом телефонные номера были предоставлены в виде строк из десяти цифр без дефисов. Создадим коллекцию `DataFrame`:

```
In [9]: contacts = [['Mike Green', 'demo1@deitel.com', '5555555555'],
   ...:             ['Sue Brown', 'demo2@deitel.com', '5555551234']]
   ...:

In [10]: contactsdf = pd.DataFrame(contacts,
    ...:                           columns=['Name', 'Email', 'Phone'])
    ...:

In [11]: contactsdf
Out[11]:
         Name             Email       Phone
0  Mike Green  demo1@deitel.com  5555555555
1   Sue Brown  demo2@deitel.com  5555551234
```

В этой коллекции `DataFrame` мы задали индексы столбцов ключевым аргументом `columns`, но *не* указали индексы строк, так что строки индексируются с 0. Кроме того, в выводе показаны значения столбцов, по умолчанию выравниваемые по правому краю. Ситуация отличается от форматирования Python, в котором числа по умолчанию выравниваются *по правому краю* поля, но нечисловые значения по умолчанию выравниваются *по левому краю*.

Теперь произведем первичную обработку данных с применением программирования в функциональном стиле. Телефонные номера можно *перевести* в правильный формат вызовом метода `map` коллекции `Series` для столбца `'Phone'` коллекции `DataFrame`. Аргументом метода `map` является *функция*, которая получает значение и возвращает *отображенное* (преобразованное) значение. Функция `get_formatted_phone` отображает десять последовательных цифр в формат `###-###-####`:

```
In [12]: import re

In [13]: def get_formatted_phone(value):
    ...:     result = re.fullmatch(r'(\d{3})(\d{3})(\d{4})', value)
    ...:     return '-'.join(result.groups()) if result else value
    ...:
    ...:
```

Регулярное выражение в первой команде блока совпадает *только* с первыми десятью последовательно идущими цифрами. Оно сохраняет подстроки, которые содержат первые три цифры, следующие три цифры и последние четыре цифры. Команда `return` работает следующим образом:

- Если результат равен `None`, то значение просто возвращается в неизменном виде.
- В противном случае вызывается метод `result.groups()` для получения кортежа, содержащего сохраненные подстроки. Кортеж передается методу `join` строк для выполнения конкатенации элементов, с разделением элементов символом `'-'` для формирования преобразованного телефонного номера.

Метод `map` коллекции `Series` новую коллекцию `Series`, которая содержит результаты вызова ее функции-аргумента для каждого значения в столбце. Фрагмент [15] выводит результаты, включающие имя и тип столбца:

```
In [14]: formatted_phone = contactsdf['Phone'].map(get_formatted_phone)

In [15]: formatted_phone
```

```
0      555-555-5555
1      555-555-1234
Name: Phone, dtype: object
```

Убедившись в том, что данные имеют правильный формат, можно обновить их в исходной коллекции `DataFrame`, присвоив новую коллекцию `Series` столбцу `'Phone'`:

```
In [16]: contactsdf['Phone'] = formatted_phone

In [17]: contactsdf
Out[17]:
         Name              Email         Phone
0  Mike Green  demo1@deitel.com  555-555-5555
1   Sue Brown  demo2@deitel.com  555-555-1234
```

Обсуждение pandas продолжится в разделе «Введение в data science» следующей главы. Кроме того, pandas будет использоваться и в других последующих главах.

8.14. Итоги

В этой главе были представлены различные средства форматирования и обработки строк. Мы занимались форматированием данных с использованием форматных строк и метода `format`. Вы увидели, как применять расширенное присваивание для конкатенации и повторения строк, научились удалять пропуски в начале и в конце строки, а также изменять регистр символов. Были рассмотрены дополнительные методы для разбиения строк и для объединения итерируемых объектов, содержащих строки. Также были представлены различные методы проверки символов.

Вы узнали, что в необработанных строках символы \ интерпретируются как литеральные символы, а не как начало служебных последовательностей. Необработанные строки особенно удобны для определения регулярных выражений, которые часто содержат много символов \.

Затем мы представили замечательные возможности поиска по шаблону, предоставляемые функциями регулярных выражений из модуля `re`. Функция `fullmatch` проверяет, что шаблон совпадает со всей строкой, и может пригодиться при проверке данных. Мы показали, как использовать функцию `replace` для поиска и замены подстрок. Функция `split` использовалась для разбиения

строк на лексемы по ограничителям, совпадающим с шаблоном регулярного выражения. Затем были представлены различные способы поиска совпадений шаблонов в строках и обращения к полученным совпадениям.

В разделе «Введение в data science» был описан процесс первичной обработки данных и продемонстрирован пример преобразования данных. Затем мы вернулись к коллекциям `Series` и `DataFrame` на примере использования регулярных выражений для проверки и первичной обработки данных.

Обсуждение средств обработки строк продолжится в следующей главе, когда мы займемся чтением текста из файлов и записью текста в файлы. Мы используем модуль `csv` для обработки файлов с данными, разделенными запятыми (CSV). Также будет представлен механизм обработки исключений, который позволяет обрабатывать ошибки при их возникновении вместо вывода трассировки.

9

Файлы и исключения

В этой главе...

- Файлы и долгосрочное хранение данных.
- Чтение, запись и обновление файлов.
- Чтение и запись CSV-файлов — наборов данных стандартного формата для машинного обучения.
- Сериализация объектов в формат JSON, часто применяемая для передачи данных по интернету, и десериализация JSON в объекты.
- Использование команды `with` для обеспечения гарантированного освобождения ресурсов и предотвращения «утечки ресурсов».
- Использование команды `try` для ограничения кода, в котором могут возникнуть исключения, и обработка этих исключений в соответствующих секциях `except`.
- Использование секции `else` команды `try` для выполнения кода, если в наборе `try` не возникают исключения.
- Использование секции `finally` команды `try` для выполнения кода независимо от того, возникло исключение в `try` или нет.
- Выдача исключений для обозначения возникающих проблем.
- Трассировка функций и методов, приведших к исключению.
- Использование pandas для загрузки в `DataFrame` и обработки набора данных катастрофы «Титаника» в формате CSV.

9.1. Введение

Переменные, списки, кортежи, словари, множества, массивы, коллекции `Series` и `DataFrame` из библиотеки pandas предоставляют возможность только *временного* хранения данных. Такие данные теряются, когда локальная переменная выходит «из области видимости» или при завершении программы. *Файлы* предоставляют средства для долгосрочного хранения больших объемов данных даже после завершения программы, создавшей эти данные. Компьютеры хранят файлы на устройствах вторичного хранения данных, включая твердотельные диски, жесткие диски и т. д. В этой главе мы расскажем, как программы Python создают, обновляют и обрабатывают файлы данных.

Нами рассматриваются текстовые файлы в нескольких популярных форматах — простой текст (по общепринятым соглашениям расширение .txt обозначает текстовый файл), JSON (JavaScript Object Notation) и CSV (значения, разделенные запятыми). Формат JSON будет использоваться для сериализации и десериализации объектов с целью упрощения сохранения этих объектов во вторичной памяти и передачи их по интернету. Непременно прочитайте раздел «Введение в data science», в котором модуль `csv` из стандартной библиотеки Python и pandas будут использоваться для загрузки и обработки CSV-данных. В частности, будет рассмотрена CSV-версия набора данных катастрофы «Титаника». Мы используем многие популярные наборы данных в практических примерах, посвященных обработке естественных языков, глубокому анализу данных Twitter, IBM Watson, машинному обучению, глубокому обучению и большим данным.

Так как в книге мы уделяем особое внимание безопасности Python, будут рассмотрены проблемы уязвимости при сериализации и десериализации данных при использовании модуля `pickle` стандартной библиотеки Python. Мы рекомендуем использовать сериализацию в формате JSON вместо `pickle`.

Кроме того, в главе будут описаны *средства обработки исключений*. Исключение свидетельствует о проблеме, возникшей во время выполнения. Вам уже встречались исключения типов `ZeroDivisionError`, `NameError`, `ValueError`, `StatisticsError`, `TypeError`, `IndexError`, `KeyError` и `RuntimeError`. Мы покажем, как *обрабатывать* возникающие исключения при помощи команд `try` и связанных с ними секций `except`. Будут рассмотрены и секции `else` и `finally` команды `try`. Средства, представленные в этой главе, помогут вам создавать *надежные, защищенные от ошибок* программы, которые могут справиться с возникшими проблемами и продолжить выполнение или *корректно завершиться*.

Программы обычно запрашивают и освобождают ресурсы (в частности, файлы) в ходе выполнения программы. Часто эти ресурсы ограниченны или могут использоваться только одной программой в любой момент времени. Мы покажем, как гарантировать, что после использования ресурса программой он будет освобожден для других программ даже в случае возникновения исключения (для решения этой задачи используется команда `with`).

9.2. Файлы

Python рассматривает *текстовый файл* как последовательность символов, а *двоичный файл* (графическое изображение, видео и т. д.) как последовательность байтов. Как и в списках и массивах, первый символ текстового файла и первый байт двоичного файла располагаются в позиции 0, так что в файле, содержащем n символов или байтов, наивысший номер позиции равен $n - 1$. На следующей диаграмме изображено концептуальное представление файла:

Для каждого *открытого* вами файла Python создает *объект файла*, который будет использоваться для работы с файлом.

Конец файла

Каждая операционная система предоставляет механизм обозначения конца файла. В некоторых ОС конец файла обозначается специальным маркером (как показано на диаграмме), тогда как другие системы поддерживают счетчик общего количества символов или байтов в файле. Языки программирования обычно скрывают от вас эти подробности реализации ОС.

Стандартные объекты файлов

В начале своего выполнения программа Python создает три *стандартных файл-объекта*:

- `sys.stdin` — *стандартный объект файла для ввода*;

- `sys.stdout` — *стандартный объект файла для вывода*;
- `sys.stderr` — *стандартный объект файла для ошибок.*

Хотя эти объекты считаются объектами файлов, по умолчанию они не читают и не записывают данные в файлы. Функция `input` неявно использует `sys.stdin` для получения пользовательского ввода с клавиатуры. Функция `print` неявно направляет вывод в `sys.stdout`, что соответствует командной строке. Python неявно направляет ошибки программы и трассировку в `sys.stderr`; эта информация также отображается в командной строке. Если потребуется явно обращаться к этим объектам в своем коде, то вам придется импортировать модуль `sys`, но такие ситуации встречаются нечасто.

9.3. Обработка текстовых файлов

В этом разделе мы запишем простой текстовый файл, который может использоваться системой управления расчетами с клиентами для отслеживания задолженностей клиентов. Затем мы прочитаем этот текстовый файл и убедимся в том, что он содержит данные. Для каждого клиента в файле будет храниться номер счета, фамилия и сумма задолженности. В совокупности эти поля данных представляют *запись* клиента. Python не определяет особую структуру файла, так что в Python не существует встроенной концепции записей. Программист должен самостоятельно структурировать файлы, чтобы они соответствовали требованиям его приложения. Мы будем создавать и вести этот файл по номеру счета. В определенном смысле номер счета может рассматриваться как *ключ записи*. В этой главе предполагается, что вы запустили IPython из папки примеров ch09.

9.3.1. Запись в текстовый файл: команда with

Создадим файл accounts.txt и запишем в него пять записей клиентов. В общем случае записи в текстовых файлах хранятся по одной на строку текста, поэтому каждая запись завершается символом новой строки:

```
In [1]: with open('accounts.txt', mode='w') as accounts:
   ...:     accounts.write('100 Jones 24.98\n')
   ...:     accounts.write('200 Doe 345.67\n')
   ...:     accounts.write('300 White 0.00\n')
   ...:     accounts.write('400 Stone -42.16\n')
   ...:     accounts.write('500 Rich 224.62\n')
   ...:
```

Данные также можно записать в файл вызовом `print` (который автоматически выводит завершающий символ `\n`):

```
print('100 Jones 24.98', file=accounts)
```

Команда with

Многие приложения *захватывают* ресурсы: файлы, сетевые подключения, подключения к базам данных и т. д. Как только необходимость в ресурсе отпала, его необходимо *освободить*. Это правило гарантирует, что ресурсами смогут воспользоваться другие приложения. *Команда* Python `with`:

- захватывает ресурс (в данном случае объект файла для accounts.txt) и присваивает соответствующий объект переменной (`accounts` в данном примере);
- позволяет приложению использовать ресурс через эту переменную;
- вызывает метод `close` для объекта ресурса, чтобы освободить ресурс при достижении конца набора команды `with` в программе.

Встроенная функция open

Встроенная *функция* `open` открывает файл accounts.txt и связывает его с объектом файла. Аргумент `mode` определяет *режим открытия файла*, то есть он указывает, будет ли файл открыт для чтения, для записи или для чтения/записи. Режим `'w'` открывает файл для *записи*; если файл не существует, то он будет создан. Если не указать путь к файлу, то Python создаст его в текущей папке (ch09). Будьте внимательны: открытие файла для записи приводит к *удалению* всех существующих данных в этом файле.

Запись в файл

Команда `with` присваивает объект, возвращенный `open`, переменной `accounts` из *секции* `as`. В наборе команды `with` переменная `accounts` используется для взаимодействия с файлом. В данном случае *метод* `write` вызывается пять раз для сохранения пяти записей в файле, каждая из которых представляет собой отдельную строку текста, завершаемую символом новой строки. В конце набора команды `with` *неявно* вызывается метод `close` объекта файла, чтобы файл был закрыт.

Содержимое файла accounts.txt

После выполнения предыдущего фрагмента в каталоге ch09 появляется файл accounts.txt со следующим содержимым (вы можете просмотреть его, открыв файл в текстовом редакторе):

```
100 Jones 24.98
200 Doe 345.67
300 White 0.00
400 Stone -42.16
500 Rich 224.62
```

В следующем разделе мы прочитаем файл и выведем его содержимое.

9.3.2. Чтение данных из текстового файла

Мы только что создали текстовый файл accounts.txt и записали в него данные. Теперь прочитаем эти данные из файла последовательно, от начала до конца. Следующий сеанс читает записи из файла accounts.txt и выводит содержимое каждой записи по столбцам: столбцы `Account` и `Name` выравниваются *по левому краю*, а столбец `Balance` выравнивается *по правому краю*, чтобы точки — разделители дробной части были выровнены по вертикали:

```
In [1]: with open('accounts.txt', mode='r') as accounts:
   ...:     print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
   ...:     for record in accounts:
   ...:         account, name, balance = record.split()
   ...:         print(f'{account:<10}{name:<10}{balance:>10}')
   ...:
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
300       White           0.00
400       Stone         -42.16
500       Rich          224.62
```

Если содержимое файла не должно изменяться, откройте файл только для чтения. Тем самым предотвращается случайная модификация файла. Чтобы открыть файл для чтения, передайте режим открытия `'r'` во втором аргументе функции open. Если каталог, в котором находится файл, не указан, то open предполагает, что файл находится в текущем каталоге.

Как показывает предыдущая команда `for`, перебор по объекту файла читает одну строку данных из файла и возвращает ее в виде строки. Для каждой записи (то есть строки) из файла метод строки `split` возвращает лексемы в строке в виде списка, который распаковывается в переменные `account`, `name` и `balance`[1]. Последняя команда в наборе команды `for` выводит эти переменные по столбцам, используя выравнивание по ширине полей.

Метод readlines файлов

Метод `readlines` объекта файла также может использоваться для чтения *всего* текстового файла. Метод возвращает каждую строку файла как элемент списка строк. Для небольших файлов такое решение работает, но в больших файлах представленный выше подход с перебором строк из объекта файла может оказаться более эффективным[2]. Вызов `readlines` для большого файла может стать продолжительной операцией, которая должна быть завершена перед тем, как вы сможете использовать список строк. Использование объекта файла в команде `for` позволяет программе обрабатывать строки текстового файла по мере их чтения.

Переход к конкретной позиции файла

В процессе чтения файла ОС поддерживает *указатель текущей позиции*, представляющий позицию следующего читаемого символа. Иногда требуется *несколько раз* последовательно обработать файл от начала во время выполнения программы. Каждый раз указатель текущей позиции необходимо вернуть в начало файла, для чего можно либо закрыть и повторно открыть файл, либо вызвать метод `seek` для объекта файла:

```
file_object.seek(0)
```

Второй способ работает быстрее.

9.4. Обновление текстовых файлов

Отформатированные данные, записанные в текстовый файл, не могут изменяться без риска повреждения других данных. Скажем, если вам потребуется

[1] При разбиении строк по пробелам (используется по умолчанию) `split` автоматически удаляет символ новой строки.

[2] https://docs.python.org/3/tutorial/inputoutput.html#methods-of-file-objects.

заменить имя 'White' на 'Williams' в файле accounts.txt, просто перезаписать старое имя не удастся. Исходная запись для клиента White хранится в виде

```
300 White 0.00
```

Если перезаписать имя 'White' именем 'Williams', запись принимает вид

```
300 Williams00
```

Новая фамилия содержит на три символа больше исходной, поэтому символы после второй буквы «i» в 'Williams' перезаписывают другие символы в строке. Проблема в том, что в модели отформатированного ввода/вывода записи и их поля могут изменяться в размерах. Например, 7, 14, -117, 2074 и 27383 — целые числа, которые во внутреннем представлении занимают одинаковое количество байтов «необработанных данных» (обычно 4 или 8 байт в современных системах). Тем не менее при выводе этих чисел в виде отформатированного текста они имеют поля разных размеров. Например, 7 при выводе занимает один символ, 14 — два символа, а 27383 — пять символов.

Чтобы внести описанное выше изменение в имя, необходимо:

- скопировать записи, предшествующие 300 White 0.00, во временный файл;
- записать обновленную и правильно отформатированную запись для счета 300 в этот файл;
- скопировать записи после 300 White 0.00 во временный файл;
- удалить старый файл;
- переименовать временный файл и присвоить ему имя исходного файла.

Процесс выглядит достаточно громоздким, потому что он требует обновления *всех* записей в файле, даже если обновляется всего одна запись. Обновление файла способом, описанным выше, работает более эффективно, если приложению нужно обновить много записей за один проход по файлу[1].

Обновление accounts.txt

Воспользуемся командой with для обновления файла accounts.txt с заменой имени 'White' именем 'Williams' для счета 300, как описано выше:

[1] В главе 16 будет показано, что системы баз данных эффективно решают эту проблему «обновления на месте».

```
In [1]: accounts = open('accounts.txt', 'r')

In [2]: temp_file = open('temp_file.txt', 'w')

In [3]: with accounts, temp_file:
   ...:     for record in accounts:
   ...:         account, name, balance = record.split()
   ...:         if account != '300':
   ...:             temp_file.write(record)
   ...:         else:
   ...:             new_record = ' '.join([account, 'Williams', balance])
   ...:             temp_file.write(new_record + '\n')
   ...:
```

Для удобочитаемости мы открыли объекты файлов (фрагменты [1] и [2]), а затем указали имена их переменных в первой строке фрагмента [3]. Команда `with` управляет двумя объектами ресурсов, указанными в списке, разделенном запятыми после `with`. Команда `for` распаковывает каждую запись на переменные `account`, `name` и `balance`. Если номер счета не равен `'300'`, то запись (содержащая символ новой строки) записывается в `temp_file`. В противном случае программа конструирует новую запись, содержащую имя `'Williams'` вместо `'White'`, и записывает ее в файл. После фрагмента [3] файл temp_file.txt содержит следующие данные:

```
100 Jones 24.98
200 Doe 345.67
300 Williams 0.00
400 Stone -42.16
500 Rich 224.62
```

Файловые функции модуля os

На этот момент у нас имеется старый файл accounts.txt и новый файл temp_file.txt. Чтобы завершить обновление, удалим старый файл accounts.txt, а затем переименуем temp_file.txt в accounts.txt. *Модуль* `os`[1] предоставляет функции для взаимодействия с ОС, включая ряд функции для работы с файлами и каталогами. Создав временный файл, воспользуемся *функцией* `remove`[2] для удаления исходного файла:

```
In [4]: import os

In [5]: os.remove('accounts.txt')
```

[1] https://docs.python.org/3/library/os.html.

[2] Будьте осторожны при использовании `remove` — функция не предупреждает вас о том, что файл удаляется навсегда.

Далее посредством *функции* `rename` переименуем файл в `'accounts.txt'`:

```
In [6]: os.rename('temp_file.txt', 'accounts.txt')
```

9.5. Сериализация в формат JSON

Многие библиотеки, используемые для взаимодействия с облачными сервисами (например, Twitter, IBM Watson и др.), обмениваются с приложением данными в виде объектов JSON. *JSON* (*JavaScript Object Notation*) — текстовый формат обмена данными, который может читаться как человеком, так и компьютером. Он используется для представления объектов в виде коллекций пар «имя-значение». Формат JSON даже может использоваться для представления объектов пользовательских классов, построением которых мы займемся в следующей главе.

Формат JSON стал основным форматом данных для передачи объектов между платформами. Это относится в первую очередь к облачным веб-сервисам — функциям и методам, которые могут вызываться по интернету. Вскоре вы освоите основы работы с данными JSON, а в главе 12 будут использоваться объекты JSON, содержащие твиты и их метаданные. В главе 13 мы будем работать с данными в ответах JSON, возвращаемых сервисами Watson. В главе 16 объекты твитов в формате JSON, полученные от Twitter, будут сохраняться в MongoDB — популярной базе данных NoSQL. В этой главе мы также будем работать с другими веб-сервисами, которые отправляют и получают данные в виде объектов JSON.

Формат данных JSON

Объекты JSON отдаленно похожи на словари Python. Каждый объект JSON содержит разделенный запятыми список *имен* и *значений свойств*, заключенный в фигурные скобки. Например, следующие пары «ключ-значение» могут представлять запись клиента:

```
{"account": 100, "name": "Jones", "balance": 24.98}
```

JSON также поддерживает массивы, которые, как и списки Python, представляют собой списки значений, разделенных запятыми, в квадратных скобках. Например, следующая строка содержит действительный массив чисел в формате JSON:

```
[100, 200, 300]
```

Значениями в объектах и массивах JSON могут быть:

- строки, заключенные в *двойные кавычки* (например, `"Jones"`);
- числа (например, `100` или `24.98`);
- логические значения JSON (`true` или `false`);
- `null` (признак отсутствия значения, аналог `None` в Python);
- массивы (например, [`100`, `200`, `300`]);
- другие объекты JSON.

Модуль json стандартной библиотеки Python

Модуль `json` обеспечивает преобразование объектов в текстовый формат JSON (JavaScript Object Notation). Этот процесс называется *сериализацией* данных. Возьмем следующий словарь, который содержит одну пару «ключ-значение»: ключ `'accounts'` с ассоциированным значением, которое представляет собой список словарей, представляющих два счета. Каждый словарь счета содержит три пары «ключ-значение» для номера счета, имени клиента и баланса:

```
In [1]: accounts_dict = {'accounts': [
   ...:     {'account': 100, 'name': 'Jones', 'balance': 24.98},
   ...:     {'account': 200, 'name': 'Doe', 'balance': 345.67}]}
```

Сериализация объекта в формат JSON

Запишем объект в формате JSON в файл:

```
In [2]: import json

In [3]: with open('accounts.json', 'w') as accounts:
   ...:     json.dump(accounts_dict, accounts)
   ...:
```

Фрагмент [3] открывает файл accounts.json и использует *функцию* `dump` модуля `json` для сериализации словаря `accounts_dict` в файл. Полученный файл содержит текст, слегка переформатированный для удобства чтения:

```
{"accounts":
  [{"account": 100, "name": "Jones", "balance": 24.98},
   {"account": 200, "name": "Doe", "balance": 345.67}]}
```

Обратите внимание: в JSON строки заключаются в *двойные кавычки*.

Десериализация текста JSON

Функция `load` модуля `json` читает все содержимое JSON из объекта файла, передаваемого в аргументе, преобразуя JSON в объект Python. Это и называется *десериализацией* данных. Восстановим исходный объект Python из следующего текста JSON:

```
In [4]: with open('accounts.json', 'r') as accounts:
   ...:     accounts_json = json.load(accounts)
   ...:
   ...:
```

Теперь можно взаимодействовать с загруженным объектом — например, вывести содержимое словаря:

```
In [5]: accounts_json
Out[5]:
{'accounts': [{'account': 100, 'name': 'Jones', 'balance': 24.98},
  {'account': 200, 'name': 'Doe', 'balance': 345.67}]}
```

Как и следовало ожидать, вы можете обратиться к содержимому словаря. Получим список словарей, ассоциированных с ключом `'accounts'`:

```
In [6]: accounts_json['accounts']
Out[6]:
[{'account': 100, 'name': 'Jones', 'balance': 24.98},
 {'account': 200, 'name': 'Doe', 'balance': 345.67}]
```

А теперь обратимся к словарям отдельных счетов:

```
In [7]: accounts_json['accounts'][0]
Out[7]: {'account': 100, 'name': 'Jones', 'balance': 24.98}

In [8]: accounts_json['accounts'][1]
Out[8]: {'account': 200, 'name': 'Doe', 'balance': 345.67}
```

Кстати, словари также можно изменять. Например, вы можете добавить счета или удалить счета из списка, а затем записать словарь обратно в файл JSON.

Вывод текста JSON

Функция `dumps` модуля `json` возвращает строковое представление Python для объекта в формате JSON. Используя `dumps` с `load`, вы можете прочитать данные JSON из файла и вывести их в удобном формате с отступами — ино-

гда это называется «структурным выводом» кода JSON. Если вызов функции `dumps` включает ключевой аргумент `indent`, строка содержит символы новой строки и отступы для структурного вывода — аргумент `indent` также может использоваться с функцией `dump` при выводе в файл:

```
In [9]: with open('accounts.json', 'r') as accounts:
   ...:     print(json.dumps(json.load(accounts), indent=4))
   ...:
{
    "accounts": [
        {
            "account": 100,
            "name": "Jones",
            "balance": 24.98
        },
        {
            "account": 200,
            "name": "Doe",
            "balance": 345.67
        }
    ]
}
```

9.6. Вопросы безопасности: сериализация и десериализация pickle

Модуль pickle стандартной библиотеки Python может сериализовать объекты в формат данных, специфический для Python. **Будьте осторожны: в документации Python приводятся следующие предупреждения относительно pickle**:

- «Файлы pickle могут подвергаться несанкционированным изменениям. Если вы получаете необработанный файл pickle по Сети, не доверяйте ему! Он может содержать вредоносный код, который выполнит произвольный код Python при попытке преобразования из формата pickle. С другой стороны, если вы выполняете запись и чтение pickle самостоятельно, то вы в безопасности (разумеется, при условии, что посторонние не имеют доступа к файлу pickle[1])»;
- «Pickle — протокол, позволяющий сериализовать объекты Python произвольной сложности. Соответственно, он специфичен для Python и не может использоваться для взаимодействия с приложениями, написанны-

[1] https://wiki.python.org/moin/UsingPickle.

ми на других языках. Он также небезопасен по умолчанию: десериализация данных pickle из ненадежного источника может привести к выполнению произвольного кода, если данные были сгенерированы опытным злоумышленником»[1].

Мы не рекомендуем использовать pickle, но этот модуль использовался в течение многих лет, поэтому, скорее всего, вы встретите его в *унаследованном коде* — старом коде, который часто уже не сопровождается.

9.7. Дополнительные замечания по поводу файлов

В табл. 9.1 приведена сводка режимов открытия текстовых файлов, включая уже описанные ранее режимы для чтения и записи. Режимы *записи* и *присоединения* создают файл, если он не существует. Если файл не существует, то режимы *чтения* выдают ошибку `FileNotFoundError`. Каждый режим текстового файла имеет соответствующий режим двоичного файла, обозначаемый суффиксом `b`, например `'rb'` или `'wb+'`. Эти режимы могут использоваться для чтения или для записи двоичных файлов: графики, аудио, видео, сжатых ZIP-файлов и др.

Таблица 9.1. Основные режимы открытия текстовых файлов

Режим	Описание
`'r'`	Текстовый файл открывается для чтения. Используется по умолчанию, если режим открытия файла не указан при вызове `open`
`'w'`	Текстовый файл открывается для записи. Существующее содержимое файла *удаляется*
`'a'`	Текстовый файл открывается для присоединения данных. Если файл не существует, то он создается. Новые данные записываются в конец файла
`'r+'`	Текстовый файл открывается для чтения и записи
`'w+'`	Текстовый файл открывается для чтения и записи. Существующее содержимое файла удаляется
`'a+'`	Текстовый файл открывается для чтения и присоединения данных. Новые данные записываются в конец файла. Если файл не существует, то он создается

[1] https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files.

Другие методы объектов файлов

Укажем еще несколько полезных методов объектов файлов:

- Для текстового файла метод `read` возвращает строку с количеством символов, заданным целочисленным аргументом метода. Для двоичного файла метод возвращает заданное количество байтов. Если аргумент не задан, то метод возвращает все содержимое файла.
- Метод `readline` возвращает одну строку текста в строковой форме, включая символ новой строки (если он присутствует). При достижении конца файла метод возвращает пустую строку.
- Метод `writelines` получает список строк и записывает его содержимое в файл.

Классы, используемые Python для создания объектов файлов, определяются в *модуле* `io` стандартной библиотеки Python (https://docs.python.org/3/library/io.html).

9.8. Обработка исключений

При работе с файлами могут происходить исключения различных типов, например:

- Исключение `FileNotFoundError` происходит при попытке открытия несуществующего файла для чтения в режиме `'r'` или `'r+'`.
- Исключение `PermissionsError` происходит при попытке выполнения операции, для которой у вас нет необходимых разрешений. Например, это может произойти при попытке открыть файл, недоступный для вашей учетной записи, или создать файл в каталоге, для которого у вашей учетной записи отсутствует разрешение записи (например, в каталоге, в котором хранится ОС).
- Исключение `ValueError` (с сообщением об ошибке `'I/O operation on closed file.'`) происходит при попытке записи в ранее закрытый файл.

9.8.1. Деление на нуль и недействительный ввод

Вернемся к двум исключениям, которые были описаны ранее в этой книге.

Деление на нуль

Вспомним, что попытка деления на 0 приводит к ошибке `ZeroDivisionError`:

```
In [1]: 10 / 0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-a243dfbf119d> in <module>()
----> 1 10 / 0

ZeroDivisionError: division by zero

In [2]:
```

В этой ситуации интерпретатор *выдает исключение* типа `ZeroDivisionError`. При возникновении исключения IPython завершает фрагмент, выводит трассировку исключения и последующее приглашение `In[]` для ввода следующего фрагмента. Если исключение возникает в сценарии, то этот сценарий завершается, а IPython выводит трассировку.

Недействительный ввод

Функция `int` выдает исключение `ValueError` при попытке преобразования в строку целого числа, которое не представляет число (например, `'hello'`):

```
In [2]: value = int(input('Enter an integer: '))
Enter an integer: hello
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-b521605464d6> in <module>()
----> 1 value = int(input('Enter an integer: '))

ValueError: invalid literal for int() with base 10: 'hello'

In [3]:
```

9.8.2. Команды try

Теперь посмотрим, как *обработать* возникшее исключение, чтобы программа могла продолжить выполнение. Возьмем следующий сценарий и пример выполнения. Его цикл пытается получить два целых числа от пользователя, а затем выводит первое число, разделенное на второе. Сценарий использует механизм обработки исключений для перехвата и обработки любых воз-

никающих исключений `ZeroDivisionError` и `ValueError` — в данном случае пользователю предлагается заново ввести данные.

```
 1 # dividebyzero.py
 2 """Простой пример обработки исключений."""
 3
 4 while True:
 5     # Пытаемся преобразовать и разделить значения
 6     try:
 7         number1 = int(input('Enter numerator: '))
 8         number2 = int(input('Enter denominator: '))
 9         result = number1 / number2
10     except ValueError:  # Попытка преобразования в int некорректного значения
11         print('You must enter two integers\n')
12     except ZeroDivisionError:  # Делитель равен 0
13         print('Attempted to divide by zero\n')
14     else:  # Выполняется только при отсутствии исключений
15         print(f'{number1:.3f} / {number2:.3f} = {result:.3f}')
16         break  # завершает цикл
```

```
Enter numerator: 100
Enter denominator: 0
Attempted to divide by zero

Enter numerator: 100
Enter denominator: hello
You must enter two integers

Enter numerator: 100
Enter denominator: 7
100.000 / 7.000 = 14.286
```

Секция try

Python использует *команды* `try` (вроде приведенной в строках 6–16) для обработки исключений. *Секция* `try` команды `try` (строки 6–9) начинается с ключевого слова `try`, за которым следует двоеточие (`:`) и набор команд, при выполнении которых *могут быть* выданы исключения.

Секция except

После секции `try` может следовать одна или несколько *секций* `except` (строки 10–11 и 12–13); они располагаются непосредственно после набора секции `try`. Эти секции называются *обработчиками исключений*. Каждая секция `except` задает тип обрабатываемого ею исключения. В данном примере каждый

обработчик исключения просто выводит сообщение с информацией о возникшей проблеме.

Секция else

За последней секцией `except` необязательная *секция* `else` (строки 14–16) задает код, выполняемый только в том случае, если код в наборе `try` не выдает исключений. Если в наборе `try` этого примера не возникло ни одно исключение, то строка 15 выводит результат деления, а строка 16 завершает цикл.

Последовательность выполнения для ошибки ZeroDivisionError

Теперь рассмотрим последовательность передачи управления в этом примере на основании первых трех строк вывода в нашем примере.

- Сначала пользователь вводит делимое `100` в ответ на строку 7 в наборе `try`.
- Затем пользователь вводит делитель `0` в ответ на строку 8 в наборе `try`.
- К этому моменту пользователь ввел два целочисленных значения, поэтому строка 9 пытается разделить 100 на 0, в результате чего Python выдает исключение `ZeroDivisionError`. Точка, в которой в программе возникает исключение, часто называется *точкой выдачи исключения*.

Если в наборе `try` происходит исключение, его выполнение немедленно завершается. Если за набором `try` следуют обработчики `except`, то программа передает управление первому подходящему. Если обработчиков нет, то происходит процесс, называемый *раскруткой стека*; он будет рассмотрен позднее в этой главе.

В данном примере обработчики `except` *присутствуют*, поэтому интерпретатор ищет первый из них, совпадающий с типом выданного исключения:

- Секция `except` в строках 10–11 обрабатывает исключения `ValueError`. Этот тип не совпадает с типом `ZeroDivisionError`, поэтому набор секции `except` не выполняется, а управление передается следующему обработчику `except`.
- Секция `except` в строках 12–13 обрабатывает исключения `ZeroDivisionError`. Совпадение *найдено*, поэтому выполняется набор секции `except`, а программа выводит сообщение о попытке деления на нуль.

Когда секция `except` успешно обработает исключение, выполнение программы продолжается с секции `finally` (если она присутствует), а затем управление передается следующей команде после команды `try`. В данном примере достигается конец цикла, поэтому выполнение продолжается со следующей итерации. Учтите, что после обработки исключения управление *не* возвращается в точку, в которой это исключение было выдано, — выполнение продолжается с точки после `try`. Вскоре секция `finally` будет рассмотрена более подробно.

Последовательность выполнения для ошибки ValueError

Теперь рассмотрим последовательность передачи управления в этом примере на основании следующих трех строк вывода в нашем примере.

- Сначала пользователь вводит делимое `100` в ответ на строку 7 в наборе `try`.
- Затем пользователь вводит текст `hello` в ответ на строку 8 в наборе `try`. Введенное значение не является допустимым целым числом, поэтому функция `int` выдает исключение `ValueError`.

Исключение завершает набор `try`, а управление передается первому обработчику `except`. В данном случае секция `except` в строках 10–11 подходит, поэтому выполняется ее набор с выводом сообщения о необходимости ввести два целых числа. Выполнение продолжается со следующей команды после `try`: управление передается в конец цикла, а выполнение продолжается со следующей итерации.

Последовательность выполнения для успешного деления

Наконец, рассмотрим последовательность передачи управления в этом примере на основании трех последних строк вывода в нашем примере:

- Сначала пользователь вводит делимое `100` в ответ на строку 7 в наборе `try`.
- Затем пользователь вводит делитель `7` в ответ на строку 8 в наборе `try`.
- К этому моменту пользователь ввел два допустимых целых числа, а делитель не равен 0, поэтому строка 9 успешно делит `100` на `7`.

Если в наборе `try` исключения не возникают, то выполнение программы продолжается в секции `else` (если она присутствует); если же ее нет, то программа выполняется со следующей команды после команды `try`. В секции

else примера выводится результат деления, после чего завершаются цикл и сама программа.

9.8.3. Перехват нескольких исключений в одной секции except

В довольно типичной ситуации за секцией try следуют несколько секций except для обработки разных типов исключений. Если несколько наборов except полностью совпадают, то вы можете перехватить все эти типы, указав их в виде кортежа в *одном* обработчике except:

```
except (mun1, mun2, …) as имя_переменной:
```

Секция as необязательна. Обычно программам не нужно обращаться к объектам перехваченных исключений напрямую. При необходимости используйте переменную из секции as для обращения к объекту исключения в наборе except.

9.8.4. Какие исключения выдают функция или метод?

Исключения могут быть порождены командами в наборе try, функциями или методами, прямо или косвенно вызванными из набора try, или интерпретатором Python в ходе выполнения кода (например, исключение ZeroDivisionError).

До использования функции или метода прочитайте электронную документацию API. В ней указано, какие исключения выдают функция или метод (и выдают ли), а также указаны возможные причины для возникновения таких исключений. Прочитайте в электронной документации API обо всех типах исключений; это позволит вам понять возможные причины для возникновения исключения.

9.8.5. Какой код должен размещаться в наборе try?

Постарайтесь разместить в наборе try значительный логический раздел программы, в котором исключения могут выдаваться несколькими командами, вместо того чтобы упаковывать каждую команду, способную выдавать исключения, в отдельную конструкцию try. Однако для правильной детализации обработки исключений каждая команда try должна включать достаточно малую

часть кода, чтобы при возникновении исключения был известен конкретный контекст, а обработчики `except` могли правильно обработать исключение. Если сразу несколько команд в наборе `try` могут выдавать одинаковые типы исключений, то для определения контекста каждого исключения могут понадобиться несколько команд `try`.

9.9. Секция finally

Операционные системы обычно запрещают нескольким программам работать с файлом одновременно. Когда программа завершает обработку файла, она должна закрыть его, чтобы освободить этот ресурс и сделать его доступным для других программ. Закрытие файла помогает предотвратить *утечку ресурсов*.

Секция finally команды try

Команда `try` может содержать секцию `finally`, которая следует после всех секций `except` или секции `else`. Выполнение секции `finally` гарантировано[1]. В других языках с поддержкой `finally` набор `finally` становится идеальным местом для размещения кода освобождения ресурсов для ресурсов, захваченных в соответствующем наборе `try`. В Python для этой цели рекомендуется использовать команду `with`, а весь прочий завершающий код размещается в наборе `finally`.

Пример

Следующий сеанс IPython демонстрирует, что секция `finally` всегда выполняется независимо от того, происходит ли исключение в соответствующем наборе `try`. Сначала рассмотрим команду `try`, в наборе которой исключения не возникают:

```
In [1]: try:
   ...:     print('try suite with no exceptions raised')
   ...: except:
   ...:     print('this will not execute')
   ...: else:
   ...:     print('else executes because no exceptions in the try suite')
```

[1] Единственная причина, по которой секция `finally` не будет выполнена после входа программы в соответствующую секцию `try`, — приложение будет завершено до этого (например, вызовом функции `exit` модуля `sys`).

```
   ...: finally:
   ...:     print('finally always executes')
   ...:
try suite with no exceptions raised
else executes because no exceptions in the try suite
finally always executes

In [2]:
```

Предшествующий набор `try` выводит сообщение, но не выдает никаких исключений. Когда управление успешно достигает конца набора `try`, секция `except` пропускается, выполняется секция `else`, а секция `finally` выводит сообщение, которое подтверждает, что она выполняется всегда. Когда секция `finally` завершается, выполнение программы продолжается со следующей команды после команды `try`, а в сеансе IPython выводится приглашение `In []`.

Рассмотрим команду `try`, у которой в наборе `try` происходит исключение:

```
In [2]: try:
   ...:     print('try suite that raises an exception')
   ...:     int('hello')
   ...:     print('this will not execute')
   ...: except ValueError:
   ...:     print('a ValueError occurred')
   ...: else:
   ...:     print('else will not execute because an exception occurred')
   ...: finally:
   ...:     print('finally always executes')
   ...:
try suite that raises an exception
a ValueError occurred
finally always executes

In [3]:
```

Набор `try` начинается с вывода сообщения. Вторая команда пытается преобразовать строку `'hello'` в целое число, в результате чего функция `int` выдает исключение `ValueError`. Набор `try` немедленно завершается, а последняя команда `print` пропускается. Секция `except` перехватывает исключение `ValueError` и выводит сообщение. Секция `else` не выполняется, потому что произошло исключение. Затем секция `finally` выводит сообщение, которое показывает, что она выполняется всегда. Секция `finally` завершается, а программа продолжается со следующей команды после команды `try`. В сеансе IPython появляется приглашение `In []`.

Объединение команд with с командами try...except

У многих ресурсов, требующих явного освобождения (файлов, сетевых подключений, подключений к базам данных), существуют потенциальные исключения, связанные с обработкой этих ресурсов. Например, программа, обрабатывающая файл, может выдавать исключения `IOError`. По этой причине надежный код, работающий с файлами, обычно заключается в набор `try`, содержащий команду `with`, которая гарантирует освобождение ресурса. Код заключен в набор `try`, так что вы можете перехватить любые возникающие исключения в обработчиках `except`, и секция `finally` не понадобится — команда `with` позаботится об освобождении ресурсов.

Чтобы продемонстрировать, как работает эта схема, допустим, что программа запрашивает у пользователя имя файла, и пользователь вводит неправильное имя, например gradez.txt вместо имени созданного ранее файла grades.txt. В этом случае вызов open выдает исключение `FileNotFoundError` при попытке открыть несуществующий файл:

```
In [3]: open('gradez.txt')
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-3-b7f41b2d5969> in <module>()
----> 1 open('gradez.txt')

FileNotFoundError: [Errno 2] No such file or directory: 'gradez.txt'
```

Чтобы перехватывать исключения вроде `FileNotFoundError`, происходящие при попытке открыть файл для чтения, заключите команду `with` в набор `try`:

```
In [4]: try:
   ...:     with open('gradez.txt', 'r') as accounts:
   ...:         print(f'{"ID":<3}{"Name":<7}{"Grade"}')
   ...:         for record in accounts:
   ...:             student_id, name, grade = record.split()
   ...:             print(f'{student_id:<3}{name:<7}{grade}')
   ...: except FileNotFoundError:
   ...:     print('The file name you specified does not exist')
   ...:
The file name you specified does not exist
```

9.10. Явная выдача исключений

Итак, код Python может выдавать различные исключения. Иногда требуется написать функции, оповещающие сторону вызова о возникших ошибках. Яв-

ная выдача исключения осуществляется командой `raise`. Простейшая форма команды `raise`:

```
raise ИмяКлассаИсключения
```

Команда `raise` создает объект заданного класса исключения. За именем класса исключения могут следовать необязательные круглые скобки с аргументами, инициализирующими объект исключения, — как правило, с целью определения нестандартной строки с сообщением об ошибке. Код, выдающий исключение, должен сначала освободить любые ресурсы, полученные перед возникновением исключения. В следующем разделе будет представлен пример явной выдачи исключения.

При необходимости выдачи исключения рекомендуется использовать один из многих встроенных типов исключений Python[1], см. полный список по адресу:

https://docs.python.org/3/library/exceptions.html

9.11. Раскрутка стека и трассировка (дополнение)

В каждом объекте исключения хранится информация о точной последовательности вызовов функций, которые привели к исключению. Это может быть полезно при отладке кода. Рассмотрим следующие определения функций — `function1` вызывает `function2`, а `function2` выдает исключение `Exception`:

```
In [1]: def function1():
   ...:     function2()
   ...:

In [2]: def function2():
   ...:     raise Exception('An exception occurred')
   ...:
```

Вызов `function1` приводит к выдаче следующей трассировки. Например, мы выделили жирным шрифтом части трассировки, обозначающие строки кода, приведшие к исключению:

[1] Возможно, вам захочется создать нестандартный класс исключения, принадлежащий вашему приложению. Нестандартные исключения будут более подробно рассмотрены в следующих главах.

```
In [3]: function1()
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-3-c0b3cafe2087> in <module>()
----> 1 function1()

<ipython-input-1-a9f4faeeeb0c> in function1()
      1 def function1():
----> 2     function2()
      3

<ipython-input-2-c65e19d6b45b> in function2()
      1 def function2():
----> 2     raise Exception('An exception occurred')

Exception: An exception occurred
```

Подробности трассировки

В трассировке указывается тип возникшего исключения (`Exception`), за которым следует полный стек вызовов функций, который привел к точке выдачи исключения. Нижний вызов функции в стеке указывается на *первом* месте, а верхний — на *последнем*, так что интерпретатор выводит следующий текст как напоминание:

```
Traceback (most recent call last)
```

В этой трассировке следующий текст обозначает нижнюю позицию стека вызовов — вызов `function1` в фрагменте [3] (обозначенный `ipython-input-3`):

```
<ipython-input-3-c0b3cafe2087> in <module>()
----> 1 function1()
```

Затем мы видим, что `function1` вызывает `function2` в строке 2 во фрагменте [1]:

```
<ipython-input-1-a9f4faeeeb0c> in function1()
      1 def function1():
----> 2     function2()
      3
```

Наконец, мы видим *точку выдачи исключения* — в данном случае строка 2 во фрагменте [2] выдает исключение:

```
<ipython-input-2-c65e19d6b45b> in function2()
      1 def function2():
----> 2     raise Exception('An exception occurred')
```

Раскрутка стека

В предшествующих примерах обработки исключений точка выдачи исключения происходила в наборе `try`, а исключение было обработано в одном из соответствующих обработчиков `except` команды `try`. Если исключение *не было* перехвачено в функции, то происходит раскрутка стека. Рассмотрим раскрутку стека в контексте данного примера:

- В `function2` команда `raise` выдает исключение. Это не набор `try`, поэтому функция `function2` завершается, ее кадр стека удаляется из стека вызовов функций, а управление возвращается команде `function1`, из которой была вызвана функция `function2`.
- В `function1` команда, вызвавшая `function2`, не принадлежит набору `try`, поэтому функция `function1` завершается, ее кадр стека удаляется из стека вызовов функций, а управление возвращается команде, из которой была вызвана функция `function1` — фрагмент [3] в сеансе IPython.
- Вызов во фрагменте [3] не принадлежит набору `try`, поэтому вызов функции завершается. Поскольку исключение не было перехвачено, IPython выводит трассировку и ожидает следующего ввода. Если это происходит в типичном сценарии, то сценарий завершается[1].

Рекомендации по чтению трассировок

Часто в своих программах вы будете вызывать функции и методы, принадлежащие библиотекам, которые вы не писали. Иногда эти функции и методы выдают исключения. При чтении трассировки начните с конца и сначала прочитайте сообщение об ошибке. Затем продолжайте читать вверх по трассировочному выводу и найдите первую строку, в которой обозначен код, написанный вами для вашей программы. Обычно это позиция вашего кода, приведшая к выдаче исключения.

Исключения в наборах finally

Выдача исключения в наборе `finally` может привести к коварным, трудно диагностируемым ошибкам. Если произойдет исключение, которое не будет обработано к моменту выполнения набора `finally`, происходит раскрутка сте-

[1] В более сложных приложениях, использующих потоки, неперехваченное исключение завершает только программный поток, в котором произошло исключение, а не все приложение (если это возможно).

ка. Если набор `finally` выдает *новое* исключение, которое не перехватывается набором, то первое исключение *теряется*, а *новое* исключение передается следующей вмещающей команде `try`. По этой причине набор `finally` всегда должен заключать в команду `try` любой код, который может выдать исключение, чтобы исключения обрабатывались внутри этого набора.

9.12. Введение в data science: работа с CSV-файлами

В этой книге мы будем работать со многими наборами данных при представлении концепций data science. *CSV* (*значения, разделенные запятыми*) — чрезвычайно популярный файловый формат. В этом разделе мы продемонстрируем обработку CSV-файлов с использованием модуля стандартной библиотеки Python и pandas.

9.12.1. Модуль csv стандартной библиотеки Python

Модуль `csv`[1] предоставляет функции для работы с CSV-файлами. Встроенная поддержка CSV также присутствует во многих других библиотеках Python.

Запись в CSV-файл

Создадим файл accounts.csv в формате CSV. Документация модуля `csv` рекомендует открывать CSV-файлы с дополнительным ключевым аргументом `newline=''`, чтобы обеспечить правильную обработку символов новой строки:

```
In [1]: import csv

In [2]: with open('accounts.csv', mode='w', newline='') as accounts:
   ...:     writer = csv.writer(accounts)
   ...:     writer.writerow([100, 'Jones', 24.98])
   ...:     writer.writerow([200, 'Doe', 345.67])
   ...:     writer.writerow([300, 'White', 0.00])
   ...:     writer.writerow([400, 'Stone', -42.16])
   ...:     writer.writerow([500, 'Rich', 224.62])
   ...:
```

[1] https://docs.python.org/3/library/csv.html.

Расширение .csv является признаком файла в формате CSV. *Функция* `writer` модуля `csv` возвращает объект, который записывает данные CSV в заданный объект файла. Каждый вызов *метода* `writerow` объекта `writer` возвращает итерируемый объект для сохранения в файле. В данном случае используются списки. По умолчанию `writerow` разделяет значения запятыми, но вы можете указать собственные нестандартные разделители[1]. После предыдущего фрагмента файл accounts.csv содержит следующие данные:

```
100,Jones,24.98
200,Doe,345.67
300,White,0.00
400,Stone,-42.16
500,Rich,224.62
```

CSV-файлы обычно не содержат пробелов после запятых, но некоторые разработчики используют их для улучшения удобочитаемости. Предшествующие вызовы `writerow` можно заменить одним вызовом `writerows`, который выводит разделенный запятыми список итерируемых объектов, представляющих записи.

Если записываемые данные содержат запятые внутри строки, то `writerow` заключает ее в двойные кавычки. Для примера возьмем следующий список Python:

```
[100, 'Jones, Sue', 24.98]
```

Строка в одинарных кавычках `'Jones, Sue'` содержит запятую, отделяющую имя от фамилии. В этом случае `writerow` выводит запись в виде

```
100,"Jones, Sue",24.98
```

Кавычки, в которые заключена строка `"Jones, Sue"`, показывают, что это *одно* значение. Программы, читающие эти данные из CSV-файла, разобьют запись на *три* части — `100`, `'Jones, Sue'` и `24.98`.

Чтение из CSV-файла

Теперь прочитаем CSV-данные из файла. Следующий фрагмент читает записи из файла accounts.csv и выводит содержимое каждой записи, в результате чего будет получен тот же вывод, что и приведенный ранее:

---

[1] https://docs.python.org/3/library/csv.html#csv-fmt-params.

```
In [3]: with open('accounts.csv', 'r', newline='') as accounts:
   ...:     print(f'{"Account":<10}{"Name":<10}{"Balance":>10}')
   ...:     reader = csv.reader(accounts)
   ...:     for record in reader:
   ...:         account, name, balance = record
   ...:         print(f'{account:<10}{name:<10}{balance:>10}')
   ...:
Account   Name         Balance
100       Jones          24.98
200       Doe           345.67
300       White            0.0
400       Stone         -42.16
500       Rich          224.62
```

Функция `reader` модуля `csv` возвращает объект, который читает данные в формате CSV по заданному объекту файла. По аналогии с тем, как можно выполнить перебор по объекту файла, возможен и перебор по объекту `reader`; при каждой итерации будет прочитана одна запись значений, разделенных запятыми. Приведенная ранее команда `for` возвращает каждую запись в виде списка значений, распаковываемых в переменные `account`, `name` и `balance`, после чего выводится программой.

Предупреждение: запятые в полях данных CSV

Будьте осторожны при работе со строками, содержащими внутренние запятые, например имя `'Jones, Sue'`. Если вы случайно введете их как две строки `'Jones'` и `'Sue'`, то `writerow`, разумеется, создаст запись CSV с *четырьмя* полями вместо *трех*. Программы, читающие CSV-файлы, обычно ожидают, что все записи состоят из *одинакового* количества полей; если это условие нарушается, то начинаются проблемы. Для примера возьмем два списка:

```
[100, 'Jones', 'Sue', 24.98]
[200, 'Doe'  , 345.67]
```

Первый список содержит *четыре* значения, а второй — только *три*. Если сохранить эти две записи в CSV-файле, а затем прочитать их в программу с использованием приведенного фрагмента, то следующая команда приведет к ошибке при попытке распаковать запись из четырех полей в три переменные:

```
account, name, balance = record
```

Предупреждение: отсутствующие и лишние запятые в CSV-файлах

Будьте осторожны при подготовке и обработке CSV-файлов. Допустим, ваш файл состоит из записей, каждая из которых содержит *четыре* значения `int`, разделенные запятыми:

```
100,85,77,9
```

Если случайно пропустить одну из запятых:

```
100,8577,9
```

запись будет состоять из *трех* полей, одно из которых содержит недопустимое значение `8577`.

Если поставить две запятые подряд там, где должна быть только одна:

```
100,85,,77,9
```

запись будет состоять из *пяти* полей вместо *четырех*, а одно из полей окажется *пустым*. Все ошибки, связанные с запятыми, могут сбить с толку программы, пытающиеся обработать записи.

9.12.2. Чтение CSV-файлов в коллекции DataFrame библиотеки pandas

В разделах «Введение в data science» предыдущих двух глав были представлены основы работы с pandas. Теперь продемонстрируем средства pandas для загрузки файлов в формате CSV, а затем выполним базовые операции анализа данных.

Наборы данных

В практических примерах data science будут использованы различные бесплатные и открытые наборы данных для демонстрации концепций машинного обучения и обработки естественного языка. В интернете доступно огромное количество разнообразных бесплатных наборов данных. Популярный *репозиторий Rdatasets* содержит ссылки на более чем 1100 бесплатных наборов данных в формате CSV. Эти наборы изначально поставлялись с языком программирования R, чтобы упростить изучение и разработку статистических

программ, тем не менее, они не связаны с языком R. Сейчас эти наборы данных доступны на GitHub по адресу:

https://vincentarelbundock.github.io/Rdatasets/datasets.html

Этот репозиторий настолько популярен, что существует *модуль* `pydataset`, предназначенный специально для обращения к Rdatasets. За инструкциями по установке `pydataset` и обращению к наборам данных обращайтесь по адресу:

https://github.com/iamaziz/PyDataset

Другой большой источник наборов данных:

https://github.com/awesomedata/awesome-public-datasets

Одним из часто используемых наборов данных машинного обучения для начинающих является *набор данных катастрофы «Титаника»*, в котором перечислены все пассажиры и указано, выжили ли они, когда «*Титаник*» столкнулся с айсбергом и затонул 14–15 апреля 1912 года. Мы воспользуемся этим набором, чтобы показать, как загрузить набор данных, просмотреть его данные и вывести характеристики описательной статистики. Другие популярные наборы данных будут исследованы в главах с примерами data science позднее в этой книге.

Работа с локальными CSV-файлами

Для загрузки набора данных CSV в `DataFrame` можно воспользоваться функцией `read_csv` библиотеки pandas. Следующий фрагмент загружает и выводит CSV-файл accounts.csv, который был создан ранее в этой главе:

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('accounts.csv',
   ...:                  names=['account', 'name', 'balance'])
   ...:

In [3]: df
Out[3]:
   account   name  balance
0      100  Jones    24.98
1      200    Doe   345.67
2      300  White     0.00
3      400  Stone   -42.16
4      500   Rich   224.62
```

Аргумент `names` задает имена столбцов `DataFrame`. Без этого аргумента `read_csv` считает, что первая строка CSV-файла содержит разделенный запятыми список имен столбцов.

Чтобы сохранить данные `DataFrame` в файле формата CSV, вызовите метод `to_csv` коллекции `DataFrame`:

```
In [4]: df.to_csv('accounts_from_dataframe.csv', index=False)
```

Ключевой аргумент `index=False` означает, что имена строк (0–4 в левой части вывода `DataFrame` в фрагменте [3]) не должны записываться в файл. Первая строка полученного файла содержит имена столбцов:

```
account,name,balance
100,Jones,24.98
200,Doe,345.67
300,White,0.0
400,Stone,-42.16
500,Rich,224.62
```

9.12.3. Чтение набора данных катастрофы «Титаника»

Набор данных катастрофы «Титаника» принадлежит к числу самых популярных наборов данных машинного обучения и доступен во многих форматах, включая CSV.

Загрузка набора данных катастрофы «Титаника» по URL-адресу

Если у вас имеется URL-адрес, представляющий набор данных в формате CSV, то вы можете загрузить его в `DataFrame` функцией `read_csv` — допустим, с GitHub:

```
In [1]: import pandas as pd

In [2]: titanic = pd.read_csv('https://vincentarelbundock.github.io/' +
   ...:     'Rdatasets/csv/carData/TitanicSurvival.csv')
   ...:
```

Просмотр некоторых строк набора данных катастрофы «Титаника»

Набор данных содержит свыше 1300 строк, каждая строка представляет одного пассажира. По данным «Википедии», на борту было приблизительно

1317 пассажиров, а 815 из них погибли[1]. Для больших наборов данных при выводе `DataFrame` показываются только первые 30 строк, потом идет многоточие «...» и последние 30 строк. Для экономии места просмотрим первые и последние пять строк при помощи методов `head` и `tail` коллекции `DataFrame`. Оба метода по умолчанию возвращают пять строк, но число выводимых строк можно передать в аргументе:

```
In [3]: pd.set_option('precision', 2)  # Формат для значений с плавающей точкой

In [4]: titanic.head()
Out[4]:
                        Unnamed: 0 survived     sex    age passengerClass
0    Allen, Miss. Elisabeth Walton      yes  female  29.00            1st
1   Allison, Master. Hudson Trevor      yes    male   0.92            1st
2     Allison, Miss. Helen Loraine       no  female   2.00            1st
3  Allison, Mr. Hudson Joshua Crei       no    male  30.00            1st
4  Allison, Mrs. Hudson J C (Bessi       no  female  25.00            1st

In [5]: titanic.tail()
Out[5]:
                     Unnamed: 0 survived     sex    age  passengerClass
1304       Zabour, Miss. Hileni       no  female  14.50             3rd
1305      Zabour, Miss. Thamine       no  female    NaN             3rd
1306  Zakarian, Mr. Mapriededer       no    male  26.50             3rd
1307        Zakarian, Mr. Ortin       no    male  27.00             3rd
1308         Zimmerman, Mr. Leo       no    male  29.00             3rd
```

Обратите внимание: pandas регулирует ширину каждого столбца на основании самого широкого значения в столбце или имени столбца (в зависимости от того, какое имеет бóльшую ширину); в столбце age строки 1305 стоит значение `NaN` — признак отсутствующего значения в наборе данных.

Настройка имен столбцов

Имя первого столбца в наборе данных выглядит довольно странно (`'Unnamed: 0'`). Эту проблему можно решить настройкой имен столбцов. Заменим `'Unnamed: 0'` на `'name'` и сократим `'passengerClass'` до `'class'`:

```
In [6]: titanic.columns = ['name', 'survived', 'sex', 'age', 'class']

In [7]: titanic.head()
Out[7]:
                             name survived     sex    age class
```

[1] https://en.wikipedia.org/wiki/Passengers_of_the_RMS_Titanic.

```
0    Allen, Miss. Elisabeth Walton      yes  female  29.00   1st
1   Allison, Master. Hudson Trevor      yes    male   0.92   1st
2     Allison, Miss. Helen Loraine       no  female   2.00   1st
3  Allison, Mr. Hudson Joshua Crei       no    male  30.00   1st
4  Allison, Mrs. Hudson J C (Bessi       no  female  25.00   1st
```

9.12.4. Простой анализ данных на примере набора данных катастрофы «Титаника»

Теперь воспользуемся pandas для проведения простого анализа данных на примере некоторых характеристик описательной статистики. При вызове `describe` для коллекции `DataFrame`, содержащей как числовые, так и нечисловые столбцы, `describe` вычисляет статистические характеристики *только для числовых столбцов* — в данном случае только для столбца `age`:

```
In [8]: titanic.describe()
Out[8]:
            age
count  1046.00
mean     29.88
std      14.41
min       0.17
25%      21.00
50%      28.00
75%      39.00
max      80.00
```

Обратите внимание на расхождения в значении `count` (1046) и количества строк данных в наборе данных (1309 — при вызове `tail` индекс последней строки был равен 1308). Только 1046 строк данных (значение `count`) содержали значение age. Остальные результаты *отсутствовали* и были помечены `NaN`, как в строке 1305. При выполнении вычислений библиотека pandas *по умолчанию игнорирует отсутствующие данные* (`NaN`). Для 1046 пассажиров с действительным значением `age` средний возраст (математическое ожидание) составил `29.88` года. Самому молодому пассажиру (`min`) было всего два месяца (`0.17 * 12` дает `2.04`), а самому старому (`max`) — 80 лет. Медианный возраст был равен `28` (обозначается 50-процентным квартилем). 25-процентный квартиль описывает медианный возраст в первой половине пассажиров (ранжированных по возрасту), а 75-процентный квартиль — медиану во второй половине пассажиров.

Допустим, вы хотите вычислить статистику о выживших пассажирах. Мы можем сравнить столбец `survived` со значением `'yes'`, чтобы получить новую

коллекцию `Series` со значениями `True`/`False`, а затем использовать `describe` для описания результатов:

```
In [9]: (titanic.survived == 'yes').describe()
Out[9]:
count      1309
unique        2
top       False
freq        809
Name: survived, dtype: object
```

Для нечисловых данных `describe` выводит различные характеристики описательной статистики:

- `count` — общее количество элементов в результате;
- `unique` — количество уникальных значений (2) в результате — `True` (пассажир выжил) или `False` (пассажир погиб);
- `top` — значение, чаще всего встречающееся в результате;
- `freq` — количество вхождений значения `top`.

9.12.5. Гистограмма возраста пассажиров

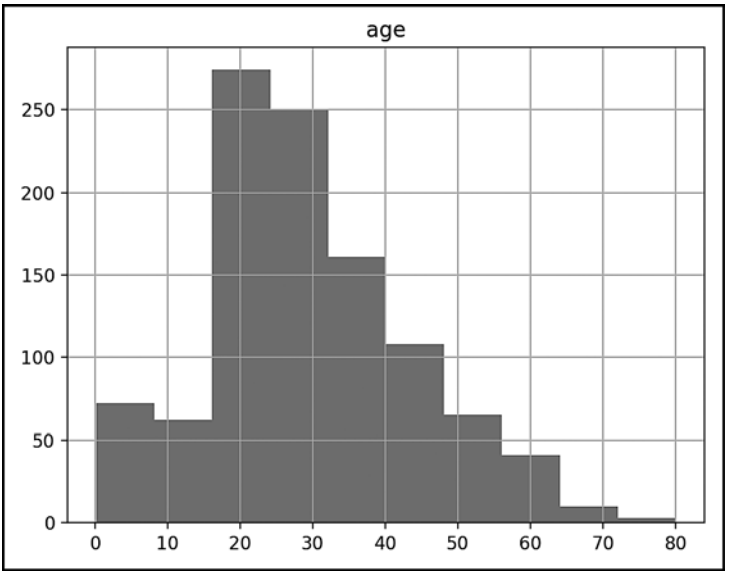
Визуализация — хороший способ поближе познакомиться с данными. Pandas содержит много встроенных средств визуализации, реализованных на базе Matplotlib. Чтобы использовать их, сначала включите поддержку Matplotlib в IPython:

```
In [10]: %matplotlib
```

Гистограмма наглядно показывает распределение числовых данных по диапазону значений. Метод `hist` коллекции `DataFrame` автоматически анализирует данные каждого числового столбца и строит соответствующую гистограмму. Чтобы просмотреть гистограммы по каждому числовому столбцу данных, вызовите `hist` для своей коллекции `DataFrame`:

```
In [11]: histogram = titanic.hist()
```

Набор данных катастрофы «Титаника» содержит только один числовой столбец данных, поэтому на диаграмме показана гистограмма для распределения возрастов. Для наборов данных с несколькими числовыми столбцами `hist` создает отдельную гистограмму для каждого числового столбца.



9.13. Итоги

Темой этой главы была обработка файлов и исключений. Файлы используются для долгосрочного хранения данных. Мы обсуждали объекты файлов и упомянули, что Python рассматривает файл как последовательность символов или байтов. Также были упомянуты стандартные объекты файлов, которые автоматически создаются при запуске программы Python.

Вы научились создавать, читать, записывать и обновлять текстовые файлы. Мы рассмотрели несколько популярных файловых форматов — txt, JSON (JavaScript Object Notation) и CSV (значения, разделенные запятыми). Встроенная функция `open` и команда `with` использовались для открытия файла, чтения и записи данных в файл, а также автоматического закрытия файла для предотвращения утечки ресурсов при завершении команды `with`. Модуль `json` стандартной библиотеки Python использовался для сериализации объектов в формат JSON и сохранения их в файле, загрузки объектов JSON из файла, десериализации их в объекты Python и структурного вывода объекта JSON для удобства чтения.

Вы узнали, что исключения указывают на возникшие проблемы во время выполнения. В главе приведен список различных исключений, с которыми вы уже сталкивались. Мы показали, как обрабатываются исключения, — для

этого код заключается в наборы команд `try` с секциями `except` для обработки конкретных типов исключений, которые могут возникнуть в наборе `try`, в результате чего ваши программы становятся более надежными и защищенными от ошибок.

Далее была рассмотрена секция `finally` команды `try` для гарантированного выполнения кода, если управление передается в соответствующий набор `try`. Для этой цели можно использовать либо команду `with`, либо секцию `finally` команды `try` — мы предпочитаем команду `with`.

В разделе «Введение в data science» модуль `csv` стандартной библиотеки Python и средства библиотеки pandas использовались для загрузки, обработки и сохранения данных в формате CSV. В завершение мы загрузили набор данных катастрофы «Титаника» в коллекцию `DataFrame`, поменяли имена некоторых столбцов для удобочитаемости, вывели начальную и конечную часть набора данных, а также провели простой анализ данных. В следующей главе будут рассмотрены средства объектно-ориентированного программирования Python.

10

Объектно-ориентированное программирование

В этой главе...

- Создание пользовательских классов и объектов этих классов.
- Преимущества создания полезных классов.
- Управление доступом к атрибутам.
- Полезность объектно-ориентированного программирования.
- Специальные методы Python `__repr__`, `__str__` и `__format__` для получения строковых представлений объекта.
- Специальные методы Python для перегрузки (переопределения) операторов для использования их с объектами новых классов.
- Наследование методов, свойств и атрибутов из существующих классов в новых классах и последующая модификация этих классов.
- Концепции базовых классов (суперклассов) и производных классов (подклассов).
- Роль утиной типизации и полиморфизма для реализации «обобщенного» программирования.
- Класс `object`, от которого все классы наследуют фундаментальные возможности.
- Сравнение композиции с наследованием.
- Встраивание тестовых сценариев в doc-строки и выполнение этих тестов с использованием `doctest`.
- Пространства имен и их влияние на область видимости.

10.1. Введение

В разделе 1.2 была представлена основная терминология и концепции объектно-ориентированного программирования. В Python нет ничего, кроме объектов, поэтому вы постоянно использовали объекты в этой книге. Подобно тому как дома строятся на основании планов, объекты строятся на основании классов, и это одна из основополагающих технологий объектно-ориентированного программирования. Заметим, что построить новый объект на основе даже сложного класса просто: обычно для этого достаточно одной команды.

Создание полезных классов

Вы уже использовали многие классы, созданные другими людьми. В этой главе мы займемся созданием своих *собственных* классов. Нас прежде всего будут интересовать «полезные классы», соответствующие потребностям ваших приложений. Вы будете использовать объектно-ориентированное программирование и его основополагающие технологии: классы, объекты, наследование и полиморфизм. В наши дни приложения становятся все более крупными и полнофункциональными. Объектно-ориентированное программирование упрощает проектирование, реализацию, тестирование, отладку и обновление таких ультрасовременных приложений. В разделах 10.1–10.9 содержится краткое введение в эти технологии с многочисленными примерами. Многие разработчики могут пропустить разделы с 10.10 по 10.15, которые содержат дополнительную информацию об этих технологиях и описывают некоторые сопутствующие возможности.

Библиотеки классов и объектно-базированное программирование

Подавляющее большинство объектно-ориентированного программирования в Python относится к области *объектно-базированного программирования*, в котором вы создаете и используете объекты *существующих* классов. Вы постоянно делали это в книге со встроенными типами, такими как `int`, `float`, `str`, `list`, `tuple`, `dict` и `set`; с типами стандартной библиотеки Python, такими как `Decimal`, и коллекциями `array` NumPy; типами Matplotlib `Figure` и `Axes`, а также коллекциями pandas `Series` и `DataFrame`.

Чтобы работать с Python с максимальной эффективностью, следует освоить многие готовые классы. За годы сообщество программирования с открытым кодом Python создало громадное количество полезных классов и упаковало их в библиотеки классов. Это позволяет повторно использовать существующие классы, вместо того чтобы «изобретать велосипед». Часто используемые классы библиотек с открытым кодом, как правило, тщательно протестированы, свободны от ошибок, оптимизированы и разработаны с учетом портирования по широкому спектру устройств, операционных систем и версий Python. Великое множество библиотек можно найти на сайтах GitHub, BitBucket, SourceForge и т. д., легко устанавливаемых при помощи `conda` или `pip`. Изобилие готовых классов стало одной из ключевых причин популярности Python. Абсолютное большинство классов, которые вам понадобятся, почти наверняка уже существуют в библиотеках с открытым кодом.

Создание собственных классов

Классы представляют собой новые типы данных. Каждый класс стандартной библиотеки Python и сторонних библиотек — тип, созданный другим разработчиком. В этой главе мы займемся разработкой классов для конкретных приложений — `CommissionEmployee`, `Time`, `Card`, `DeckOfCards` и т. д.

В большинстве приложений, которые вы будете строить для собственного использования, либо не понадобится создавать собственные классы, либо их количество будет минимальным. Если вы станете участником профессиональной группы разработки, то вам придется работать над приложениями с сотнями и даже тысячами классов. Вы можете публиковать написанные вами классы в сообществе библиотек с открытым кодом Python, но не обязаны это делать. Во многих организациях действуют свои политики и процедуры, относящиеся к разработке с открытым кодом.

Наследование

Возможно, наибольший интерес здесь представляет возможность формирования новых классов посредством наследования и композиции из классов многочисленных библиотек. Вероятно, в будущем программы будут практически полностью строиться из *стандартизированных компонентов, рассчитанных на повторное использование* (подобно тому как современные устройства строятся из взаимозаменяемых частей). Такой подход позволит справиться с трудностями разработки еще более мощных программных продуктов.

При создании нового класса, вместо того чтобы писать совершенно новый код, вы можете указать, что новый класс должен *наследовать* свои атрибуты (переменные) и методы (функции, принадлежащие классам) от ранее определенного *базового класса* (также называемого *суперклассом*). Новый класс, созданный посредством наследования, называется *производным классом* (или *подклассом*). После наследования вы можете модифицировать производный класс «под потребности» вашего приложения. Чтобы свести к минимуму объем дополнительной работы, всегда старайтесь наследовать от базового класса, который наиболее близок к вашим потребностям. Чтобы ваш выбор был эффективным, следует ознакомиться с библиотеками классов, предназначенными для приложений такого типа.

Полиморфизм

Мы объясним и продемонстрируем концепцию *полиморфизма*, позволяющую удобно программировать «на общем уровне» без привязки к подробностям. Вы просто адресуете вызов *одного* метода объектам, которые могут относиться ко многим *разным* типам. Каждый объект реагирует на вызов, выполняя соответствующую операцию. Таким образом, один вызов метода существует «во многих формах» — отсюда и термин «полиморфизм». Мы расскажем, как реализовать полиморфизм посредством наследования и возможности Python, называемой «утиной типизацией», приведя соответствующие примеры.

Практический пример: моделирование тасования и сдачи карт

Ранее мы уже использовали моделирование бросков кубиков с применением генератора случайных чисел для реализации популярной игры «крэпс». Ниже в этой главе будет рассмотрено моделирование процесса тасования и сдачи карт, который можно использовать для программирования карточных игр. Для этого мы, в частности, воспользуемся Matplotlib с изображениями карт, находящимися в открытом доступе, для вывода полной колоды карт до и после тасования.

Классы данных

Новые *классы данных* Python 3.7 упрощают построение классов благодаря более компактной записи и автоматическому генерированию частей классов. Первая реакция сообщества Python на классы данных была положительной. Как и с любой новой возможностью, может понадобиться какое-то время,

чтобы нововведение получило широкое распространение. Мы рассмотрим разработку классов как со старой, так и с новой технологией.

Другие концепции, представленные в этой главе

В этой главе представлены и некоторые другие концепции, суть которых сводится к:

- определению некоторых идентификаторов, используемых только внутри класса и остающихся недоступными для клиентов класса;
- применению специальных методов для создания строковых представлений объектов ваших классов и определению того, как ваши классы должны работать со встроенными операторами Python (процесс, называемый *перегрузкой операторов*);
- вхождению в иерархию классов исключений Python и созданию собственных классов исключений;
- тестированию кода посредством модуля `doctest` стандартной библиотеки Python;
- использованию пространств имен в Python для определения областей видимости идентификаторов.

10.2. Класс Account

Начнем с класса `Account` для представления банковского счета; в нем должны храниться имя владельца счета и баланс. Вероятно, в реальный класс банковского счета также будет включено много дополнительной информации: адрес, дата рождения, номер телефона, номер счета и т. д. Класс `Account` поддерживает возможность внесения средств с увеличением баланса, а также снятия средств с уменьшением баланса.

10.2.1. Класс Account в действии

Каждый новый класс становится и новым *типом данных*, который может использоваться для создания объектов. Это — одна из причин, по которым Python называется *расширяемым языком*. Прежде чем рассматривать определение класса `Account`, продемонстрируем его возможности.

Импортирование классов Account и Decimal

Чтобы использовать новый класс `Account`, запустите сеанс IPython из папки примеров `ch10`, а затем импортируйте класс `Account`:

```
In [1]: from account import Account
```

Класс `Account` ведет баланс счета и работает с ним как со значением формата `Decimal`, поэтому мы также импортируем класс `Decimal`:

```
In [2]: from decimal import Decimal
```

Создание объекта Account выражением-конструктором

Команда создания объекта `Decimal` выглядит примерно так:

```
value = Decimal('12.34')
```

Это выражение, называемое *выражением-конструктором*; строит и инициализирует объект класса по аналогии с тем, как дом строится на основе плана, а затем раскрашивается в цвета по желанию покупателя. Выражения-конструкторы создают новые объекты и инициализируют данные аргументами, заданными в круглых скобках. Отметим, что круглые скобки, следующие за именем класса, обязательны даже при отсутствии аргументов.

Воспользуемся выражением-конструктором для создания объекта `Account` и инициализируем его именем владельца счета (строка) и балансом (`Decimal`):

```
In [3]: account1 = Account('John Green', Decimal('50.00'))
```

Получение имени и баланса из объекта Account

Получим значения атрибутов `name` и `balance` объекта `Account`:

```
In [4]: account1.name
Out[4]: 'John Green'

In [5]: account1.balance
Out[5]: Decimal('50.00')
```

Внесение средств

Метод `deposit` класса `Account` получает положительную сумму в долларах и прибавляет ее к балансу:

```
In [6]: account1.deposit(Decimal('25.53'))

In [7]: account1.balance
Out[7]: Decimal('75.53')
```

Проверка данных в методах Account

Методы класса `Account` проверяют свои аргументы. Например, если вносимая сумма отрицательная, то метод `deposit` выдает исключение `ValueError`:

```
In [8]: account1.deposit(Decimal('-123.45'))
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-27dc468365a7> in <module>()
----> 1 account1.deposit(Decimal('-123.45'))

~/Documents/examples/ch10/account.py in deposit(self, amount)
     21         # Если amount меньше 0.00, выдать исключение
     22         if amount < Decimal('0.00'):
---> 23             raise ValueError('Deposit amount must be positive.')
     24
     25         self.balance += amount

ValueError: Deposit amount must be positive.
```

10.2.2. Определение класса Account

Теперь рассмотрим определение класса `Account`, находящееся в файле account.py.

Определение класса

Определение класса начинается с ключевого слова `class` (строка 5), за которым следует имя класса и двоеточие (`:`). Эта строка называется *заголовком класса*. «*Руководство по стилю для кода Python*» рекомендует начинать каждое слово в имени класса, состоящем из нескольких слов, с буквы верхнего регистра (например, `CommissionEmployee`). Каждая команда в наборе класса снабжается отступом.

```
1 # account.py
2 """Определение класса Account."""
3 from decimal import Decimal
4
5 class Account:
6     """Класс Account для ведения банковского счета."""
7
```

Каждый класс обычно предоставляет doc-строку с описанием (строка 6). Если doc-строка присутствует, то она обычно должна следовать непосредственно после заголовка класса. Чтобы просмотреть doc-строку любого класса в IPython, введите имя класса и вопросительный знак, после чего нажмите Enter:

```
In [9]: Account?
Init signature: Account(name, balance)
Docstring:      Account class for maintaining a bank account balance.
Init docstring: Initialize an Account object.
File:           ~/Documents/examples/ch10/account.py
Type:           type
```

Идентификатор `Account` определяет как имя класса, так и имя, используемое в выражении-конструкторе для создания объекта `Account` и вызова метода `__init__` класса. По этой причине механизм справки IPython выводит как doc-строку класса (`"Docstring:"`), так и doc-строку метода `__init__` (`"Init docstring:"`).

Инициализация объектов Account: метод __init__

Выражение-конструктор во фрагменте [3] из предыдущего раздела:

```
account1 = Account('John Green', Decimal('50.00'))
```

создает новый объект, а затем инициализирует его данные вызовом метода `__init__` класса. Каждый новый класс, созданный вами, может предоставлять метод `__init__`, который определяет, как должны инициализироваться атрибуты данных объекта. Возвращение методом `__init__` значения, отличного от `None`, приводит к исключению `TypeError`. Напомним, значение `None` возвращается любой функцией или методом, не содержащим команды `return`. Если значение `balance` действительно, то метод `__init__` класса `Account` (строки 8–16) инициализирует атрибуты `name` и `balance` объекта `Account`:

```
 8     def __init__(self, name, balance):
 9         """Инициализация объекта Account."""
10
11         # Если balance меньше 0.00, выдать исключение
12         if balance < Decimal('0.00'):
13             raise ValueError('Initial balance must be >= to 0.00.')
14
15         self.name = name
16         self.balance = balance
17
```

При вызове метода для конкретного объекта Python неявно передает ссылку на этот объект в первом аргументе метода. По этой причине все методы класса должны определяться хотя бы с одним параметром. По общепринятым соглашениям программисты Python присваивают первому параметру метода имя `self`. Методы класса должны использовать эту ссылку (`self`) для обращения к атрибутам и другим методам объекта. Метод `__init__` класса `Account` также определяет параметры `name` и `balance` для имени владельца счета и баланса.

Команда `if` *проверяет* параметр `balance`. Если значение `balance` меньше `0.00`, то `__init__` выдает ошибку `ValueError`, которая завершает метод `__init__`. В противном случае метод создает и инициализирует атрибуты `name` и `balance` нового объекта `Account`.

В момент создания объект класса `Account` еще не имеет никаких атрибутов. Они добавляются *динамически* присваиванием вида:

```
self.имя_атрибута = значение
```

Классы Python могут определять много *специальных методов* (таких, как `__init__`), обозначаемых начальными и конечными двойными подчеркиваниями (`__`) в имени метода. Класс Python `object`, который будет рассматриваться позднее в этой главе, определяет специальные методы, доступные для *всех* объектов Python.

Метод deposit

Метод `deposit` класса `Account` прибавляет положительную величину к атрибуту `balance` счета. Если аргумент `amount` меньше `0.00`, то метод выдает ошибку `ValueError`, указывающую на то, что разрешены только положительные вносимые суммы. Если `amount` проходит проверку, то строка 25 добавляет это значение к атрибуту `balance` объекта.

```
18     def deposit(self, amount):
19         """Внесение средств на счет."""
20
21         # Если amount меньше 0.00, выдать исключение
22         if amount < Decimal('0.00'):
23             raise ValueError('amount must be positive.')
24
25         self.balance += amount
```

10.2.3. Композиция: ссылка на объекты как компоненты классов

Объект `Account` *содержит* имя владельца (`name`) и баланс (`balance`.) Вспомните, что «в Python нет ничего, кроме объектов». Это означает, что атрибуты объектов являются ссылками на объекты других классов. Например, атрибут `name` объекта `Account` представляет собой ссылку на объект строки, а атрибут `balance` содержит ссылку на объект `Decimal`. Внедрение ссылки на объекты других типов является разновидностью повторного использования программных компонентов, которая обычно называется *композицией* (реже — *отношением «содержит»*). Позднее в этой главе будет рассмотрен механизм наследования, который создает *отношение «является»*.

10.3. Управление доступом к атрибутам

Методы класса `Account` проверяют свои аргументы, с тем чтобы значение `balance` *всегда* оставалось действительным, то есть бóльшим или равным `0.00`. В предыдущем примере атрибуты `name` и `balance` использовались только для *получения* их значений. Оказывается, атрибуты могут использоваться и для *изменения* этих значений. Рассмотрим объект `Account` в следующем сеансе IPython:

```
In [1]: from account import Account

In [2]: from decimal import Decimal

In [3]: account1 = Account('John Green', Decimal('50.00'))

In [4]: account1.balance
Out[4]: Decimal('50.00')
```

Изначально `account1` содержит действительный баланс. Теперь попробуем присвоить атрибуту `balance` *недопустимое* отрицательное значение, а затем вывести `balance`:

```
In [5]: account1.balance = Decimal('-1000.00')

In [6]: account1.balance
Out[6]: Decimal('-1000.00')
```

Вывод фрагмента [6] показывает, что баланс `account1` стал отрицательным. Таким образом, в отличие от методов, атрибуты данных не могут проверять присвоенные им значения.

Инкапсуляция

Клиентским кодом класса называется любой код, использующий объекты этого класса. Большинство объектно-ориентированных языков программирования позволяет *инкапсулировать* (или *скрыть*) от клиентского кода данные объекта, называемые по этой причине *приватными данными*.

Соглашение об именах с начальными подчеркиваниями (_)

В Python *не существует* приватных данных. Вместо этого при проектировании классов используется *соглашение (соглашения) об именах*, способствующее правильному использованию. Программисты Python знают, что по распространенным соглашениям любое имя атрибута, начинающееся с символа подчеркивания (_), предназначено *только* для *внутреннего использования* классом. Клиентский код должен использовать методы класса и (как будет показано в следующем разделе) свойства класса для взаимодействия с атрибутами данных каждого объекта, предназначенными исключительно для внутреннего использования. Атрибуты, идентификаторы которых *не* начинаются с символа подчеркивания (_), считаются *общедоступными* для использования в клиентском коде. В следующем разделе мы определим класс `Time` и воспользуемся этими соглашениями. Тем не менее и при использовании соглашений атрибуты все равно остаются доступными.

10.4. Использование свойств для доступа к данным

Разработаем класс `Time`, предназначенный для хранения времени в 24-часовом формате: часы в диапазоне 0–23, а минуты и секунды в диапазоне 0–59. Для этого класса мы определим *свойства*, которые с точки зрения клиентских программистов похожи на атрибуты данных, но управляют способом чтения и записи данных объекта. Предполагается, что другие программисты соблюдают соглашения Python для правильного использования объектов вашего класса.

10.4.1. Класс Time в действии

Прежде чем рассматривать определение класса `Time`, продемонстрируем его возможности. Для начала убедитесь в том, что текущим каталогом является ch10, и импортируйте класс `Time` из файла timewithproperties.py:

```
In [1]: from timewithproperties import Time
```

Создание объекта Time

Теперь создадим объект `Time`. Метод `__init__` класса `Time` получает параметры `hour`, `minute` и `second`, каждому из которых по умолчанию соответствует значение аргумента 0. В данном случае мы задаем значения для `hour` и `minute` — `second` по умолчанию использует значение 0:

```
In [2]: wake_up = Time(hour=6, minute=30)
```

Вывод объекта Time

Класс `Time` определяет два метода для формирования строковых представлений объекта `Time`. Когда вы выводите переменную в IPython, как во фрагменте [3], IPython вызывает специальный метод `__repr__` объекта для получения строкового представления объекта. Реализация `__repr__` создает строку в таком формате:

```
In [3]: wake_up
Out[3]: Time(hour=6, minute=30, second=0)
```

Мы также предоставили специальный метод `__str__`, который вызывается при преобразовании объекта в строку, например при выводе объекта вызовом `print`[1]. Наша реализация `__str__` создает строку в 12-часовом формате времени:

```
In [4]: print(wake_up)
6:30:00 AM
```

Получение атрибута через свойство

Класс `Time` предоставляет *свойства* `hour`, `minute` и `second`, которые не уступают по удобству атрибутам данных для получения и изменения данных объекта. Тем не менее, как вы вскоре увидите, свойства реализуются как методы и поэтому могут содержать дополнительную логику — например, для определения формата, в котором должно возвращаться значение атрибута данных, или для проверки нового значения перед его использованием для изменения атрибута данных. В следующем примере мы получаем значение `hour` объекта `wake_up`:

```
In [5]: wake_up.hour
Out[5]: 6
```

---

[1] Если класс не предоставляет метод `__str__`, то при преобразовании объекта класса в строку будет вызван метод `__repr__` класса.

И хотя этот фрагмент вроде бы получает значение атрибута данных `hour`, в действительности это вызов *метода* `hour`, который возвращает значение атрибута данных (который мы назвали `_hour`, как будет показано в следующем разделе).

Присваивание времени

Вы можете задать новое значение времени при помощи метода `set_time` объекта `Time`. Как и метод `__init__`, метод `set_time` имеет параметры `hour`, `minute` и `second`, каждый из которых имеет значение по умолчанию 0:

```
In [6]: wake_up.set_time(hour=7, minute=45)

In [7]: wake_up
Out[7]: Time(hour=7, minute=45, second=0)
```

Присваивание значения атрибута через свойство

Класс `Time` позволяет задать значения `hour`, `minute` и `second` по отдельности при помощи свойств. Присвоим `hour` значение `6`:

```
In [8]: wake_up.hour = 6

In [9]: wake_up
Out[9]: Time(hour=6, minute=45, second=0)
```

Хотя на первый взгляд фрагмент [8] просто присваивает значение атрибуту данных, в действительности это вызов метода `hour`, который получает значение `6` в аргументе. Метод проверяет значение, а затем присваивает его соответствующему атрибуту данных (который мы назвали `_hour`, как будет показано в следующем разделе).

Попытка присваивания недействительного значения

Чтобы доказать, что свойства класса `Time` *проверяют* значения, которые вы пытаетесь им присвоить, попробуем задать некорректное значение для свойства `hour`. Это приводит к ошибке `ValueError`:

```
In [10]: wake_up.hour = 100
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-1fce0716ef14> in <module>()
----> 1 wake_up.hour = 100
```

```
~/Documents/examples/ch10/timewithproperties.py in hour(self, hour)
     20         """Настройка hour."""
     21         if not (0 <= hour < 24):
---> 22             raise ValueError(f'Hour ({hour}) must be 0-23')
     23
     24         self._hour = hour

ValueError: Hour (100) must be 0-23
```

10.4.2. Определение класса Time

Итак, вы увидели класс `Time` в действии. Обратимся к его определению.

Класс Time: метод __init__ со значениями параметров по умолчанию

Метод `__init__` класса `Time` получает параметры `hour`, `minute` и `second` со значением 0 по умолчанию. Как и в случае с методом `__init__` класса `Account`, параметр `self` содержит ссылку на инициализируемый объект `Time`. Команды, содержащие `self.hour`, `self.minute` и `self.second`, *вроде бы* создают атрибуты `hour`, `minute` и `second` для нового объекта `Time` (`self`). Тем не менее эти команды в действительности вызывают методы, реализующие *свойства* `hour`, `minute` и `second` класса (строки 13–50). Затем эти методы создают атрибуты с именами `_hour`, `_minute` и `_second`, предназначенные только для использования внутри класса:

```
 1 # timewithproperties.py
 2 """Класс Time со свойствами, доступными для чтения/записи."""
 3
 4 class Time:
 5     """Класс Time со свойствами, доступными для чтения/записи."""
 6
 7     def __init__(self, hour=0, minute=0, second=0):
 8         """Инициализация каждого атрибута."""
 9         self.hour = hour  # 0-23
10         self.minute = minute  # 0-59
11         self.second = second  # 0-59
12
```

Класс Time: свойство hour

Строки 13–24 определяют *общедоступное* свойство `hour`, *доступное для чтения записи*, которое работает с атрибутом данных с именем `_hour`. Соглашение о начальном символе подчеркивания (`_`) указывает, что клиентский код не

должен обращаться к `_hour` напрямую. Как показывают фрагменты [5] и [8] предыдущего раздела, с точки зрения программиста, работающего с объектами `Time`, свойства очень похожи на атрибуты данных. Тем не менее следует помнить, что свойства реализованы в виде *методов*. Каждое свойство определяет *get-метод*, который *получает* значение атрибута данных, а также может определить *set-метод*, который *задает* значение атрибута данных:

```
13     @property
14     def hour(self):
15         """Возвращает значение часов."""
16         return self._hour
17
18     @hour.setter
19     def hour(self, hour):
20         """Присваивает значение часов."""
21         if not (0 <= hour < 24):
22             raise ValueError(f'Hour ({hour}) must be 0-23')
23
24         self._hour = hour
25
```

Декоратор `@property` предшествует *get-методу* свойства, который получает только параметр `self`. Во внутренней реализации декоратор добавляет в декорированную функцию специальный код — в данном случае для того, чтобы функция `hour` работала с синтаксисом атрибутов. Имя *get-метода* совпадает с именем свойства. *Get-метод* возвращает значение атрибута данных `hour`. Следующее выражение в клиентском коде приводит к вызову *get-метода*:

```
wake_up.hour
```

Как вы вскоре увидите, *get-метод* также может использоваться внутри класса.

Декоратор в форме `@`*имя_свойства*`.setter` (в данном случае `@hour.setter`) предшествует *set-методу* свойства. Метод получает два параметра — `self` и параметр (`hour`), представляющий значение, присваиваемое свойству. Если значение параметра `hour` *действительно*, то этот метод присваивает его атрибуту `_hour` объекта `self`; в противном случае метод выдает ошибку `ValueError`. Следующее выражение в клиентском коде приводит к вызову *set-метода* при присваивании значения свойству:

```
wake_up.hour = 8
```

Также `set-метод` вызывается внутри класса в строке 9 метода `__init__`:

```
self.hour = hour
```

Использование *set-метода* позволило *проверить* аргумент `hour` метода `__init__` *перед* созданием и инициализацией атрибута `_hour` объекта; это происходит при *первом* выполнении *set-метода* свойства `hour` в результате выполнения строки 9. *Свойство, доступное для чтения/записи*, имеет как *get-*, так и *set-метод*. *Свойство, доступное только для чтения*, имеет только *get-метод*.

Класс Time: свойства minute и second

Строки 26–37 и 39–50 определяют свойства `minute` и `second`, доступные для чтения/записи. Set-метод каждого свойства проверяет, что его второй аргумент принадлежит диапазону 0–59 (допустимый диапазон значений для минут и секунд):

```
26     @property
27     def minute(self):
28         """Возвращает значение минут."""
29         return self._minute
30
31     @minute.setter
32     def minute(self, minute):
33         """Присваивает значение минут."""
34         if not (0 <= minute < 60):
35             raise ValueError(f'Minute ({minute}) must be 0-59')
36
37         self._minute = minute
38
39     @property
40     def second(self):
41         """Возвращает значение секунд."""
42         return self._second
43
44     @second.setter
45     def second(self, second):
46         """Присваивает значение секунд."""
47         if not (0 <= second < 60):
48             raise ValueError(f'Second ({second}) must be 0-59')
49
50         self._second = second
51
```

Класс Time: метод set_time

Метод `set_time` предоставляется как удобный способ изменения *всех трех* атрибутов *одним* вызовом метода. В строках 54–56 вызываются *set-методы* для свойств `hour`, `minute` и `second`:

```
52     def set_time(self, hour=0, minute=0, second=0):
53         """Присваивает значения часов, минут и секунд."""
54         self.hour = hour
55         self.minute = minute
56         self.second = second
57
```

Класс Time: специальный метод __repr__

При передаче объекта встроенной функции `repr`, что неявно происходит при выводе значения переменной в сеансе IPython, вызывается *специальный метод* `__repr__` соответствующего класса для получения строкового представления объекта:

```
58     def __repr__(self):
59         """Возвращает строку Time для repr()."""
60         return (f'Time(hour={self.hour}, minute={self.minute}, ' +
61                 f'second={self.second})')
62
```

В документации Python сказано, что `__repr__` возвращает «официальное» строковое представление объекта. Как правило, эта строка выглядит как выражение-конструктор, которое создает и инициализирует объект[1], как в следующем примере:

```
'Time(hour=6, minute=30, second=0)',
```

что похоже на выражение-конструктор из фрагмента [2] в предыдущем разделе. В Python существует встроенная функция `eval`, позволяющая получить эту строку в аргументе и использовать ее для создания и инициализации объекта `Time`, который содержит значения, заданные в строке.

Класс Time: специальный метод __str__

Для класса `Time` мы также определяем специальный метод `__str__`. Этот метод неявно вызывается при преобразовании объекта в строку встроенной функцией `str`, как при выводе объекта или явном вызове `str`. Наша реализация `__str__` создает строку в 12-часовом формате времени вида `'7:59:59 AM'` или `'12:30:45 PM'`:

---

[1] https://docs.python.org/3/reference/datamodel.html.

```
63     def __str__(self):
64         """Выводит объект Time в 12-часовом формате времени."""
65         return (('12' if self.hour in (0, 12) else str(self.hour % 12)) +
66                 f':{self.minute:0>2}:{self.second:0>2}' +
67                 (' AM' if self.hour < 12 else ' PM'))
```

10.4.3. Замечания по проектированию определения класса Time

Рассмотрим некоторые аспекты проектирования классов в контексте класса `Time`.

Интерфейс класса

Свойства и методы класса `Time` определяют *открытый интерфейс*, то есть набор свойств и методов, которые должны использоваться программистами для взаимодействия с объектами класса.

Атрибуты всегда доступны

Хотя мы предоставили четко определенный интерфейс, Python *не* помешает напрямую обратиться к атрибутам данных `_hour`, `_minute` и `_second`:

```
In [1]: from timewithproperties import Time

In [2]: wake_up = Time(hour=7, minute=45, second=30)

In [3]: wake_up._hour
Out[3]: 7

In [4]: wake_up._hour = 100

In [5]: wake_up
Out[5]: Time(hour=100, minute=45, second=30)
```

После выполнения фрагмента [4] объект `wake_up` содержит *недействительные* данные. В отличие от многих других языков объектно-ориентированного программирования (таких как C++, Java и C#), атрибуты данных в Python не могут быть скрыты от клиентского кода. В документации Python сказано: «**В Python нет ничего, что сделало бы возможным сокрытие данных**, — **все основано на соглашениях**»[1].

[1] https://docs.python.org/3/tutorial/classes.html#random-remarks.

Внутреннее представление данных

Итак, мы выбрали представление времени в виде трех целочисленных значений — для часов, минут и секунд. Было бы совершенно нормально использовать внутреннее представление времени как число секунд, прошедших с полуночи. И хотя нам пришлось бы заново реализовать свойства `hour`, `minute` и `second`, программисты смогли бы использовать *тот же* интерфейс и получить *те же* результаты, ничего не зная об этих изменениях. Вы можете самостоятельно внести это изменение и убедиться в том, что клиентский код, использующий объекты `Time`, при этом остается неизменным.

Усовершенствование подробностей реализации класса

Занимаясь проектированием класса, тщательно продумайте интерфейс класса, перед тем как сделать этот класс доступным для других программистов. В идеале интерфейс должен быть спроектирован так, чтобы существующий код сохранял работоспособность при обновлении подробностей реализации класса (представления внутренних данных или способа реализации тел методов).

Если программисты Python следуют соглашениям и не обращаются к атрибутам, начинающимся с символа `_`, то проектировщики класса смогут совершенствовать подробности реализации без нарушения работоспособности клиентского кода.

Свойства

Может показаться, что определение свойств с *set-* и *get-методами* не дает особых преимуществ перед прямым обращением к атрибутам данных, но здесь существуют тонкие различия. *Get-метод* вроде бы позволяет клиентам читать данные по своему усмотрению, но в то же время *get-метод* может управлять форматированием данных. *Set-метод* может анализировать попытки изменения значения атрибута данных, предотвращая присваивание данным недопустимых значений.

Вспомогательные методы

Не все методы служат частью интерфейса класса. Некоторые из них, так называемые *вспомогательные методы*, используются только *внутри* класса и не предназначены быть частью открытого интерфейса класса, используемого клиентским кодом. Имена таких методов должны начинаться с одного сим-

вола подчеркивания. В других объектно-ориентированных языках (таких, как C++, Java и C#) такие методы обычно реализуются как приватные (см. ниже).

Модуль datetime

Профессиональные разработчики Python обычно не создают собственные классы для представления даты и времени — как правило, они пользуются функциональностью модуля `datetime` стандартной библиотеки Python. За дополнительной информацией обращайтесь по адресу:

https://docs.python.org/3/library/datetime.html

10.5. Моделирование «приватных» атрибутов

В таких языках программирования, как C++, Java и C#, классы явно объявляют, какие их компоненты являются общедоступными, то есть *открытыми*. Компоненты класса, к которым невозможно обратиться за пределами определения класса, называются *приватными*; они видимы только в том классе, который их определяет. Программисты Python часто используют «приватные» атрибуты для данных и вспомогательных методов, которые необходимы для работы внутренних механизмов класса, но не являются частью его открытого интерфейса.

Атрибуты объектов Python доступны *всегда*. Тем не менее в Python предусмотрены соглашения имен для обозначения «приватных» атрибутов. Допустим, вы хотите создать объект класса `Time` и *запретить* команды присваивания следующего вида:

```
wake_up._hour = 100
```

что привело бы к присваиванию недействительного значения часа. Вместо `_hour` можно присвоить атрибуту имя `__hour` с *двумя* начальными символами подчеркивания. Это соглашение означает, что `__hour` является «приватным» компонентом, который не должен быть доступен для клиентов класса. Чтобы помешать клиентам обращаться к «приватным» атрибутам, Python *переименовывает* их, ставя перед именем атрибута префикс *_ИмяКласса* — например, `_Time__hour`. Эта процедура называется *преобразованием имени*. Если вы попытаетесь выполнить присваивание `__hour`:

```
wake_up.__hour = 100
```

Python выдаст ошибку `AttributeError`, которая указывает на то, что у класса нет атрибута `__hour` (вскоре мы продемонстрируем этот факт).

Механизм автозаполнения IPython выводит только «открытые» атрибуты

Кроме того, IPython не включает атрибуты с одним или двумя начальными символами _ в список автозаполнения при нажатии *Tab* для выражений вида

```
wake_up
```

В списке автозаполнения IPython присутствуют только атрибуты, являющиеся частью «открытого» интерфейса объекта `wake_up`.

Демонстрация «приватных» атрибутов

Чтобы показать, как работает преобразование имени, возьмем класс `PrivateClass` с одним «открытым» атрибутом данных `public_data` и одним «приватным» атрибутом данных `__private_data`:

```
 1 # private.py
 2 """Класс с открытыми и приватными атрибутами."""
 3
 4 class PrivateClass:
 5     """Класс с открытыми и приватными атрибутами."""
 6
 7     def __init__(self):
 8         """Инициализировать открытые и приватные атрибуты."""
 9         self.public_data = "public"  # Открытый атрибут
10         self.__private_data = "private"  # Приватный атрибут
```

Создадим объект класса `PrivateData` для демонстрации этих атрибутов данных:

```
In [1]: from private import PrivateClass

In [2]: my_object = PrivateClass()
```

Фрагмент [3] показывает, что к атрибуту `public_data` можно обратиться напрямую:

```
In [3]: my_object.public_data
Out[3]: 'public'
```

Тем не менее при попытке обратиться напрямую к `__private_data` в фрагменте [4] будет получена ошибка `AttributeError` с сообщением о том, что класс не содержит атрибута с таким именем:

```
In [4]: my_object.__private_data
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-d896bfdf2053> in <module>()
----> 1 my_object.__private_data

AttributeError: 'PrivateClass' object has no attribute '__private_data'
```

Это происходит из-за того, что Python изменил имя атрибута. К сожалению, атрибут `__private_data` при этом остается косвенно доступным.

10.6. Практический пример: моделирование тасования и сдачи карт

В следующем примере представлены два класса, которые могут использоваться для тасования и сдачи карт. Класс `Card` представляет игральную карту с номиналом (`'Ace'`, `'2'`, `'3'`, ..., `'Jack'`, `'Queen'`, `'King'`) и мастью (`'Hearts'`, `'Diamonds'`, `'Clubs'`, `'Spades'`). Класс `DeckOfCards` представляет колоду из 52 карт в виде списка объектов `Card`. Сначала мы протестируем эти классы в сеансе IPython, чтобы продемонстрировать функции тасования и сдачи, а также вывода карт в текстовом виде. Затем будут рассмотрены определения классов. Наконец, мы используем другой сеанс IPython для вывода 52 карт в графическом виде при помощи библиотеки Matplotlib. Заодно вы узнаете, где можно найти красивые изображения карт, находящиеся в открытом доступе.

10.6.1. Классы Card и DeckOfCards в действии

Прежде чем рассматривать код классов `Card` и `DeckOfCards`, воспользуемся сеансом IPython для демонстрации их возможностей.

Создание объектов, тасование и сдача карт

Импортируйте класс `DeckOfCards` из deck.py и создайте объект этого класса:

```
In [1]: from deck import DeckOfCards

In [2]: deck_of_cards = DeckOfCards()
```

Метод `__init__` класса `DeckOfCards` создает 52 объекта `Card`, упорядоченных по мастям и по номиналу в каждой масти. Чтобы убедиться в этом, можно вывести объект `deck_of_cards`, который вызывает метод `__str__` класса `DeckOfCards` для получения строкового представления колоды карт. Прочитайте каждую строку слева направо и убедитесь, что все карты идут по порядку в каждой масти:

```
In [3]: print(deck_of_cards)
Ace of Hearts      2 of Hearts        3 of Hearts        4 of Hearts
5 of Hearts        6 of Hearts        7 of Hearts        8 of Hearts
9 of Hearts        10 of Hearts       Jack of Hearts     Queen of Hearts
King of Hearts     Ace of Diamonds    2 of Diamonds      3 of Diamonds
4 of Diamonds      5 of Diamonds      6 of Diamonds      7 of Diamonds
8 of Diamonds      9 of Diamonds      10 of Diamonds     Jack of Diamonds
Queen of Diamonds  King of Diamonds   Ace of Clubs       2 of Clubs
3 of Clubs         4 of Clubs         5 of Clubs         6 of Clubs
7 of Clubs         8 of Clubs         9 of Clubs         10 of Clubs
Jack of Clubs      Queen of Clubs     King of Clubs      Ace of Spades
2 of Spades        3 of Spades        4 of Spades        5 of Spades
6 of Spades        7 of Spades        8 of Spades        9 of Spades
10 of Spades       Jack of Spades     Queen of Spades    King of Spades
```

Перетасуем колоду и снова выведем объект `deck_of_cards`. Мы не стали задавать значение для инициализации генератора, поэтому при каждом тасовании вы будете получать разные результаты:

```
In [4]: deck_of_cards.shuffle()

In [5]: print(deck_of_cards)
King of Hearts     Queen of Clubs     Queen of Diamonds  10 of Clubs
5 of Hearts        7 of Hearts        4 of Hearts        2 of Hearts
5 of Clubs         8 of Diamonds      3 of Hearts        10 of Hearts
8 of Spades        5 of Spades        Queen of Spades    Ace of Clubs
8 of Clubs         7 of Spades        Jack of Diamonds   10 of Spades
4 of Diamonds      8 of Hearts        6 of Spades        King of Spades
9 of Hearts        4 of Spades        6 of Clubs         King of Clubs
3 of Spades        9 of Diamonds      3 of Clubs         Ace of Spades
Ace of Hearts      3 of Diamonds      2 of Diamonds      6 of Hearts
King of Diamonds   Jack of Spades     Jack of Clubs      2 of Spades
5 of Diamonds      4 of Clubs         Queen of Hearts    9 of Clubs
10 of Diamonds     2 of Clubs         Ace of Diamonds    7 of Diamonds
9 of Spades        Jack of Hearts     6 of Diamonds      7 of Clubs
```

Сдача карт

Вызов метода `deal_card` сдает карты по одной. IPython вызывает метод `__repr__` возвращенного объекта для получения строкового вывода, показанного в приглашении `Out[]`:

```
In [6]: deck_of_cards.deal_card()
Out[6]: Card(face='King', suit='Hearts')
```

Другие возможности класса Card

Чтобы продемонстрировать работу метода `__str__` класса `Card`, сдадим еще одну карту и передадим ее встроенной функции `str`:

```
In [7]: card = deck_of_cards.deal_card()

In [8]: str(card)
Out[8]: 'Queen of Clubs'
```

У каждого объекта `Card` существует соответствующий файл с графическим изображением, которое можно получить из свойства `image_name`, доступного только для чтения. Вскоре мы воспользуемся этой возможностью, когда будем выводить объекты `Card` в графическом виде:

```
In [9]: card.image_name
Out[9]: 'Queen_of_Clubs.png'
```

10.6.2. Класс Card — знакомство с атрибутами класса

Каждый объект `Card` содержит три строковых свойства, представляющих номинал карты (`face`), масть (`suit`) и имя файла, содержащего соответствующее изображение (`image_name`). Как было показано в сеансе IPython из предыдущего раздела, класс `Card` также предоставляет методы для инициализации `Card` и получения различных строковых представлений.

Атрибуты класса FACES и SUITS

Каждый объект класса содержит собственные копии атрибутов данных класса. Например, каждый объект `Account` содержит собственные атрибуты имени владельца (`name`) и баланса (`balance`). Иногда атрибут должен совместно использоваться *всеми* объектами класса. *Атрибут класса* (также называемый *переменной класса*) представляет информацию *уровня класса*, которая принадлежит *классу*, а не конкретному объекту этого класса. Класс `Card` определяет два атрибута класса (строки 5–7):

- `FACES` — список имен номиналов карт.
- `SUITS` — список имен мастей карт.

```
1 # card.py
2 """Класс Card представляет игральную карту и имя файла с ее изображением."""
3
4 class Card:
5     FACES = ['Ace', '2', '3', '4', '5', '6',
6              '7', '8', '9', '10', 'Jack', 'Queen', 'King']
7     SUITS = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
8
```

Чтобы определить атрибут класса, присвойте ему значение в определении класса, но не в методах или свойствах класса (в этом случае они станут локальными переменными). *Константы* `FACES` и `SUITS` не должны изменяться. Вспомните, что «*Руководство по стилю для кода Python*» рекомендует присваивать константам имена, состоящие из букв верхнего регистра[1].

Мы будем использовать элементы этих списков для инициализации всех создаваемых объектов `Card`. Тем не менее хранить копию каждого списка в каждом объекте `Card` не нужно. К атрибутам классов можно обращаться из любого объекта класса, но обычно программы обращаются к ним с указанием имени класса: `Card.FACES`, `Card.SUITS` и т. п. Атрибуты классов начинают существовать сразу же с импортирования определений их классов.

Метод __init__

При создании объекта `Card` метод `__init__` определяет атрибуты данных `_face` и `_suit`:

```
 9     def __init__(self, face, suit):
10         """Инициализирует карту номиналом и мастью."""
11         self._face = face
12         self._suit = suit
13
```

Свойства face, suit и image_name

После того как объект `Card` будет создан, значения `face`, `suit` и `image_name` не изменяются, поэтому мы реализуем их в виде свойств, доступных только для чтения (строки 14–17, 19–22 и 24–27). Свойства `face` и `suit` возвращают соответствующие атрибуты данных `_face` и `_suit`. Свойство не обязано иметь соответствующий атрибут данных. Для демонстрации этой возмож-

[1] Напомним, что в Python нет полноценных констант, так что значения FACES и SUITS могут изменяться.

ности значение свойства `image_name` объекта `Card` *создается динамически*: для этого свойство получает строковое представление объекта `Card` вызовом `str(self)`, заменяет все пробелы символами подчеркивания и присоединяет расширение `'.png'`. Таким образом, `'Ace of Spades'` преобразуется в `'Ace_of_Spades.png'`. Это имя файла будет использоваться для загрузки изображения в формате PNG, представляющего карту. PNG (Portable Network Graphics) — популярный графический формат для изображений, размещаемых в интернете.

```
14      @property
15      def face(self):
16          """Возвращает значение self._face объекта Card."""
17          return self._face
18
19      @property
20      def suit(self):
21          """Возвращает значение self._suit объекта Card."""
22          return self._suit
23
24      @property
25      def image_name(self):
26          """Возвращает имя файла изображения объекта Card."""
27          return str(self).replace(' ', '_') + '.png'
28
```

Методы, возвращающие строковое представление Card

Класс `Card` предоставляет три специальных метода, возвращающих строковые представления. Как и в классе `Time`, метод `__repr__` возвращает строковое представление, которое выглядит как выражение-конструктор для создания и инициализации объекта `Card`:

```
29      def __repr__(self):
30          """Возвращает строковое представление для repr()."""
31          return f"Card(face='{self.face}', suit='{self.suit}')"
32
```

Метод `__str__` возвращает строку в формате '*face* `of` *suit*' — например, `'Ace of Hearts'`:

```
33      def __str__(self):
34          """Возвращает строковое представление для str()."""
35          return f'{self.face} of {self.suit}'
36
```

Когда сеанс IPython из предыдущего раздела выводил всю колоду, вы видели, что объекты `Card` выводились в четыре столбца, выровненных по левому краю. Как будет показано в методе `__str__` класса `DeckOfCards`, для форматирования объектов `Card` по полям, состоящим из 19 символов, используются форматные строки. Специальный метод `__format__` класса `Card` вызывается при *форматировании* объекта `Card` как строки — например, в форматной строке:

```
37     def __format__(self, format):
38         """Возвращает отформатированное строковое представление для str()."""
39         return f'{str(self):{format}}'
```

Второй аргумент этого метода содержит форматную строку, используемую для форматирования объекта. Чтобы использовать значение параметра `format` как спецификатор формата, заключите имя параметра в фигурные скобки *справа* от двоеточия. В данном случае форматируется строковое представление объекта `Card`, возвращаемое `str(self)`. Мы снова обсудим `__format__` при описании метода `__str__` в классе `DeckOfCards`.

10.6.3. Класс DeckOfCards

Класс `DeckOfCards` содержит атрибут класса `NUMBER_OF_CARDS`, представляющий количество объектов `Card` в колоде, и создает два атрибута данных:

- атрибут `_current_card` отслеживает карту, которая будет сдана следующей (0–51), и;
- атрибут `_deck` (строка 12) содержит список 52 объектов `Card`.

Метод __init__

Метод `__init__` класса `DeckOfCards` инициализирует колоду (`_deck`) объектов `Card`. Команда `for` заполняет список `_deck` присоединением новых объектов `Card`, каждый из которых инициализируется двумя строками — из списка `Card.FACES` и из списка `Card.SUITS`. Выражение `count % 13` *всегда* дает значение от 0 до 12 (13 индексов `Card.FACES`), а выражение `count // 13` *всегда* дает значение от 0 до 3 (четыре индекса `Card.SUITS`). При инициализации списка `_deck` он содержит объекты `Card` с номиналами от `'Ace'` до `'King'` по порядку для всех четырех мастей (`Hearts`, `Diamonds`, `Clubs`, `Spades`).

```
1 # deck.py
2 """Класс DeckofCards представляет колоду карт."""
3 import random
```

```
 4 from card import Card
 5
 6 class DeckOfCards:
 7     NUMBER_OF_CARDS = 52  # число карт - константа
 8
 9     def __init__(self):
10         """Инициализирует колоду."""
11         self._current_card = 0
12         self._deck = []
13
14         for count in range(DeckOfCards.NUMBER_OF_CARDS):
15             self._deck.append(Card(Card.FACES[count % 13],
16                 Card.SUITS[count // 13]))
17
```

Метод shuffle

Метод `shuffle` обнуляет `_current_card`, после чего перетасовывает карты в колоде `_deck` при помощи функции `shuffle` модуля `random`:

```
18     def shuffle(self):
19         """Тасует колоду."""
20         self._current_card = 0
21         random.shuffle(self._deck)
22
```

Метод deal_card

Метод `deal_card` сдает одну карту (объект `Card`) из колоды (`_deck`). Напомним, `_current_card` определяет индекс (0–51) следующего объекта `Card` для сдачи (то есть карты на верху колоды). Строка 26 пытается получить элемент `_deck` с индексом `_current_card`. Если попытка оказалась успешной, то метод увеличивает `_current_card` на 1, после чего возвращает объект `Card`; в противном случае метод возвращает `None`, чтобы показать, что в колоде не осталось ни одной карты.

```
23     def deal_card(self):
24         """Возвращает одну карту."""
25         try:
26             card = self._deck[self._current_card]
27             self._current_card += 1
28             return card
29         except:
30             return None
31
```

Метод __str__

Класс DeckOfCards также определяет специальный метод __str__ для получения строкового представления колоды из четырех столбцов, в котором каждый объект Card выравнивается по левому краю поля из 19 символов. Когда строка 37 форматирует объект Card, его специальный метод __format__ вызывается с передачей форматного спецификатора '<19' в аргументе format. Затем метод __format__ использует значение '<19' для создания отформатированного строкового представления Card.

```
32     def __str__(self):
33         """Возвращает строковое представление текущей колоды _deck."""
34         s = ''
35
36         for index, card in enumerate(self._deck):
37             s += f'{self._deck[index]:<19}'
38             if (index + 1) % 4 == 0:
39                 s += '\n'
40
41         return s
```

10.6.4. Вывод изображений карт средствами Matplotlib

До настоящего момента объекты Card выводились в текстовом виде. Теперь перейдем на вывод изображений Card. Для этой демонстрации мы загрузили изображения карт, находящиеся в открытом доступе[1], из Wikimedia Commons:

https://commons.wikimedia.org/wiki/Category:SVG_English_pattern_playing_cards

В каталоге примеров ch10 находится подкаталог card_images с изображениями карт. Начнем с создания объекта DeckOfCards:

```
In [1]: from deck import DeckOfCards

In [2]: deck_of_cards = DeckOfCards()
```

Включение поддержки Matplotlib в IPython

Затем включим поддержку Matplotlib в IPython при помощи магической команды %matplotlib:

```
In [3]: %matplotlib
Using matplotlib backend: Qt5Agg
```

[1] https://creativecommons.org/publicdomain/zero/1.0/deed.en.

Создание базового пути для каждого изображения

Прежде чем выводить каждое изображение, необходимо загрузить его из каталога card_images. Мы используем *класс* `Path` модуля `pathlib` для построения полного пути к каждому изображению в нашей системе. Фрагмент [5] создает объект `Path` для текущего каталога (каталог `ch10` с примерами), который представлен обозначением `'.'`, а затем используем метод `joinpath` класса `Path` для присоединения подкаталога с изображениями карт:

```
In [4]: from pathlib import Path

In [5]: path = Path('.').joinpath('card_images')
```

Импортирование функциональности Matplotlib

Затем необходимо импортировать модули Matplotlib, необходимые для вывода изображений. Нам понадобится функция из `matplotlib.image` для загрузки изображений:

```
In [6]: import matplotlib.pyplot as plt

In [7]: import matplotlib.image as mpimg
```

Создание объектов Figure и Axes

Следующий фрагмент использует функцию `subplots` библиотеки Matplotlib для создания объекта `Figure`, на котором будут выводиться изображения в виде 52 *поддиаграмм*, объединенных в четыре строки (`nrows`) и 13 столбцов (`ncols`). Функция возвращает кортеж с объектом `Figure` и массивом объектов `Axes` для поддиаграмм. Кортеж распаковывается в переменные `figure` и `axes_list`:

```
In [8]: figure, axes_list = plt.subplots(nrows=4, ncols=13)
```

Если выполнить эту команду в IPython, немедленно появляется окно Matplotlib с 52 пустыми поддиаграммами.

Настройка объектов Axes и вывод изображений

Затем мы переберем все объекты `Axes` в списке `axes_list`. Напомним, `ravel` создает одномерное представление многомерного массива. Для каждого объекта `Axes` выполняются следующие операции:

- Первые две команды в цикле скрывают оси *x* и *y*: мы не собираемся выводить данные на диаграммах, поэтому оси и метки не понадобятся.

- Третья команда сдает карту и получает значение `image_name` объекта `Card`.
- Четвертая команда при помощи метода `joinpath` класса `Path` присоединяет `image_name` к пути `Path`, а затем вызывает метод `resolve` для определения полного пути к изображению в вашей системе. Полученный объект `Path` передается встроенной функции `str` для получения строкового представления местоположения файла. Эта строка передается функции `imread` модуля `matplotlib.image`, которая загружает изображение.
- Последняя команда вызывает метод `imshow` класса `Axes`, чтобы вывести текущее изображение на текущей поддиаграмме.

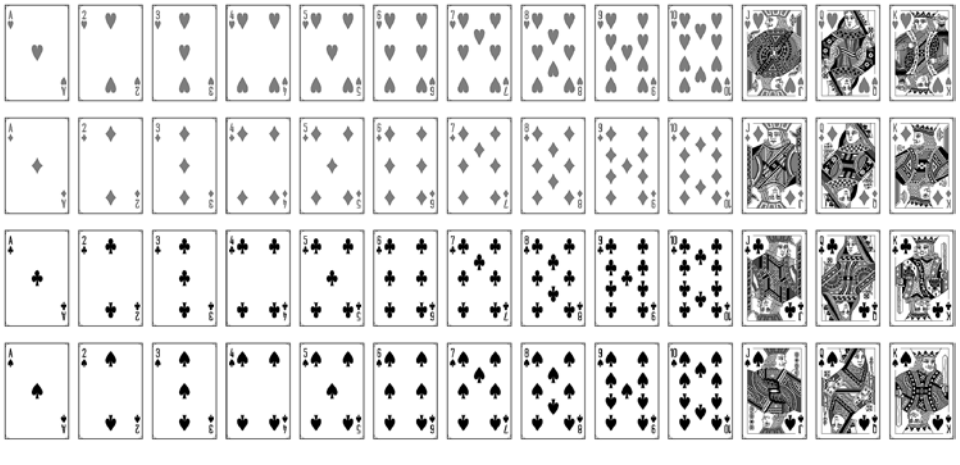
```
In [9]: for axes in axes_list.ravel():
   ...:     axes.get_xaxis().set_visible(False)
   ...:     axes.get_yaxis().set_visible(False)
   ...:     image_name = deck_of_cards.deal_card().image_name
   ...:     img = mpimg.imread(str(path.joinpath(image_name).resolve()))
   ...:     axes.imshow(img)
   ...:
```

Вывод изображений с максимальным размером

На этот момент все изображения выводятся успешно. Чтобы изображения карт занимали как можно большую площадь, вы можете развернуть окно, а затем вызвать *метод* `tight_layout` класса `Figure` из библиотеки Matplotlib. При этом из окна пропадает бóльшая часть промежутков:

```
In [10]: figure.tight_layout()
```

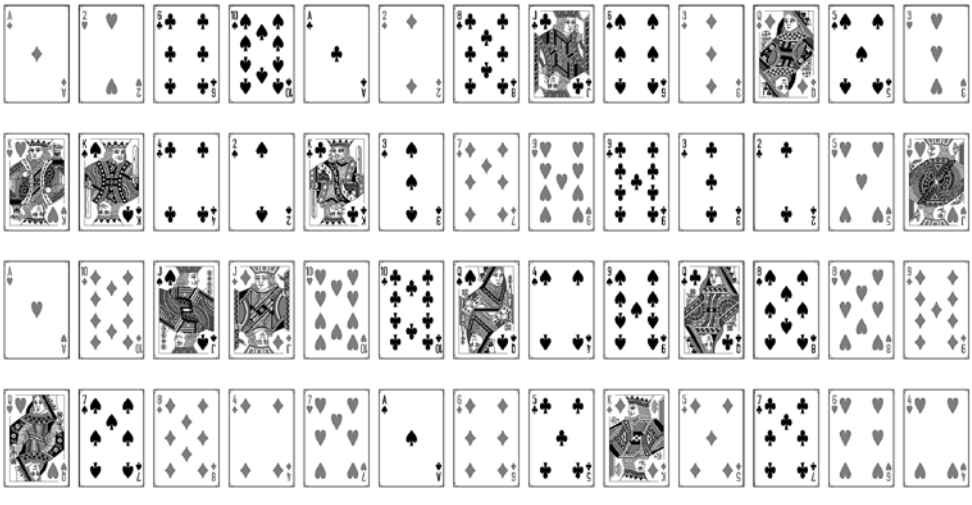
На следующем изображении показано содержимое полученного окна:



Тасование и повторная сдача карт

Чтобы увидеть перетасованные изображения, вызовите метод `shuffle`, а затем снова выполните код фрагмента [9]:

```
In [11]: deck_of_cards.shuffle()

In [12]: for axes in axes_list.ravel():
   ...:     axes.get_xaxis().set_visible(False)
   ...:     axes.get_yaxis().set_visible(False)
   ...:     image_name = deck_of_cards.deal_card().image_name
   ...:     img = mpimg.imread(str(path.joinpath(image_name).resolve()))
   ...:     axes.imshow(img)
   ...:
```



10.7. Наследование: базовые классы и подклассы

Часто объект одного класса одновременно *является* и объектом другого класса. Скажем, `CarLoan` (кредит на покупку машины) может рассматриваться как разновидность `Loan` (кредит) наряду с `HomeImprovementLoan` (кредит на обустройство жилья) или с `MortgageLoan` (ипотечный кредит). Можно сказать, что класс `CarLoan` наследует от класса `Loan`. В этом контексте класс `Loan` является базовым классом (или суперклассом), а класс `CarLoan` — подклассом. `CarLoan` *является* конкретной разновидностью `Loan`, но было бы неправильно утверждать, что каждый объект `Loan` *является* `CarLoan` — `Loan` может представ-

лять любой тип кредита. В табл. 10.1 перечислены простые примеры базовых классов и подклассов — базовые классы обычно являются более «общими», а подклассы — более «конкретными»:

Таблица 10.1. Примеры базовых классов и подклассов

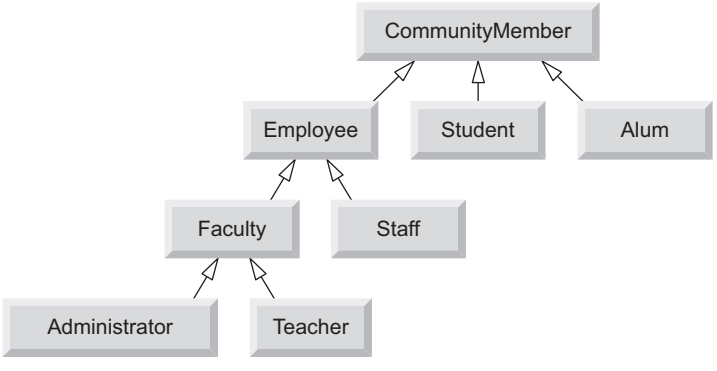
Базовый класс	Подклассы
`Student`	`GraduateStudent, UndergraduateStudent`
`Shape`	`Circle, Triangle, Rectangle, Sphere, Cube`
`Loan`	`CarLoan, HomeImprovementLoan, MortgageLoan`
`Employee`	`Faculty, Staff`
`BankAccount`	`CheckingAccount, SavingsAccount`

Так как каждый объект подкласса одновременно *является* объектом своего базового класса, а один базовый класс может иметь много подклассов, множество объектов, представляемых базовым классом, часто шире множества объектов, представляемых любым из его подклассов. Например, базовый класс `Vehicle` представляет *любые* виды транспортных средств: машины, грузовики, лодки, велосипеды и т. д. С другой стороны, подкласс `Car` представляет меньшее, более конкретное подмножество транспортных средств.

Иерархия наследования CommunityMember

Отношения наследования образуют древовидные *иерархические* структуры. Базовый класс находится в иерархических отношениях со своими подклассами. Разработаем простую иерархию классов (изображена на следующей диаграмме), которая также называется *иерархией наследования*. Участниками университетского сообщества (`CommunityMember`) могут быть тысячи человек, включая работников (`Employee`), студентов (`Student`) и выпускников (`Alum`). Работники (`Employee`) относятся либо к профессорско-преподавательскому составу (`Faculty`), либо к штатному персоналу (`Staff`). Профессорско-преподавательский состав включает администраторов (`Administrator`), например деканы, заведующие кафедрами и т. д., и преподавателей (`Teacher`). Иерархия может содержать множество других классов. Так, в категории студентов могут быть как студенты магистратуры, так и студенты бакалавриата, а студенты бакалавриата могут делиться на первокурсников, второкурсников и т. д. При *одиночном наследовании* класс может иметь только *один* непосредственный базовый класс. При *множественном наследовании* подкласс наследует от *двух*

и более базовых классов. Одиночное наследование реализуется достаточно прямолинейно, а множественное наследование выходит за рамки книги, поэтому предварительно поищите в интернете информацию о «проблеме ромбовидного наследования» при множественном наследовании в Python.

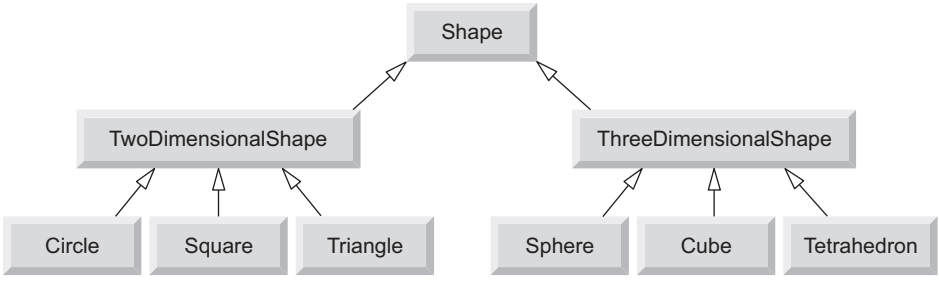


Каждая стрелка в иерархии представляет отношение типа «*является*» (точнее, «является частным случаем»). Например, переходя по стрелкам снизу вверх в этой иерархии классов, можно утверждать, что «`Employee` *является* `CommunityMember`», а «`Teacher` *является* `Faculty`». `CommunityMember` является непосредственным базовым классом `Employee`, `Student` и `Alum` и опосредованным базовым классом всех остальных классов на диаграмме. Начиная от низа диаграммы, вы можете переходить по стрелкам и применять отношения «является» вплоть до верхнего суперкласса. Например, `Administrator` *является* и `Faculty`, и `Employee`, и `CommunityMember`, и, конечно, в конечном итоге *является* объектом.

Иерархия наследования Shape

Рассмотрим иерархию наследования геометрических фигур, изображенную на следующей диаграмме классов. Она начинается с базового класса фигуры `Shape`, за которым следуют подклассы `TwoDimensionalShape` (двумерная фигура) и `ThreeDimensionalShape` (трехмерная фигура). Каждый объект `Shape` является `TwoDimensionalShape` или `ThreeDimensionalShape`. На третьем уровне иерархии находятся *конкретные* разновидности `TwoDimensionalShape` и `ThreeDimensionalShape`. И снова можно проследовать по стрелкам от нижней части диаграммы до верхнего базового класса в иерархии классов для выявления нескольких отношений типа «*является*». Например, `Triangle` (тре-

угольник) *является* и `TwoDimensionalShape`, и `Shape`, тогда как сфера (`Sphere`) *является* и `ThreeDimensionalShape`, и `Shape`. Эта иерархия может содержать много других классов. Например, эллипсы и параллелепипеды относятся к двумерным фигурам (`TwoDimensionalShape`), а конусы и цилиндры — к трехмерным фигурам (`ThreeDimensionalShape`).



Отношение «is a» и отношение «has a»

Наследование создает *отношения is-a*, в которых объект типа подкласса может также рассматриваться как объект типа базового класса. Вы также видели (составные) отношения *has-a*, в которых класс имеет ссылки на один или несколько объектов других классов в качестве членов.

10.8. Построение иерархии наследования. Концепция полиморфизма

Иерархия типов работников в программе начисления зарплаты поможет понять отношения между базовым классом и его подклассами. У всех работников компании есть много общего, но *внештатные работники* (которые будут представлены объектами базового класса) получают проценты от продаж, тогда как *штатные работники* (которые будут представлены объектами подкласса) получают и базовую зарплату, и процент от продаж.

Начнем с представления *базового класса* `CommissionEmployee`. Затем создадим *подкласс* `SalariedCommissionEmployee`, наследующий от класса `CommissionEmployee`. После этого создадим в сеансе IPython объект `SalariedCommissionEmployee` и продемонстрируем, что он обладает всей

функциональностью своего базового класса *и* подкласса, но вычисляет заработок по другим правилам.

10.8.1. Базовый класс CommissionEmployee

Рассмотрим класс `CommissionEmployee`, который содержит следующие компоненты:

- Метод `__init__` (строки 8–15), который создает атрибуты данных `_first_name`, `_last_name` и `_ssn` (номер социального страхования), и использует set-методы свойств `gross_sales` и `commission_rate` для создания соответствующих атрибутов данных.
- Доступные только для чтения свойства `first_name` (строки 17–19), `last_name` (строки 21–23) и `ssn` (строки 25–27), возвращающие соответствующие атрибуты данных.
- Доступные только для чтения свойства `gross_sales` (строки 29–39) и `commission_rate` (строки 41–52), у которых set-методы выполняют проверку данных.
- Метод `earnings` (строки 54–56), который вычисляет и возвращает заработок `CommissionEmployee`.
- Метод `__repr__` (строки 58–64), который возвращает строковое представление объекта `CommissionEmployee`.

```
 1 # commmissionemployee.py
 2 """Базовый класс CommissionEmployee."""
 3 from decimal import Decimal
 4
 5 class CommissionEmployee:
 6     """Сотрудник, получающий процент от продаж."""
 7
 8     def __init__(self, first_name, last_name, ssn,
 9                  gross_sales, commission_rate):
10         """Инициализирует атрибуты CommissionEmployee."""
11         self._first_name = first_name
12         self._last_name = last_name
13         self._ssn = ssn
14         self.gross_sales = gross_sales  # Проверка через свойство
15         self.commission_rate = commission_rate  # Проверка через свойство
16
17     @property
18     def first_name(self):
19         return self._first_name
```

```
20
21      @property
22      def last_name(self):
23          return self._last_name
24
25      @property
26      def ssn(self):
27          return self._ssn
28
29      @property
30      def gross_sales(self):
31          return self._gross_sales
32
33      @gross_sales.setter
34      def gross_sales(self, sales):
35          """Задает объем продаж или выдает ошибку ValueError."""
36          if sales < Decimal('0.00'):
37              raise ValueError('Gross sales must be >= to 0')
38
39          self._gross_sales = sales
40
41      @property
42      def commission_rate(self):
43          return self._commission_rate
44
45      @commission_rate.setter
46      def commission_rate(self, rate):
47          """Задает комиссионную ставку или выдает ошибку ValueError."""
48          if not (Decimal('0.0') < rate < Decimal('1.0')):
49              raise ValueError(
50                 'Interest rate must be greater than 0 and less than 1')
51
52          self._commission_rate = rate
53
54      def earnings(self):
55          """Вычисляет заработок."""
56          return self.gross_sales * self.commission_rate
57
58      def __repr__(self):
59          """Возвращает строковое представление для repr()."""
60          return ('CommissionEmployee: ' +
61              f'{self.first_name} {self.last_name}\n' +
62              f'social security number: {self.ssn}\n' +
63              f'gross sales: {self.gross_sales:.2f}\n' +
64              f'commission rate: {self.commission_rate:.2f}')
```

Свойства `first_name`, `last_name` и `ssn` доступны только для чтения. Мы не стали проверять их, хотя это можно было сделать. Например, можно проверить

имя и фамилию `first_name` и `last_name` — допустим, убедиться, что они имеют разумную длину. Или проверить номер социального страхования и убедиться в том, что он состоит из 9 цифр с дефисами или без (в формате `###-##-####` или `#########`, где каждый знак # соответствует одной цифре).

Все классы наследуют прямо или опосредованно от класса object

Наследование используется для создания новых классов на базе существующих. Собственно, *каждый* класс Python наследует от существующего класса. Если базовый класс не задается явно для нового класса, то Python считает, что класс наследует непосредственно от класса `object`. Иерархия классов Python начинается с класса `object`, непосредственного или опосредованного базового класса *любого* другого класса. Таким образом, заголовок класса `CommissionEmployee` можно было бы записать в виде

```
class CommissionEmployee(object):
```

Круглые скобки после `CommissionEmployee` обозначают наследование; в них может быть указан один класс для одиночного наследования или разделенный запятыми список базовых классов для множественного наследования. Как говорилось ранее, тема множественного наследования выходит за рамки книги.

Класс `CommissionEmployee` наследует все методы класса `object`. Класс `object` не содержит атрибутов данных. Среди многочисленных методов, наследуемых от `object`, можно выделить `__repr__` и `__str__`. Таким образом, *каждый* класс содержит эти методы, которые возвращают строковые представления тех объектов, для которых они вызываются. Если реализация метода базового класса не подходит для произвольного класса, этот метод может быть *переопределен* нужной реализацией в производном классе. Метод `__repr__` (строки 58–64) переопределяет реализацию по умолчанию, наследуемую классом `CommissionEmployee` от класса `object`.[1]

Тестирование класса CommissionEmployee

Протестируем некоторые возможности `CommissionEmployee`. Начнем с создания и вывода `CommissionEmployee`:

---

[1] Список переопределяемых методов object доступен по адресу https://docs.python.org/3/reference/datamodel.html.

```
In [1]: from commissionemployee import CommissionEmployee

In [2]: from decimal import Decimal

In [3]: c = CommissionEmployee('Sue', 'Jones', '333-33-3333',
   ...:     Decimal('10000.00'), Decimal('0.06'))
   ...:

In [4]: c
Out[4]:
CommissionEmployee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00
commission rate: 0.06
```

Затем вычислим и выведем заработок `CommissionEmployee`:

```
In [5]: print(f'{c.earnings():,.2f}')
600.00
```

Наконец, изменим объем продаж и комиссионную ставку `CommissionEmployee`, а затем снова вычислим заработок:

```
In [6]: c.gross_sales = Decimal('20000.00')

In [7]: c.commission_rate = Decimal('0.1')

In [8]: print(f'{c.earnings():,.2f}')
2,000.00
```

10.8.2. Подкласс SalariedCommissionEmployee

При одиночном наследовании подкласс начинает свое существование практически в том же виде, в котором находится базовый класс. Настоящая сила наследования происходит от возможности определять в подклассе добавления, замены или уточнения для средств, унаследованных от базового класса.

Многие возможности `SalariedCommissionEmployee` похожи, если не идентичны, возможностям класса `CommissionEmployee`. Оба типа работников содержат атрибуты данных для имени, фамилии, номера социального страхования, объема продаж, комиссионной ставки, а также свойств и методов для работы с этими данными. Чтобы создать класс `SalariedCommissionEmployee` *без* применения наследования, нам пришлось бы *скопировать* код класса `CommissionEmployee` и *вставить* его в класс `SalariedCommissionEmployee`. Затем новый класс пришлось бы модифицировать, включить в него атрибут данных для базовой

зарплаты, а также свойства и методы для работы с базовой зарплатой, включая новый метод `earnings`. *Решение с копированием/вставкой* кода часто подвержено высокому риску ошибок. Что еще хуже, оно может привести к появлению в системе многих физических копий одного кода (вместе с ошибками), что усложнит сопровождение вашего кода. Наследование позволяет «впитать» функциональность существующего класса *без* дублирования кода. Посмотрим, как это делается.

Объявление класса SalariedCommissionEmployee

Объявим подкласс `SalariedCommissionEmployee`, *наследующий* большую часть своей функциональности от класса `CommissionEmployee` (строка 6). Класс `SalariedCommissionEmployee` *является* `CommissionEmployee` (потому что наследование передает функциональность класса `CommissionEmployee`), но класс `SalariedCommissionEmployee` также обладает следующими дополнительными функциональными аспектами:

- Метод `__init__` (строки 10–15) инициализирует все данные, унаследованные от класса `CommissionEmployee` (вскоре мы расскажем об этом подробнее), а затем использует `set`-метод свойства `base_salary` для создания атрибута данных `base_salary`.
- Свойство `base_salary`, доступное только для чтения (строки 17–27), `set`-метод которого выполняет проверку данных.
- Видоизмененная версия метода `earnings` (строки 29–31).
- Видоизмененная версия метода `__repr__` (строки 33–36).

```
 1 # salariedcommissionemployee.py
 2 """Класс SalariedCommissionEmployee наследует от CommissionEmployee."""
 3 from commissionemployee import CommissionEmployee
 4 from decimal import Decimal
 5
 6 class SalariedCommissionEmployee(CommissionEmployee):
 7     """Работник, получающий зарплату и комиссионные,
 8     вычисляемые как процент от продаж."""
 9
10     def __init__(self, first_name, last_name, ssn,
11                  gross_sales, commission_rate, base_salary):
12         """Инициализирует атрибуты SalariedCommissionEmployee."""
13         super().__init__(first_name, last_name, ssn,
14                          gross_sales, commission_rate)
15         self.base_salary = base_salary  # validate via property
16
```

```
17      @property
18      def base_salary(self):
19          return self._base_salary
20
21      @base_salary.setter
22      def base_salary(self, salary):
23          """Задает базовую зарплату или выдает ошибку ValueError."""
24          if salary < Decimal('0.00'):
25              raise ValueError('Base salary must be >= to 0')
26
27          self._base_salary = salary
28
29      def earnings(self):
30          """Вычисляем доход."""
31          return super().earnings() + self.base_salary
32
33      def __repr__(self):
34          """Возвращает строковое представление для repr()."""
35          return ('Salaried' + super().__repr__() +
36              f'\nbase salary: {self.base_salary:.2f}')
```

Наследование от Class CommissionEmployee

Чтобы наследовать от класса, сначала необходимо импортировать его определение (строка 3). Строка 6

```
class SalariedCommissionEmployee(CommissionEmployee):
```

указывает, что класс `SalariedCommissionEmployee` *наследует* от `CommissionEmployee`. Хотя вы не видите атрибуты данных, свойства и методы класса `CommissionEmployee` в классе `SalariedCommissionEmployee`, они все равно являются частью нового класса.

Метод __init__ и встроенная функция super

Метод `__init__` *каждого подкласса должен явно вызвать метод* `__init__` *своего базового класса, чтобы инициализировать атрибуты данных, унаследованные от базового класса.* Этот вызов должен быть первой командой в методе `__init__` подкласса. Метод `__init__` класса `SalariedCommissionEmployee` явно вызывает метод `__init__` класса `CommissionEmployee` (строки 13–14) для инициализации части объекта `SalariedCommissionEmployee`, принадлежащей базовому классу (то есть пяти атрибутов данных, унаследованных от класса `CommissionEmployee`). Запись `super().__init__` использует встроенную функцию `super` для поиска и вызова метода `__init__` базового класса, передавая пять аргументов, которые инициализируют унаследованные атрибуты данных.

Переопределение метода earnings

Метод `earnings` класса `SalariedCommissionEmployee` (строки 29–31) переопределяет метод `earnings` класса `CommissionEmployee` (раздел 10.8.1, строки 54–56) для вычисления заработка `SalariedCommissionEmployee`. Новая версия получает часть заработка, *состоящую из комиссионных*, вызовом метода `earnings` класса `CommissionEmployee`; для этого используется выражение `super().earnings()` (строка 31). Затем метод `earnings` класса `SalariedCommissionEmployee` прибавляет `base_salary` к этому значению для вычисления суммарного заработка. Благодаря тому что метод `earnings` класса `SalariedCommissionEmployee` вызывает метод `earnings` класса `CommissionEmployee` для вычисления части заработка, относящейся к `SalariedCommissionEmployee`, мы избегаем дублирования кода и снижаем риск проблем, связанных с сопровождением кода.

Переопределение метода __repr__

Метод `__repr__` класса `SalariedCommissionEmployee` (строки 33–36) переопределяет метод `__repr__` класса `CommissionEmployee` (раздел 10.8.1, строки 58–64) для возвращения объекта `String`, соответствующего `SalariedCommissionEmployee`. Подкласс создает часть строкового представления, объединяя строку `'Salaried'` со строкой, возвращаемой `super().__repr__()`, для которой вызывается метод `__repr__` класса `CommissionEmployee`. Затем переопределенный метод присоединяет данные базовой зарплаты и возвращает полученную строку.

Тестирование класса SalariedCommissionEmployee

Протестируем класс `SalariedCommissionEmployee`, чтобы показать, что он действительно унаследовал функциональность от класса `CommissionEmployee`. Начнем с создания объекта `SalariedCommissionEmployee` и вывода всех его свойств:

```
In [9]: from salariedcommissionemployee import SalariedCommissionEmployee

In [10]: s = SalariedCommissionEmployee('Bob', 'Lewis', '444-44-4444',
    ...:         Decimal('5000.00'), Decimal('0.04'), Decimal('300.00'))
    ...:

In [11]: print(s.first_name, s.last_name, s.ssn, s.gross_sales,
    ...:       s.commission_rate, s.base_salary)
Bob Lewis 444-44-4444 5000.00 0.04 300.00
```

Обратите внимание: объект `SalariedCommissionEmployee` содержит *все* свойства классов `CommissionEmployee` *и* `SalariedCommissionEmployee`.

Теперь вычислим и выведем заработок `SalariedCommissionEmployee`. Так как метод `earnings` вызывается для объекта `SalariedCommissionEmployee`, будет выполнена *версия* метода *из подкласса*:

```
In [12]: print(f'{s.earnings():,.2f}')
500.00
```

Изменим свойства `gross_sales`, `commission_rate` и `base_salary`, а затем выведем обновленные данные при помощи метода `__repr__` класса `SalariedCommissionEmployee`:

```
In [13]: s.gross_sales = Decimal('10000.00')

In [14]: s.commission_rate = Decimal('0.05')

In [15]: s.base_salary = Decimal('1000.00')

In [16]: print(s)
SalariedCommissionEmployee: Bob Lewis
social security number: 444-44-4444
gross sales: 10000.00
commission rate: 0.05
base salary: 1000.00
```

И снова этот метод вызывается для объекта `SalariedCommissionEmployee`, поэтому выполняется *версия* метода *из подкласса*. Наконец, вычислим и выведем обновленный заработок для объекта `SalariedCommissionEmployee`:

```
In [17]: print(f'{s.earnings():,.2f}')
1,500.00
```

Проверка отношений «является»

Python предоставляет две встроенные функции — `issubclass` и `isinstance` — для проверки отношений типа «является». Функция `issubclass` определяет, является ли один класс производным от другого:

```
In [18]: issubclass(SalariedCommissionEmployee, CommissionEmployee)
Out[18]: True
```

Функция `isinstance` определяет, находится ли объект в отношении «является» с конкретным типом. Так как класс `SalariedCommissionEmployee` наследует от

CommissionEmployee, оба следующих фрагмента возвращают True, подтверждая существование отношения «является»:

```
In [19]: isinstance(s, CommissionEmployee)
Out[19]: True

In [20]: isinstance(s, SalariedCommissionEmployee)
Out[20]: True
```

10.8.3. Полиморфная обработка CommissionEmployee и SalariedCommissionEmployee

При наследовании каждый объект подкласса может интерпретироваться и как объект базового класса этого подкласса. Мы воспользуемся этим отношением «объект подкласса является объектом базового класса» для выполнения некоторых интересных операций. Например, можно поместить объекты, связанные с наследованием, в список, а затем перебрать содержимое списка и интерпретировать каждый элемент списка как объект базового класса. Это позволяет организовать *обобщенную* обработку различных объектов. Чтобы продемонстрировать этот факт, поместим объекты CommissionEmployee и SalariedCommissionEmployee в список, а затем для каждого элемента выведем его строковое представление и величину заработка:

```
In [21]: employees = [c, s]

In [22]: for employee in employees:
    ...:     print(employee)
    ...:     print(f'{employee.earnings():,.2f}\n')
    ...:
CommissionEmployee: Sue Jones
social security number: 333-33-3333
gross sales: 20000.00
commission rate: 0.10
2,000.00

SalariedCommissionEmployee: Bob Lewis
social security number: 444-44-4444
gross sales: 10000.00
commission rate: 0.05
base salary: 1000.00
1,500.00
```

Как видите, для каждого работника выводится правильное строковое представление и величина заработка. Этот механизм, называемый *полиморфизмом*,

является одним из ключевых аспектов объектно-ориентированного программирования (ООП).

10.8.4. Объектно-базированное и объектно-ориентированное программирование

Наследование с переопределением методов — мощный механизм построения программных компонентов, *похожих* на существующие компоненты, но адаптированных для специфических потребностей вашего приложения. В мире разработки с открытым кодом Python существует огромное количество качественно разработанных библиотек классов, для использования которых вы должны:

- знать доступные библиотеки;
- знать доступные классы;
- создавать объекты существующих классов;
- отправлять им сообщения (то есть вызывать их методы).

Такой стиль программирования называется *объектно-базированным программированием* (*ОБП*). Выполняя композицию объектов известных классов, вы используете объектно-базированное программирование. Добавление наследования с переопределением методов под уникальные потребности вашего приложения и, возможно, с полиморфной обработкой объектов называется *объектно-ориентированным программированием* (*ООП*). Если вы реализуете композицию с объектами унаследованных классов, то это также относится к объектно-ориентированному программированию.

10.9. Утиная типизация и полиморфизм

Многим другим языкам объектно-ориентированного программирования для реализации полиморфного поведения требуются отношения типа «является», основанные на наследовании. Python обладает большей гибкостью. В нем используется концепция *утиной типизации*, которая в документации Python описывается следующим образом:

> *Стиль программирования, который не проверяет тип объекта для определения того, обладает ли этот объект нужным интерфейсом; вместо этого*

просто вызывается или используется метод или атрибут («Если что-то выглядит как утка и крякает как утка, то это и есть утка»[1]*).*

Таким образом, при обработке объекта во время выполнения его тип роли не играет. Если объект содержит атрибут данных, свойство или метод (с подходящими параметрами), к которому вы хотите обратиться, то ваш код будет работать.

Вернемся к циклу в конце раздела 10.8.3, в котором обрабатывается список работников:

```
for employee in employees:
    print(employee)
    print(f'{employee.earnings():,.2f}\n')
```

В Python этот цикл будет правильно работать при условии, что `employees` содержит только объекты, которые:

- могут быть выведены вызовом `print` (то есть имеют строковое представление);
- содержат метод `earnings`, который может вызываться без аргументов.

Все классы непосредственно или опосредованно наследуют от `object`, поэтому *все* они наследуют методы по умолчанию для получения строковых представлений, которые могут выводиться при вызове `print`. Если класс содержит метод `earnings`, который может вызываться без аргументов, то вы можете включить объекты этого класса в список `employees`, даже если класс объекта не находится в отношении «является» с классом `CommissionEmployee`. Для демонстрации этого факта рассмотрим класс `WellPaidDuck`:

```
In [1]: class WellPaidDuck:
   ...:     def __repr__(self):
   ...:         return 'I am a well-paid duck'
   ...:     def earnings(self):
   ...:         return Decimal('1_000_000.00')
   ...:
```

Объекты `WellPaidDuck`, которые, очевидно, не должны представлять работников, будут работать с приведенным циклом. Чтобы доказать этот факт, создадим объекты классов `CommissionEmployee`, `SalariedCommissionEmployee` и `WellPaidDuck`, а затем поместим их в список:

---

[1] https://docs.python.org/3/glossary.html#term-duck-typing.

```
In [2]: from decimal import Decimal

In [3]: from commissionemployee import CommissionEmployee

In [4]: from salariedcommissionemployee import SalariedCommissionEmployee

In [5]: c = CommissionEmployee('Sue', 'Jones', '333-33-3333',
   ...:                        Decimal('10000.00'), Decimal('0.06'))
   ...:

In [6]: s = SalariedCommissionEmployee('Bob', 'Lewis', '444-44-4444',
   ...:     Decimal('5000.00'), Decimal('0.04'), Decimal('300.00'))
   ...:

In [7]: d = WellPaidDuck()

In [8]: employees = [c, s, d]
```

Теперь обработаем список с использованием цикла из раздела 10.8.3. Как видно из вывода, Python сможет использовать утиную типизацию для *полиморфной* обработки всех трех объектов в списке:

```
In [9]: for employee in employees:
   ...:     print(employee)
   ...:     print(f'{employee.earnings():,.2f}\n')
   ...:
CommissionEmployee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00
commission rate: 0.06
600.00

SalariedCommissionEmployee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00
500.00

I am a well-paid duck
1,000,000.00
```

10.10. Перегрузка операторов

Вы уже видели, как взаимодействовать с объектами в программе: нужно обращаться к их атрибутам и свойствам и вызывать их методы. Синтаксис вызова методов может быть слишком громоздким для некоторых операций, напри-

мер арифметических. В таких ситуациях удобнее воспользоваться богатым набором встроенных операторов Python.

В этом разделе показано, как использовать *перегрузку операторов* для определения того, как операторы Python должны работать с объектами ваших типов. Вы уже часто использовали перегрузку операторов для разных типов:

- оператор `+` для суммирования числовых значений, конкатенации списков, конкатенации строк и прибавления значения к каждому элементу массива NumPy;
- оператор `[]` для обращения к элементам списков, кортежей, строк и массивов, а также для обращения к значению, связанному с конкретным ключом в словаре;
- оператор `*` для умножения числовых значений, повторения последовательностей и умножения каждого элемента в массивах NumPy на конкретное значение.

Большую часть операторов можно перегружать. Для каждого перегружаемого оператора класс `object` определяет специальный метод: например, `__add__` для оператора сложения (`+`) или `__mul__` для оператора умножения (`*`). Переопределение этих методов позволит вам определить, как этот оператор должен работать с объектами вашего класса. За полным списком специальных методов обращайтесь по адресу:

https://docs.python.org/3/reference/datamodel.html#special-method-names

Ограничения перегрузки операторов

Для перегрузки операторов установлены следующие ограничения:

- Приоритет операторов не может изменяться при перегрузке. Тем не менее круглые скобки позволяют принудительно установить нужный порядок вычисления выражения.
- Группировка операторов (слева направо или справа налево) не может изменяться при перегрузке.
- «Арность» оператора (признак, указывающий на то, является оператор унарным или бинарным) изменяться не может.
- Создавать новые операторы нельзя — возможна только перегрузка существующих операторов.

- Смысл операции, выполняемой оператором с объектами встроенных типов, изменяться не может. Например, вы не сможете изменить оператор `+` так, чтобы он вычитал одно целое число из другого.
- Перегрузка операторов работает только с объектами пользовательских классов или в комбинациях объекта пользовательского класса и объекта встроенного типа.

Комплексные числа

Для демонстрации перегрузки операторов мы определим класс `Complex`, представляющий комплексное число[1]. Комплексные числа (например, $-3 + 4i$ или $6{,}2 - 11{,}73i$) записываются в форме

$$действительнаяЧасть\ +\ мнимаяЧасть\ *\ i,$$

где i — квадратный корень из -1. Комплексные числа, как и `int`, `float` и `Decimal`, являются арифметическими типами. В этом разделе мы создадим класс `Complex`, который перегружает только оператор сложения `+` и расширенное присваивание `+=`, поэтому объекты `Complex` можно складывать с использованием синтаксиса математических операций Python.

10.10.1. Класс Complex в действии

Для начала воспользуемся классом `Complex` для демонстрации его возможностей. Класс будет подробно рассмотрен в следующем разделе. Импортируйте класс `Complex` из файла complexnumber.py:

```
In [1]: from complexnumber import Complex
```

Затем создадим и выведем пару объектов `Complex`. Фрагменты [3] и [5] неявно вызывают метод `__repr__` класса `Complex` для получения строкового представления каждого объекта:

```
In [2]: x = Complex(real=2, imaginary=4)

In [3]: x
Out[3]: (2 + 4i)
```

[1] В Python существует встроенная поддержка комплексных чисел, так что класс создается исключительно в демонстрационных целях.

```
In [4]: y = Complex(real=5, imaginary=-1)

In [5]: y
Out[5]: (5 - 1i)
```

Мы выбрали формат строки `__repr__`, показанный в фрагментах [3] и [5], для имитации строк `__repr__`, генерируемых встроенным типом Python `complex`[1].

Теперь воспользуемся оператором `+` для суммирования объектов `x` и `y` класса `Complex`. Выражение суммирует действительные части двух операндов (`2` и `5`) и мнимые части двух операндов (`4i` и `-1i`), после чего возвращает новый объект `Complex` с результатом:

```
In [6]: x + y
Out[6]: (7 + 3i)
```

Оператор `+` не изменяет ни один из своих операндов:

```
In [7]: x
Out[7]: (2 + 4i)

In [8]: y
Out[8]: (5 - 1i)
```

Наконец, воспользуемся оператором `+=` для прибавления `y` к `x` и сохранения результата в `x`. Оператор `+=` *изменяет* свой левый операнд, но не изменяет правый:

```
In [9]: x += y

In [10]: x
Out[10]: (7 + 3i)

In [11]: y
Out[11]: (5 - 1i)
```

10.10.2. Определение класса Complex

Итак, вы увидели класс `Complex` в действии. Теперь обратимся к его определению и посмотрим, как были реализованы эти возможности.

---

[1] Python использует `j` вместо `i` для обозначения $\sqrt{-1}$. Например, `3+4j` (без пробелов вокруг оператора) создает объект complex с атрибутами `real` и `imag`. Строка `__repr__` для этого значения complex имеет вид `'(3+4j)'`.

Метод __init__

Метод `__init__` получает параметры для инициализации атрибутов данных, представляющих `действительную` и `мнимую` часть:

```
 1 # complexnumber.py
 2 """Класс Complex с перегруженными операторами."""
 3
 4 class Complex:
 5     """Класс Complex, представляющий комплексное число
 6     с действительной и мнимой частью."""
 7
 8     def __init__(self, real, imaginary):
 9         """Инициализирует атрибуты класса Complex."""
10         self.real = real
11         self.imaginary = imaginary
12
```

Перегруженный оператор +

Следующий перегруженный специальный метод `__add__` определяет, как оператор `+` должен работать с двумя объектами `Complex`:

```
13     def __add__(self, right):
14         """Переопределяет оператор +."""
15         return Complex(self.real + right.real,
16                        self.imaginary + right.imaginary)
17
```

Методы, перегружающие бинарные операторы, должны предоставить два параметра: *первый* (`self`) предназначен для *левого*, а *второй* (`right`) — для *правого* операнда. Метод `__add__` класса `Complex` получает два объекта `Complex` в аргументах и возвращает новый объект `Complex` с суммами `действительных` и `мнимых` частей операндов.

Содержимое исходных операндов при этом *не* изменяется, что соответствует нашим интуитивным представлениям о том, как должен вести себя этот оператор. Суммирование двух чисел не изменяет ни одно из исходных слагаемых.

Перегруженное расширенное присваивание +=

Строки 18–22 перегружают специальный метод `__iadd__` для определения того, как оператор `+=` должен суммировать два объекта `Complex`:

```
18     def __iadd__(self, right):
19         """Перегружает оператор +=."""
20         self.real += right.real
21         self.imaginary += right.imaginary
22         return self
23
```

Расширенное присваивание изменяет свои левые операнды, поэтому метод `__iadd__` изменяет объект `self`, соответствующий левому операнду, после чего возвращает `self`.

Метод `__repr__`

Строки 24–28 возвращают строковое представление объекта `Complex`.

```
24     def __repr__(self):
25         """Возвращает строковое представление для repr()."""
26         return (f'({self.real} ' +
27                 ('+' if self.imaginary >= 0 else '-') +
28                 f' {abs(self.imaginary)}i)')
```

10.11. Иерархия классов исключений и пользовательские исключения

В предыдущей главе был описан механизм обработки исключений. Каждое исключение представляет собой объект класса, входящего в иерархию классов исключений Python[1], или объект класса, производного от одного из этих классов. Классы исключений наследуют — непосредственно или опосредованно — от базового класса `BaseException` и определяются в модуле `exceptions`.

В Python определены четыре основных подкласса `BaseException` — `SystemExit`, `KeyboardInterrupt`, `GeneratorExit` и `Exception`:

- Исключения `SystemExit` завершают выполнение программы (или завершают интерактивный сеанс); если исключение остается неперехваченным, для него не выводится трассировка, как для других типов исключений.

[1] https://docs.python.org/3/library/exceptions.html.

- Исключения `KeyboardInterrupt` происходят при вводе пользователем команды прерывания — Ctrl + C (или control + C) в большинстве систем.
- Исключения `GeneratorExit` происходят при закрытии генератора — как правило, когда генератор завершает производство значений или его метод `close` вызывается явно.
- `Exception` — базовый класс для всех основных исключений, с которыми вы столкнетесь. Вы уже видели исключения различных подклассов `Exception`: `ZeroDivisionError`, `NameError`, `ValueError`, `StatisticsError`, `TypeError`, `IndexError`, `KeyError`, `RuntimeError` и `AttributeError`. Часто исключения `StandardError` можно перехватить и обработать, чтобы программа продолжила выполнение.

Перехват исключений базовых классов

Одно из преимуществ иерархии классов исключений заключается в том, что обработка исключений может перехватывать исключения конкретного типа или использовать тип базового класса для перехвата исключений базового класса и исключений всех его подклассов. Например, обработчик исключений базового класса `Exception` может перехватывать объекты *любого* подкласса `Exception`. Размещение обработчика, перехватывающего тип `Exception`, до других обработчиков `except` является логической ошибкой, потому что все исключения будут перехвачены еще до достижения других обработчиков. Таким образом, все последующие обработчики будут недостижимы.

Пользовательские классы исключений

Когда вы выдаете исключение в своем коде, обычно рекомендуется использовать один из существующих классов исключений из стандартной библиотеки Python. Тем не менее при помощи средств наследования, представленных ранее в этой главе, вы сможете создавать собственные классы исключений, наследующие (непосредственно или опосредованно) от класса `Exception`. В общем случае так поступать не рекомендуется, особенно начинающим программистам. Прежде чем создавать пользовательские классы исключений, поищите существующие классы исключений в иерархии исключений Python. Новые классы исключений должны определяться только в случае, если правила перехвата и обработки этих исключений отличаются от других существующих типов исключений (что встречается достаточно редко).

10.12. Именованные кортежи

Мы использовали кортежи для сведения нескольких атрибутов данных в один объект. *Модуль* `collections` стандартной библиотеки Python также предоставляет *именованные кортежи*, позволяющие ссылаться на компоненты кортежа по именам (вместо индексов).

Создадим простой именованный кортеж, который может использоваться для представления карты в колоде. Начнем с импортирования функции `namedtuple`:

```
In [1]: from collections import namedtuple
```

Функция `namedtuple` создает подкласс встроенного типа кортежа. В первом аргументе функции передается имя нового типа, а во втором — список строк, представляющих идентификаторы, которые будут использоваться для обращения к компонентам нового типа:

```
In [2]: Card = namedtuple('Card', ['face', 'suit'])
```

У вас появился новый тип кортежа с именем `Card`, который может использоваться в любом месте, где может использоваться кортеж. Создадим объект `Card`, обратимся к его компонентам и выведем его строковое представление:

```
In [3]: card = Card(face='Ace', suit='Spades')

In [4]: card.face
Out[4]: 'Ace'

In [5]: card.suit
Out[5]: 'Spades'

In [6]: card
Out[6]: Card(face='Ace', suit='Spades')
```

Другие возможности именованных кортежей

Каждый тип именованного кортежа содержит дополнительные методы. *Метод класса* `_make` этого типа (то есть метод, вызываемый для *класса*) получает итерируемый объект значений и возвращает объект с типом именованного кортежа:

```
In [7]: values = ['Queen', 'Hearts']

In [8]: card = Card._make(values)

In [9]: card
Out[9]: Card(face='Queen', suit='Hearts')
```

Например, это может быть полезно при работе с типом именованного кортежа, представляющим записи в CSV-файле. При чтении и разборе на лексемы записей CSV их можно преобразовать в объекты именованных кортежей.

Для заданного объекта с типом именованного кортежа можно получить представление имен и значений объекта в виде словаря `OrderedDict`. `OrderedDict` сохраняет порядок вставки пар «ключ-значение» в словарь:

```
In [10]: card._asdict()
Out[10]: OrderedDict([('face', 'Queen'), ('suit', 'Hearts')])
```

За дополнительной информацией о возможностях именованных кортежей обращайтесь по адресу:

https://docs.python.org/3/library/collections.html#collections.namedtuple

10.13. Краткое введение в новые классы данных Python 3.7

Хотя именованные кортежи позволяют ссылаться на свои компоненты по имени, они остаются кортежами, а не классами. Если вы хотите пользоваться преимуществами именованных кортежей, а также возможностями, предоставляемыми традиционными классами Python, то имеет смысл воспользоваться новыми *классами данных*[1] Python 3.7 из *модуля* `dataclasses` стандартной библиотеки Python.

Классы данных входят в число важнейших нововведений Python 3.7. Они помогают *быстрее* строить классы с использованием более *компактной* записи и с *автоматическим генерированием* «шаблонного» кода, общего для большинства классов. Они могут стать предпочтительным вариантом определения большинства классов Python. В этом разделе будут представлены основные принципы классов данных. В конце раздела будут приведены ссылки на дополнительную информацию.

Классы данных и автоматическое генерирование кода

Большинство классов, которые вы определяете, предоставляют метод `__init__` для создания и инициализации атрибутов объекта, а также метод `__repr__` для

[1] https://www.python.org/dev/peps/pep-0557/.

определения нестандартного строкового представления объекта. Если класс содержит много атрибутов данных, создание этих атрибутов может стать довольно утомительным и однообразным.

Классы данных *автоматически генерируют* атрибуты данных, а также методы `__init__` и `__repr__` за вас. Это может быть особенно полезно для классов, основная задача которых — агрегирование взаимосвязанных элементов данных. Так, в приложении, обрабатывающем записи в формате CSV, может присутствовать класс, представляющий поля каждой записи в виде атрибутов данных объекта. Классы данных также могут генерироваться *динамически* по списку имен полей.

Классы данных также автоматически генерируют метод `__eq__`, который перегружает оператор `==`. Любой класс, содержащий метод `__eq__`, также неявно поддерживает `!=`. *Все* классы наследуют реализацию по умолчанию метода `__ne__` (не равно), который возвращает величину, обратную `__eq__` (или `NotImplemented`, если класс не определяет `__eq__`). Классы данных *не* генерируют методы для операторов сравнения `<`, `<=`, `>` и `>=` автоматически, но такая возможность существует.

10.13.1. Создание класса данных Card

Давайте снова реализуем класс `Card` из раздела 10.6.2 в виде класса данных. Новый класс определяется в файле carddataclass.py. Определение класса данных потребует нового синтаксиса. В последующих разделах мы используем новый класс данных `Card` в классе `DeckOfCards`, чтобы показать, что он взаимозаменяем с исходным классом `Card`, а затем обсудим некоторые преимущества классов данных перед именованными кортежами и традиционными классами Python.

Импортирование из модулей dataclasses и typing

Модуль `dataclasses` стандартной библиотеки Python определяет декораторы и функции для реализации классов данных. Мы используем *декоратор* `@dataclass` (импортируемый в строке 4), чтобы указать, что новый класс является классом данных, и автоматически сгенерировать различный код. Вспомним, что исходный класс `Card` определял *переменные класса* `FACES` и `SUITS`, в которых хранились списки строк для инициализации `Card`. Мы используем `ClassVar` и `List` из *модуля* `typing` стандартной библиотеки Python (импортируется в строке 5) для обозначения того, что `FACES` и `SUITS` являются

переменными класса, которые содержат ссылки на *списки*. Вскоре мы поговорим о них более подробно:

```
1 # carddataclass.py
2 """Класс данных Card с атрибутами класса, атрибутами данных,
3 автоматически сгенерированными и явно определенными методами."""
4 from dataclasses import dataclass
5 from typing import ClassVar, List
6
```

Использование декоратора @dataclass

Чтобы указать, что класс является *классом данных*, поставьте перед его определением декоратор `@dataclass`[1]:

```
7 @dataclass
8 class Card:
```

Также декоратор `@dataclass` может содержать круглые скобки с аргументами, которые помогают классу данных определить, какие именно методы должны быть сгенерированы автоматически. Например, декоратор `@dataclass(order=True)` заставит класс данных автоматически сгенерировать методы для перегруженных операторов сравнения `<`, `<=`, `>` и `>=`. Это может пригодиться, если вы собираетесь сортировать свои объекты класса данных.

Аннотации переменных: атрибуты класса

В отличие от обычных классов, в классах данных как атрибуты класса, так и атрибуты данных объявляются *внутри* класса, но *вне* методов класса. В обычном классе только *атрибуты класса* объявляются таким способом, а атрибуты данных обычно создаются в `__init__`. Классам данных необходима дополнительная информация, или *рекомендации* (hints), с тем чтобы они могли отличить атрибуты класса от атрибутов данных, что также влияет на подробности реализации автоматически генерируемых методов.

Строки 9–11 определяют и инициализируют *атрибуты класса* `FACES` и `SUITS`:

```
 9     FACES: ClassVar[List[str]] = ['Ace', '2', '3', '4', '5', '6', '7',
10                                   '8', '9', '10', 'Jack', 'Queen', 'King']
11     SUITS: ClassVar[List[str]] = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
12
```

[1] https://docs.python.org/3/library/dataclasses.html#module-level-decorators-classes-and-functions.

Используемый в строках 9 и 11 синтаксис

```
: ClassVar[List[str]]
```

является *аннотацией переменной*[12] (иногда называемой *рекомендацией типа*), которая указывает, что `FACES` является атрибутом класса (`ClassVar`), содержащим ссылку на *список* строк (`List[str]`). `SUITS` также является атрибутом класса, который содержит ссылку на список строк.

Переменные класса инициализируются в своих определениях и относятся к *классу*, а не к отдельным *объектам* класса. При этом методы `__init__`, `__repr__` и `__eq__` предназначены для использования с объектами класса. Когда класс данных генерирует эти методы, он анализирует все аннотации переменных и включает в реализацию метода только *атрибуты данных*.

Аннотации переменных: атрибуты данных

Обычно атрибуты данных создаются в методе `__init__` класса (или методах, вызываемых `__init__`) присваиваниями в форме `self.`*имя_атрибута* `=` *значение*. Так как класс данных *автоматически генерирует* свой метод `__init__`, необходим другой способ задания атрибутов данных в определении класса данных. Вы не можете просто разместить их имена внутри класса, потому что при этом происходит ошибка `NameError`:

```
In [1]: from dataclasses import dataclass

In [2]: @dataclass
   ...: class Demo:
   ...:     x  # Попытка создания атрибута данных x
   ...:
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-79ffe37b1ba2> in <module>()
----> 1 @dataclass
      2 class Demo:
      3     x  # Попытка создания атрибута данных x
      4
```

[1] https://www.python.org/dev/peps/pep-0526/.

[2] Аннотации переменных — относительно новая языковая возможность, необязательная для обычных классов. В наследном коде Python они практически не встречаются.

```
<ipython-input-2-79ffe37b1ba2> in Demo()
      1 @dataclass
      2 class Demo:
----> 3     x  # Попытка создания атрибута данных x
      4

NameError: name 'x' is not defined
```

Каждый атрибут данных, как и атрибут класса, должен быть объявлен с аннотацией переменной. Строки 13–14 определяют атрибуты данных `face` и `suit`. Аннотация переменной `": str"` означает, что каждый атрибут содержит ссылку на объект строки:

```
13     face: str
14     suit: str
```

Определение свойств и других методов

Классы данных являются классами, то есть они могут содержать свойства и методы и участвовать в иерархиях классов. Для этого класса данных `Card` мы определяем то же свойство `image_name`, доступное только для чтения, и специальные методы `__str__` и `__format__`, как в исходном классе `Card` ранее в этой главе:

```
15     @property
16     def image_name(self):
17         """Возвращает имя файла с изображением карты."""
18         return str(self).replace(' ', '_') + '.png'
19
20     def __str__(self):
21         """Возвращает строковое представление для str()."""
22         return f'{self.face} of {self.suit}'
23
24     def __format__(self, format):
25         """Возвращает отформатированное строковое представление."""
26         return f'{str(self):{format}}'
```

Примечания по поводу аннотаций переменных

Аннотации переменных могут задаваться с использованием имен встроенных типов (например, `str`, `int` и `float`), типов классов или типов, определенных в модуле `typing` (вроле приведенных выше `ClassVar` и `List`). Даже с аннотациями типов Python является *языком с динамической типизацией*. Таким об-

разом, аннотации типов *не* проверяются во время выполнения. Следовательно, хотя атрибут `face` класса `Card` должен быть строкой, `face` можно присвоить объект любого типа.

10.13.2. Использование класса данных Card

Продемонстрируем работу с новым классом данных `Card`. Начнем с создания объекта `Card`:

```
In [1]: from carddataclass import Card

In [2]: c1 = Card(Card.FACES[0], Card.SUITS[3])
```

Затем используем автоматически сгенерированный метод `__repr__` класса `Card` для вывода объекта `Card`:

```
In [3]: c1
Out[3]: Card(face='Ace', suit='Spades')
```

Наш собственный метод `__str__`, который вызывается `print` при передаче объекта `Card`, возвращает строку в форме '*номинал* `of` *масть*':

```
In [4]: print(c1)
Ace of Spades
```

Обратимся к атрибутам класса данных и свойству, доступному только для чтения:

```
In [5]: c1.face
Out[5]: 'Ace'

In [6]: c1.suit
Out[6]: 'Spades'

In [7]: c1.image_name
Out[7]: 'Ace_of_Spades.png'
```

Затем продемонстрируем, что объекты `Card` можно сравнивать *автоматически сгенерированным* оператором `==` и унаследованным оператором `!=`. Создадим два дополнительных объекта `Card` — идентичный первому и отличный от него:

```
In [8]: c2 = Card(Card.FACES[0], Card.SUITS[3])

In [9]: c2
```

```
Out[9]: Card(face='Ace', suit='Spades')

In [10]: c3 = Card(Card.FACES[0], Card.SUITS[0])

In [11]: c3
Out[11]: Card(face='Ace', suit='Hearts')
```

Сравним объекты с использованием `==` и `!=`:

```
In [12]: c1 == c2
Out[12]: True

In [13]: c1 == c3
Out[13]: False

In [14]: c1 != c3
Out[14]: True
```

Наш класс данных `Card` взаимозаменяем с классом `Card`, разработанным ранее в этой главе. Чтобы продемонстрировать этот факт, мы создали файл `deck2.py` с копией класса `DeckOfCards`, описанного ранее в этой главе, и импортировали в него класс данных `Card`. Следующие фрагменты импортируют класс `DeckOfCards`, создают объект класса и выводят его. Вспомним, что `print` неявно вызывает метод `__str__` класса `DeckOfCards`, который форматирует каждый объект `Card` в поле шириной 19 символов, что приводит к вызову метода `__format__` класса `Card`. Прочитайте каждую строку вывода слева направо и убедитесь, что все объекты `Card` выводятся по порядку внутри каждой масти (`Hearts`, `Diamonds`, `Clubs` и `Spades`):

```
In [15]: from deck2 import DeckOfCards  # Используем класс данных Card

In [16]: deck_of_cards = DeckOfCards()

In [17]: print(deck_of_cards)
Ace of Hearts      2 of Hearts        3 of Hearts        4 of Hearts
5 of Hearts        6 of Hearts        7 of Hearts        8 of Hearts
9 of Hearts        10 of Hearts       Jack of Hearts     Queen of Hearts
King of Hearts     Ace of Diamonds    2 of Diamonds      3 of Diamonds
4 of Diamonds      5 of Diamonds      6 of Diamonds      7 of Diamonds
8 of Diamonds      9 of Diamonds      10 of Diamonds     Jack of Diamonds
Queen of Diamonds  King of Diamonds   Ace of Clubs       2 of Clubs
3 of Clubs         4 of Clubs         5 of Clubs         6 of Clubs
7 of Clubs         8 of Clubs         9 of Clubs         10 of Clubs
Jack of Clubs      Queen of Clubs     King of Clubs      Ace of Spades
2 of Spades        3 of Spades        4 of Spades        5 of Spades
6 of Spades        7 of Spades        8 of Spades        9 of Spades
10 of Spades       Jack of Spades     Queen of Spades    King of Spades
```

10.13.3. Преимущества классов данных перед именованными кортежами

Классы данных обладают рядом преимуществ перед именованными кортежами[1]:

- Хотя каждый именованный кортеж с технической точки зрения представляет отдельный тип, именованный кортеж *является* кортежем, а *все* кортежи могут сравниваться друг с другом. Таким образом, объекты *разных* типов именованных кортежей могут при сравнении считаться равными, если они содержат одинаковое количество элементов и эти элементы имеют одинаковые значения. Сравнение объектов разных классов данных *всегда* возвращает `False`, как и сравнение объекта класса данных с объектом кортежа.
- Если у вас имеется код, который распаковывает кортеж, то добавление новых элементов в кортеж нарушает работоспособность кода распаковки. Объекты класса данных распаковываться не могут. Таким образом, в класс данных можно добавить новые атрибуты данных без нарушения работоспособности существующего кода.
- Класс данных может быть базовым классом или подклассом в иерархии наследования.

10.13.4. Преимущества класса данных перед традиционными классами

Классы данных также обладают различными преимуществами перед традиционными классами Python, которые были представлены ранее в этой главе:

- Класс данных автоматически генерирует методы `__init__`, `__repr__` и `__eq__`, экономя ваше время.
- Класс данных может автоматически сгенерировать специальные методы, перегружающие операторы сравнения `<`, `<=`, `>` и `>=`.
- Если вы измените атрибуты данных, определенные в классе данных, а затем используете их в сценарии или интерактивном сеансе, то сгенери-

[1] https://www.python.org/dev/peps/pep-0526/.

рованный код обновится автоматически. Таким образом, вам останется меньше работы по отладке и сопровождению.

- Необходимые аннотации переменных для атрибутов классов и атрибутов данных позволяют применять средства статического анализа кода. Возможно, это позволит устранить большее количество ошибок до того, как они произойдут на стадии выполнения.
- Некоторые средства статического анализа кода и IDE могут анализировать аннотации переменных и выдавать предупреждения при использовании ошибочного типа в коде. Это поможет обнаружить логические ошибки в коде *до того,* как он будет выполнен.

Дополнительная информация

Классы данных могут обладать дополнительными возможностями, например возможностью создания «фиксированных» экземпляров, которые не позволяют присвоить значения атрибутов объекта класса данных после его создания. За полным списком преимуществ и возможностей классов данных обращайтесь по адресу:

https://www.python.org/dev/peps/pep-0557/

или

https://docs.python.org/3/library/dataclasses.html

10.14. Модульное тестирование с doc-строками и doctest

Одним из ключевых аспектов разработки программного обеспечения является тестирование вашего кода, проверяющее правильность его работы. Впрочем, даже при самом тщательном тестировании ваш код все равно может содержать ошибки. По словам знаменитого голландского ученого Эдсгера Дейкстры (Edsger Dijkstra), «тестирование выявляет наличие ошибок, но не их отсутствие»[1].

[1] J. N. Buxton and B. Randell, eds, *Software Engineering Techniques*, апрель 1970 г., с. 16. Отчет на конференции, проведенной научным комитетом НАТО, Рим, Италия, 27–31 октября 1969 г.

Модуль doctest и функция testmod

Стандартная библиотека Python содержит *модуль* `doctest`, который помогает в тестировании кода и позволяет удобно провести повторное тестирование после внесения изменений. Когда вы выполняете *функцию* `testmod` модуля `doctest`, она анализирует doc-строки ваших функций, методов и классов в поисках команд Python с префиксом `>>>`, за которыми в следующей строке идет предполагаемый вывод команды (если он есть[1]). Затем функция `testmod` выполняет эти команды и убеждается в том, что их вывод совпадает с ожидаемым. Если вывод не совпадает, то `testmod` сообщает об ошибках и указывает, какие тесты не прошли, чтобы вы могли найти и исправить ошибки в своем коде. Каждый тест, определяемый в doc-строке, обычно предназначен для тестирования конкретной *программной единицы*, например функции, модуля или класса. Такие тесты называются *модульными*.

Обновленный класс Account

Файл accountdoctest.py содержит класс `Account` из первого примера этой главы. Мы изменили doc-строку метода `__init__` и добавили четыре теста для проверки правильности работы метода:

- тест в строке 11 создает объект `Account` с именем `account1`. Эта команда не создает вывод;
- тест в строке 12 показывает, каким должно быть значение атрибута `name` объекта `account1` в случае успешного выполнения строки 11. Пример вывода показан в строке 13;
- тест в строке 14 показывает, каким должно быть значение атрибута `balance` объекта `account1` в случае успешного выполнения строки 11. Пример вывода показан в строке 15;
- тест в строке 18 создает объект `Account` с недействительным исходным балансом. Вывод показывает, что в этом случае должно произойти исключение `ValueError`. Для исключений в документации модуля `doctest` рекомендуется выводить только первую и последнюю строки трассировки[2].

Тесты можно чередовать с описательным текстом, как в строке 17.

---

[1] Синтаксис >>> моделирует приглашения ввода стандартного интерпретатора Python.

[2] https://docs.python.org/3/library/doctest.html?highlight=doctest#module-doctest.

```
# accountdoctest.py
"""Определение класса Account."""
from decimal import Decimal

class Account:
    """Класс Account для демонстрации doctest."""

    def __init__(self, name, balance):
        """Инициализирует объект Account.

        >>> account1 = Account('John Green', Decimal('50.00'))
        >>> account1.name
        'John Green'
        >>> account1.balance
        Decimal('50.00')

        The balance argument must be greater than or equal to 0.
        >>> account2 = Account('John Green', Decimal('-50.00'))
        Traceback (most recent call last):
            ...
        ValueError: Initial balance must be >= to 0.00.
        """

        # Если balance меньше 0.00, выдать исключение
        if balance < Decimal('0.00'):
            raise ValueError('Initial balance must be >= to 0.00.')

        self.name = name
        self.balance = balance

    def deposit(self, amount):
        """Внесение средств на счет."""

        # Если amount меньше 0.00, выдать исключение
        if amount < Decimal('0.00'):
            raise ValueError('amount must be positive.')

        self.balance += amount

if __name__ == '__main__':
    import doctest
    doctest.testmod(verbose=True)
```

Модуль __main__

При загрузке любого модуля Python присваивает строку, содержащую имя модуля, глобальному атрибуту модуля с именем `__name__`. При выполнении исходного файла Python (например, accountdoctest.py) в виде *сценария* Python

использует строку `'__main__'` как имя модуля. Вы можете использовать `__name__` в команде `if` (строки 40–42), чтобы указать код, который должен выполняться только в случае выполнения исходного файла в форме *сценария*. В данном примере строка 41 импортирует модуль `doctest`, а строка 42 вызывает функцию `testmod` модуля для выполнения модульных тестов из doc-строк.

Выполнение тестов

Запустите файл accountdoctest.py в виде сценария, чтобы выполнить тесты. По умолчанию при вызове `testmod` без аргументов результаты *успешно прошедших* тестов не выводятся. В этом случае отсутствие вывода означает, что все тесты были выполнены успешно. В данном примере строка 42 вызывает `testmod` с ключевым аргументом `verbose=True`. Он приказывает `testmod` выдавать подробный вывод с результатами *каждого* теста:

```
Trying:
    account1 = Account('John Green', Decimal('50.00'))
Expecting nothing
ok
Trying:
    account1.name
Expecting:
    'John Green'
ok
Trying:
    account1.balance
Expecting:
    Decimal('50.00')
ok
Trying:
    account2 = Account('John Green', Decimal('-50.00'))
Expecting:
    Traceback (most recent call last):
        ...
    ValueError: Initial balance must be >= to 0.00.
ok
3 items had no tests:
    __main__
    __main__.Account
    __main__.Account.deposit
1 items passed all tests:
   4 tests in __main__.Account.__init__
4 tests in 4 items.
4 passed and 0 failed.
Test passed.
```

В режиме подробного вывода функция `testmod` для каждого теста сообщает, что она пытается сделать (`"Trying"`) и что она ожидает получить в результате (`"Expecting"`), а в случае успешного прохождения теста выводит `"ok"`. После завершения тестов в режиме подробного вывода `testmod` выводит сводку результатов.

Чтобы продемонстрировать *неудачу* при прохождении теста, закомментируйте строки 25–26 в файле accountdoctest.py, поставив в начало строки знак #, после чего выполните accountdoctest.py в форме сценария. Для экономии места мы приводим только части вывода `doctest`, относящиеся к сбойному тесту:

```
...
**********************************************************************
File "accountdoctest.py", line 18, in __main__.Account.__init__
Failed example:
    account2 = Account('John Green', Decimal('-50.00'))
Expected:
    Traceback (most recent call last):
        ...
    ValueError: Initial balance must be >= to 0.00.
Got nothing
**********************************************************************
1 items had failures:
   1 of   4 in __main__.Account.__init__
4 tests in 4 items.
3 passed and 1 failed.
***Test Failed*** 1 failures.
```

В данном случае мы видим, что тест в строке 18 не прошел. Функция `testmod` *ожидала* получить трассировку, которая указывает, что из-за недопустимого исходного баланса была инициирована ошибка `ValueError`. Это исключение *не* произошло, что и привело к провалу теста. Для программиста, отвечающего за определение этого класса, ошибка при прохождении этого теста указывает на то, что в коде проверки из метода `__init__` что-то пошло не так.

Магическая команда IPython %doctest_mode

Удобный способ создания doc-тестов для существующего кода основан на использовании интерактивного сеанса IPython при тестировании кода с последующим копированием/вставкой сеанса в doc-строку. Приглашения IPython `In []` и `Out[]` несовместимы с `doctest`, поэтому для вывода приглашений в формате `doctest` IPython предоставляет магическую команду `%doctest_mode`, которая переключается между двумя стилями оформления приглашений. При

первом выполнении `%doctest_mode` IPython переключается в режим с приглашениями `>>>` для ввода и без приглашений при выводе. При втором выполнении `%doctest_mode` IPython возвращается к приглашениям `In []` и `Out[]`.

10.15. Пространства имен и области видимости

Каждый идентификатор имеет область видимости, определяющую, в каких местах программы он может использоваться; напомним, области видимости могут быть локальными и глобальными. Продолжим изучение темы областей видимости и рассмотрим пространства имен.

Области видимости определяются *пространствами имен*, существующими независимо друг от друга, связывающими идентификаторы с объектами и строящими внутреннюю реализацию на базе словарей. Таким образом, один идентификатор может входить в несколько пространств имен. Основные пространства имен: локальное, глобальное и встроенное.

Локальное пространство имен

Каждая функция и метод обладают *локальным пространством имен*, связывающим локальные идентификаторы (например, параметры и локальные переменые) с объектами. Локальное пространство имен существует от момента вызова функции (метода) до его завершения и доступно *только* для этой функции (метода). Внутри набора функции или метода *присваивание* несуществующей переменной приводит к созданию локальной переменной и включению ее в локальное пространство имен. Идентификаторы в локальном пространстве имен находятся в *области видимости* от той точки, в которой они определяются, и до завершения функции или метода.

Глобальное пространство имен

Каждый модуль обладает *глобальным пространством имен*, связывающим глобальные идентификаторы модуля (глобальные переменные, имена функций и имена классов) с объектами. Python создает глобальное пространство имен при загрузке модуля. Глобальное пространство имен модуля существует, а его идентификаторы находятся в *области видимости* для кода внутри этого модуля до завершения программы (интерактивного сеанса). Сеанс IPython обладает собственным глобальным пространством имен для всех идентификаторов, созданных в этом сеансе.

Глобальное пространство имен каждого модуля также включает идентификатор с именем `__name__`, содержащий имя модуля (например, `'math'` для модуля `math` или `'random'` для модуля `random`). Как показано в примере `doctest` из предыдущего раздела, `__name__` содержит `'__main__'` для файла `.py`, запускаемого в виде сценария.

Встроенное пространство имен

Встроенное пространство имен содержит сопутствующие идентификаторы для встроенных функций Python (например, `input` и `range`) и типов (например, `int`, `float` и `str`) с объектами, определяющими эти функции и типы. Python создает встроенное пространство имен в начале выполнения интерпретатора. Идентификаторы встроенного пространства имен остаются в *области видимости* для всего кода вплоть до завершения программы (или интерактивного сеанса[1]).

Поиск идентификаторов в пространствах имен

При использовании идентификатора Python ищет этот идентификатор в пространствах имен, доступных в настоящий момент; поиск начинается с *локального* пространства имен, переходит к *глобальному*, а затем к *встроенному*. Для лучшего понимания порядка поиска в пространствах имен возьмем следующий сеанс IPython:

```
In [1]: z = 'global z'

In [2]: def print_variables():
   ...:     y = 'local y in print_variables'
   ...:     print(y)
   ...:     print(z)
   ...:

In [3]: print_variables()
local y in print_variables
global z
```

Идентификаторы, определяемые в сеансе IPython, размещаются в *глобальном* пространстве имен сеанса. Когда фрагмент [3] вызывает `print_variables`,

[1] Предполагается, что вы не замещаете встроенные функции или типы, переопределяя их идентификаторы в локальном или глобальном пространстве имен. Замещение рассматривалось в главе 4.

Python проводит поиск по *локальному*, *глобальному* и *встроенному* пространствам имен:

- Фрагмент [3] не является функцией или методом, поэтому *глобальное* и *встроенное* пространство имен сеанса доступны. Python сначала просматривает *глобальное* пространство имен сеанса, содержащее `print_variables`. Таким образом, `print_variables` находится *в области видимости*, и Python использует соответствующий объект для вызова `print_variables`.
- В начале выполнения `print_variables` Python создает *локальное* пространство имен функции. Когда функция `print_variables` определяет локальную переменную `y`, Python добавляет `y` в *локальное* пространство имен функции. Переменная `y` находится *в области видимости*, пока функция не завершится.
- Затем `print_variables` вызывает *встроенную* функцию `print`, передавая `y` в аргументе. Чтобы выполнить этот вызов, Python должен найти идентификаторы `y` и `print`. Идентификатор `y` определяется в *локальном* пространстве имен, поэтому он находится *в области видимости*, и Python сможет использовать соответствующий объект (строка `'local y in print_variables'`) как аргумент `print`. Чтобы вызвать функцию, Python должен найти объект, соответствующий `print`. Сначала поиск ведется по локальному пространству имен, в котором `print` *не* определяется. Затем поиск продолжается по *глобальному* пространству имен сеанса, в котором тоже *не* определяется `print`. Наконец, поиск продолжается во *встроенном* пространстве имен, в котором идентификатор `print` *определяется*. Следовательно, `print` находится *в области видимости*, и Python использует соответствующий объект для вызова `print`.
- Затем `print_variables` снова вызывает *встроенную* функцию `print` — уже с аргументом `z`, который *не* определяется в *локальном* пространстве имен. Python переходит к *глобальному* пространству имен. Аргумент `z` определяется в *глобальном* пространстве имен, так что `z` находится *в области видимости*, и Python использует соответствующий объект (строка `'global z'`) в качестве аргумента `print`. И снова Python находит идентификатор `print` во *встроенном* пространстве имен, используя соответствующий объект для вызова `print`.
- К этому моменту достигнут конец набора функции `print_variables`, функция завершается, а ее *локальное* пространство имен перестает существовать; это означает, что локальная переменная `y` стала неопределенной.

Чтобы доказать, что идентификатор y не определен, попробуем вывести y:

```
In [4]: y
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-9063a9f0e032> in <module>()
----> 1 y

NameError: name 'y' is not defined
```

В этом случае локального пространства имен нет, поэтому Python ищет y в *глобальном* пространстве имен сеанса. Идентификатор y здесь не определен, и Python продолжает поиск y во *встроенном* пространстве имен, так и не находя y. Пространств имен не осталось, поэтому Python выдает исключение NameError, которое показывает, что идентификатор y не определен.

Но идентификаторы print_variables и z все еще существуют в *глобальном* пространстве имен сеанса, и их по-прежнему можно использовать. Например, вычислим идентификатор z, чтобы узнать его значение:

```
In [5]: z
Out[5]: 'global z'
```

Вложенные функции

Одним из пространств имен, которые не были рассмотрены в предшествующем обсуждении, является *вмещающее пространство имен*. Python позволяет определять *вложенные функции* внутри других функций или методов. Например, если функция или метод выполняют одну операцию несколько раз, можно определить вложенную функцию, чтобы избежать повторения кода во вмещающей функции. Когда вы обращаетесь к идентификатору внутри вложенной функции, Python сначала просматривает *локальное* пространство имен вложенной функции, затем пространство имен *вмещающей* функции, после этого *глобальное* пространство имен и, наконец, *встроенное* пространство имен. Иногда эта последовательность называется *правилом LEGB* (*Local, Enclosing, Global, Built-in*).

Пространство имен класса

Класс содержит пространство имен, в котором хранятся все атрибуты его класса. При обращении к атрибуту класса Python сначала ищет этот атрибут в пространстве имен класса, затем в пространстве имен базового класса и т. д.,

пока атрибут не будет найден или не будет достигнут класс `object`. Если атрибут не найден, то происходит ошибка `NameError`.

Пространство имен объекта

Каждый объект имеет собственное пространство имен, содержащее методы и атрибуты данных объекта. Метод `__init__` класса начинает с пустого объекта (`self`) и последовательно добавляет атрибуты в пространство имен объекта. После того как атрибут определен в пространстве имен объекта, клиенты, использующие объект, могут обращаться к значению атрибута.

10.16. Введение в data science: временные ряды и простая линейная регрессия

Ранее нами рассматривались последовательности: списки, кортежи и массивы. В этом разделе рассматриваются *временны́е ряды* — последовательности значений (называемых *наблюдениями*), связанных с определенными моментами времени. Примерами такого рода являются котировки на момент закрытия операций на бирже, почасовые измерения температуры, изменения локации летящего самолета, годовая урожайность и квартальная прибыль компании. Пожалуй, самым выразительным примером временного ряда является поток твитов с временными метками, поступающих от пользователей Twitter со всего мира (подробный анализ данных Twitter представлен в главе 12).

Для формирования прогнозов на основе данных временных рядов (точнее, для прогнозирования средних январских температур в будущем и средних январских температур до 1895 года) воспользуемся методом простой линейной регрессии и данными средней январской температуры в Нью-Йорке с 1895 по 2018 год. Кстати, в главе 14 мы вернемся к этому примеру с библиотекой `scikit-learn`, а в главе 15 для анализа временных рядов применены *рекуррентные нейронные сети* (*RNN*). В главе 16 рассматриваются временные ряды, часто встречающиеся в финансовых приложениях и в области «интернета вещей» (IoT).

В этом разделе для вывода диаграмм будут использоваться библиотеки Seaborn и pandas, работающие на базе Matplotlib. Запустите IPython с поддержкой Matplotlib:

```
ipython --matplotlib
```

Временные ряды

Данные, которые будут использованы в примере, представляют собой временной ряд, в котором наблюдения *упорядочены* по годам. *Одномерные временные ряды* содержат *одно* наблюдение на один момент времени, например среднюю январскую температуру в Нью-Йорке за конкретный год. *Многомерные временные ряды* содержат по *два и более* наблюдения на один момент времени (например, температуру, влажность и атмосферное давление в метеорологическом приложении). В этой главе будут анализироваться одномерные временные ряды.

С временными рядами часто выполняются две операции:

- *Анализ временных рядов* выявляет закономерности в существующих данных временных рядов, помогая аналитикам понять суть данных. Стандартная аналитическая задача — выявление *сезонности* в данных. Например, в Нью-Йорке ежемесячная температура существенно изменяется в зависимости от времени года (зима, весна, лето или осень).
- *Прогнозирование временных рядов* использует прошлые данные для прогнозирования будущего.

В этом разделе рассматривается прогнозирование временных рядов.

Простая линейная регрессия

При помощи метода, называемого *простой линейной регрессией*, построим прогнозы, выявляющие линейные отношения между месяцами (январь каждого года) и средней температурой в Нью-Йорке. Для заданной коллекции значений, представляющих *независимую переменную* (комбинация «месяц/год») и *зависимую переменную* (средняя температура за этот месяц/год), простая линейная регрессия описывает отношение между этими переменными прямой линией — *регрессионной прямой*.

Линейные отношения

Чтобы понять общую концепцию линейных отношений, рассмотрим температуры по шкале Фаренгейта и Цельсия. Для заданной температуры по Фаренгейту соответствующая температура по Цельсию вычисляется по следующей формуле:

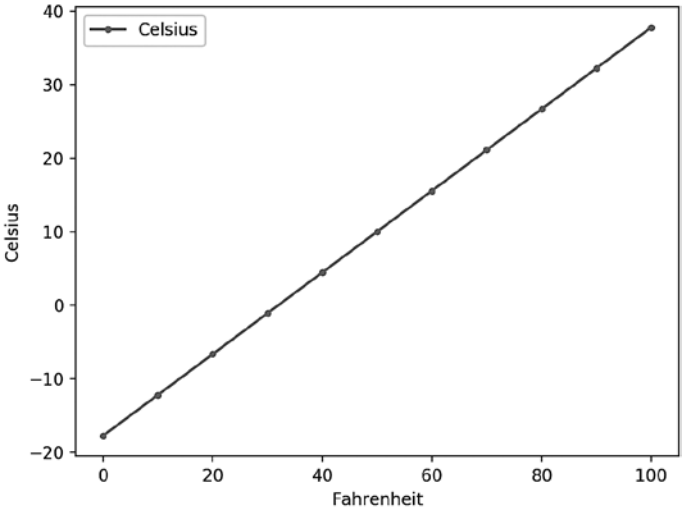
```
c = 5 / 9 * (f - 32)
```

В этой формуле `f` (температура по Фаренгейту) — *независимая переменная*, а `c` (температура по Цельсию) — *зависимая переменная*; каждое значение `c` *зависит* от значения `f`, использованного при вычислении.

График температур по Фаренгейту и соответствующих им температур по Цельсию представляет собой прямую линию. Чтобы убедиться в этом, сначала создадим лямбда-выражение для предыдущей формулы и воспользуемся им для вычисления эквивалентов по шкале Цельсия для температур по Фаренгейту 0–100 с приращением 10 градусов. Каждая пара температур по Фаренгейту/Цельсию будет храниться в виде кортежа в `temps`:

```
In [1]: c = lambda f: 5 / 9 * (f - 32)

In [2]: temps = [(f, c(f)) for f in range(0, 101, 10)]
```

Поместим данные в `DataFrame` и используем *метод* `plot` для вывода линейной зависимости между температурами по Фаренгейту и по Цельсию. Ключевой аргумент `style` метода `plot` управляет внешним видом данных. Точка в строке `'.-'` указывает, что каждая точка данных должна обозначаться точкой на графике, а дефис — что точки должны соединяться линиями. Оси *y* вручную назначается метка `'Celsius'`, потому что метод `plot` по умолчанию выводит `'Celsius'` только в левом верхнем углу условных обозначений на графике:




```
In [3]: import pandas as pd

In [4]: temps_df = pd.DataFrame(temps, columns=['Fahrenheit', 'Celsius'])
```

```
In [5]: axes = temps_df.plot(x='Fahrenheit', y='Celsius', style='.-')
In [6]: y_label = axes.set_ylabel('Celsius')
```

Компоненты уравнения простой линейной регрессии

Точки на любой прямой линии (в двумерном пространстве) могут быть описаны уравнением:

$$y = mx + b,$$

где m — коэффициент *наклона* линии;

b — *точка пересечения* линии с осью y (при $x = 0$);

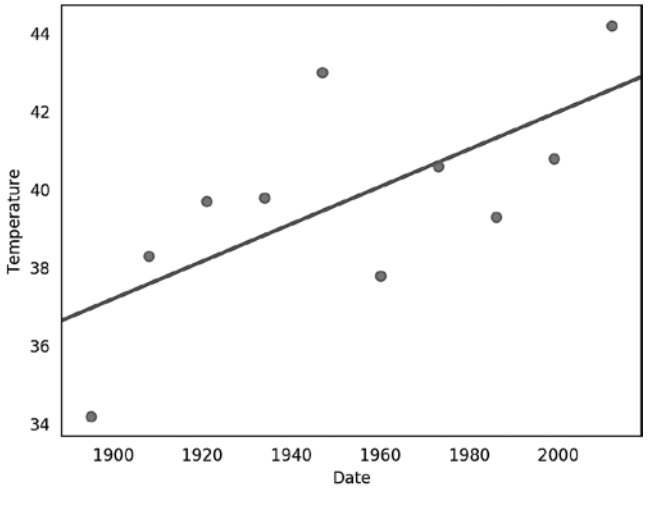
x — независимая переменная (дата в нашем примере);

y — зависимая переменная (температура в нашем примере).

В простой линейной регрессии y — *прогнозируемое значение* для заданного x.

Функция linregress из модуля stats библиотеки SciPy

Простая линейная регрессия определяет коэффициент наклона (m) и точку пересечения (b) прямой линии, лучше всего подходящую к вашим данным. На следующей диаграмме показаны некоторые точки данных временного ряда, обрабатываемые в этом разделе, и соответствующая линия регрессии. Мы добавили вертикальные линии для обозначения расстояний каждой строки от регрессионной прямой:



Алгоритм простой линейной регрессии производит итерационную настройку угла наклона и точки пересечения, вычисляя для каждой корректировки квадрат расстояния каждой точки от линии. «Наилучшая подгонка» достигается, когда значения угла наклона и точки пересечения минимизируют сумму квадратов расстояний. Этот принцип называется *обычным методом наименьших квадратов*[1].

Библиотека SciPy (*Scientific Python*) широко применяется в инженерных, научных и математических вычислениях на языке Python. Функция `linregress` этой библиотеки (из *модуля* `scipy.stats`) выполняет простую линейную регрессию за вас. После вызова `linregress` остается подставить полученные значения угла наклона и точки пересечения в формулу $y = mx + b$ для получения прогноза.

Pandas

В трех предыдущих разделах «Введение в data science» для работы с данными использовалась библиотека pandas. Мы продолжим использовать pandas в оставшейся части книги. В этом примере мы загрузим данные средних январских температур в Нью-Йорке в 1895–2018 годах из CSV-файла в `DataFrame`. Затем данные будут отформатированы для использования в примере.

Визуализация в Seaborn

Библиотека Seaborn будет использована для графического представления данных `DataFrame` в виде регрессионной прямой, представляющей график изменения средней температуры за период 1895–2018 годов.

Получение метеорологических данных от NOAA

Загрузим данные для исследования. Национальное управление по исследованию океанов и атмосферы (NOAA, National Oceanic and Atmospheric Administration[2], или Национальное управление по исследованию океанов и атмосферы) предоставляет доступ к обширным статистическим данным, включая временные ряды для средних температур в конкретных городах с различными интервалами времени.

---

[1] https://en.wikipedia.org/wiki/Ordinary_least_squares.

[2] http://www.noaa.gov.

Мы получили средние январские температуры в Нью-Йорке с 1895 по 2018 год из временных рядов NOAA «Climate at a Glance»:

https://www.ncdc.noaa.gov/cag/

На этой веб-странице можно выбрать температуру, уровень осадков и другие данные для целых регионов США, штатов, городов и т. д. Выбрав зону и период времени, щелкните на кнопке Plot, чтобы вывести диаграмму и просмотреть таблицу с выбранными данными. В верхней части таблицы размещаются ссылки для загрузки данных в нескольких форматах, включая формат CSV (см. главу 9). На момент написания книги максимальный диапазон доступных данных NOAA — с 1895 до 2018 года. Для удобства мы разместили данные в каталоге примеров ch10 в файле ave_hi_nyc_jan_1895-2018.csv. Если вы загрузите данные самостоятельно, то удалите строки, лежащие выше строки `"Date,Value,Anomaly"`. Данные содержат три столбца для каждого наблюдения:

- `Date` — значение в форме `'YYYYMM'` (например, `'201801'`). Часть `MM` всегда содержит `01`, потому что мы загружали данные только за январь каждого года.
- `Value` — температура по Фаренгейту в формате с плавающей точкой.
- `Anomaly` — разность между значением для заданной даты и средними значениями для всех дат. В нашем примере значение `Anomaly` не используется, поэтому мы его проигнорируем.

Загрузка средних температур в DataFrame

Загрузим и выведем данные для Нью-Йорка из файла ave_hi_nyc_jan_1895-2018.csv:

```
In [7]: nyc = pd.read_csv('ave_hi_nyc_jan_1895-2018.csv')
```

Чтобы получить общее представление о данных, можно просмотреть начальные и конечные записи `DataFrame`:

```
In [8]: nyc.head()
Out[8]:
     Date  Value  Anomaly
0  189501   34.2     -3.2
1  189601   34.7     -2.7
2  189701   35.5     -1.9
3  189801   39.6      2.2
4  189901   36.4     -1.0
```

```
In [9]: nyc.tail()
Out[9]:
       Date  Value  Anomaly
119  201401   35.5     -1.9
120  201501   36.1     -1.3
121  201601   40.8      3.4
122  201701   42.8      5.4
123  201801   38.7      1.3
```

Очистка данных

Скоро мы воспользуемся Seaborn для графического представления пар `Date-Value` и регрессионной прямой. При отображении данных из `DataFrame` Seaborn помечает оси графика именами столбцов `DataFrame`. Для удобочитаемости переименуем столбец `'Value'` в `'Temperature'`:

```
In [10]: nyc.columns = ['Date', 'Temperature', 'Anomaly']

In [11]: nyc.head(3)
Out[11]:
     Date  Temperature  Anomaly
0  189501         34.2     -3.2
1  189601         34.7     -2.7
2  189701         35.5     -1.9
```

Seaborn помечает деления на оси *x* значениями `Date`. Поскольку в примере обрабатываются только январские данные, метки оси *x* будут лучше читаться без обозначения `01` (для января); удалим его из `Date`. Сначала проверим тип столбца:

```
In [12]: nyc.Date.dtype
Out[12]: dtype('int64')
```

Значения являются целыми числами, поэтому мы можем разделить их на 100 для отсечения двух последних цифр. Вспомните, что каждый столбец в `DataFrame` представляет собой коллекцию `Series`. Вызов метода `floordiv` коллекции `Series` выполняет *целочисленное деление* с каждым элементом `Series`:

```
In [13]: nyc.Date = nyc.Date.floordiv(100)

In [14]: nyc.head(3)
Out[14]:
   Date  Temperature  Anomaly
0  1895         34.2     -3.2
1  1896         34.7     -2.7
2  1897         35.5     -1.9
```

Вычисление базовых описательных статистик для наборов данных

Чтобы быстро получить некоторые статистики для температур из набора данных, вызовем `describe` для столбца `Temperature`. В коллекции присутствуют 124 наблюдения, среднее значение наблюдений равно `37.60`, а наименьшее и наибольшее наблюдение равны `26.10` и `47.60` градусам соответственно:

```
In [15]: pd.set_option('precision', 2)

In [16]: nyc.Temperature.describe()
Out[16]:
count    124.00
mean      37.60
std        4.54
min       26.10
25%       34.58
50%       37.60
75%       40.60
max       47.60
Name: Temperature, dtype: float64
```

Прогнозирование будущих январских температур

Библиотека SciPy (*Scientific Python*) широко применяется в инженерных, научных и математических вычислениях на языке Python. Ее *модуль* `stats` предоставляет функцию `linregress`, которая вычисляет *наклон* и *точку пересечения* регрессионной прямой для заданного набора точек данных:

```
In [17]: from scipy import stats

In [18]: linear_regression = stats.linregress(x=nyc.Date,
    ...:                                       y=nyc.Temperature)
    ...:
```

Функция `linregress` получает два одномерных массива[1] одинаковой длины, представляющих координаты *x* и *y* точек данных. Ключевые аргументы `x` и `y` представляют независимые и зависимые переменные соответственно. Объект, возвращаемый `linregress`, содержит угол наклона и точку пересечения регрессионной прямой:

---

[1] Эти аргументы также могут быть объектами, сходными с одномерными массивами, например списками или коллекциями `Series`.

```
In [19]: linear_regression.slope
Out[19]: 0.01407713611329661677

In [20]: linear_regression.intercept
Out[20]: 8.694845520062952
```

Эти значения можно объединить с уравнением простой линейной регрессии для прямой линии, $y = mx + b$ при прогнозировании средней январской температуры в Нью-Йорке для заданного года. Спрогнозируем среднюю температуру по Фаренгейту за январь 2019 года. В следующих вычислениях `linear_regression.slope` соответствует m, 2019 соответствует x (значение, для которого прогнозируется температура), а `linear_regression.intercept` соответствует b:

```
In [21]: linear_regression.slope * 2019 + linear_regression.intercept
Out[21]: 38.51837136113298
```

Также по формуле можно приближенно оценить, какой могла быть средняя температура до 1895 года. Например, оценка средней температуры за январь 1890 года может быть получена следующим образом:

```
In [22]: linear_regression.slope * 1890 + linear_regression.intercept
Out[22]: 36.612865774980335
```

Напомним, в примере были доступны данные за 1895–2018 годы, и чем дальше вы выходите за границы диапазона, тем менее надежными становятся прогнозы.

Построение графика со средними температурами и регрессионной прямой

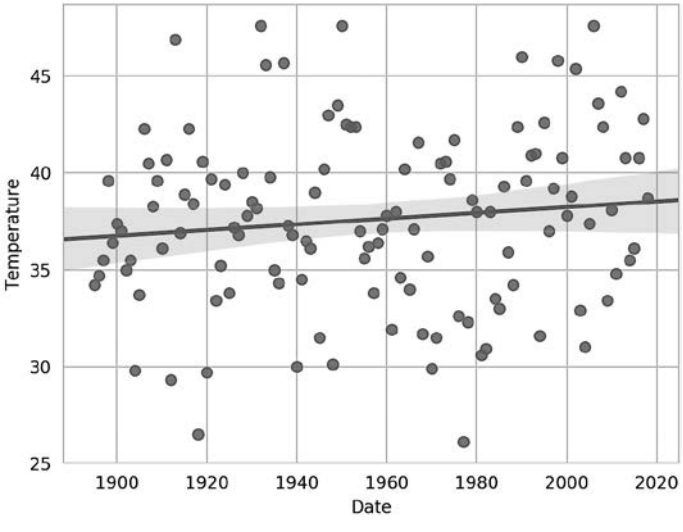
Затем воспользуемся *функцией* `regplot` библиотеки Seaborn для вывода всех точек данных; даты представляются на оси x, а температуры на оси y. Функция `regplot` строит *диаграмму разброса данных*, на которой точки представляют температуры за заданный год, а прямая линия — регрессионную прямую.

Сначала закройте предыдущее окно Matplotlib, если это не было сделано ранее, в противном случае `regplot` будет использовать существующее окно, в котором уже находится график. Ключевые аргументы `x` и `y` функции `regplot` — одномерные массивы[1] совпадающей длины, представляющие пары координат x-y

[1] Эти аргументы также могут быть объектами, сходными с одномерными массивами, например списками или коллекциями Series библиотеки pandas.

для нанесения на график. Напомним, pandas автоматически создает атрибуты для каждого имени столбца, если оно является действительным идентификатором Python[1]:

```
In [23]: import seaborn as sns

In [24]: sns.set_style('whitegrid')

In [25]: axes = sns.regplot(x=nyc.Date, y=nyc.Temperature)
```

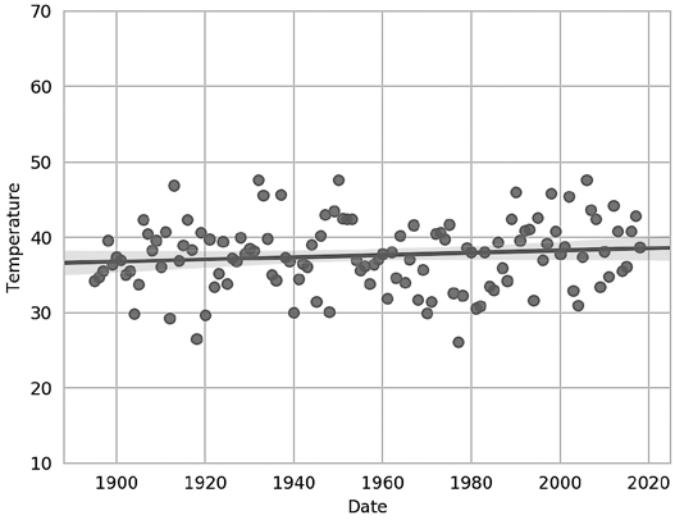


Наклон регрессионной прямой (подъем от левой части к правой) указывает на то, что в последние 124 года средняя температура повышалась. На графике ось *y* представляет температурный диапазон между минимумом `26.1` и максимумом `47.6` по Фаренгейту, в результате чего наблюдается значительный разброс данных вокруг регрессионной прямой, из-за которого труднее выявить линейную связь. Эта проблема типична для визуализаций в области аналитики данных. Если оси на графике отражают разные типы данных (даты и температуры в данном случае), то как разумно определить их относительные масштабы? На предыдущем графике все определяется исключительно высотой графика — Seaborn и Matplotlib *автоматически масштабируют* оси

[1] Затененная область рядом с регрессионной прямой демонстрирует 95-процентный *доверительный интервал* для регрессионной прямой (https://en.wikipedia.org/wiki/Simple_linear_regression#Confidence_intervals). Для вывода графика без доверительного интервала добавьте ключевой аргумент `ci=None` в список аргументов функции `regplot`.

на основании диапазонов значений данных. Ось значений *y* можно масштабировать, чтобы подчеркнуть линейность отношения. В следующем примере ось *y* была масштабирована от 21,5-градусного диапазона до 60-градусного диапазона (от 10 до 70 градусов):

```
In [26]: axes.set_ylim(10, 70)
Out[26]: (10, 70)
```



Загрузка наборов данных временных рядов

Ниже перечислены некоторые популярные сайты, на которых вы сможете загрузить временные ряды для своих исследований.

Исходные наборы данных с временными рядами

- https://data.gov/ — портал открытых данных правительства США. Поиск текста «time series» дает свыше 7200 наборов данных с временными рядами.
- https://www.ncdc.noaa.gov/cag/ — климатический портал NOAA предоставляет данные временных рядов — как глобальные, так и для США.
- https://www.esrl.noaa.gov/psd/data/timeseries/ — портал Земной научно-исследовательской лаборатории (ESRL, Earth System Research Laboratory) администрации NOAA предоставляет ежемесячные и сезонные временные ряды с климатическими данными.

- https://www.quandl.com/search — Quandl предоставляет сотни бесплатных временных рядов с финансовыми данными, а также наборы данных с платным доступом.
- https://datamarket.com/data/list/?q=provider:tsdl — библиотека данных TSDL (Time Series Data Library) содержит ссылки на сотни наборов данных временных рядов по многим промышленным областям.
- http://archive.ics.uci.edu/ml/datasets.html — репозиторий машинного обучения Калифорнийского университета в Ирвайне (UCI, University of California Irvine) содержит десятки наборов данных временных рядов из разнообразных областей.
- http://inforumweb.umd.edu/econdata/econdata.html — сервис EcoData Университета Мэриленда предоставляет ссылки на тысячи экономических временных рядов, собранных различными правительственными агентствами США.

10.17. Итоги

В этой главе подробно рассматривалась тема построения полезных классов. Вы узнали, как определить класс, создать объекты класса, обратиться к атрибутам объекта и вызвать его методы. Мы определили специальный метод `__init__` для создания и инициализации атрибутов данных нового объекта, рассмотрели тему управления доступом к атрибутам и использования свойств, показав, что клиент может напрямую обращаться ко всем атрибутам объектов. Идентификаторы, начинающиеся с одного символа подчеркивания (_), обозначают атрибуты, не предназначенные для обращения со стороны клиентского кода. Мы показали, как реализовать «приватные» атрибуты при помощи соглашения о двойном начальном символе подчеркивания (__), предписывающем Python преобразовать имя атрибута.

Затем была реализована модель тасования и сдачи карт, в которой был задействован класс `Card` и класс `DeckOfCards`, поддерживающий список `Card`. Колода карт выводилась как в строковом виде, так и в виде изображений карт с использованием Matplotlib. Мы реализовали специальные методы `__repr__`, `__str__` и `__format__` для создания строковых представлений объектов.

Далее мы перешли к средствам Python для создания базовых классов и подклассов, показав, как создать подкласс, наследующий многие возможности суперкласса, а затем добавить новые средства — возможно, с переопределением

методов базового класса. В частности, был создан список, содержащий объекты базового класса и подкласса, для демонстрации возможностей полиморфного программирования Python.

Перегрузка операторов определяет, как встроенные операторы Python должны работать с объектами пользовательских классов. Вы узнали, что перегруженные методы операторов реализуются перегрузкой различных специальных методов, наследуемых всеми классами от класса `object`. Мы обсудили иерархию классов исключений Python и проблемы создания пользовательских классов исключений.

Мы показали, как создать именованный кортеж, позволяющий обращаться к элементам кортежа по именам атрибутов вместо индексов. Затем были представлены новые классы данных Python 3.7, которые могут автоматически генерировать различный шаблонный код, который обычно предоставляется в определениях классов, в частности специальные методы `__init__`, `__repr__` и `__eq__`.

Вы узнали, как писать модульные тесты для вашего кода в doc-строках, а затем выполнять их при помощи функции `testmod` модуля `doctest`. Наконец, мы обсудили различные пространства имен, используемые Python для определения области видимости идентификаторов.

В следующей части книги будут представлены практические примеры, использующие смесь искусственного интеллекта и технологий больших данных. Мы исследуем обработку естественного языка, анализ данных Twitter, IBM Watson и когнитивных вычислений, машинное обучение с учителем и без, а также глубокое обучение с применением сверточных и рекуррентных нейронных сетей. Будет обсуждаться программное обеспечение и аппаратная инфраструктура больших данных, включая базы данных NoSQL, Hadoop и Spark, уделяющие первостепенное внимание производительности. Словом, вы узнаете много интересного!

11

Обработка естественного языка (NLP)

В этой главе...

- Задачи обработки естественного языка (NLP), имеющие фундаментальное значение для многих последующих глав с практическими применениями data science.
- Выполнение демонстрационных приложений NLP.
- Использование NLP-библиотек TextBlob, NLTK, Textatistic и spaCy, а также их предварительно обученных моделей для выполнения различных задач NLP.
- Разбиение текста на слова и предложения.
- Пометка частей речи.
- Использование анализа эмоциональной окраски высказываний для определения положительной, отрицательной или нейтральной окраски текста.
- Распознавание языка текста и перевод на другие языки с использованием поддержки TextBlob Google Translate.
- Определение корней слов посредством выделения основы и лемматизации.
- Средства проверки орфографии и исправления ошибок в TextBlob.
- Получение определений слов, синонимов и антонимов.
- Удаление игнорируемых слов из текста.
- Построение словарных облаков.
- Определение удобочитаемости текста при помощи Textatistic.
- Использование библиотеки spaCy для распознавания именованных сущностей и выявления сходства.

11.1. Введение

Вы просыпаетесь от звонка будильника и нажимаете кнопку «Выкл.». Вы берете в руки смартфон, читаете текстовые сообщения и просматриваете свежие новости. Вы слушаете, как ведущие телепрограммы берут интервью у знаменитостей. Вы общаетесь с друзьями, коллегами и членами семьи, слушаете их ответы. У вас есть друг с нарушением слуха, с которым вы общаетесь на языке знаков и который смотрит видеопрограммы с субтитрами. У вас есть слепой коллега, который читает текст, написанный алфавитом Брайля, слушает книги, которые читает вслух специальная программа, и экранного диктора, который описывает содержимое экрана компьютера. Вы читаете сообщения электронной почты, отделяя «мусор» от важной информации, и отправляете ответы. Вы ведете машину, обращая внимание на дорожные знаки: «Стоп», «Ограничение скорости 60», «Дорожные работы» и т. д. Вы отдаете своей машине голосовые команды: «Позвонить домой», «Включить классическую музыку» или задаете вопросы типа «Где находится ближайшая заправка?». Вы учите ребенка говорить и читать. Отправляете открытку другу. Читаете художественные и научные тексты: книги, газеты и журналы. Вы делаете заметки во время лекции или встречи. Вы учите иностранный язык, готовясь к заграничной поездке. Вы получаете сообщение от клиента на испанском языке и обрабатываете его бесплатной программой-переводчиком. Вы отвечаете на английском языке, зная, что клиент может легко перевести его на испанский. Вы не уверены в том, на каком языке написано полученное сообщение, но программа моментально определяет это за вас и переводит текст на английский.

Все перечисленное — примеры общения на *естественном языке* в форме текста, голоса, видео, знака жестов, алфавита Брайля и в других формах на разных языках: английском, испанском, французском, русском, китайском, японском и сотнях других. В этой главе вы освоите ряд средств обработки естественного языка (NLP) на нескольких практических примерах и сеансах NLP. Многие из этих средств NLP будут использоваться в практических примерах data science в последующих главах.

Обработка естественного языка может применяться к текстовым коллекциям, состоящим из твитов, сообщений в Facebook, диалогов, рецензий на фильмы, пьес Шекспира, исторических документов, новостей, протоколов собраний и т. д. Текстовая коллекция также называется *корпусом*.

Естественные языки не обладают математической точностью. Смысловые нюансы могут усложнить понимание естественных языков. Смысл текста может

зависеть от контекста и «мировоззрения» читателя. Например, поисковые системы могут «изучить» вас на основании предыдущих запросов. Положительной стороной такого изучения может стать повышение качества результатов поиска, а отрицательной — нарушение неприкосновенности частной жизни.

11.2. TextBlob[1]

TextBlob — объектно-ориентированная библиотека NLP-обработки текста, построенная на базе NLP-библиотек *NLTK* и *pattern* и упрощающая некоторые аспекты их функциональности. Вот примеры операций NLP, которые можно выполнять при помощи TextBlob:

- *разбиение на лексемы* — разбиение текста на содержательные блоки (например, слова и числа);
- *пометка частей речи* — идентификация части речи каждого слова (существительное, глагол, прилагательное и т. д.);
- *извлечение именных конструкций* — обнаружение групп слов, представляющих имена существительные;
- *анализ эмоциональной окраски* — определение положительной, отрицательной или нейтральной окраски текста;
- *перевод на другие языки* и *распознавание языка* на базе Google Translate;
- *формообразование* — образование множественного и единственного числа. У формообразования существуют и другие аспекты, которые не поддерживаются TextBlob;
- *проверка орфографии* и *исправление ошибок*;
- *выделение основы* — исключение приставок, суффиксов и т. д.; например, при выделении основы из слова «varieties» будет получен результат «varieti»;
- *лемматизация* — аналог выделения основы, но с формированием реальных слов на основании контекста исходных слов; например, результатом лемматизации «varieties» является слово «variety»;
- *определение частот слов* — определение того, сколько раз каждое слово встречается в корпусе;

[1] https://textblob.readthedocs.io/en/latest/.

- *интеграция с WordNet* для поиска определений слов, синонимов и антонимов;
- *устранение игнорируемых слов* — исключение таких слов, как a, an, the, I, we, you и т. д., с целью анализа важных слов в корпусе;
- *n-граммы* — построение множеств последовательно идущих слов в корпусе для выявления слов, часто располагающихся по соседству друг с другом.

Многие из этих возможностей используются как составная часть более сложных задач NLP. В этом разделе эти NLP-задачи будут выполняться при помощи TextBlob и NLTK.

Установка модуля TextBlob

Чтобы установить TextBlob, откройте приглашение Anaconda (Windows), терминал (macOS/Linux) или командную оболочку (Linux), после чего выполните команду:

```
conda install -c conda-forge textblob
```

Возможно, пользователям Windows придется запустить приглашение Anaconda с правами администратора для получения необходимых привилегий для установки программного обеспечения. Щелкните правой кнопкой мыши на команде Anaconda Prompt в меню Пуск и выберите команду More ▸ Run as administrator.

После завершения установки выполните загрузку корпусов NLTK, используемых TextBlob:

```
ipython -m textblob.download_corpora
```

К их числу относятся:

- Brown Corpus (создан в Университете Брауна[1]) — для пометки частей речи.
- Punkt — для разбиения английских предложений на лексемы.
- WordNet — для определений слов, синонимов и антонимов.
- Averaged Perceptron Tagger — для пометки частей речи.

[1] https://en.wikipedia.org/wiki/Brown_Corpus.

- `conll2000` — для разбиения текста на компоненты (существительные, глаголы и т. д.). Имя `conll2000` происходит от конференции, на которой были созданы эти данные (Conference on Computational Natural Language Learning).
- Movie Reviews — для анализа эмоциональной окраски.

Проект «Гутенберг»

Превосходным источником текстов для анализа станут бесплатные электронные книги на сайте проекта «Гутенберг»:

https://www.gutenberg.org

Сайт содержит свыше 57 000 электронных книг в различных форматах, включая простые текстовые файлы. В США на эти книги не распространяется авторское право. За информацией об условиях использования сайта проекта «Гутенберг» и авторских правах в других странах обращайтесь по адресу:

https://www.gutenberg.org/wiki/Gutenberg:Terms_of_Use

В некоторых примерах этой книги используется простой текстовый файл с текстом пьесы Шекспира «*Ромео и Джульетта*», который можно загрузить по адресу:

https://www.gutenberg.org/ebooks/1513

Проект «Гутенберг» не предоставляет программного доступа к своим электронным книгам; для этого необходимо скопировать книги[1]. Чтобы загрузить «*Ромео и Джульетту*» в виде простого текстового файла, щелкните правой кнопкой мыши на ссылке Plain Text UTF-8 на веб-странице книги, после чего выберите команду Save Link As... (Chrome/FireFox), Download Linked File As... (Safari) или Save target as (Microsoft Edge), чтобы сохранить книгу в своей системе. Сохраните файл с именем RomeoAndJuliet.txt в каталоге ch11, чтобы примеры этой главы правильно работали. Для целей анализа мы удалили текст проекта «Гутенберг» перед строкой `"THE TRAGEDY OF ROMEO AND JULIET"`, а также информацию проекта «Гутенберг» в конце файла, начиная с текста:

```
End of the Project Gutenberg EBook of Romeo and Juliet,
by William Shakespeare
```

[1] https://www.gutenberg.org/wiki/Gutenberg:Information_About_Robot_Access_to_our_Pages.

11.2.1. Создание TextBlob

`TextBlob`[1] — фундаментальный класс для NLP-операций в *модуле* `textblob`. Создадим объект `TextBlob` с двумя предложениями:

```
In [1]: from textblob import TextBlob

In [2]: text = 'Today is a beautiful day. Tomorrow looks like bad weather.'

In [3]: blob = TextBlob(text)

In [4]: blob
Out[4]: TextBlob("Today is a beautiful day. Tomorrow looks like bad
weather.")
```

Объекты `TextBlob` (и, как вы вскоре убедитесь, объекты `Sentence` и `Word`) поддерживают строковые методы и могут сравниваться со строками. Они также предоставляют методы для различных NLP-операций. Классы `Sentence`, `Word` и `TextBlob` наследуют от `BaseBlob` и потому содержат много общих методов и свойств.

11.2.2. Разбиение текста на предложения и слова

Обработка естественного языка часто требует разбиения текста на лексемы перед выполнением других NLP-операций. `TextBlob` предоставляет удобные свойства для обращения к предложениям и словам в `TextBlob`. Используем *свойство* `sentence` для получения списка объектов `Sentence`:

```
In [5]: blob.sentences
Out[5]:
[Sentence("Today is a beautiful day."),
 Sentence("Tomorrow looks like bad weather.")]
```

Свойство `words` возвращает объект `WordList` со списком объектов `Word`, представляющим каждое слово в `TextBlob` после удаления знаков препинания:

```
In [6]: blob.words
Out[6]: WordList(['Today', 'is', 'a', 'beautiful', 'day', 'Tomorrow',
'looks', 'like', 'bad', 'weather'])
```

---

[1] http://textblob.readthedocs.io/en/latest/api_reference.html#textblob.blob.TextBlob.

11.2.3. Пометка частей речи

Пометка частей речи — процесс проверки слов с учетом контекста для определения части речи каждого слова. В английском языке существуют восемь основных частей речи — существительные, местоимения, глаголы, прилагательные, наречия, предлоги, союзы и междометия (слова, выражающие эмоции, за которыми обычно следует знак препинания, например «Yes!» или «Ha!»). В каждой категории существует множество подкатегорий.

Некоторые слова имеют несколько значений — так, у слов «set» или «run» возможные значения исчисляются сотнями! Взглянув на определения слова «run» на сайте dictionary.com, вы увидите, что оно может быть глаголом, существительным, прилагательным или частью глагольной группы. Одно из важных применений пометки частей речи — определение смысла слова из множества возможных вариантов. Эта операция играет важную роль в «понимании» естественных языков компьютерами.

Свойство `tags` возвращает список кортежей, каждый из которых содержит слово и строку, представляющую его пометку части речи:

```
In [7]: blob
Out[7]: TextBlob("Today is a beautiful day. Tomorrow looks like bad
weather.")

In [8]: blob.tags
Out[8]:
[('Today', 'NN'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('beautiful', 'JJ'),
 ('day', 'NN'),
 ('Tomorrow', 'NNP'),
 ('looks', 'VBZ'),
 ('like', 'IN'),
 ('bad', 'JJ'),
 ('weather', 'NN')]
```

По умолчанию `TextBlob` использует `PatternTagger` для определения частей речи, используя функциональность определения частей речи *библиотеки pattern*:

https://www.clips.uantwerpen.be/pattern

Шестьдесят три поддерживаемые пометки частей речи этой библиотеки можно найти по адресу:

https://www.clips.uantwerpen.be/pages/MBSP-tags

В выводе приведенного фрагмента:

- `Today`, `day` и `weather` имеют пометку `NN` — признак существительного в единственном или множественном числе.
- `is` и `looks` имеют пометку `VBZ` — признак глагола третьего лица единственного числа.
- `a` имеет пометку `DT` — признак определяющего слова[1].
- `beautiful` и `bad` имеют пометку `JJ` — признак прилагательного.
- `Tomorrow` имеет пометку `NNP` — признак имени собственного единственного числа.
- `like` имеет пометку `IN` — признак подчинительного союза или предлога.

11.2.4. Извлечение именных конструкций

Допустим, вы собираетесь купить водные лыжи и ищете информацию о них в интернете — скажем, по строке «best water ski». В данном случае группа «water ski» является именной конструкцией. Если поисковая система не сможет правильно выделить именную конструкцию, то вероятно, что полученные результаты не будут оптимальными. Попробуйте поискать информацию по строкам «best water», «best ski» и «best water ski» и посмотрите, что вы найдете.

Свойство `noun_phrases` класса `TextBlob` возвращает объект `WordList` со списком объектов `Word` — по одному для каждой именной конструкции в тексте:

```
In [9]: blob
Out[9]: TextBlob("Today is a beautiful day. Tomorrow looks like bad
weather.")

In [10]: blob.noun_phrases
Out[10]: WordList(['beautiful day', 'tomorrow', 'bad weather'])
```

Обратите внимание: объект `Word`, представляющий именную конструкцию, может содержать несколько слов. Класс `WordList` является расширением

[1] https://en.wikipedia.org/wiki/Determiner.

встроенного типа списка Python. Объекты `WordList` предоставляют дополнительные методы для выделения основы, лемматизации, образования формы единственного и множественного числа.

11.2.5. Анализ эмоциональной окраски с использованием анализатора TextBlob по умолчанию

Одна из самых частых и полезных NLP-операций — анализ эмоциональной окраски, определяющий положительную, отрицательную или нейтральную тональность текста. Например, компании могут использовать их для определения того, как потребители отзываются в интернете об их продуктах — положительно или отрицательно. Будем считать слово «good» положительным, а слово «bad» — отрицательным. Но сам факт присутствия слова «good» или «bad» в предложении еще не означает, что предложение в целом эмоционально окрашено положительно или отрицательно. Например, предложение

```
The food is not good.
```

безусловно, обладает отрицательной эмоциональной окраской. Аналогичным образом предложение

```
The movie was not bad.
```

явно обладает положительной эмоциональной окраской, хотя, возможно, и не настолько положительной, как предложение вида

```
The movie was excellent!
```

Анализ эмоциональной окраски — сложная задача из области машинного обучения. Тем не менее такие библиотеки, как TextBlob, содержат предварительно обученные модели для ее выполнения.

Оценка эмоциональной окраски средствами TextBlob

Свойство `sentiment` класса `TextBlob` возвращает объект `Sentiment`, который сообщает, имеет текст положительную или отрицательную эмоциональную окраску и является ли он объективным или субъективным:

```
In [11]: blob
Out[11]: TextBlob("Today is a beautiful day. Tomorrow looks like bad
weather.")
```

```
In [12]: blob.sentiment
Out[12]: Sentiment(polarity=0.07500000000000007,
subjectivity=0.8333333333333333)
```

В показанном выводе полярность (показатель `polarity`) означает эмоциональную окраску со значениями от `-1.0` (отрицательная) до `1.0` (положительная); значение `0.0` соответствует нейтральной эмоциональной окраске. На основании данных `TextBlob` общая эмоциональная окраска близка к нейтральной, а текст в целом субъективен.

Получение значений polarity и subjectivity из объекта Sentiment

Вероятно, показанные значения обладают большей точностью, чем необходимо в большинстве случаев. Излишняя точность может усложнить чтение числового вывода. Магическая команда IPython `%precision` позволяет задать точность по умолчанию для *автономных* объектов `float` и объектов `float` во *встроенных типах*, таких как списки, словари и кортежи. Воспользуемся этой магической командой для *округления* значений `polarity` и `subjectivity` до трех цифр в дробной части:

```
In [13]: %precision 3
Out[13]: '%.3f'

In [14]: blob.sentiment.polarity
Out[14]: 0.075

In [15]: blob.sentiment.subjectivity
Out[15]: 0.833
```

Получение эмоциональной окраски предложения

Также можно получить данные эмоциональной окраски на уровне отдельных предложений. Воспользуемся свойством `sentence` для получения списка объектов `Sequence`[1], а затем переберем их и выведем свойство `sentiment` каждого объекта `Sentence`:

```
In [16]: for sentence in blob.sentences:
    ...:     print(sentence.sentiment)
    ...:
Sentiment(polarity=0.85, subjectivity=1.0)
Sentiment(polarity=-0.6999999999999998, subjectivity=0.6666666666666666)
```

[1] http://textblob.readthedocs.io/en/latest/api_reference.html#textblob.blob.Sentence.

Это может объяснить, почему эмоциональная окраска `TextBlob` близка к `0.0` (нейтральная) — одно предложение имеет положительную окраску (`0.85`), а другое отрицательную (`-0.6999999999999998`).

11.2.6. Анализ эмоциональной окраски с использованием NaiveBayesAnalyzer

По умолчанию объект `TextBlob` и объекты `Sentence` и `Word`, получаемые от него, определяют эмоциональную окраску при помощи объекта `PatternAnalyzer`, который использует те же методы анализа эмоциональной окраски, что и библиотека Pattern. Библиотека TextBlob также содержит объект `NaiveBayesAnalyzer`[1] (модуль `text-blob.sentiments`), прошедший обучение на базе данных рецензий о фильмах. Наивный байесовский классификатор[2] — часто применяемый алгоритм классификации текста с машинным обучением. Следующий пример использует ключевой аргумент `analyzer` для назначения анализатора эмоциональной окраски `TextBlob`. Напомним, что в текущем сеансе IPython `text` содержит `'Today is a beautiful day. Tomorrow looks like bad weather.'`:

```
In [17]: from textblob.sentiments import NaiveBayesAnalyzer

In [18]: blob = TextBlob(text, analyzer=NaiveBayesAnalyzer())

In [19]: blob
Out[19]: TextBlob("Today is a beautiful day. Tomorrow looks like bad
weather.")
```

Воспользуемся свойством `sentiment` объекта `TextBlob`, чтобы вывести данные эмоциональной окраски текста с использованием `NaiveBayesAnalyzer`:

```
In [20]: blob.sentiment
Out[20]: Sentiment(classification='neg', p_pos=0.47662917962091056,
p_neg=0.5233708203790892)
```

В данном случае общая эмоциональная окраска классифицируется как отрицательная (`classification='neg'`.) Свойство `p_pos` объекта `Sentiment` показывает, что текст `TextBlob` положителен на 47,66%, а свойство `p_neg` показывает, что текст `TextBlob` отрицателен на 52,34%. Так как общая эмоциональная окраска всего лишь немногим более отрицательна, эмоциональная окраска `TextBlob` может рассматриваться как в целом нейтральная.

---

[1] https://textblob.readthedocs.io/en/latest/api_reference.html#module-textblob.en.sentiments.

[2] https://ru.wikipedia.org/wiki/Наивный_байесовский_классификатор.

А теперь получим данные об эмоциональной окраске каждого объекта `Sentence`:

```
In [21]: for sentence in blob.sentences:
    ...:     print(sentence.sentiment)
    ...:
Sentiment(classification='pos', p_pos=0.8117563121751951,
p_neg=0.18824368782480477)
Sentiment(classification='neg', p_pos=0.174363226578349,
p_neg=0.8256367734216521)
```

Вместо значений `polarity` и `subjectivity` объекты `Sentiment`, получаемые от `NaiveBayesAnalyzer`, содержат *классификацию* (`'pos'` (положительная) или `'neg'` (отрицательная)) и значения `p_pos` (положительный процент) и `p_neg` (отрицательный процент) в диапазоне от `0.0` до `1.0`. И снова мы видим, что первое предложение имеет положительную, а второе — отрицательную эмоциональную окраску.

11.2.7. Распознавание языка и перевод

Перевод на другой язык — довольно сложная задача из области обработки естественного языка и искусственного интеллекта. Благодаря новейшим достижениям в области машинного обучения, искусственного интеллекта и обработки естественных языков такие сервисы, как Google Translate (100+ языков) и Microsoft Bing Translator (60+ языков), могут мгновенно переводить тексты на другие языки.

Автоматический перевод также очень удобен для людей, посещающих зарубежные страны. Они могут использовать приложение-переводчик для перевода меню, дорожных знаков и т. д. Также ведутся исследования в области перевода речи, чтобы вы могли общаться в реальном времени с людьми, которые не знают ваш основной язык[1,2]. Некоторые современные смартфоны в сочетании с наушниками-вкладышами обеспечивают практически мгновенный перевод со многих языков[3,4,5]. В главе 13 будет разработан сценарий, который практически в реальном времени осуществляет перевод на другие языки из числа поддерживаемых Watson.

---

1 https://www.skype.com/en/features/skype-translator/.

2 https://www.microsoft.com/en-us/translator/business/live/.

3 https://www.telegraph.co.uk/technology/2017/10/04/googles-new-headphones-can-translate-foreign-languages-real/.

4 https://store.google.com/us/product/google_pixel_buds?hl=en-US.

5 http://www.chicagotribune.com/bluesky/originals/ct-bsi-google-pixel-buds-review-20171115-story.html.

Библиотека TextBlob использует сервис Google Translate для распознавания языка текста и перевода объектов `TextBlob`, `Sentence` и `Word` на другие языки[1]. Воспользуемся *методом* `detect_language` для распознавания языка текста (`'en'` — английский):

```
In [22]: blob
Out[22]: TextBlob("Today is a beautiful day. Tomorrow looks like bad
weather.")

In [23]: blob.detect_language()
Out[23]: 'en'
```

А теперь воспользуемся *методом* `translate` для перевода текста на испанский язык (`'es'`) с последующим распознаванием языка результата. Ключевой аргумент `to` задает целевой язык.

```
In [24]: spanish = blob.translate(to='es')

In [25]: spanish
Out[25]: TextBlob("Hoy es un hermoso dia. Mañana parece mal tiempo.")

In [26]: spanish.detect_language()
Out[26]: 'es'
```

На следующем шаге переведем текст `TextBlob` на упрощенный китайский (`'zh'` или `'zh-CN'`) с последующим распознаванием языка результата:

```
In [27]: chinese = blob.translate(to='zh')

In [28]: chinese
Out[28]: TextBlob("今天是美好的一天。明天看起来像恶劣的天气。")

In [29]: chinese.detect_language()
Out[29]: 'zh-CN'
```

В выводе `detect_language` упрощенный китайский всегда обозначается `'zh-CN'`, несмотря на то что функция `translate` может получать упрощенный китайский в виде `'zh'` или `'zh-CN'`.

В каждом из предыдущих случаев Google Translate автоматически распознает исходный язык. Вы можете явно задать исходный язык, передав ключевой аргумент `from_lang` методу `translate`:

```
chinese = blob.translate(from_lang='en', to='zh')
```

[1] Для этого необходимо подключение к интернету.

Google Translate использует коды языков ISO-639-1[1]:

https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

Для поддерживаемых языков эти коды можно использовать в значениях аргументов `from_lang` и `to`. Список поддерживаемых языков Google Translate доступен по адресу:

https://cloud.google.com/translate/docs/languages

При вызове `translate` без аргументов осуществляется перевод с распознанного исходного языка на английский:

```
In [30]: spanish.translate()
Out[30]: TextBlob("Today is a beautiful day. Tomorrow seems like bad
weather.")

In [31]: chinese.translate()
Out[31]: TextBlob("Today is a beautiful day. Tomorrow looks like bad
weather.")
```

Обратите внимание на небольшие отличия в английских результатах.

11.2.8. Формообразование: образование единственного и множественного числа

У одного слова может быть несколько разных форм — например, единственное или множественное число («person» и «people») или разное время глагола («run» и «ran»). Возможно, при вычислении частот слов в тексте вы захотите сначала преобразовать все формы слов к одной форме для получения более точных данных частот. `Word` и `WordList` поддерживают преобразование слов к форме единственного или множественного числа. Попробуем выполнить это преобразование с парой объектов `Word`:

```
In [1]: from textblob import Word

In [2]: index = Word('index')

In [3]: index.pluralize()
Out[3]: 'indices'

In [4]: cacti = Word('cacti')

In [5]: cacti.singularize()
Out[5]: 'cactus'
```

[1] ISO — Международная организация по стандартизации (https://www.iso.org/).

Образование формы множественного и единственного числа — сложные задачи, которые, как было показано ранее, не сводятся к простому добавлению или удалению «s» или «es» в конце слова.

То же самое можно сделать с `WordList`:

```
In [6]: from textblob import TextBlob

In [7]: animals = TextBlob('dog cat fish bird').words

In [8]: animals.pluralize()
Out[8]: WordList(['dogs', 'cats', 'fish', 'birds'])
```

11.2.9. Проверка орфографии и исправление ошибок

Для задач обработки естественного языка очень важно, чтобы текст был свободен от орфографических ошибок. Программные пакеты для ввода и редактирования текста, такие как Microsoft Word, Google Docs и им подобные, автоматически проверяют орфографию во время ввода текста и обычно подчеркивают неправильные слова красной волнистой линией. Другие инструменты позволяют выполнить проверку орфографии вручную.

Вы можете проверить орфографию текста в `Word` *методом* `spellcheck` этого объекта. Этот метод возвращает список кортежей, содержащих возможные варианты написания слова и уровень достоверности. Предположим, вы хотели написать слово «they», но случайно ввели его в виде «theyr». Результаты проверки предлагают два возможных исправления, при этом вариант `'they'` имеет наивысший уровень достоверности:

```
In [1]: from textblob import Word

In [2]: word = Word('theyr')

In [3]: %precision 2
Out[3]: '%.2f'

In [4]: word.spellcheck()
Out[4]: [('they', 0.57), ('their', 0.43)]
```

Следует учитывать, что слово с наивысшим уровнем достоверности может и не быть правильным словом для заданного контекста.

Объекты `TextBlob`, `Sentence` и `Word` содержат *метод* `correct`, который можно вызвать для исправления ошибки. Вызов `correct` для `Word` возвращает пра-

вильно написанное слово с наибольшим уровнем достоверности (по данным проверки орфографии):

```
In [5]: word.correct()  # Выбирает слово с наибольшим уровнем достоверности
Out[5]: 'they'
```

Вызов `correct` для объекта `TextBlob` или `Sentence` проверяет орфографию каждого слова. Для каждого неправильного слова `correct` заменяет его правильно написанным вариантом с наибольшим уровнем достоверности:

```
In [6]: from textblob import Word

In [7]: sentence = TextBlob('Ths sentense has missplled wrds.')

In [8]: sentence.correct()
Out[8]: TextBlob("The sentence has misspelled words.")
```

11.2.10. Нормализация: выделение основы и лемматизация

В результате *выделения основы* из слова удаляется префикс или суффикс и остается только основа, которая может быть реальным словом (но может и не быть). *Лемматизация* выполняется аналогичным образом, но ее результатом является осмысленная часть речи, то есть реальное слово.

Выделение основы и лемматизация относятся к операциям *нормализации*, готовящим слова для анализа. Например, перед вычислением статистики вхождения слов в корпусе текста все слова могут быть преобразованы к нижнему регистру, чтобы слова, начинающиеся с букв нижнего и верхнего регистра, обрабатывались одинаково. Иногда для представления разных форм слова приходится использовать только корень. Например, в некотором приложении все следующие слова могут рассматриваться как слово «program»: program, programs, programmer, programming и programmed (и, возможно, английские варианты написания — скажем, programmes).

У объектов `Word` и `WordList` для выделения основы и лемматизации используются методы `stem` и `lemmatize`. Попробуем использовать их с `Word`:

```
In [1]: from textblob import Word

In [2]: word = Word('varieties')

In [3]: word.stem()
Out[3]: 'varieti'
```

```
In [4]: word.lemmatize()
Out[4]: 'variety'
```

11.2.11. Частоты слов

Различные методы выявления сходства между документами основаны на частотах вхождения слов. Как вы вскоре узнаете, `TextBlob` подсчитывает частоты автоматически. Начнем с загрузки электронного текста пьесы Шекспира «Ромео и Джульетта» в `TextBlob`. Для этого будет использоваться *класс* `Path` из *модуля* `pathlib` стандартной библиотеки Python:

```
In [1]: from pathlib import Path

In [2]: from textblob import TextBlob

In [3]: blob = TextBlob(Path('RomeoAndJuliet.txt').read_text())
```

Используйте загруженный ранее файл RomeoAndJuliet.txt[1]. Предполагается, что вы запустили сеанс IPython из этой папки. Когда вы читаете файл *методом* `read_text` объекта `Path`, файл будет немедленно закрыт после завершения чтения файла.

Частоты вхождения слов в тексте `TextBlob` хранятся в *словаре* `word_counts`. Подсчитаем вхождения некоторых слов в пьесе:

```
In [4]: blob.word_counts['juliet']
Out[4]: 190

In [5]: blob.word_counts['romeo']
Out[5]: 315

In [6]: blob.word_counts['thou']
Out[6]: 278
```

Если вы уже разобрали `TextBlob` в список `WordList`, то для подсчета вхождений конкретных слов в список можно воспользоваться *методом* `count`:

```
In [7]: blob.words.count('joy')
Out[7]: 14

In [8]: blob.noun_phrases.count('lady capulet')
Out[8]: 46
```

[1] Каждая электронная книга проекта «Гутенберг» включает дополнительный текст (в частности, лицензионную информацию), которая не является частью самой книги. В данном примере мы воспользовались текстовым редактором для удаления текста из нашей копии электронной книги.

11.2.12. Получение определений, синонимов и антонимов из WordNet

WordNet[1] – база данных слов, созданная в Принстонском университете. Библиотека TextBlob использует интерфейс WordNet библиотеки NLTK, позволяющий искать определения слов, синонимы и антонимы. За дополнительной информацией обращайтесь к документации интерфейса NLTK WordNet по адресу:

https://www.nltk.org/api/nltk.corpus.reader.html#module-nltk.corpus.reader.wordnet

Получение определений

Начнем с создания объекта `Word`:

```
In [1]: from textblob import Word

In [2]: happy = Word('happy')
```

Свойство `definitions` класса `Word` возвращает список всех определений слова в базе данных WordNet:

```
In [3]: happy.definitions
Out[3]:
['enjoying or showing or marked by joy or pleasure',
 'marked by good fortune',
 'eagerly disposed to act or to be of service',
 'well expressed and to the point']
```

База данных необязательно содержит все возможные словарные определения для заданного слова. Также имеется *метод* `define`, который позволяет передать в аргументе часть речи, с тем чтобы вы могли получить определения, соответствующие только этой части слова.

Получение синонимов

Наборы синонимов `Word` доступны в *свойстве* `synsets`. Результат представляет собой список объектов `Synset`:

```
In [4]: happy.synsets
Out[4]:
[Synset('happy.a.01'),
```

[1] https://wordnet.princeton.edu/.

```
Synset('felicitous.s.02'),
Synset('glad.s.02'),
Synset('happy.s.04')]
```

Каждый объект `Synset` представляет группу синонимов. В записи `happy.a.01`:

- `happy` — лемматизированная форма исходного объекта `Word` (в данном случае они совпадают).
- `a` — часть речи: `a` — прилагательное, `n` — существительное, `v` — глагол, `r` — наречие, `s` — прилагательное-сателлит. Многие наборы синонимов прилагательных в WordNet имеют сателлитные наборы синонимов, представляющие похожие прилагательные.
- `01` — индекс, начинающийся с 0. Многие слова обладают несколькими смыслами; значение является индексом соответствующего смысла в базе данных WordNet.

Также имеется *метод* `get_synsets`, который позволяет передать часть речи в аргументе, чтобы получить набор `Synset` только для указанной части речи.

Вы можете перебрать список `synsets`, чтобы найти синонимы исходного слова. Каждый объект `Synset` содержит *метод* `lemmas`, который возвращает список объектов `Lemma`, представляющих синонимы. Метод `name` объекта `Lemma` возвращает слово-синоним в виде строки. В следующем коде для каждого объекта `Synset` в списке `synsets` вложенный цикл `for` перебирает объекты `Lemma` из этого объекта `Synset` (при наличии). Затем синоним добавляется в множество с именем `synonyms`. Мы использовали множество, потому что оно автоматически устраняет все добавленные в него дубликаты:

```
In [5]: synonyms = set()

In [6]: for synset in happy.synsets:
   ...:     for lemma in synset.lemmas():
   ...:         synonyms.add(lemma.name())
   ...:

In [7]: synonyms
Out[7]: {'felicitous', 'glad', 'happy', 'well-chosen'}
```

Получение антонимов

Если слово, представленное объектом `Lemma`, имеет антонимы в базе данных WordNet, то вызов метода `antonyms` объекта `Lemma` возвращает список объектов `Lemma`, представляющих антонимы (или пустой список, если в базе данных

нет ни одного антонима). Во фрагменте [4] показано, что для `'happy'` было найдено четыре объекта `Synset`. Сначала найдем объекты `Lemma` для `Synset` с индексом 0 в списке `synsets`:

```
In [8]: lemmas = happy.synsets[0].lemmas()

In [9]: lemmas
Out[9]: [Lemma('happy.a.01.happy')]
```

В данном случае `lemmas` возвращает список из одного элемента `Lemma`. Мы можем проверить, содержит ли база данных какие-либо антонимы для этого объекта `Lemma`:

```
In [10]: lemmas[0].antonyms()
Out[10]: [Lemma('unhappy.a.01.unhappy')]
```

Результат является списком объектов `Lemma`, представляющих антоним (-ы). В данном случае мы видим, что база данных содержит для слова `'happy'` один антоним `'unhappy'`.

11.2.13. Удаление игнорируемых слов

Игнорируемые слова (стоп-слова) — часто встречающиеся в тексте слова, которые часто удаляются из текста перед анализом, поскольку обычно не несут полезной информации. Ниже приведен список игнорируемых слов английского языка из NLTK, возвращаемый функцией `words`[1] модуля `stopwords` (которой мы вскоре воспользуемся на практике):

Список игнорируемых слов английского языка из NLTK

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an',
'and', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be', 'because', 'been',
'before', 'being', 'below', 'between', 'both', 'but', 'by', 'can', 'couldn',
"couldn't", 'd', 'did', 'didn', "didn't", 'do', 'does', 'doesn', "doesn't",
'doing', 'don', "don't", 'down', 'during', 'each', 'few', 'for', 'from',
'further', 'had', 'hadn', "hadn't", 'has', 'hasn', "hasn't", 'have', 'haven',
"haven't", 'having', 'he', 'her', 'here', 'hers', 'herself', 'him', 'himself',
'his', 'how', 'i', 'if', 'in', 'into', 'is', 'isn', "isn't", 'it', "it's", 'its',
'itself', 'just', 'll', 'm', 'ma', 'me', 'mightn', "mightn't", 'more', 'most',
'mustn', "mustn't", 'my', 'myself', 'needn', "needn't", 'no', 'nor', 'not',
```

[1] https://www.nltk.org/book/ch02.html.

```
'now', 'o', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'our', 'ours',
'ourselves', 'out', 'over', 'own', 're', 's', 'same', 'shan', "shan't", 'she',
"she's", 'should', "should've", 'shouldn', "shouldn't", 'so', 'some', 'such',
't', 'than', 'that', "that'll", 'the', 'their', 'theirs', 'them', 'themselves',
'then', 'there', 'these', 'they', 'this', 'those', 'through', 'to', 'too',
'under', 'until', 'up', 've', 'very', 'was', 'wasn', "wasn't", 'we', 'were',
'weren', "weren't", 'what', 'when', 'where', 'which', 'while', 'who', 'whom',
'why', 'will', 'with', 'won', "won't", 'wouldn', "wouldn't", 'y', 'you', "you'd",
"you'll", "you're", "you've", 'your', 'yours', 'yourself', 'yourselves']
```

Библиотека NLTK также содержит списки игнорируемых слов для ряда других естественных языков. Прежде чем использовать списки игнорируемых слов NLTK, их необходимо загрузить при помощи *функции* `download` модуля `nltk`:

```
In [1]: import nltk

In [2]: nltk.download('stopwords')
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\PaulDeitel\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
Out[2]: True
```

В этом примере будет загружен список игнорируемых слов английского языка `'english'`. Импортируйте `stopwords` из модуля `nltk.corpus`, а затем используйте метод `words` класса `stopwords` для загрузки списка игнорируемых слов `'english'`:

```
In [3]: from nltk.corpus import stopwords

In [4]: stops = stopwords.words('english')
```

Затем создадим объект `TextBlob`, из которого будут удаляться игнорируемые слова:

```
In [5]: from textblob import TextBlob

In [6]: blob = TextBlob('Today is a beautiful day.')
```

Наконец, чтобы удалить игнорируемые слова, используйте слова `TextBlob` в трансформации списка, которая добавляет каждое слово в полученный список только в том случае, если слово не входит в `stops`:

```
In [7]: [word for word in blob.words if word not in stops]
Out[7]: ['Today', 'beautiful', 'day']
```

11.2.14. n-граммы

N-грамма[1] представляет собой последовательность из *n* текстовых элементов, например букв в словах или слов в предложении. В обработке естественных языков *n*-граммы могут использоваться для выявления букв или слов, часто располагающихся рядом друг с другом. Для текстового ввода это поможет предсказать следующую букву или слово, вводимое пользователем, например при завершении элементов в IPython по нажатии клавиши Tab или при вводе текстового сообщения в вашем любимом месседжере для смартфона. При преобразовании речи в текст *n*-граммы могут использоваться для повышения качества текста. N-граммы представляют собой форму *лексической солидарности*, то есть расположения букв или слов рядом друг с другом.

Метод `ngrams` объекта `TextBlob` выдает список `WordList` *n*-грамм, которые по умолчанию имеют длину 3 (они называются *триграммами*). Передавая ключевой аргумент `n`, вы сможете получать *n*-граммы любой длины по своему усмотрению. Вывод показывает, что первая триграмма содержит первые три слова предложения (`'Today'`, `'is'` и `'a'`.) Затем `ngrams` создает триграмму, начинающуюся со второго слова (`'is'`, `'a'` и `'beautiful'`), и т. д. — до тех пор, пока не будет создана триграмма с тремя последними словами `TextBlob`:

```
In [1]: from textblob import TextBlob

In [2]: text = 'Today is a beautiful day. Tomorrow looks like bad weather.'

In [3]: blob = TextBlob(text)

In [4]: blob.ngrams()
Out[4]:
[WordList(['Today', 'is', 'a']),
 WordList(['is', 'a', 'beautiful']),
 WordList(['a', 'beautiful', 'day']),
 WordList(['beautiful', 'day', 'Tomorrow']),
 WordList(['day', 'Tomorrow', 'looks']),
 WordList(['Tomorrow', 'looks', 'like']),
 WordList(['looks', 'like', 'bad']),
 WordList(['like', 'bad', 'weather'])]
```

Следующий пример строит *n*-граммы из пяти слов:

```
In [5]: blob.ngrams(n=5)
Out[5]:
[WordList(['Today', 'is', 'a', 'beautiful', 'day']),
```

[1] https://en.wikipedia.org/wiki/N-gram.

```
WordList(['is', 'a', 'beautiful', 'day', 'Tomorrow']),
WordList(['a', 'beautiful', 'day', 'Tomorrow', 'looks']),
WordList(['beautiful', 'day', 'Tomorrow', 'looks', 'like']),
WordList(['day', 'Tomorrow', 'looks', 'like', 'bad']),
WordList(['Tomorrow', 'looks', 'like', 'bad', 'weather'])]
```

11.3. Визуализация частот вхождения слов с использованием гистограмм и словарных облаков

Ранее мы получили частоты вхождения некоторых слов в пьесе «Ромео и Джульетта». В некоторых случаях визуализации частот слов повышают качество анализа текстового корпуса. Часто существует несколько возможных вариантов визуализации данных, и один из них оказывается лучше других. Например, вас могут интересовать частоты вхождения слов друг относительно друга или же данные относительного использования слов в корпусе. В этом разделе будут рассмотрены два способа наглядного представления частот слов:

- Гистограмма для наглядного *количественного* представления частот самых частых слов в пьесе «Ромео и Джульетта» в виде столбцов разной длины.
- Словарное облако для наглядного *качественного* представления частот: более частые слова выводятся более крупным шрифтом, а более редкие слова — шрифтом меньшего размера.

11.3.1. Визуализация частот вхождения слов средствами Pandas

Построим наглядное представление частот в тексте «Ромео и Джульетты» для 20 самых частых слов, которые *не являются* игнорируемыми словами. Для этого воспользуемся функциональностью TextBlob, NLTK и Pandas. Средства визуализации Pandas базируются на Matplotlib, поэтому в этом сеансе IPython следует запустить следующей командой:

```
ipython --matplotlib
```

Загрузка данных

Начнем с загрузки текста «Ромео и Джульетты». Прежде чем выполнять следующий код, запустите IPython из каталога ch11, чтобы вы могли обратиться к файлу электронной книги RomeoAndJuliet.txt, загруженному ранее в этой главе:

```
In [1]: from pathlib import Path

In [2]: from textblob import TextBlob

In [3]: blob = TextBlob(Path('RomeoAndJuliet.txt').read_text())
```

Затем загрузите список игнорируемых слов NLTK:

```
In [4]: from nltk.corpus import stopwords

In [5]: stop_words = stopwords.words('english')
```

Получение частот слов

Чтобы построить визуализацию частот 20 слов, необходимо знать все слова и их частоты. Вызовем метод `items` словаря `blob.word_counts`, чтобы получить список кортежей «слово-частота»:

```
In [6]: items = blob.word_counts.items()
```

Исключение игнорируемых слов

Воспользуемся трансформацией списка для удаления кортежей, содержащих стоп-слова:

```
In [7]: items = [item for item in items if item[0] not in stop_words]
```

Выражение `item[0]` получает слово из каждого кортежа, чтобы проверить, входит ли оно в список `stop_words`.

Сортировка слов по частотам

Чтобы определить 20 наиболее частых слов, отсортируем кортежи в `items` по убыванию частоты. Мы можем воспользоваться встроенной функцией `sorted` с аргументом `key` для сортировки кортежей по элементу частоты в каждом кортеже. Чтобы задать элемент кортежа для выполнения сортировки, используйте *функцию* `itemgetter` из *модуля* `operator` стандартной библиотеки Python:

```
In [8]: from operator import itemgetter

In [9]: sorted_items = sorted(items, key=itemgetter(1), reverse=True)
```

В процессе упорядочения элементов `items` функция `sorted` обращается к элементу с индексом `1` в каждом кортеже с использованием выражения `itemgetter(1)`. Ключевой аргумент `reverse=True` означает, что кортежи должны сортироваться *по убыванию*.

Получение 20 самых частых слов

Затем сегмент используется для получения 20 самых частых слов из `sorted_items`. Когда объект `TextBlob` проводит разбиение корпуса на лексемы, он разбивает все сокращенные формы по апострофам и подсчитывает общее количество апострофов как одно из «слов». Текст «Ромео и Джульетты» содержит множество сокращенных форм. Если вывести `sorted_items[0]`, вы увидите, что это самое часто встречающееся «слово» с 867 вхождениями[1]. Выводиться должны только слова, поэтому мы игнорируем элемент 0 и получаем сегмент с элементами с 1 по 20 списка `sorted_items`:

```
In [10]: top20 = sorted_items[1:21]
```

Преобразование top20 в DataFrame

Затем преобразуем список кортежей `top20` в коллекцию `DataFrame` библиотеки Pandas для удобства его представления в визуальном виде:

```
In [11]: import pandas as pd

In [12]: df = pd.DataFrame(top20, columns=['word', 'count'])

In [13]: df
Out[13]:
        word  count
0      romeo    315
1       thou    278
2     juliet    190
3        thy    170
4    capulet    163
5      nurse    149
6       love    148
7       thee    138
8       lady    117
9      shall    110
10     friar    105
11      come     94
12  mercutio     88
13  lawrence     82
14      good     80
15  benvolio     79
16    tybalt     79
17     enter     75
18        go     75
19     night     73
```

[1] В некоторых локальных контекстах этого не происходит, и элементом 0 становится слово `'romeo'`.

Визуализация DataFrame

Чтобы построить визуализацию данных, используем *метод* `bar` *свойства* `plot` коллекции `DataFrame`. Аргументы указывают, какие данные столбца должны выводиться по осям x и y, и что на диаграмме не должны выводиться условные обозначения:

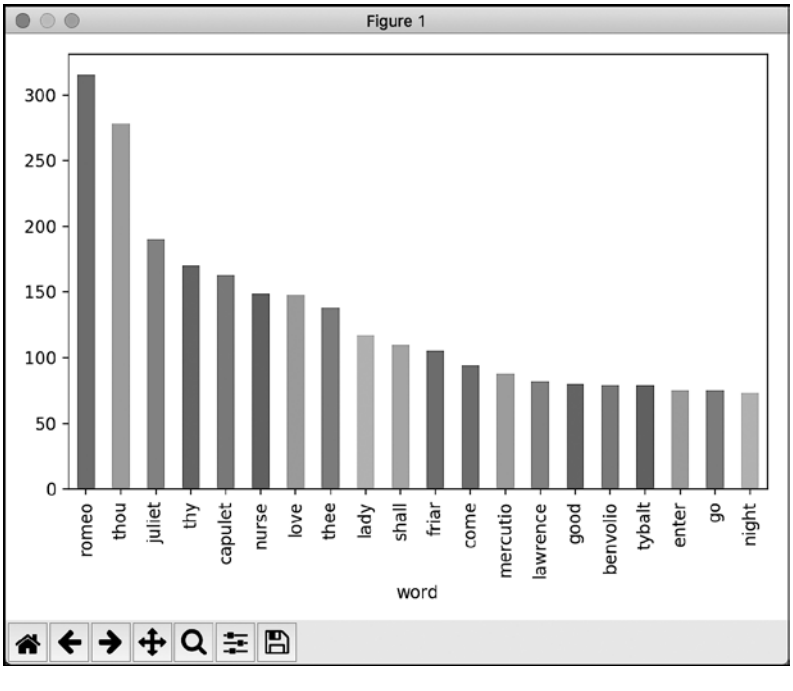
```
In [14]: axes = df.plot.bar(x='word', y='count', legend=False)
```

Метод `bar` создает и выводит гистограмму Matplotlib.

Взглянув на исходную гистограмму, которая появляется на экране, можно заметить, что некоторые из ее столбцов усечены. Чтобы решить эту проблему, используйте функцию `gcf` (Get Current Figure) для получения рисунка Matlpotlib, выведенного pandas, а затем вызовите метод `tight_layout`. Вызов «сжимает» гистограмму, чтобы все ее компоненты поместились в окне:

```
In [15]: import matplotlib.pyplot as plt

In [16]: plt.gcf().tight_layout()
```

Окончательный вид гистограммы показан ниже:



11.3.2. Визуализация частот слов в словарных облаках

Теперь построим словарное облако для визуализации наиболее частых 200 слов в тексте «Ромео и Джульетты». Мы можем воспользоваться для этого *классом* `WordCloud` *модуля* `wordcloud` [1] с открытым кодом для генерирования словарных облаков всего в нескольких строках кода. По умолчанию `wordcloud` создает прямоугольные словарные облака, но, как вы вскоре увидите, библиотека также может создавать словарные облака произвольной формы.

Установка модуля wordcloud

Чтобы установить `wordcloud`, откройте приглашение Anaconda (Windows), терминал (macOS/Linux) или командную оболочку (Linux), после чего выполните следующую команду:

```
conda install -c conda-forge wordcloud
```

Возможно, пользователям Windows придется запустить приглашение Anaconda с правами администратора для получения необходимых привилегий при установке программного обеспечения. Щелкните правой кнопкой мыши на команде Anaconda Prompt в меню Пуск и выберите команду More ▸ Run as administrator.

Загрузка текста

Сначала нужно загрузить текст пьесы «Ромео и Джульетты». Прежде чем выполнять следующий код, запустите IPython из каталога ch11, чтобы вы могли обратиться к файлу электронной книги RomeoAndJuliet.txt, загруженному ранее в этой главе:

```
In [1]: from pathlib import Path

In [2]: text = Path('RomeoAndJuliet.txt').read_text()
```

Загрузка маски, определяющей форму словарного облака

Чтобы создать словарное облако нужной формы, нужно инициализировать объект `WordCloud` изображением, которое называется *маской*. `WordCloud` за-

[1] https://github.com/amueller/word_cloud.

полняет текстом области изображения маски, цвет которых отличен от белого. В нашем примере будет использована форма сердца, определяемая файлом mask_heart.png из каталога примеров ch11. Более сложным маскам для создания словарного облака требуется больше времени.

Загрузим изображение маски *функцией* `imread` из модуля `imageio`, включенного в поставку Anaconda:

```
In [3]: import imageio

In [4]: mask_image = imageio.imread('mask_heart.png')
```

Функция возвращает изображение в виде коллекции `array` библиотеки NumPy, как того требует `WordCloud`.

Настройка объекта WordCloud

Затем создадим и настроим объект `WordCloud`:

```
In [5]: from wordcloud import WordCloud

In [6]: wordcloud = WordCloud(colormap='prism', mask=mask_image,
   ...:     background_color='white')
   ...:
```

По умолчанию ширина и высота объекта `WordCloud` составляет 400 × 200 пикселов, если только другие значения не заданы ключевыми аргументами `width` и `height` или маской. При использовании маски размер `WordCloud` определяется размером изображения. `WordCloud` использует Matplotlib во внутренней реализации. WordCloud назначает словам случайные цвета с цветовой карты. Вы можете передать ключевой аргумент `colormap` и воспользоваться одной из именованных цветовых карт Matplotlib. Список имен цветовых карт и их цветов доступен по адресу:

https://matplotlib.org/examples/color/colormaps_reference.html

Ключевой аргумент `mask` задает загруженное ранее изображение `mask_image`. По умолчанию слова выводятся на черном фоне, но мы изменили цвет фона при помощи ключевого аргумента `background_color` и выбрали белый цвет (`'white'`). За полным списком ключевых аргументов `WordCloud` обращайтесь по адресу:

http://amueller.github.io/word_cloud/generated/wordcloud.WordCloud.html

Генерирование словарного облака

Метод `generate` объекта `WordCloud` получает используемый текст в аргументе и строит словарное облако, которое возвращается в виде объекта `WordCloud`:

```
In [7]: wordcloud = wordcloud.generate(text)
```

Прежде чем строить словарное облако, `generate` сначала удаляет игнорируемые слова из аргумента `text` по встроенному списку игнорируемых слов модуля `wordcloud`. Затем `generate` вычисляет частоты для оставшихся слов. По умолчанию метод использует не более 200 слов для словарного облака, но это количество можно изменить при помощи ключевого аргумента `max_words`.

Сохранение словарного облака в файле

Наконец, воспользуемся *методом* `to_file` объекта `WordCloud` для сохранения изображения словарного облака в заданном файле:

```
In [8]: wordcloud = wordcloud.to_file('RomeoAndJulietHeart.png')
```

Откройте каталог примеров `ch11` в своей системе и сделайте двойной щелчок на файле RomeoAndJuliet.png, чтобы просмотреть изображение, — вероятно, в вашей версии слова будут располагаться в других позициях и выводиться другим цветом:

Генерирование словарного облака по словарю

Если у вас уже имеется словарь с парами «ключ-значение», представляющими счетчики слов, то их можно передать *методу* `fit_words` объекта `WordCloud`. Этот метод предполагает, что игнорируемые слова уже были удалены ранее.

Вывод изображения средствами Matplotlib

Для вывода изображения на экран используйте магическую команду IPython

```
%matplotlib
```

чтобы включить интерактивную поддержку Matplotlib в IPython, после чего выполните следующие команды:

```
import matplotlib.pyplot as plt
plt.imshow(wordcloud)
```

11.4. Оценка удобочитаемости с использованием Textatistic

Интересным применением обработки естественного языка является оценка *удобочитаемости* текста, которая зависит от используемого словаря, структуры предложений, длины предложений, темы и многих других факторов. При написании этой книги мы воспользовались платным инструментом Grammarly, чтобы улучшить текст и добиться того, чтобы он нормально воспринимался широкой аудиторией.

В этом разделе для оценки удобочитаемости будет использоваться *библиотека Textatistic*[1,2]. Существует много разных формул, используемых при обработке естественного языка для вычисления индекса удобочитаемости.

Установка Textatistic

Чтобы установить Textatistic, откройте приглашение Anaconda (Windows), терминал (macOS/Linux) или командную оболочку (Linux), после чего выполните команду:

```
pip install textatistic
```

---

[1] https://github.com/erinhengel/Textatistic.

[2] Другие библиотеки оценки удобочитаемости для Python — readability-score, textstat, readability и pylinguistics.

Возможно, пользователям Windows придется запустить приглашение Anaconda с правами администратора для получения необходимых привилегий при установке программного обеспечения. Щелкните правой кнопкой мыши на команде Anaconda Prompt в меню Пуск и выберите команду More ▸ Run as administrator.

Вычисление статистики и индексов удобочитаемости

Загрузим текст «Ромео и Джульетты» в переменную `text`:

```
In [1]: from pathlib import Path

In [2]: text = Path('RomeoAndJuliet.txt').read_text()
```

Для вычисления статистики и индексов удобочитаемости потребуется объект `Textatistic`, инициализированный текстом, который вы хотите обработать:

```
In [3]: from textatistic import Textatistic

In [4]: readability = Textatistic(text)
```

Метод `dict` объекта `Textatistic` возвращает словарь с различными статистическими характеристиками и индексами удобочитаемости[1]:

```
In [5]: %precision 3
Out[5]: '%.3f'

In [6]: readability.dict()
Out[6]:
{'char_count': 115141,
 'word_count': 26120,
 'sent_count': 3218,
 'sybl_count': 30166,
 'notdalechall_count': 5823,
 'polysyblword_count': 549,
 'flesch_score': 100.892,
 'fleschkincaid_score': 1.203,
 'gunningfog_score': 4.087,
 'smog_score': 5.489,
 'dalechall_score': 7.559}
```

[1] Каждая электронная книга проекта «Гутенберг» включает дополнительный текст (в частности, лицензионную информацию), которая не является частью самой книги. В данном примере мы воспользовались текстовым редактором для удаления текста из нашей копии электронной книги.

Каждое значение в словаре также доступно через свойство `Textatistic` с именем, совпадающим с ключом в показанном выводе. Вот некоторые статистические показатели:

- `char_count` — количество символов в тексте.
- `word_count` — количество слов в тексте.
- `sent_count` — количество предложений в тексте.
- `sybl_count` — количество слогов в тексте.
- `notdalechall_count` — количество слов, не входящих в список Дейла — Челла (список слов, понятных 80 % пятиклассников[1]). Чем выше это число по сравнению с общим количеством слов, тем менее понятным считается текст.
- `polysyblword_count` — количество слов из трех и более слогов.
- `flesch_score` — индекс удобочитаемости Флеша, который может быть связан с уровнем образования. Считается, что тексты со значением индекса более 90 понятны пятиклассникам. При значениях ниже 30 для усвоения текста обычно требуется образование на уровне колледжа. Промежуточные диапазоны соответствуют другим образовательным уровням.
- `fleschkincaid_score` — индекс Флеша — Кинкейда, соответствующий уровню образования.
- `gunningfog_score` — индекс туманности Ганнинга, соответствующий уровню образования.
- `smog_score` — индекс SMOG (Simple Measure of Gobbledygook), соответствующий годам образования, необходимым для понимания текста. Эта метрика считается особенно эффективной для материалов по здравоохранению[2].
- `dalechall_score` — индекс Дейла — Челла, который может быть связан с уровнями образования от 4 и ниже до выпускника колледжа (уровень 16) и выше. Этот индекс считается наиболее надежным для широкого диапазона типов текста[3,4].

---

[1] http://www.readabilityformulas.com/articles/dale-chall-readability-word-list.php.

[2] https://en.wikipedia.org/wiki/SMOG.

[3] https://en.wikipedia.org/wiki/Readability#The_Dale%E2%80%93Chall_formula.

[4] http://www.readabilityformulas.com/articles/how-do-i-decide-which-readability-formula-to-use.php.

За дополнительной информацией о каждом индексе удобочитаемости, представленном здесь, а также нескольких других обращайтесь по адресу:

https://en.wikipedia.org/wiki/Readability

В документации Textatistic приведены и использованные формулы индексов удобочитаемости:

http://www.erinhengel.com/software/textatistic/

11.5. Распознавание именованных сущностей с использованием spaCy

NLP может определить, о чем говорится в тексте. Ключевой аспект этого процесса — *распознавание именованных сущностей* в целях поиска и классификации таких элементов, как даты, время, количества, места, имена людей, названия предметов, организаций и т. д. В этом разделе для анализа текста используются средства распознавания именованных сущностей из *NLP-библиотеки spaCy*[1,2].

Установка spaCy

Для установки spaCy откройте приглашение Anaconda (Windows), терминал (macOS/Linux) или командную оболочку (Linux), после чего выполните команду:

```
conda install -c conda-forge spacy
```

Возможно, пользователям Windows придется запустить приглашение Anaconda с правами администратора для получения необходимых привилегий для установки программного обеспечения. Щелкните правой кнопкой мыши на команде Anaconda Prompt в меню Пуск и выберите команду More ▸ Run as administrator.

Чтобы библиотека spaCy могла загрузить дополнительные необходимые компоненты для обработки английского (en) текста, по завершении установки необходимо выполнить команду,:

```
python -m spacy download en
```

---

[1] https://spacy.io/.

[2] Также стоит обратить внимание на Textacy (https://github.com/chartbeat-labs/textacy) — библиотеку NLP, построенную на базе spaCy и поддерживающую другие операции NLP.

Загрузка языковой модели

Первый шаг использования spaCy — загрузка языковой модели, представляющей естественный язык анализируемого текста. Для этого следует вызвать *функцию* `load` модуля `spacy`. Загрузим ранее примененную модель английского языка:

```
In [1]: import spacy

In [2]: nlp = spacy.load('en')
```

Документация spaCy рекомендует использовать имя переменной `nlp`.

Создание объекта spaCy Doc

Затем объект `nlp` используется для создания объекта spaCy `Doc` [1], представляющего обрабатываемый документ. В данном случае используется предложение из вводного описания Всемирной паутины, встречающегося во многих наших книгах:

```
In [3]: document = nlp('In 1994, Tim Berners-Lee founded the ' +
   ...:     'World Wide Web Consortium (W3C), devoted to ' +
   ...:     'developing web technologies')
   ...:
```

Получение именованных сущностей

Свойство `ents` объекта `Doc` возвращает кортеж объектов `Span`, представляющих именованные сущности, встречающиеся в `Doc`. Каждый объект `Span` содержит многочисленные свойства[2]. Переберем объекты `Span` и выведем свойства `text` и `label_`:

```
In [4]: for entity in document.ents:
   ...:     print(f'{entity.text}: {entity.label_}')
   ...:
1994: DATE
Tim Berners-Lee: PERSON
the World Wide Web Consortium: ORG
```

Свойство `text` каждого объекта `Span` возвращает сущность в виде строки, а *свойство* `label_` возвращает строку, обозначающую классификацию сущ-

[1] https://spacy.io/api/doc.

[2] https://spacy.io/api/span.

ности. В данном случае spaCy обнаруживает три сущности, представляющие дату (`DATE — 1994`), человека (`PERSON — Tim Berners-Lee`) и организацию (`ORG — World Wide Web Consortium`). За дополнительной информацией о spaCy и кратким руководством по Quickstart обращайтесь по адресу:

https://spacy.io/usage/models#section-quickstart

11.6. Выявление сходства средствами spaCy

Выявление сходства — процесс анализа документов для определения того, насколько они похожи друг на друга. Один из возможных методов выявления сходства — подсчет частот слов. Так, некоторые специалисты считают, что работы Уильяма Шекспира в действительности могли быть написаны сэром Фрэнсисом Бэконом, Кристофером Марло или другими людьми[1]. Сравнение частот слов в их работах с частотами слов в работах Шекспира может открыть сходство в авторском стиле.

Для анализа сходства документов могут применяться различные методы машинного обучения, которые будут рассмотрены в дальнейших главах. Тем не менее, как это часто бывает в Python, некоторые библиотеки, такие как spaCy и Gensim, могут сделать это за вас. Мы воспользуемся средствами выявления сходства spaCy для сравнения объектов `Doc`, представляющих пьесу Шекспира «Ромео и Джульетта», с работой Кристофера Марло «Эдуард II». Текст «Эдуарда II» можно загрузить с сайта проекта «Гутенберг» так же, как мы это сделали ранее для «Ромео и Джульетты»[2].

Загрузка языковой модели и создание объекта spaCy Doc

Как и в предыдущем разделе, начнем с загрузки языковой модели для английского языка:

```
In [1]: import spacy

In [2]: nlp = spacy.load('en')
```

[1] https://en.wikipedia.org/wiki/Shakespeare_authorship_question.

[2] Каждая электронная книга проекта «Гутенберг» включает дополнительный текст (в частности, лицензионную информацию), которая не является частью самой книги. В данном примере мы воспользовались текстовым редактором для удаления текста из нашей копии электронной книги.

Создание объекта spaCy Doc

Затем создаются два объекта `Doc` — для «*Ромео и Джульетты*» и для «*Эдуарда II*»:

```
In [3]: from pathlib import Path

In [4]: document1 = nlp(Path('RomeoAndJuliet.txt').read_text())

In [5]: document2 = nlp(Path('EdwardTheSecond.txt').read_text())
```

Сравнение сходства книг

Наконец, *метод* `similarity` класса `Doc` используется для получения значения в диапазоне от `0.0` (сходство отсутствует) до `1.0` (полная идентичность), которое показывает, насколько похожи документы:

```
In [6]: document1.similarity(document2)
Out[6]: 0.9349950179100041
```

spaCy считает, что два документа обладают высокой степенью сходства. Для чистоты эксперимента мы создали объект `Doc` для текста новостной заметки и сравнили его с «Ромео и Джульеттой». Как и предполагалось, spaCy возвращает низкое значение, свидетельствующее о незначительном сходстве между ними. Попробуйте скопировать текст заметки в текстовый файл и провести сравнение самостоятельно.

11.7. Другие библиотеки и инструменты NLP

В этой главе были представлены различные библиотеки NLP, но вам всегда стоит самостоятельно исследовать возможные варианты, чтобы выбрать оптимальный инструмент для ваших задач. Ниже перечислены некоторые (в основном бесплатные и распространяемые с открытым кодом) библиотеки NLP и API:

- Gensim — выявление сходства и тематическое моделирование.
- Google Cloud Natural Language API — облачный API для задач NLP (распознавание именованных сущностей, анализ эмоциональной окраски, анализ частей речи и визуализация, определение категорий контента и т. д.).
- Microsoft Linguistic Analysis API.

- Анализ эмоциональной окраски Bing — поисковая система Bing компании Microsoft использует эмоциональную окраску при выборе результатов поиска (на момент написания книги эта возможность была доступна только в США).
- PyTorch NLP — библиотека глубокого обучения для NLP.
- Stanford CoreNLP — NLP-библиотека для языка Java, которая также предоставляет обертку для Python. Включает разрешение кореференции, то есть поиск всех ссылок на один ресурс.
- Apache OpenNLP — другая NLP-библиотека на базе Java для выполнения распространенных операций, включая разрешение кореференции. Доступны обертки для Python.
- PyNLPl — NLP-библиотека для языка Python; включает как базовую, так и более сложную функциональность NLP.
- SnowNLP — библиотека Python, упрощающая обработку текста на китайском языке.
- KoNLPy — NLP для корейского языка.
- stop-words — библиотека Python с игнорируемыми словами для многих языков. В этой главе использовались списки игнорируемых слов NLTK.
- TextRazor — коммерческий облачный NLP API с бесплатным уровнем.

11.8. Машинное обучение и NLP-приложения с глубоким обучением

Существует множество областей обработки естественного языка, требующих применения машинного обучения и методов глубокого обучения. Некоторые из них рассмотрены в главах, посвященных машинному обучению и глубокому обучению:

- Ответы на вопросы на естественном языке — например, издатель Pearson Education использует IBM Watson в качестве виртуального преподавателя. Студенты задают Watson вопросы на естественном языке и получают ответы.
- Составление сводки документа — анализ документа и построение краткой сводки (также называемой конспектом), которая, например, может вклю-

чаться в результаты поиска, чтобы упростить пользователю выбор материал для чтения.

- Синтез речи (текст в речь) и распознавание речи (речь в текст) — эти средства будут использоваться в главе 13 наряду с переводом текста в текст на другом языке, для разработки голосового переводчика, функционирующего практически в реальном времени.
- Совместная фильтрация — используется для реализации рекомендательных систем («если вам понравился этот фильм, то вам также может понравиться...»).
- Классификация текста — например, классификация новостных заметок по категориям: мировые новости, национальные новости, местные новости, спорт, бизнес, развлечения и т. д.
- Тематическое моделирование — определение тем, обсуждаемых в документах.
- Распознавание сарказма — часто используется в сочетании с анализом эмоциональной окраски.
- Упрощение текста — преобразование текста, которое делает его более компактным и простым для чтения.
- Преобразование речи на язык жестов и наоборот — для общения со слабослышащими людьми.
- Технология чтения по губам — преобразование движения губ в текст или речь для общения с людьми, которые не могут говорить.
- Формирование субтитров — включение субтитров в видео.

11.9. Наборы данных естественных языков

Разработчикам доступно огромное количество источников данных для обработки естественных языков:

- «Википедия» — https://meta.wikimedia.org/wiki/Datasets.
- IMDB (Internet Movie Database) — различные наборы данных с информацией о фильмах и ТВ-шоу.
- Текстовые наборы данных UCIs — множество наборов данных, включая Spambase.

- Проект «Гутенберг» — 50 000+ бесплатных электронных книг, не защищенных авторским правом в США.
- Набор данных Jeopardy! — 200 000+ вопросов телевикторины Jeopardy! Одной из важных вех в развитии искусственного интеллекта стала победа IBM Watson над двумя сильнейшими игроками Jeopardy! в 2011 году.
- Наборы данных для обработки естественных языков: https://machinelearningmastery.com/datasets-natural-language-processing/.
- Данные NLTK: https://www.nltk.org/data.html.
- Набор данных предложений с эмоциональной разметкой (из различных источников, включая IMDB.com, amazon.com, yelp.com).
- Реестр открытых данных AWS — каталог наборов данных, размещенных в Amazon Web Services (https://registry.opendata.aws).
- Набор данных с отзывами клиентов Amazon — 130+ миллионов отзывов о продуктах (https://registry.opendata.aws/amazon-reviews/).
- Корпус Pitt.edu (http://mpqa.cs.pitt.edu/corpora/).

11.10. Итоги

В этой главе были рассмотрены разнообразные операции обработки естественного языка (NLP) с использованием нескольких библиотек NLP, продемонстрировано использование NLP с текстовыми коллекциями, которые называются корпусами. Мы обсудили, почему смысловые нюансы затрудняют понимание естественных языков.

Основное внимание было уделено NLP-библиотеке TextBlob, которая построена на базе библиотек NLTK и `pattern`, но более проста в использовании. Мы создали объекты `TextBlob` и разбили их на объекты `Sentence` и `Word`, определили части речи каждого слова в `TextBlob` и выделили в тексте именные конструкции.

Далее показано, как оценить положительную или отрицательную эмоциональную окраску `TextBlob` и `Sentence` при помощи стандартного анализатора `TextBlob` и при помощи анализатора `NaiveBayesAnalyzer`. Средства интеграции библиотеки TextBlob с Google Translate были использованы для выявления языка текста и перевода текста на другой язык.

Также было продемонстрировано выполнение других операций NLP, включая образование множественного и единственного числа, проверку орфографии

и исправление ошибок, нормализацию с выделением основы и лемматизацией, а также определение частот вхождения слов. Определения слов, синонимы и антонимы были загружены из WordNet. Также список игнорируемых слов NLTK был использован для устранения этих слов из текста, и мы создали n-граммы, содержащие группы последовательных слов.

Далее вы узнали, как создать количественную визуализацию частот вхождения слов в виде гистограммы с использованием встроенных средств создания диаграмм pandas. Затем библиотека `wordcloud` была использована для качественного представления частот вхождения слов в виде словарных облаков. Для оценки удобочитаемости текста использовалась библиотека Textatistic. Наконец, библиотека spaCy использовалась для поиска именованных сущностей и выявления сходства между документами. В следующей главе мы продолжим использование обработки естественных языков при глубоком анализе твитов средствами Twitter API.

12

Глубокий анализ данных Twitter

В этой главе...

- Влияние Twitter на бизнес, бренды, репутацию, анализ эмоциональной окраски, прогнозы и т. д.
- Использование Tweepy, одного из самых популярных Python-клиентов Twitter API, для глубокого анализа данных Twitter.
- Использование Twitter Search API для загрузки прошлых твитов, удовлетворяющих заданному критерию.
- Использование Twitter Streaming API для работы с потоком твитов по мере их появления.
- Полезная информация в объектах твитов, возвращаемых Twitter.
- Использование методов обработки естественных языков из предыдущей главы с целью очистки и предварительной обработки твитов для их подготовки к анализу.
- Выполнение анализа эмоциональной окраски твитов.
- Выявление тенденций с использованием Twitter Trends API.
- Отображение твитов с использованием folium и OpenStreetMap.
- Различные способы хранения твитов.

12.1. Введение

Мы часто стремимся предугадать будущее. Пойдет ли дождь в день предстоящего пикника? Будут ли подниматься или снижаться биржевые индексы или котировки отдельных ценных бумаг, когда и насколько? Как люди будут голосовать на следующих выборах? С какой вероятностью экспедиция откроет новое месторождение и сколько нефти оно сможет произвести? Будет ли бейсбольная команда чаще выигрывать при переходе на новую тактику? Сколько клиентов воспользуется услугами авиакомпании за ближайшие месяцы? По какой траектории пойдет ураган и какую силу он наберет: категория 1, 2, 3, 4 или 5? Согласитесь, такая информация жизненно важна для подготовленности к чрезвычайным ситуациям. Или, скажем, с какой вероятностью та или иная финансовая операция может оказаться мошеннической? Будет ли ипотечный кредит погашен вовремя? Насколько вероятно быстрое распространение болезни, и если вероятно, то в какой географической области следует ожидать вспышки? И так далее и тому подобное.

Прогнозирование — сложный и нередко дорогостоящий процесс, но потенциальный выигрыш может быть очень большим. Благодаря технологиям этой и следующих глав вы увидите, как искусственный интеллект — часто в сочетании с большими данными — стремительно расширяет возможности прогнозирования.

В этой главе мы сосредоточимся на глубоком анализе данных Twitter и определении эмоциональной окраски твитов. *Глубоким анализом данных* называется процесс поиска в больших коллекциях данных (часто больших данных) с целью выявления закономерностей, которые могут представлять интерес для отдельных лиц и организаций. Информация об эмоциональной окраске, извлеченная из твитов, поможет спрогнозировать результаты выборов, вероятные доходы от нового фильма или успех рекламной кампании, выявить слабости в продуктах, предлагаемых конкурентами, и многое другое.

Установление связи с Twitter осуществляется через веб-сервисы. Мы воспользуемся Twitter Search API для подключения к гигантской базе данных прошлых твитов. Twitter Streaming API будет использоваться для выборки информации из потока новых твитов по мере их появления. Twitter Trends API поможет узнать наиболее актуальные темы. Вы увидите, что большая часть информации, представленной в главе 11, пригодится для построения приложений для Twitter.

Как уже показано в книге, благодаря мощным библиотекам достаточно серьезные задачи часто решаются всего в нескольких строках кода, и это — одна из самых привлекательных сторон языка Python и его сообщества разработки с открытым кодом.

Twitter постепенно вытесняет крупные информационные агентства и занимает место основного источника достоверных новостей. Многие публикации в Twitter общедоступны и происходят в реальном времени одновременно с глобальными событиями. Люди откровенно высказываются по любым темам и пишут о своей личной и деловой жизни, оставляя комментарии по социальным, развлекательным, политическим темам и любым иным мыслимым вопросам. Они снимают на свои мобильные телефоны происходящие вокруг события. Часто встречающиеся термины «*Twitter-сфера*» и «*мир Twitter*» обозначают сотни миллионов пользователей, имеющих отношение к отправке, получению и анализу твитов.

Что такое Twitter?

Компания *Twitter* основана в 2006 году как сервис микроблогов; сейчас она стала одним из самых популярных сайтов в интернете. Концепция проста: люди пишут короткие сообщения — *твиты* (изначально их длина была ограничена 140 символами, но теперь для многих языков она увеличилась до 280 символов). Каждый желающий может читать сообщения любого другого пользователя. В этом отношении Twitter отличается от замкнутых сообществ других социальных сетей вроде Facebook, LinkedIn и т. д., в которых «отношения подписки» должны быть взаимными.

Статистика Twitter

У Twitter сотни миллионов пользователей. Ежедневно отправляются сотни миллионов твитов, по несколько тысяч в секунду[1]. Поиск в интернете по критериям «интернет-статистика» и «статистика Twitter» поможет объективно оценить эти числа. Некоторые авторы имеют более 100 миллионов подписчиков. Обычно они публикуют по несколько сообщений в день, чтобы подписчики не теряли интереса. Как правило, наибольшее количество подписчиков имеют артисты и политики. Разработчики могут получать информацию из «живого» потока твитов прямо во время его появления. Из-за невероятной скорости

[1] http://www.internetlivestats.com/twitter-statistics/.

появления твитов этот способ получения информации иногда сравнивают с «питьем из пожарного шланга».

Twitter и большие данные

Twitter стал излюбленным источником больших данных для исследователей и бизнесменов по всему миру, предоставляя обычным пользователям бесплатный доступ к небольшой части самых последних твитов. Посредством специального соглашения с Twitter некоторые сторонние компании (и сама компания Twitter) предоставляют платный доступ к намного большей базе данных твитов за все время.

Предупреждение

Нельзя доверять всему, что вы читаете в интернете, и твиты в этом смысле не являются исключением. Например, ложная информация может использоваться для махинаций с финансовыми рынками или влияния на политические выборы. Хеджевые фонды часто выполняют операции с ценными бумагами с учетом потоков твитов, на которые они подписаны, действуя, впрочем, осторожно. Заметим, это — одна из проблем построения *особо важных* или *критических* систем, основанных на контенте из социальных сетей.

В своей работе мы широко используем веб-сервисы. Подключения к интернету могут теряться, сервисы могут изменяться, а некоторые сервисы могут быть недоступны в некоторых странах. Таков реальный мир облачного программирования: при использовании веб-сервисов мы не можем программировать с такой же надежностью, как при разработке настольных приложений.

12.2. Обзор Twitter APIs

Twitter-API реализуются в форме облачных веб-сервисов, поэтому для выполнения кода этой главы потребуется подключение к интернету. *Веб-сервисы* представляют собой методы, которые вызываются в облаке, как это будет делаться с Twitter-API в этой главе, с IBM Watson API — в следующей главе, а также с другими API, которыми вы будете пользоваться при переносе вычислений на облачные платформы. Каждый метод API имеет *конечную точку* веб-сервиса, представленную URL-адресом для вызова метода по интернету.

Различные Twitter-API включают разные категории функциональности; одни из них бесплатны, другие требуют оплаты. У многих предусмотрены *ограни-*

чения частоты использования, то есть максимальное количество возможных обращений за 15-минутный интервал. В этой главе *библиотека Tweepy* будет использоваться для вызова методов из следующих Twitter API:

- Authentication API — передача своих регистрационных данных Twitter (см. далее) для использования других API.
- Accounts and Users API — обращение к информации учетных записей.
- Tweets-API — поиск по старым твитам, обращение к потокам текущих твитов и т. д.
- Trends-API — поиск актуальных тем и получение списков актуальных тем по местоположению.

Полный список категорий, подкатегорий и отдельных методов Twitter-API — здесь:

https://developer.twitter.com/en/docs/api-reference-index.html

Ограничения частоты использования: предупреждение

Twitter ожидает, что разработчики будут ответственно пользоваться его сервисами. У каждого метода Twitter API существует *ограничение частоты использования* — максимальное количество запросов (то есть вызовов), которые могут быть выданы за 15-минутное окно. Если вы продолжите вызвать метод API после превышения ограничения частоты использования этого метода, Twitter может заблокировать вам доступ к API.

До использования метода API прочитайте документацию и поймите его ограничения частоты использования[1]. Мы настроим Tweepy так, чтобы при достижении ограничения библиотека переходила в ожидание, чтобы предотвратить возможное превышение ограничения частоты. В некоторых списках указываются ограничения частоты использования как для пользователей, так и для приложений. Во всех примерах этой главы используются *ограничения частоты для приложений*. Они предназначены для приложений, позволяющих отдельному пользователю подключиться к Twitter (например, сторонним приложениям, взаимодействующим с Twitter от вашего имени, скажем, приложениям на смартфонах). За подробностями об ограничениях частоты использования обращайтесь по адресу:

https://developer.twitter.com/en/docs/basics/rate-limiting

[1] Учтите, что в будущем Twitter может изменить эти ограничения.

За информацией об ограничениях частоты использования конкретных методов API обращайтесь по адресу:

https://developer.twitter.com/en/docs/basics/rate-limits

и к документации методов API.

Другие ограничения

Twitter — настоящая золотая жила для глубокого анализа данных, и с помощью бесплатных сервисов можно сделать очень много. Вы не поверите, насколько полезные приложения можно построить на этой основе и насколько они способны улучшить ваши личные и карьерные достижения. **Тем не менее если вы будете нарушать правила Twitter, то ваша учетная запись разработчика может быть заблокирована. Внимательно ознакомьтесь со следующими текстами и теми документами, на которые они ссылаются:**

- Условия обслуживания: https://twitter.com/tos
- Соглашение с разработчиком: https://developer.twitter.com/en/developer-terms/agreement-and-policy.html
- Политика для разработчиков: https://developer.twitter.com/en/developer-terms/policy.html
- Другие ограничения: https://developer.twitter.com/en/developer-terms/more-on-restricted-use-cases

Позднее в этой главе будет показано, что бесплатный Twitter-API позволяет проводить поиск только по твитам за последние семь дней с получением ограниченного количества твитов. Некоторые книги и статьи сообщают, что эти ограничения можно обойти прямым извлечением данных с twitter.com. Тем не менее в условиях обслуживания явно указано, что «**извлечение данных с сервисов без предварительного согласования с Twitter категорически запрещается**».

12.3. Создание учетной записи Twitter

Twitter требует подачи заявок на создание учетной записи разработчика для использования своих API. Откройте страницу

https://developer.twitter.com/en/apply-for-access

и отправьте свое приложение. Если у вас еще нет учетной записи Twitter, нужно зарегистрироваться в ходе этого процесса. Вам придется ответить на вопросы о предназначении этого приложения. Вы должны *тщательно* прочитать условия Twitter и согласиться на них, а затем подтвердить свой адрес электронной почты.

Twitter проверяет все приложения, а получение разрешения не гарантировано. На момент написания книги разрешения для *персональных учетных записей* предоставлялись немедленно. По информации с форумов разработчиков Twitter, для учетных записей компаний процесс мог занимать от нескольких дней до нескольких недель.

12.4. Получение регистрационных данных Twitter — создание приложения

Когда у вас появится учетная запись разработчика Twitter, вы должны получить *регистрационные данные* для взаимодействия с Twitter API. Для этого необходимо создать *приложение*; разные приложения используют разные регистрационные данные. Чтобы создать приложение, выполните вход по адресу:

https://developer.twitter.com

и следующие действия:

1. Щелкните на раскрывающемся меню своей учетной записи в правой верхней части страницы и выберите пункт Apps.
2. Щелкните на ссылке Create an app.
3. В поле App name укажите имя своего приложения. Если вы *отправляете* твиты через API, то имя вашего приложения будет использоваться в качестве отправителя твитов. Оно также будет выводиться для пользователей, если вы создаете приложение, требующее входа пользователя через Twitter. В этой главе мы не будем делать ни того ни другого, поэтому для ее целей будет достаточно имени вида «*ВашеИмя* Test App».
4. В поле Application description введите описание приложения. При создании приложений на базе Twitter, которые будут использоваться другими людьми, оно будет содержать информацию о том, что делает ваше приложение. В этой главе будет использоваться строка "Learning to use the Twitter API.".

5. В поле Website URL введите URL-адрес своего сайта. При создании приложений на базе Twitter это должен быть веб-сайт, на котором размещается ваше приложение. Вы можете использовать URL-адрес Twitter: https://twitter.com/ВашеИмяПользователя, где *ВашеИмяПользователя* — имя вашей учетной записи Twitter. Например, адрес https://twitter.com/nasa соответствует имени NASA `@nasa`.
6. В поле Tell us how this app will be used вводится описание, содержащее не менее 100 символов. Оно поможет работникам Twitter понять, что делает ваше приложение. Мы ввели строку с информацией о том, что приложение предназначено исключительно для учебных целей (`"I am new to Twitter app development and am simply learning how to use the Twitter APIs for educational purposes"`).
7. Оставьте остальные поля пустыми и щелкните на кнопке Create, а затем внимательно просмотрите (длинные) условия использования сервиса для разработчиков, после чего снова щелкните на кнопке Create.

Получение регистрационных данных

После выполнения *шага 7* в предыдущем списке Twitter выводит веб-страницу для управления приложением. В начале страницы располагаются вкладки App details (Подробное описание приложения), Keys and tokens (Ключи и маркеры) и Permissions (Разрешения). Щелкните на кнопке вкладки Keys and tokens, чтобы просмотреть регистрационные данные вашего приложения. Изначально на странице выводятся ключи *Consumer API — ключ API* и *секретный ключ API*. Щелкните на кнопке Create, чтобы получить *маркер доступа* и *секретный маркер доступа*. Все четыре ключа будут использоваться для выполнения аутентификации Twitter с целью использования методов API.

Сохранение регистрационных данных

Не включайте ключи API и маркеры доступа (как и другую информацию регистрационных данных, например имена пользователей и пароли) в исходный код, так как при этом они будут открыты каждому, кто читает ваш код. Всегда храните свои ключи в отдельном файле и никогда никому не передавайте этот файл[1]. Код, который будет выполняться в последующих разделах, предполагает, что значения ключа пользователя, секретного ключа пользователя,

[1] Рекомендуется зашифровать ключи, маркеры доступа и другие регистрационные данные, которые будут использоваться в вашем коде, при помощи библиотеки шифрования

маркера доступа и секретного маркера доступа хранятся в приведенном ниже файле keys.py. Вы найдете этот файл в каталоге примеров ch12:

```
consumer_key='ВашКлючПользователя'
consumer_secret='ВашСекретныйКлючПользователя'
access_token='ВашМаркерДоступа'
access_token_secret='ВашСекретныйМаркерДоступа'
```

Отредактируйте файл и замените *ВашКлючПользователя*, *ВашСекретный КлючПользователя*, *ВашМаркерДоступа* и *ВашСекретныйМаркерДоступа* соответствующими значениями. Сохраните полученный файл.

OAuth 2.0

Ключ пользователя, секретный ключ пользователя, маркер доступа и секретный маркер доступа являются частью процесса аутентификации *OAuth 2.0*[1,2], используемого Twitter для обращения к API. Библиотека Tweepy позволяет вам передать ключ пользователя, секретный ключ пользователя, маркер доступа и секретный маркер доступа, принимая на себя все детали аутентификации OAuth 2.0.

12.5. Какую информацию содержит объект Tweet?

Методы Twitter API возвращают объекты JSON. *JSON* (*JavaScript Object Notation*) — текстовый формат передачи данных, используемый для представления объектов в виде коллекций пар «имя-значение». Он часто применяется при обращениях к веб-сервисам. Формат JSON может читаться как человеком, так и компьютером и позволяет легко отправлять и получать данные по интернету.

Объекты JSON сходны со словарями Python. Каждый объект JSON содержит список *имен свойств* и их *значений*, заключенный в фигурные скобки:

```
{имяСвойства1: значение1, имяСвойства2: значение2}
```

(например, bcrypt — https://github.com/pyca/bcrypt/), а затем читать и расшифровывать их только при передаче Twitter.

[1] https://developer.twitter.com/en/docs/basics/authentication/overview.

[2] https://oauth.net/.

Как и в Python, списки JSON — это значения, разделенные запятыми в квадратных скобках:

```
[значение1, значение2, значение3]
```

Для вашего удобства Tweepy берет на себя все операции с JSON, преобразуя JSON в объекты Python с использованием классов, определенных в библиотеке Tweepy.

Ключевые свойства объекта Tweet

Твит (также называемый *обновлением статуса*) может содержать не более 280 символов, но объекты, возвращаемые Twitter API, содержат множество атрибутов *метаданных*, описывающих разные аспекты твита:

- когда он был создан;
- кто его создал;
- списки хештегов, URL, @-упоминаний и аудиовизуальные материалы (например, изображения и видео, заданные URL-адресами), входящие в твит;
- и многое другое.

В табл. 12.1 перечислены некоторые ключевые атрибуты объекта твита.

Таблица 12.1. Некоторые ключевые атрибуты объекта твита

Атрибут	Описание
`created_at`	Дата и время создания в формате UTC (Всемирное координированное время)
`entities`	Twitter извлекает из твитов хештеги, URL, @-упоминания, ссылки на аудиовизуальные материалы (графика, видео), условные обозначения и опросы и помещает их в словарь entities в форме списков, чтобы вы могли обращаться к ним по соответствующим ключам
`extended_tweet`	Дополнительная информация для твитов, длина которых превышает 140 символов (например, `full_text` и `entities`)
`favorite_count`	Счетчик помещений твита в «Избранное» другими пользователями
`coordinates`	Координаты (широта и долгота) отправки твита. Часто содержит null (None в Python), потому что многие пользователи отключают отправку данных своего местоположения

Атрибут	Описание
`place`	Пользователи могут связать с твитом определенное место. В таком случае этот атрибут будет содержать объект place: https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/geo-objects#place-dictionary; в противном случае он равен null (None в Python)
`id`	Целочисленный идентификатор твита. Twitter рекомедует использовать `id_str` для обеспечения портируемости
`id_str`	Строковое представление целочисленного идентификатора твита
`lang`	Язык твита (например, `'en'` для английского или `'fr'` для французского)
`retweet_count`	Количество ретвитов данного твита другими пользователями
`text`	Текст твита. Если твит использует новое 280-символьное ограничение длины и содержит более 140 символов, это свойство будет усечено, а свойство `truncated` будет равно *true*. Также это может произойти в том случае, если в результате ретвита 140-символьный твит превысил длину в 140 символов
`user`	Объект User, представляющий пользователя, отправившего твит. За списком JSON-свойств объекта User обращайтесь по адресу: https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/user-object

Пример объекта твита в формате JSON

Рассмотрим пример JSON для следующего твита от учетной записи `@nasa`:

```
@NoFear1075 Great question, Anthony! Throughout its seven-year mis-
sion, our Parker #SolarProbe spacecraft... https://t.co/xKd6ym8waT'
```

Мы добавили нумерацию строк и переформатировали часть разметки JSON для переноса строк. Отметим, что не все поля JSON-разметки поддерживаются всеми методами Twitter-API; различия объясняются в электронной документации каждого метода.

```
1 {'created_at': 'Wed Sep 05 18:19:34 +0000 2018',
2  'id': 1037404890354606082,
3  'id_str': '1037404890354606082',
4  'text': '@NoFear1075 Great question, Anthony! Throughout its seven-year
         mission, our Parker #SolarProbe spacecraft… https://t.co/xKd6ym8waT',
5  'truncated': True,
6  'entities': {'hashtags': [{'text': 'SolarProbe', 'indices': [84, 95]}],
7     'symbols': [],
```

```
    'user_mentions': [{'screen_name': 'NoFear1075',
        'name': 'Anthony Perrone',
        'id': 284339791,
        'id_str': '284339791',
        'indices': [0, 11]}],
    'urls': [{'url': 'https://t.co/xKd6ym8waT',
        'expanded_url': 'https://twitter.com/i/web/status/
         1037404890354606082',
        'display_url': 'twitter.com/i/web/status/1…',
        'indices': [117, 140]}]},
'source': '<a href="http://twitter.com" rel="nofollow">Twitter Web
      Client</a>',
'in_reply_to_status_id': 1037390542424956928,
'in_reply_to_status_id_str': '1037390542424956928',
'in_reply_to_user_id': 284339791,
'in_reply_to_user_id_str': '284339791',
'in_reply_to_screen_name': 'NoFear1075',
'user': {'id': 11348282,
   'id_str': '11348282',
   'name': 'NASA',
   'screen_name': 'NASA',
   'location': '',
   'description': 'Explore the universe and discover our home planet with
        @NASA. We usually post in EST (UTC-5)',
   'url': 'https://t.co/TcEE6NS8nD',
   'entities': {'url': {'urls': [{'url': 'https://t.co/TcEE6NS8nD',
           'expanded_url': 'http://www.nasa.gov',
           'display_url': 'nasa.gov',
           'indices': [0, 23]}]},
    'description': {'urls': []}},
   'protected': False,
   'followers_count': 29486081,
   'friends_count': 287,
   'listed_count': 91928,
   'created_at': 'Wed Dec 19 20:20:32 +0000 2007',
   'favourites_count': 3963,
   'time_zone': None,
   'geo_enabled': False,
   'verified': True,
   'statuses_count': 53147,
   'lang': 'en',
   'contributors_enabled': False,
   'is_translator': False,
   'is_translation_enabled': False,
   'profile_background_color': '000000',
   'profile_background_image_url': 'http://abs.twimg.com/images/themes/
        theme1/bg.png',
   'profile_background_image_url_https': 'https://abs.twimg.com/images/
        themes/theme1/bg.png',
```

```
52     'profile_image_url': 'http://pbs.twimg.com/profile_images/188302352/
           nasalogo_twitter_normal.jpg',
53     'profile_image_url_https': 'https://pbs.twimg.com/profile_images/
           188302352/nasalogo_twitter_normal.jpg',
54     'profile_banner_url': 'https://pbs.twimg.com/profile_banners/11348282/
            1535145490',
55     'profile_link_color': '205BA7',
56     'profile_sidebar_border_color': '000000',
57     'profile_sidebar_fill_color': 'F3F2F2',
58     'profile_text_color': '000000',
59     'profile_use_background_image': True,
60     'has_extended_profile': True,
61     'default_profile': False,
62     'default_profile_image': False,
63     'following': True,
64     'follow_request_sent': False,
65     'notifications': False,
66     'translator_type': 'regular'},
67  'geo': None,
68  'coordinates': None,
69  'place': None,
70  'contributors': None,
71  'is_quote_status': False,
72  'retweet_count': 7,
73  'favorite_count': 19,
74  'favorited': False,
75  'retweeted': False,
76  'possibly_sensitive': False,
77  'lang': 'en'}
```

Ресурсы с информацией по JSON-объектам Twitter

Полный и более удобочитаемый список атрибутов объекта твита доступен по адресу:

https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.html

Дополнительная информация об изменениях, произошедших при переходе Twitter на новое ограничение длины твита (с 140 на 280 символов), здесь:

https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json.html#extendedtweet

Общий обзор всех объектов JSON, возвращаемых разными Twitter-API, а также ссылки на информацию о конкретных объектах, доступны по адресу:

https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json

12.6. Tweepy

Мы будем использовать библиотеку Tweepy[1] (http://www.tweepy.org/) — одну из самых популярных библиотек Python для взаимодействия с разными Twitter-API. Tweepy упрощает доступ к функциональности Twitter, скрывая подробности обработки объектов JSON, возвращаемых Twitter-API. Документацию Tweepy[2] можно просмотреть по адресу:

http://docs.tweepy.org/en/latest/

За дополнительной информацией и исходным кодом Tweepy обращайтесь по адресу:

https://github.com/tweepy/tweepy

Установка Tweepy

Для установки Tweepy откройте приглашение Anaconda (Windows), терминал (macOS/Linux) или командную оболочку (Linux), после чего выполните команду:

```
pip install tweepy==3.7
```

Возможно, пользователям Windows придется запустить приглашение Anaconda с правами администратора для получения необходимых привилегий для установки программного обеспечения. Щелкните правой кнопкой мыши на команде Anaconda Prompt в меню Пуск и выберите команду More ▸ Run as administrator.

Установка geopy

В процессе работы с Tweepy вы также будете использовать функции из файла tweetutilities.py (файл включен в примеры кода этой главы). Одна из вспомо-

---

[1] Среди других библиотек Python, рекомендуемых Twitter, — Birdy, python-twitter, Python Twitter Tools, TweetPony, TwitterAPI, twitter-gobject, TwitterSearch и twython. За подробностями обращайтесь по адресу https://developer.twitter.com/en/docs/developer-utilities/twitter-libraries.html.

[2] Работа над документацией Tweepy еще не завершена. На момент написания книги у Tweepy еще не было документации по классам, соответствующим объектам JSON, возвращаемым Twitter-API. Классы Tweepy используют те же имена атрибутов и структуру, что и объекты JSON. Правильные имена атрибутов можно определить по документации JSON компании Twitter. Мы будем пояснять все атрибуты, используемые нами в коде, предоставив ссылки на описания JSON-объектов Twitter.

гательных функций файла зависит от *библиотеки* `geopy` (https://github.com/geopy/geopy, см. раздел 12.15). Чтобы установить geopy, выполните следующую команду:

```
conda install -c conda-forge geopy
```

12.7. Аутентификация Twitter с использованием Tweepy

В нескольких ближайших разделах мы будем обращаться к различным облачным Twitter-API через Tweepy. Здесь мы начнем использовать Tweepy для выполнения аутентификации Twitter и создания `объекта Tweepy API`, который становится шлюзом для использования Twitter API по интернету. Далее мы будем работать с разными Twitter API, вызывая методы объекта `API`.

До вызова методов Twitter-API необходимо использовать ключ API, секретный ключ API, маркер доступа и секретный маркер доступа для выполнения аутентификации Twitter[1]. Запустите IPython из папки примеров ch12, затем импортируйте *модуль* `tweepy` и файл keys.py, который был изменен ранее в этой главе. Вы можете импортировать любой файл .py в виде модуля, для чего имя файла указывается *без* расширения .py в команде `import`:

```
In [1]: import tweepy

In [2]: import keys
```

Если файл keys.py был импортирован как модуль, то можно обращаться к каждой из четырех переменных, определенных в этом файле, в форме `keys`. *имя_переменной*.

Создание и настройка OAuthHandler для выполнения аутентификации Twitter

Аутентификация Twitter с использованием Tweepy состоит из двух этапов. Создайте объект *класса* `OAuthHandler` модуля `tweepy`, передав свой ключ API

[1] Возможно, вы займетесь созданием приложений, которые позволяют пользователям входить в учетные записи Twitter, управлять ими, отправлять твиты, читать твиты других пользователей, искать твиты и т. д. Дополнительная информация об аутентификации пользователей доступна в учебном руководстве по адресу http://docs.tweepy.org/en/latest/auth_tutorial.html.

и секретный ключ API его конструктору. *Конструктором* называется функция, имя которой совпадает с именем класса (в данном случае `OAuthHandler`) и которая получает аргументы для настройки нового объекта:

```
In [3]: auth = tweepy.OAuthHandler(keys.consumer_key,
   ...:                            keys.consumer_secret)
   ...:
```

Чтобы передать маркер доступа и секретный маркер доступа, вызовите *метод* `set_access_token` объекта `OAuthHandler`:

```
In [4]: auth.set_access_token(keys.access_token,
   ...:                       keys.access_token_secret)
   ...:
```

Создание объекта API

Теперь создайте объект `API`, который используется для взаимодействия с Twitter:

```
In [5]: api = tweepy.API(auth, wait_on_rate_limit=True,
   ...:                  wait_on_rate_limit_notify=True)
   ...:
```

При вызове конструктора `API` передаются три объекта:

- `auth` — объект `OAuthHandler`, содержащий регистрационные данные.
- Ключевой аргумент `wait_on_rate_limit=True` сообщает Tweepy, чтобы при каждом превышении ограничения частоты использования метода `API` делалась пауза продолжительностью 15 минут. Тем самым предотвращаются возможные нарушения ограничений частоты использования Twitter.
- Ключевой аргумент `wait_on_rate_limit_notify=True` сообщает Tweepy, что при необходимости ожидания из-за превышения частоты использования библиотека должна оповестить вас об этом выводом сообщения в командной строке.

Теперь все готово для взаимодействия с Twitter через Tweepy. Обратите внимание: примеры кода в нескольких ближайших разделах представлены в виде повторяющегося сеанса IPython, чтобы вам не приходилось повторять ранее пройденный процесс авторизации.

12.8. Получение информации об учетной записи Twitter

После аутентификации Twitter *метод* `get_user` объекта Tweepy `API` может использоваться для получения *объекта* `tweepy.models.User` с информацией об учетной записи пользователя Twitter. Получим объект `User` для учетной записи NASA `@nasa`:

```
In [6]: nasa = api.get_user('nasa')
```

Метод `get_user` вызывает метод `users/show` из Twitter API[1]. Для каждого метода Twitter, вызываемого через Tweepy, устанавливается ограничение частоты использования. Метод `users/show`, предназначенный для получения информации о конкретных учетных записях пользователей, может вызываться до 900 раз каждые 15 минут. При упоминании других методов Twitter API мы будем предоставлять сноску со ссылкой на документацию каждого метода, в которой можно узнать его ограничения частоты использования.

Каждый класс `tweepy.models` соответствует объекту JSON, возвращаемому Twitter. Например, класс `User` соответствует *объекту* Twitter `user`:

https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/user-object

У каждого класса `tweepy.models` существует метод, который читает разметку JSON и преобразует ее в объект соответствующего класса Tweepy.

Получение основной информации об учетной записи

Обратимся к некоторым свойствам объекта `User` для вывода информации об учетной записи `@nasa`:

- *Свойство* `id` содержит числовой идентификатор учетной записи, который создается Twitter при регистрации пользователя.
- *Свойство* `name` содержит имя, связанное с учетной записью пользователя.
- *Свойство* `screen_name` содержит экранное имя пользователя в Twitter (`@nasa`). Как `name`, так и `screen_name` могут быть вымышленными именами для защиты конфиденциальности пользователя.
- *Свойство* `description` содержит описание из профиля пользователя.

---

[1] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-users-show.

```
In [7]: nasa.id
Out[7]: 11348282

In [8]: nasa.name
Out[8]: 'NASA'

In [9]: nasa.screen_name
Out[9]: 'NASA'

In [10]: nasa.description
Out[10]: 'Explore the universe and discover our home planet with @NASA.
We usually post in EST (UTC-5)'
```

Получение последних обновлений статуса

Свойство `status` объекта `User` возвращает объект `tweepy.models.Status`, соответствующий *объекту твита* Twitter:

https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object

Свойство `text` объекта `Status` содержит текст последнего твита учетной записи:

```
In [11]: nasa.status.text
Out[11]: 'The interaction of a high-velocity young star with the cloud of
gas and dust may have created this unusually sharp-... https://t.co/
J6uUf7MYMI'
```

Свойство `text` изначально предназначалось для твитов длиной до 140 символов. Многоточие `...` в приведенном фрагменте означает, что текст твита был усечен. Увеличив длину сообщения до 280 символов, компания Twitter добавила *свойство* `extended_tweet` (описанное ниже) для обращения к тексту и другой информации о твитах между 141-м и 280-м символами. В этом случае Twitter присваивает `text` усеченную версию текста `extended_tweet`. Ретвит также приводит к усечению текста, поскольку добавляет символы, которые могут превысить ограничения по количеству символов.

Получение количества подписчиков

Количество подписчиков учетной записи можно получить из *свойства* `followers_count`:

```
In [12]: nasa.followers_count
Out[12]: 29453541
```

Хотя это количество велико, некоторые учетные записи, напомним, имеют более 100 миллионов подписчиков[1].

Получение количества друзей

Аналогичным образом можно узнать и количество друзей учетной записи (то есть количество учетных записей, на которые она подписана) в *свойстве* `friends_count`:

```
In [13]: nasa.friends_count
Out[13]: 287
```

Получение информации вашей учетной записи

Свойства из этого раздела также могут использоваться с вашей учетной записью. Для этого следует вызвать *метод* `me` объекта Tweepy `API`:

```
me = api.me()
```

Оно возвращает объект `User` для той учетной записи, которая была использована для аутентификации Twitter из предыдущего раздела.

12.9. Введение в курсоры Tweepy: получение подписчиков и друзей учетной записи

При вызове методов Twitter API вы часто получаете в результатах коллекции объектов — например, коллекции твитов на вашей временной шкале Twitter, твитов на временной шкале другой учетной записи или списков твитов, соответствующих заданным критериям поиска. *Временная шкала* состоит из твитов, отправленных этим пользователем и друзьями этого пользователя, то есть другими учетными записями, на которые подписан данный пользователь.

В документации каждого метода API упоминается максимальное количество элементов, которые метод может вернуть за один вызов, — оно называется *страницей* результатов. Когда по вашему запросу выбирается больше результатов, чем может вернуть метод, в JSON-ответы Twitter включается информация о наличии других страниц. Класс Tweepy `Cursor` берет на себя все подробности. `Cursor` вызывает указанный метод и проверяет, сообщает

[1] https://friendorfollow.com/twitter/most-followers/.

ли Twitter о наличии других страниц результатов. Если они есть, то `Cursor` снова автоматически вызывает метод для получения этих результатов. Это продолжается до тех пор, пока не останется дополнительных результатов для обработки (с учетом ограничений частоты использования). Если вы настроили объект `API` для перехода в ожидание при достижении ограничений частоты использования (как это сделано в нашем примере), то объект `Cursor` будет соблюдать ограничения частоты использования и делать необходимые паузы между вызовами. В следующих подразделах изложены основы работы с `Cursor`. За дополнительной информацией обращайтесь к учебному руководству `Cursor` по адресу:

http://docs.tweepy.org/en/latest/cursor_tutorial.html

12.9.1. Определение подписчиков учетной записи

Используем объект Tweepy `Cursor` для вызова *метода* `followers` объекта `API`, вызывающий метод Twitter-API `followers/list`[1] для получения подписчиков учетной записи. По умолчанию Twitter возвращает их группами по двадцать, но вы можете запрашивать до 200 записей за прием. Для демонстрационных целей мы загрузим информацию о 10 подписчиках NASA.

Метод `followers` возвращает объекты `tweepy.models.User` с информацией о каждом подписчике. Начнем с создания списка для хранения объектов `User`:

```
In [14]: followers = []
```

Создание объекта Cursor

Затем создадим объект `Cursor`, который вызывает метод `followers` для учетной записи NASA, задаваемой ключевым аргументом `screen_name`:

```
In [15]: cursor = tweepy.Cursor(api.followers, screen_name='nasa')
```

Конструктор `Cursor` получает в аргументе имя вызываемого метода — значение `api.followers` означает, что `Cursor` будет вызывать метод `followers` объекта `api`. Если конструктор `Cursor` получает какие-либо дополнительные ключевые аргументы (скажем, `screen_name`), то они будут переданы методу, заданному первым аргументом конструктора. Таким образом, в результате `Cursor` получит данные подписчиков для учетной записи Twitter `@nasa`.

---

[1] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-list.

Получение результатов

Теперь объект `Cursor` можно использовать для получения подписчиков. Следующая команда `for` перебирает результаты выражения `cursor.items(10)`. *Метод* `items` объекта `Cursor` инициирует вызов `api.followers` и возвращает результаты метода `followers`. В данном случае методу `items` передается значение 10, чтобы получить только 10 результатов:

```
In [16]: for account in cursor.items(10):
    ...:     followers.append(account.screen_name)
    ...:

In [17]: print('Followers:',
    ...:       ' '.join(sorted(followers, key=lambda s: s.lower())))
    ...:
Followers: abhinavborra BHood1976 Eshwar12341 Harish90469614 heshamkisha
Highyaan2407 JiraaJaarra KimYooJ91459029 Lindsey06771483 Wendy_UAE_NL
```

Предыдущий фрагмент выводит подписчиков в списке, упорядоченном по возрастанию, для чего вызывается встроенная функция `sorted`. Во втором аргументе функции передается функция, используемая для определения возможной сортировки элементов followers. В данном случае используется лямбда-выражение, преобразующее каждое имя подписчика к нижнему регистру для выполнения сортировки без учета регистра символов.

Автоматическая разбивка на страницы

Если запрашиваемое количество результатов больше того, что может быть возвращено одним вызовом `followers`, то метод `items` автоматически «перелистывает» результаты многократными вызовами `api.followers`. Вспомните, что `followers` по умолчанию возвращает до 20 подписчиков, поэтому предыдущему коду достаточно вызвать `followers` всего один раз. Для получения до 200 подписчиков за прием создадим объект `Cursor` с ключевым аргументом `count`:

```
cursor = tweepy.Cursor(api.followers, screen_name='nasa', count=200)
```

Если аргумент метода `items` не задан, то `Cursor` пытается получить *всех* подписчиков учетной записи. Однако при большом количестве подписчиков это может потребовать значительного времени из-за ограничений частоты использования Twitter. Метод Twitter API `followers/list` может вернуть не более 200 подписчиков, а Twitter допускает не более 15 вызовов каждые 15 минут. Следовательно, при использовании бесплатных Twitter API за 15 минут можно получить данные не более 3000 подписчиков. Вспомните,

что мы настроили объект API для автоматического перехода в ожидание при достижении ограничения частоты, так что если вы попытаетесь получить всех подписчиков, а у учетной записи их более 3000, Tweepy после каждых 3000 подписчиков автоматически делает 15-минутную паузу и выводит сообщение. На момент написания книги у NASA было более 29,5 миллиона подписчиков. Следовательно, при загрузке данных 12 000 подписчиков в час для получения всех данных потребуется более 100 дней.

Обратите внимание: в данном случае мы могли напрямую вызвать метод `followers` вместо использования `Cursor`, так как мы получаем лишь небольшое количество подписчиков. В данном случае `Cursor` используется только для демонстрации того, как обычно вызывается `followers`. В дальнейших примерах для получения небольшого количества результатов мы будем вызывать методы `API` напрямую без использования `Cursor`.

Получение идентификаторов вместо подписчиков

Хотя за один раз можно получить объекты `User` не более чем для 200 подписчиков, вы можете получить существенно больше числовых идентификаторов Twitter вызовом *метода* `followers_ids` объекта `API`. Он вызывает метод Twitter API `followers/ids`, возвращающий до 5000 идентификаторов за раз (напомним, что в будущем эти ограничения могут измениться)[1]. Этот метод можно вызывать до 15 раз за 15 минут, так что за один интервал ограничения можно получить до 75 000 учетных записей. Он может быть особенно полезен в сочетании с *методом* `lookup_users` объекта `API`. Этот метод вызывает метод Twitter API `users/lookup`[2], который может возвращать до 100 объектов `User` за раз и может вызываться до 300 раз за каждые 15 минут. Таким образом, комбинация этих методов позволяет получить до 30 000 объектов `User` за один интервал ограничения частоты использования.

12.9.2. Определение друзей учетной записи

Метод `friends` объекта `API` вызывает метод Twitter API `friends/list` [3] для получения списка объектов `User`, представляющих друзей учетной записи.

---

[1] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-ids.

[2] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-users-lookup.

[3] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-list.

Twitter по умолчанию возвращает данные группами по двадцать, но вы можете запросить до 200 записей за раз, как было описано выше для метода `followers`. Twitter позволяет вызывать метод `friends/list` до 15 раз каждые 15 минут. Получим 10 учетных записей друзей для учетной записи NASA:

```
In [18]: friends = []

In [19]: cursor = tweepy.Cursor(api.friends, screen_name='nasa')

In [20]: for friend in cursor.items(10):
    ...:     friends.append(friend.screen_name)
    ...:

In [21]: print('Friends:',
    ...:       ' '.join(sorted(friends, key=lambda s: s.lower())))
    ...:
Friends: AFSpace Astro2fish Astro_Kimiya AstroAnnimal AstroDuke
NASA3DPrinter NASASMAP Outpost_42 POTUS44 VicGlover
```

12.9.3. Получение недавних твитов пользователя

Метод `user_timeline` объекта `API` возвращает твиты с временной шкалы конкретной учетной записи. Временная шкала включает твиты учетной записи и твиты ее друзей. Метод вызывает метод Twitter API `statuses/user_timeline`[1], который возвращает последние 20 твитов, но может возвращать до 200 записей за раз. Метод может вернуть до 3200 последних твитов учетной записи. Приложения, в которых используется этот метод, могут вызывать его до 1500 раз за 15 минут.

Метод `user_timeline` возвращает объекты `Status`, каждый из которых представляет один твит. Свойство `user` каждого объекта `Status` содержит ссылку на объект `tweepy.models.User` с информацией о пользователе, который отправил этот твит, например экранное имя `screen_name` пользователя. Свойство `text` объекта `Status` содержит текст твита. Выведем значения `screen_name` и `text` для трех твитов от `@nasa`:

```
In [22]: nasa_tweets = api.user_timeline(screen_name='nasa', count=3)

In [23]: for tweet in nasa_tweets:
    ...:     print(f'{tweet.user.screen_name}: {tweet.text}\n')
    ...:
NASA: Your Gut in Space: Microorganisms in the intestinal tract play an
especially important role in human health. But wh… https://t.co/
uLOsUhwn5p
```

[1] https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-status-es-user_timeline.

```
NASA: We need your help! Want to see panels at @SXSW related to space
exploration? There are a number of exciting panels… https://t.co/
ycqMMdGKUB

NASA: "You are as good as anyone in this town, but you are no better than
any of them," says retired @NASA_Langley mathem… https://t.co/nhMD4n84Nf
```

Эти твиты были усечены (на что указывает многоточие …); скорее всего, они используют более новое ограничение на длину твита в 280 символов. Вскоре мы покажем, как использовать свойство `extended_tweet` для обращения к полному тексту таких твитов.

В предшествующих фрагментах мы решили вызывать метод `user_timeline` напрямую и использовать ключевой аргумент `count` для указания количества загружаемых твитов. Если вы хотите получить максимальное количество твитов на прием (200), то используйте `Cursor` для вызова `user_timeline`, как было продемонстрировано выше. Вспомните, что `Cursor` при необходимости автоматически «пролистывает» результаты повторными вызовами метода.

Получение недавних твитов со своей временной шкалы

Вызов метода `home_timeline` объекта `API`:

```
api.home_timeline()
```

позволяет получить твиты с *вашей* домашней временной шкалы[1], то есть ваших твитов и твитов людей, на которых *вы* подписаны, вызывая метод Twitter `statuses/home_timeline`[2]. По умолчанию `home_timeline` возвращает 20 последних твитов, но может вернуть до 200 твитов за прием. Как и прежде, если вам потребуется получить более 200 твитов с домашней временной шкалы, то лучше использовать объект Tweepy `Cursor` для вызова `home_timeline`.

12.10. Поиск недавних твитов

Метод `search` объекта Tweepy `API` возвращает твиты, соответствующие строке запроса. Согласно документации метода, Twitter ведет поисковый индекс только для твитов за последние 7 дней, поэтому возвращение всех подходящих твитов не гарантировано. Метод `search` вызывает метод Twitter

---

[1] Для учетной записи, использованной для аутентификации Twitter.

[2] https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-home_timeline.

`search/tweets`[1], который по умолчанию возвращает 15 твитов, но может возвращать до ста.

Вспомогательная функция print_tweets из tweetutilities.py

Для этого раздела мы создали вспомогательную функцию `print_tweets`, которая получает результаты вызова метода API `search` и выводит для каждого твита экранное имя пользователя и текст твита. Если твит написан не на английском языке, а значение `tweet.lang` отлично от `'und'` (не определено), то твит также будет переведен на английский язык средствами `TextBlob`, как в главе 11. Чтобы использовать эту функцию, импортируйте ее из файла tweetutilities.py:

```
In [24]: from tweetutilities import print_tweets
```

Ниже выводится только определение функции `print_tweets` из этого файла:

```
def print_tweets(tweets):
    """Для каждого объекта Tweepy Status выводит значение
    screen_name пользователя и текст твита. Если язык отличен
    от английского, то текст переводится средствами TextBlob."""
    for tweet in tweets:
        print(f'{tweet.screen_name}:', end=' ')

        if 'en' in tweet.lang:
            print(f'{tweet.text}\n')
        elif 'und' not in tweet.lang:  # Сначала переводится на английский
            print(f'\n  ORIGINAL: {tweet.text}')
            print(f'TRANSLATED: {TextBlob(tweet.text).translate()}\n')
```

Поиск по конкретным словам

Проведем поиск трех последних твитов, относящихся к NASA Mars Opportunity Rover. Ключевой аргумент `q` метода `search` задает строку запроса для поиска, а ключевой аргумент `count` задает количество возвращаемых твитов:

```
In [25]: tweets = api.search(q='Mars Opportunity Rover', count=3)

In [26]: print_tweets(tweets)
Jacker760: NASA set a deadline on the Mars Rover opportunity! As the dust
on Mars settles the Rover will start to regain power… https://t.co/
KQ7xaFgrzr

Shivak32637174: RT @Gadgets360: NASA 'Cautiously Optimistic' of Hearing
```

[1] https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.

```
Back From Opportunity Rover as Mars Dust Storm Settles
https://t.co/O1iTTwRvFq

ladyanakina: NASA's Opportunity Rover Still Silent on #Mars. https://
t.co/njcyP6zCm3
```

Если вам потребуется получить больше результатов, чем может вернуть один прием `search`, то, как и ранее, лучше использовать объект `Cursor`.

Поиск с использованием поисковых операторов Twitter

В строку запроса можно включать различные поисковые операторы Twitter для уточнения результатов поиска. В табл. 12.2 перечислены некоторые поисковые операторы Twitter. Объединяя несколько операторов, вы сможете строить более сложные запросы. Чтобы увидеть все операторы, откройте страницу

https://twitter.com/search-home

и щелкните по ссылке operators.

Таблица 12.2. Некоторые поисковые операторы Twitter

Пример	Находит твиты, содержащие
`python twitter`	Неявный «*логический* оператор *И*» — находит твиты, содержащие `python` *и* `twitter`
`python OR twitter`	Логический оператор ИЛИ — находит твиты, содержащие `python` *или* `twitter` *или и то и другое*
`python ?`	? (вопросительный знак) — находит твиты, содержащие вопросы о `python`
`planets -mars`	– (минус) — находит твиты, содержащие `planets`, но не содержащие `mars`
`python :)`	:) (веселый смайлик) — находит твиты, содержащие `python` и имеющие *положительную эмоциональную окраску*
`python :(`	:((грустный смайлик) — находит твиты, содержащие `python` и имеющие *отрицательную эмоциональную окраску*
`since:2018-09-01`	Находит твиты, написанные *в указанную дату или позднее* (дата должна задаваться в форме `ГГГГ-ММ-ДД`)
`near:"New York City"`	Находит твиты, отправленные недалеко от заданного места
`from:nasa`	Находит твиты от учетной записи `@nasa`
`to:nasa`	Находит твиты, обращенные к учетной записи `@nasa`

Воспользуемся операторами `from` и `since` для получения трех твитов от NASA начиная с 1 сентября 2018 года (используйте дату на неделю ранее дня выполнения этого кода):

```
In [27]: tweets = api.search(q='from:nasa since:2018-09-01', count=3)

In [28]: print_tweets(tweets)
NASA: @WYSIW Our missions detect active burning fires, track the transport
of fire smoke, provide info for fire managemen… https://t.co/jx2iUoMlIy

NASA: Scarring of the landscape is evident in the wake of the Mendocino
Complex fire, the largest #wildfire in California… https://t.co/Nboo5GD90m

NASA: RT @NASAglenn: To celebrate the #NASA60th anniversary, we're
exploring our history. In this image, Research Pilot Bill Swann prepares
for a…
```

Поиск по хештегу

Твиты часто содержат *хештеги*, начинающиеся со знака #; они обозначают некий важный аспект (например, актуальную тему). Получим два твита с хештегом `#collegefootball`:

```
In [29]: tweets = api.search(q='#collegefootball', count=2)

In [30]: print_tweets(tweets)
dmcreek: So much for #FAU giving #OU a game. #Oklahoma #FloridaAtlantic
#CollegeFootball #LWOS

theangrychef: It's game day folks! And our BBQ game is strong. #bbq
#atlanta #collegefootball #gameday @ Smoke Ring https://t.co/J4lkKhCQE7
```

12.11. Выявление тенденций: Twitter Trends API

Если некоторая тема «взрывает интернет», то по ней могут одновременно писать многие тысячи и даже миллионы людей. Twitter называет такие *темы актуальными*, поддерживая списки актуальных тем по всему миру. При помощи Twitter Trends API можно получить списки географических мест с актуальными темами и списки верхних 50 актуальных тем для каждого места.

12.11.1. Места с актуальными темами

Метод `trends_available` объекта `API` вызывает метод Twitter API `trends/available`[1] для получения списка всех мест, для которых у Twitter существуют актуальные темы. Метод `trends_available` возвращает *список словарей*, представляющих эти места. При выполнении следующего кода было найдено 467 мест с актуальными темами:

```
In [31]: trends_available = api.trends_available()

In [32]: len(trends_available)
Out[32]: 467
```

Словарь в каждом элементе списка, возвращаемый `trends_available`, содержит разнообразную информацию, включая название места и идентификатор `woeid` (см. ниже):

```
In [33]: trends_available[0]
Out[33]:
{'name': 'Worldwide',
 'placeType': {'code': 19, 'name': 'Supername'},
 'url': 'http://where.yahooapis.com/v1/place/1',
 'parentid': 0,
 'country': '',
 'woeid': 1,
 'countryCode': None}

In [34]: trends_available[1]
Out[34]:
{'name': 'Winnipeg',
 'placeType': {'code': 7, 'name': 'Town'},
 'url': 'http://where.yahooapis.com/v1/place/2972',
 'parentid': 23424775,
 'country': 'Canada',
 'woeid': 2972,
 'countryCode': 'CA'}
```

Метод Twitter Trends API `trends/place` (о котором будет рассказано ниже) использует идентификаторы *WOEID* (*Yahoo! Where on Earth ID*) для поиска актуальных тем. WOEID 1 соответствует *миру в целом*. Другие места имеют уникальные WOEID со значением больше 1. В двух следующих подразделах используются значения WOEID для получения общемировых актуальных тем

[1] https://developer.twitter.com/en/docs/trends/locations-with-trending-topics/api-reference/get-trends-available.

и актуальных тем для конкретного города. В табл. 12.3 приведены значения WOEID для некоторых достопримечательностей, городов, государств и континентов. Хотя все эти значения WOEID действительны, это не означает, что в Twitter обязательно имеются актуальные темы для всех этих мест.

Таблица 12.3. Значения WOEID для некоторых достопримечательностей, городов, государств и континентов

Место	WOEID
Статуя Свободы	23617050
Лос-Анджелес (штат Калифорния)	2442047
Вашингтон (федеральный округ Колумбия)	2514815
Париж (Франция)	615702
Водопад Игуасу	468785
Соединенные Штаты	23424977
Северная Америка	24865672
Европа	24865675

Вы также можете искать места, близкие к точке, заданной широтой и долготой. Для этого вызывается *метод* `trends_closest` объекта `API`, который, в свою очередь, вызывает метод Twitter API `trends/closest`[1].

12.11.2. Получение списка актуальных тем

Метод `trends_place` объекта `API` вызывает метод Twitter Trends API `trends/place`[2] для получения первых 50 актуальных тем для места с заданным WOEID. Значения WOEID можно получить из атрибута `woeid` каждого словаря, возвращаемого методами `trends_available` или `trends_closest` из предыдущего раздела, или же найти WOEID конкретного места, проведя поиск по названию города, государства, страны, адресу, почтовому индексу или достопримечательностям на сайте:

http://www.woeidlookup.com

[1] https://developer.twitter.com/en/docs/trends/locations-with-trending-topics/api-reference/get-trends-closest.

[2] https://developer.twitter.com/en/docs/trends/trends-for-location/api-reference/get-trends-place.

Также можно провести программный поиск WOEID при помощи веб-сервисов Yahoo! из таких библиотек Python, как `woeid`[1]:

https://github.com/Ray-SunR/woeid

Общемировые актуальные темы

Получим список общемировых актуальных тем на сегодняшний день (ваши результаты будут другими):

```
In [35]: world_trends = api.trends_place(id=1)
```

Метод `trends_place` возвращает одноэлементный список, который содержит словарь. Ключ словаря `'trends'` ссылается на список словарей, каждый из которых представляет одну актуальную тему:

```
In [36]: trends_list = world_trends[0]['trends']
```

Каждый словарь актуальной темы содержит ключи `name`, `url`, `promoted_content` (признак рекламного твита), `query` и `tweet_volume` (см. далее). Следующая актуальная тема пишется на испанском языке — `#BienvenidoSeptiembre` означает «Добро пожаловать, сентябрь»:

```
In [37]: trends_list[0]
Out[37]:
{'name': '#BienvenidoSeptiembre',
 'url': 'http://twitter.com/search?q=%23BienvenidoSeptiembre',
 'promoted_content': None,
 'query': '%23BienvenidoSeptiembre',
 'tweet_volume': 15186}
```

Для актуальных тем с более чем 10 000 твитов `tweet_volume` содержит число твитов; в противном случае оно равно `None`. Воспользуемся трансформацией списка для его фильтрации, чтобы список содержал только актуальные темы с более чем 10 000 твитов:

```
In [38]: trends_list = [t for t in trends_list if t['tweet_volume']]
```

Затем отсортируем актуальные темы *по убыванию* `tweet_volume`:

```
In [39]: from operator import itemgetter

In [40]: trends_list.sort(key=itemgetter('tweet_volume'), reverse=True)
```

[1] Для этого, согласно документации модуля `woeid`, понадобится ключ API Yahoo!

Теперь выведем названия первых пяти актуальных тем:

```
In [41]: for trend in trends_list[:5]:
    ...:     print(trend['name'])
    ...:
#HBDJanaSenaniPawanKalyan
#BackToHogwarts
Khalil Mack
#ItalianGP
Alisson
```

Актуальные темы для Нью-Йорка

Получим первые пять актуальных тем для Нью-Йорка (WOEID 2459115). Этот код выполняет те же операции, что и выше, но с другим значением WOEID:

```
In [42]: nyc_trends = api.trends_place(id=2459115)  # WOEID Нью-Йорка

In [43]: nyc_list = nyc_trends[0]['trends']

In [44]: nyc_list = [t for t in nyc_list if t['tweet_volume']]

In [45]: nyc_list.sort(key=itemgetter('tweet_volume'), reverse=True)

In [46]: for trend in nyc_list[:5]:
    ...:     print(trend['name'])
    ...:
#IDOL100M
#TuesdayThoughts
#HappyBirthdayLiam
NAFTA
#USOpen
```

12.11.3. Создание словарного облака по актуальным темам

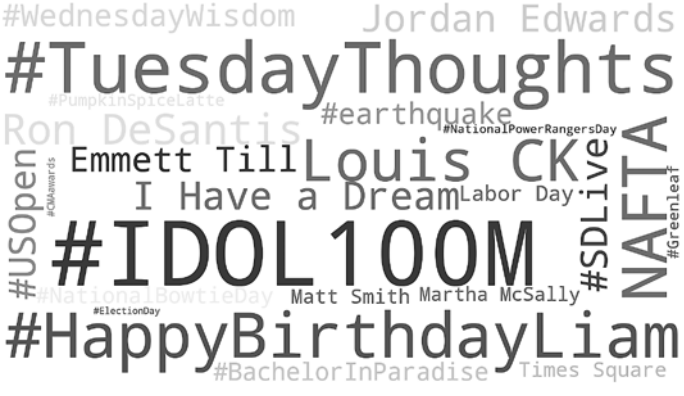
В главе 11 библиотека WordCloud использовалась для построения словарных облаков. Сейчас мы снова используем ее для визуализации актуальных тем Нью-Йорка, содержащих более 10 000 твитов. Начнем с создания словаря пар «ключ-значение», содержащих названия актуальных тем и значений `tweet_volume`:

```
In [47]: topics = {}

In [48]: for trend in nyc_list:
    ...:     topics[trend['name']] = trend['tweet_volume']
    ...:
```

Создадим объект `WordCloud` на основе пар «ключ-значение» словаря `topics`, а затем сохраним словарное облако в графическом файле TrendingTwitter.png (см. иллюстрацию после кода). Аргумент `prefer_horizontal=0.5` *предполагает*, что 50% слов должны выводиться по горизонтали, хотя программа может проигнорировать его для размещения содержимого:

```
In [49]: from wordcloud import WordCloud

In [50]: wordcloud = WordCloud(width=1600, height=900,
    ...:     prefer_horizontal=0.5, min_font_size=10, colormap='prism',
    ...:     background_color='white')
    ...:

In [51]: wordcloud = wordcloud.fit_words(topics)

In [52]: wordcloud = wordcloud.to_file('TrendingTwitter.png')
```

Полученное словарное облако показано ниже — ваш результат будет отличаться в зависимости от состава актуальных тем на день выполнения кода:



12.12. Очистка / предварительная обработка твитов для анализа

Очистка данных — одна из самых распространенных задач, выполняемых специалистами data science. В зависимости от того, как вы собираетесь обрабатывать твиты, вам придется использовать обработку естественного языка для их нормализации с выполнением операций очистки данных (всех или части) из следующей таблицы. Многие операции могут выполняться с использованием библиотек, представленных в главе 11.

Операции очистки данных

- Приведение всего текста к одному регистру.
- Удаление символа # из хештегов.
- Удаление @-упоминаний.
- Удаление дубликатов.
- Удаление лишних пропусков.
- Удаление хештегов.
- Удаление знаков препинания.
- Удаление игнорируемых слов.
- Удаление RT (ретвит) и FAV (избранное).
- Удаление URL-адресов.
- Выделение основы.
- Лемматизация.
- Разбиение на лексемы.

Библиотека tweet-preprocessor и вспомогательные функции TextBlob

В этом разделе *библиотека tweet-preprocessor*:

https://github.com/s/preprocessor

будет использоваться для выполнения базовой очистки данных твитов. Она может автоматически удалять любые комбинации следующих элементов:

- URL-адреса;
- @-упоминания (например, @nasa);
- хештеги (например, #mars);
- зарезервированные слова Twitter (например, RT = ретвит или FAV = избранное, аналог лайков в других социальных сетях);
- эмодзи (все или только смайлики) и
- числа.

В табл. 12.4 перечислены константы модуля, представляющие каждый из вариантов.

Таблица 12.4. Константы модуля, представляющие варианты базовой очистки данных твитов

Вариант	Константа
@-упоминания (например, `@nasa`)	`OPT.MENTION`
Эмодзи	`OPT.EMOJI`
Хештеги (например, `#mars`)	`OPT.HASHTAG`
Числа	`OPT.NUMBER`
Зарезервированные слова (`RT` и `FAV`)	`OPT.RESERVED`
Смайлики	`OPT.SMILEY`
URL	`OPT.URL`

Установка tweet-preprocessor

Для установки tweet-preprocessor откройте приглашение Anaconda (Windows), терминал (macOS/Linux) или командную оболочку (Linux) и выполните команду:

```
pip install tweet-preprocessor
```

Возможно, пользователям Windows придется запустить приглашение Anaconda с правами администратора для получения необходимых привилегий для установки программного обеспечения. Щелкните правой кнопкой мыши на команде Anaconda Prompt в меню Пуск и выберите команду More ▸ Run as administrator.

Очистка твитов

Выполним базовую очистку твита, использующегося в последующем примере этой главы. Библиотеке tweet-preprocessor соответствует имя модуля `preprocessor`. Документация рекомендует импортировать модуль следующим образом:

```
In [1]: import preprocessor as p
```

Чтобы выбрать используемые режимы очистки, используйте функцию `set_options` модуля. В данном примере мы хотим удалить URL и зарезервированные слова Twitter:

```
In [2]: p.set_options(p.OPT.URL, p.OPT.RESERVED)
```

А теперь очистим твит, содержащий зарезервированное слово (RT) и URL-адрес:

```
In [3]: tweet_text = 'RT A sample retweet with a URL https://nasa.gov'

In [4]: p.clean(tweet_text)
Out[4]: 'A sample retweet with a URL'
```

12.13. Twitter Streaming API

Бесплатный Twitter Streaming API динамически передает вашему приложению *случайно выбранные* твиты по мере их появления — до 1% твитов в день. По данным `InternetLiveStats.com`, в секунду пишется приблизительно 6000 твитов, или свыше 500 миллионов твитов в день[1]. Таким образом, Streaming API предоставляет доступ приблизительно к 5 миллионам твитов в день. Когда-то сервис Twitter предоставлял бесплатный доступ к 10% потоковых твитов, но сейчас этот сервис доступен только на платной основе. В этом разделе мы используем определение класса и сеанс IPython для описания основных этапов обработки потоковых твитов. Обратите внимание: для получения потока твитов необходимо создать *пользовательский класс*, *наследующий* от другого класса. Эти темы, напомним, рассмотрены в главе 10.

12.13.1. Создание подкласса StreamListener

Streaming API возвращает появляющиеся твиты, подходящие по критерию поиска. Вместо того чтобы подключаться к Twitter при каждом вызове метода, поток использует *постоянное* подключение для *отправки* твитов вашему приложению. Скорость, с которой поступают твиты, очень сильно изменяется — это зависит от критерия поиска. Чем популярнее тема, тем больше вероятность того, что твиты будут поступать быстро.

Теперь создайте подкласс *класса* Tweepy `StreamListener` для обработки потока твитов. Объектом этого класса является *слушатель*, оповещаемый о по-

[1] http://www.internetlivestats.com/twitter-statistics/.

ступлении каждого нового твита (или другого сообщения, отправленного Twitter[1]). Для каждого сообщения, отправляемого Twitter, вызывается метод `StreamListener`. В табл. 12.5 приведена сводка нескольких таких методов. `StreamListener` уже определяет каждый метод, так что вы переопределяете только те методы, которые вам нужны, — это называется *переопределением*. За информацией о других методах `StreamListener` обращайтесь по адресу:

https://github.com/tweepy/tweepy/blob/master/tweepy/streaming.py

Таблица 12.5. Сводка методов StreamListener

Метод	Описание
`on_connect(self)`	Вызывается при успешном подключении к потоку Twitter. Используется для команд, которые должны выполняться, только если приложение подключено к потоку
`on_status(self, status)`	Вызывается при поступлении твита — `status` является объектом класса Tweepy `Status`
`on_limit(self, track)`	Вызывается при поступлении уведомления об ограничении. Это происходит, когда под ваш запрос подходит большее количество твитов, чем Twitter может доставить при текущих ограничениях на скорость потока. В этом случае в уведомлении указывается количество подходящих твитов, которые не могут быть доставлены
`on_error(self, status_code)`	Вызывается в ответ на отправку Twitter кодов ошибок
`on_timeout(self)`	Вызывается при тайм-ауте подключения, то есть если сервер Twitter не отвечает
`on_warning(self, notice)`	Вызывается, если Twitter отправляет предупреждение об отключении, которое указывает, что подключение может быть закрыто. Например, Twitter поддерживает очередь твитов, отправляемых вашему приложению. Если приложение читает твиты недостаточно быстро, аргумент `notice` функции `on_warning` будет содержать сообщение с предупреждением о том, что подключение будет разорвано при переполнении очереди

[1] За подробной информацией о сообщениях обращайтесь по адресу https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/streaming-message-types.html.

Класс TweetListener

Наш подкласс `StreamListener`, которому присвоено имя `TweetListener`, определяется в файле tweetlistener.py. Рассмотрим компоненты `TweetListener`. Строка 6 означает, что класс `TweetListener` является подклассом `tweepy.StreamListener`, то есть наш новый класс содержит реализации по умолчанию для методов `StreamListener`.

```
1 # tweetlistener.py
2 """Подкласс tweepy.StreamListener для обработки поступающих твитов."""
3 import tweepy
4 from textblob import TextBlob
5
6 class TweetListener(tweepy.StreamListener):
7     """Обрабатывает входной поток твитов."""
8
```

Класс TweetListener: метод __init__

В следующих строках определяется метод `__init__` класса `TweetListener`, который вызывается при создании нового объекта `TweetListener`. Параметр `api` содержит объект Tweepy `API`, используемый `TweetListener` для взаимодействия с `Twitter`. Параметр `limit` содержит общее количество обрабатываемых твитов (10 по умолчанию). Этот параметр добавлен для того, чтобы вы могли управлять количеством получаемых твитов. Как вы вскоре увидите, при достижении этого ограничения поток завершается. Если присвоить `limit` значение `None`, то поток не будет завершаться автоматически. Строка 11 создает переменную экземпляра для отслеживания количества твитов, обработанных до настоящего момента, а строка 12 создает константу для хранения этого ограничения. Если вы не знакомы с методами `__init__` и `super()` из предыдущих глав, то строка 13 гарантирует, что объект `api` правильно сохранен для использования объектом слушателя.

```
 9     def __init__(self, api, limit=10):
10         """Создает переменные экземпляров для отслеживания количества твитов."""
11         self.tweet_count = 0
12         self.TWEET_LIMIT = limit
13         super().__init__(api)  # вызывает версию суперкласса
14
```

Класс TweetListener: метод on_connect

Метод `on_connect` вызывается при успешном подключении вашего приложения к потоку Twitter. Реализация по умолчанию переопределяется для вывода сообщения «Connection successful».

```
    def on_connect(self):
        """Вызывается при успешной попытке подключения, чтобы вы могли
        выполнить соответствующие операции приложения в этот момент."""
        print('Connection successful\n')

```

Класс TweetListener: метод on_status

Метод `on_status` вызывается Tweepy при каждом поступлении твита. Во втором параметре этого метода передается объект Tweepy `Status`, представляющий твит. В строках 23–26 извлекается текст твита. Сначала мы предполагаем, что твит использует новое 280-символьное ограничение длины, и поэтому пытаемся обратиться к свойству твита `extended_tweet` и получить его полный текст `full_text`. Если твит не содержит свойства `extended_tweet`, то происходит исключение. В этом случае вместо него будет получено свойство `text`. В строках 28–30 выводится экранное имя `screen_name` пользователя, отправившего твит, язык твита (`lang`) и `tweet_text`. Если язык отличен от английского (`'en'`), то в строках 32–33 объект `TextBlob` используется для перевода твита и его вывода на английском языке. Мы увеличиваем `self.tweet_count` (строка 36), после чего сравниваем его с `self.TWEET_LIMIT` в команде return. Если `on_status` возвращает `True`, то поток остается открытым. Если же `on_status` возвращает `False`, то Tweepy отсоединяется от потока.

```
    def on_status(self, status):
        """Вызывается, когда Twitter отправляет вам новый твит."""
        # Получение текста твита
        try:
            tweet_text = status.extended_tweet.full_text
        except:
            tweet_text = status.text

        print(f'Screen name: {status.user.screen_name}:')
        print(f'   Language: {status.lang}')
        print(f'     Status: {tweet_text}')

        if status.lang != 'en':
            print(f' Translated: {TextBlob(tweet_text).translate()}')

        print()
        self.tweet_count += 1  # Счетчик обработанных твитов

        # При достижении TWEET_LIMIT вернуть False, чтобы завершить
        # работу с потоком
        return self.tweet_count != self.TWEET_LIMIT
```

12.13.2. Запуск обработки потока

Воспользуемся сеансом IPython для тестирования нового класса `TweetListener`.

Аутентификация

Начнем с аутентификации Twitter и создания объекта Tweepy `API`:

```
In [1]: import tweepy

In [2]: import keys

In [3]: auth = tweepy.OAuthHandler(keys.consumer_key,
   ...:                            keys.consumer_secret)
   ...:

In [4]: auth.set_access_token(keys.access_token,
   ...:                       keys.access_token_secret)
   ...:

In [5]: api = tweepy.API(auth, wait_on_rate_limit=True,
   ...:                  wait_on_rate_limit_notify=True)
   ...:
```

Создание TweetListener

Затем создадим объект класса `TweetListener` и инициализируем его объектом `api`:

```
In [6]: from tweetlistener import TweetListener

In [7]: tweet_listener = TweetListener(api)
```

Аргумент `limit` не задан, поэтому `TweetListener` завершит работу после 10 твитов.

Создание Stream

Объект Tweepy `Stream` управляет подключением к потоку Twitter и передает сообщения `TweetListener`. Ключевой аргумент `auth` конструктора `Stream` получает свойство `auth` объекта `api`, содержащее ранее настроенный объект `OAuthHandler`. Ключевой аргумент `listener` получает объект слушателя:

```
In [8]: tweet_stream = tweepy.Stream(auth=api.auth,
   ...:                              listener=tweet_listener)
   ...:
```

Запуск потока твитов

Метод `filter` объекта `Stream` начинает процесс потоковой передачи. Займемся отслеживанием твитов о марсоходах NASA. В следующем примере параметр `track` используется для передачи списка поисковых критериев:

```
In [9]: tweet_stream.filter(track=['Mars Rover'], is_async=True)
```

Streaming API возвращает полные JSON-объекты для твитов, подходящих по критерию, не только в тексте твита, но также в @-упоминаниях, хештегах, расширенных URL и другой информации, поддерживаемой Twitter в данных JSON объекта твита. Следовательно, вы можете не увидеть искомые критерии, если будете просматривать только текст твита.

Асинхронные и синхронные потоки

Аргумент `is_async=True` означает, что `фильтр` должен инициировать *асинхронный поток твитов*. Это позволяет программе продолжить выполнение, пока слушатель ожидает получения твитов; данная возможность может пригодиться, если вы решите заранее завершить поток. При выполнении асинхронного потока твитов в IPython вы увидите следующее приглашение `In []` и сможете завершить поток твитов, присваивая *свойству* `running` объекта `Stream` значение `False`:

```
tweet_stream.running=False
```

Без аргумента `is_async=True` `фильтр` запускает *синхронный поток твитов*. В этом случае IPython выведет приглашение `In []` *после* завершения потока. Асинхронные потоки особенно полезны в GUI-приложениях, чтобы в процессе поступления твитов пользователи могли продолжать работу с другими частями приложения. Ниже приведена часть вывода, состоящая из двух твитов:

```
Connection successful

Screen name: bevjoy:
   Language: en
     Status: RT @SPACEdotcom: With Mars Dust Storm Clearing, Opportunity
Rover Could Finally Wake Up https://t.co/OIRP9UyB8C https://t.co/
gTfFR3RUkG

Screen name: tourmaline1973:
   Language: en
     Status: RT @BennuBirdy: Our beloved Mars rover isn't done yet, but
she urgently needs our support! Spread the word that you want to keep
calling ou…
...
```

Другие параметры метода filter

Метод `filter` также имеет другие параметры для уточнения критерия поиска твитов по идентификаторам пользователей Twitter (для отслеживания твитов от конкретных пользователей) и местоположению. Подробности — по адресу:

https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/basic-stream-parameters

Ограничения Twitter

Маркетологи, исследователи и другие специалисты часто сохраняют твиты, полученные от Streaming API. Twitter требует, чтобы при сохранении твитов удалялись все сообщения или данные местоположения, для которых было получено сообщение об удалении. Это происходит, если пользователь удалил твит или данные местоположения после того, как Twitter отправил этот твит вам. В этом случае будет вызван *метод* `on_delete` вашего слушателя. За информацией о правилах удаления и подробностях сообщений обращайтесь по адресу:

https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/streaming-message-types

12.14. Анализ эмоциональной окраски твитов

В главе 11 был продемонстрирован анализ эмоциональной окраски для текста. Многие аналитики и компании выполняют анализ эмоциональной окраски твитов. Например, политические аналитики могут проверить эмоциональную окраску во время выборов, чтобы понять, как люди относятся к конкретным политикам и темам. Компании могут проверять эмоциональную окраску твитов, чтобы понять, что люди говорят об их продуктах и продуктах конкурентов.

В этом разделе мы воспользуемся методами, представленными ранее, для создания сценария (sentimentlistener.py), который позволяет проверить эмоциональную окраску конкретной темы. Сценарий подсчитывает сумму обработанных положительных, отрицательных и нейтральных твитов и выводит результаты.

Сценарий получает два аргумента командной строки, представляющие тему получаемых твитов и количество твитов, для которых должна проверяться эмоциональная окраска, — подсчет ведется только для тех твитов, которые не были исключены. Для вирусных тем характерно большое количество ретвитов,

которые мы не подсчитываем, поэтому для получения заданного количества твитов может потребоваться некоторое время. Сценарий запускается из каталога ch12 следующей командой:

```
ipython sentimentlistener.py football 10
```

Примерный вывод команды приведен ниже. Положительные твиты помечены знаком +, отрицательные — знаком -, а нейтральные — пробелом:

```
- ftblNeutral: Awful game of football. So boring slow hoofball complete
waste of another 90 minutes of my life that I'll never get back #BURMUN

+ TBulmer28: I've seen 2 successful onside kicks within a 40 minute span.
I love college football

+ CMayADay12: The last normal Sunday for the next couple months. Don't
text me don't call me. I am busy. Football season is finally here?

  rpimusic: My heart legitimately hurts for Kansas football fans

+ DSCunningham30: @LeahShieldsWPSD It's awsome that u like college
football, but my favorite team is ND - GO IRISH!!!

  damanr: I'm bummed I don't know enough about football to roast
@samesfandiari properly about the Raiders

+ jamesianosborne: @TheRochaSays @WatfordFC @JackHind Haha.... just when
you think an American understands Football.... so close. Wat…

+ Tshanerbeer: @PennStateFball @PennStateOnBTN Ah yes, welcome back
college football. You've been missed.

- cougarhokie: @hokiehack @skiptyler I can verify the badness of that
football

+ Unite_Reddevils: @Pablo_di_Don Well make yourself clear it's football
not soccer we follow European football not MLS soccer

Tweet sentiment for "football"
Positive: 6
 Neutral: 2
Negative: 2
```

Сценарий (sentimentlistener.py) приведен ниже. Мы подробно рассмотрим только новую функциональность этого примера.

Импортирование

Строки 4–8 импортируют файл keys.py и библиотеки, использованные в сценарии:

```
1 # sentimentlisener.py
2 """Сценарий для поиска твитов, соответствующих строке поиска,
3 и суммирования количества положительных, отрицательных и нейтральных твитов."""
4 import keys
5 import preprocessor as p
6 import sys
7 from textblob import TextBlob
8 import tweepy
9
```

Класс SentimentListener: метод __init__

Кроме объекта `API`, обеспечивающего взаимодействие с Twitter, метод `__init__` получает еще три параметра:

- `sentiment_dict` — словарь для хранения счетчиков эмоциональной окраски;
- `topic` — искомая тема, которая должна присутствовать в тексте твита;
- `limit` — количество обрабатываемых твитов (без учета исключаемых твитов).

Каждое из этих значений сохраняется в текущем объекта `SentimentListener` (`self`).

```
10 class SentimentListener(tweepy.StreamListener):
11     """Обрабатывает входной поток твитов."""
12
13     def __init__(self, api, sentiment_dict, topic, limit=10):
14         """Инициализирует объект SentimentListener."""
15         self.sentiment_dict = sentiment_dict
16         self.tweet_count = 0
17         self.topic = topic
18         self.TWEET_LIMIT = limit
19
20         # Настройка tweet-preprocessor для удаления URL/зарезервированных слов
21         p.set_options(p.OPT.URL, p.OPT.RESERVED)
22         super().__init__(api)  # вызывает версию суперкласса
23
```

Метод on_status

При получении твита метод `on_status`:

- получает текст твита (строки 27–30);
- пропускает твит в том случае, если он является ретвитом (строки 33–34);
- проводит очистку твита с удалением URL и зарезервированных слов, таких как RT и FAV (строка 36);
- пропускает твит, если искомая тема не входит в текст твита (строки 39–40);
- использует объект `TextBlob` для проверки эмоциональной окраски твита и обновляет `sentiment_dict` (строки 43–52);
- выводит текст твита (строка 55) с префиксом `+` для положительной эмоциональной окраски, пробелом для нейтральной или `-` для отрицательной, а также проверяет, было ли обработано заданное количество твитов (строки 57–60).

```
24      def on_status(self, status):
25          """Вызывается, когда Twitter отправляет вам новый твит."""
26          # Получение текста твита
27          try:
28              tweet_text = status.extended_tweet.full_text
29          except:
30              tweet_text = status.text
31
32          # Ретвиты игнорируются
33          if tweet_text.startswith('RT'):
34              return
35
36          tweet_text = p.clean(tweet_text)  # Очистка твита
37
38          # Игнорировать твит, если тема не входит в текст твита
39          if self.topic.lower() not in tweet_text.lower():
40              return
41
42          # Обновить self.sentiment_dict данными полярности
43          blob = TextBlob(tweet_text)
44          if blob.sentiment.polarity > 0:
45              sentiment = '+'
46              self.sentiment_dict['positive'] += 1
47          elif blob.sentiment.polarity == 0:
48              sentiment = ' '
49              self.sentiment_dict['neutral'] += 1
50          else:
51              sentiment = '-'
52              self.sentiment_dict['negative'] += 1
```

```
53
54          # Вывести твит
55          print(f'{sentiment} {status.user.screen_name}: {tweet_text}\n')
56
57          self.tweet_count += 1  # Счетчик обработанных твитов
58
59          # При достижении TWEET_LIMIT вернуть False, чтобы завершить работу
            # с потоком
60          return self.tweet_count != self.TWEET_LIMIT
61
```

Основное приложение

Основное приложение определяется в функции `main` (строки 62–87; см. описание после листинга), которая вызывается в строках 90–91 при выполнении файла как сценария. Это позволяет импортировать sentimentlistener.py в IPython или другие модули для использования класса `SentimentListener`, как было сделано с `TweetListener` в предыдущем разделе:

```
62 def main():
63     # Настройка OAuthHandler
64     auth = tweepy.OAuthHandler(keys.consumer_key, keys.consumer_secret)
65     auth.set_access_token(keys.access_token, keys.access_token_secret)
66
67     # Получить объект API
68     api = tweepy.API(auth, wait_on_rate_limit=True,
69                      wait_on_rate_limit_notify=True)
70
71     # Создать объект подкласса StreamListener
72     search_key = sys.argv[1]
73     limit = int(sys.argv[2])  # Количество отслеживаемых твитов
74     sentiment_dict = {'positive': 0, 'neutral': 0, 'negative': 0}
75     sentiment_listener = SentimentListener(api,
76         sentiment_dict, search_key, limit)
77
78     # Создание Stream
79     stream = tweepy.Stream(auth=api.auth, listener=sentiment_listener)
80
81     # Начать фильтрацию твитов на английском языке, содержащих search_key
82     stream.filter(track=[search_key], languages=['en'], is_async=False)
83
84     print(f'Tweet sentiment for "{search_key}"')
85     print('Positive:', sentiment_dict['positive'])
86     print(' Neutral:', sentiment_dict['neutral'])
87     print('Negative:', sentiment_dict['negative'])
88
89 # Вызвать main, если файл выполняется как сценарий
90 if __name__ == '__main__':
91     main()
```

Строки 72–73 получают аргументы командной строки. Строка 74 создает словарь `sentiment_dict` для хранения данных эмоциональной окраски твитов. В строках 75–76 создается объект `SentimentListener`. Строка 79 создает объект `Stream`. Поток, как и прежде, запускается вызовом метода `filter` класса `Stream` (строка 82). Тем не менее в этом примере используется синхронный поток, чтобы в строках 84–87 отчет об эмоциональной окраске выводился только после обработки заданного количества твитов (`limit`). При вызове `filter` также передается ключевой аргумент `languages`, в котором задается список кодов языков. Единственный код языка `'en'` означает, что Twitter должен вернуть только твиты на английском языке.

12.15. Геокодирование и вывод информации на карте

В этом разделе мы соберем потоковые твиты, а затем выведем для них данные местоположения. Большинство твитов не включает данные широты и долготы, потому что Twitter по умолчанию отключает эту возможность для всех пользователей. Тот, кто захочет включить в твит свое точное местоположение, должен сознательно включить эту возможность. Хотя в большинстве твитов точная информация о местоположении отсутствует, значительная их часть включает данные о нахождении дома пользователя; впрочем, даже здесь часто содержится недействительная информация — скажем, «Где-то далеко» или название вымышленного места из любимого фильма пользователя.

В целях упрощения мы далее будем использовать свойство `location` объекта `User` для нанесения местоположения пользователя на интерактивную карту, позволяющую изменять масштаб и перетаскивать изображение, чтобы просматривать другие области. Для каждого твита на карте будет отображаться маркер; если щелкнуть на нем, то появится временное окно с экранным именем пользователя и текстом твита.

Ретвиты и твиты, не содержащие искомой темы, будут игнорироваться. Для других твитов мы будем отслеживать процент твитов, содержащих информацию о местоположении. При получении данных широты и долготы также будет отслеживаться процент твитов с недействительными данными местоположения.

Библиотека geopy

Библиотека geopy (https://github.com/geopy/geopy) будет использоваться для преобразования информации местоположения в широту и долготу (этот процесс

называется *геокодированием*), чтобы маркеры можно было разместить на карте. Библиотека поддерживает десятки веб-сервисов геокодирования, многие из которых имеют бесплатные или упрощенные уровни доступа. В данном примере будет использоваться *сервис геокодирования OpenMapQuest* (см. далее). Библиотека geopy была установлена в разделе 12.6.

OpenMapQuest

Мы используем API геокодирования OpenMapQuest для преобразования местоположения (например, Boston, MA) в широту и долготу (например, 42,3602534 и –71,0582912) для нанесения на карту. В настоящее время OpenMapQuest разрешает выполнить на бесплатном уровне до 15 000 операций в месяц.

Чтобы пользоваться сервисом, сначала зарегистрируйтесь по адресу:

https://developer.mapquest.com/

После регистрации перейдите по адресу:

https://developer.mapquest.com/user/me/apps

Затем щелкните на кнопке Create a New Key, введите в поле App Name любое имя на ваш выбор, оставьте поле Callback URL пустым и щелкните на кнопке Create App для создания ключа API. Затем щелкните на имени приложения на веб-странице, чтобы просмотреть ключ пользователя. Сохраните ключ пользователя в файле keys.py, использованном ранее в этой главе; замените *ВашКлюч* в строке

```
mapquest_key = 'ВашКлюч'
```

Как и ранее в этой главе, мы импортируем keys.py для работы с этим ключом.

Библиотека Library и библиотека Leaflet.js

Для работы с картами в этом примере задействовалась *библиотека folium*:

https://github.com/python-visualization/folium,

использующая популярную картографическую JavaScript-библиотеку Leaflet.js для вывода карт. Карты, построенные folium, сохраняются в файлах HTML, которые можно просматривать в браузере. Чтобы установить folium, выполните команду:

```
pip install folium
```

Карты OpenStreetMap.org

По умолчанию Leaflet.js использует находящиеся в свободном доступе карты `OpenStreetMap.org`, права на которые принадлежат участникам одноименного проекта. Чтобы использовать эти карты[1], вы должны включить следующее уведомление об авторском праве:

```
Map data © OpenStreetMap contributors
```

В условиях использования сервиса указано:

Вы обязаны ясно заявить, что данные предоставляются на условиях лицензии Open Database License. Для этого можно предоставить ссылку «Лицензия» или «Условия», ведущую на

www.openstreetmap.org/copyright *или* www.opendatacommons.org/licenses/odbl.

12.15.1. Получение твитов и нанесение их на карту

Выполним интерактивную разработку кода вывода местоположения твитов. Для этого будут использоваться вспомогательные функции из файла tweetutilities.py и класс `LocationListener` из locationlistener.py. Вспомогательные функции и класс будут более подробно рассмотрены в последующих разделах.

Получение объекта API

Как и в остальных примерах работы с потоками, выполним аутентификацию Twitter и получим объект Tweepy `API`. На этот раз будет использоваться вспомогательная функция `get_API` из tweetutilities.py:

```
In [1]: from tweetutilities import get_API

In [2]: api = get_API()
```

Коллекции, необходимые для LocationListener

Нашему классу `LocationListener` требуются две коллекции: список (`tweets`) для хранения собранных твитов и словарь (`counts`) для хранения общего количества собранных твитов и количества твитов с данными местоположения:

```
In [3]: tweets = []

In [4]: counts = {'total_tweets': 0, 'locations': 0}
```

---

[1] https://wiki.osmfoundation.org/wiki/Licence/Licence_and_Legal_FAQ.

Создание LocationListener

В данном примере `LocationListener` собирает 50 твитов по теме `'football'`:

```
In [5]: from locationlistener import LocationListener

In [6]: location_listener = LocationListener(api, counts_dict=counts,
   ...:     tweets_list=tweets, topic='football', limit=50)
   ...:
```

`LocationListener` при помощи вспомогательной функции `get_tweet_content` извлекает из каждого твита экранное имя, текст твита и местоположение и помещает эти данные в словарь.

Настройка и запуск потока твитов

Создадим объект `Stream` для поиска англоязычных твитов по теме `'football'`:

```
In [7]: import tweepy

In [8]: stream = tweepy.Stream(auth=api.auth, listener=location_listener)

In [9]: stream.filter(track=['football'], languages=['en'], is_async=False)
```

Дождитесь получения твитов. Хотя здесь результаты не показаны (для экономии места), `LocationListener` выводит для каждого твита экранное имя и текст, чтобы вы видели живой поток твитов. Если твиты не появляются (например, если сейчас не футбольный сезон), то завершите предыдущий фрагмент нажатием *Ctrl* + *C* и попробуйте снова с другим условием поиска.

Вывод статистики местоположения

Когда появится следующее приглашение `In []`, вы сможете проверить количество обработанных твитов, количество твитов с данными местоположения и их процент:

```
In [10]: counts['total_tweets']
Out[10]: 63

In [11]: counts['locations']
Out[11]: 50

In [12]: print(f'{counts["locations"] / counts["total_tweets"]:.1%}')
79.4%
```

При запуске 79,4% твитов содержали данные местоположения.

Геокодирование местоположения

Теперь воспользуемся вспомогательной функцией `get_geocodes` из tweetutilities.py для геокодирования местоположения каждого твита из списка `tweets`:

```
In [13]: from tweetutilities import get_geocodes

In [14]: bad_locations = get_geocodes(tweets)
Getting coordinates for tweet locations...
OpenMapQuest service timed out. Waiting.
OpenMapQuest service timed out. Waiting.
Done geocoding
```

Иногда при использовании сервиса геокодирования OpenMapQuest происходит тайм-аут; это означает, что запрос не может быть обработан немедленно, и вам придется попробовать снова. В этом случае функция `get_geocodes` выводит сообщение, делает короткую паузу, а затем снова пытается выдать запрос геокодирования.

Как вы вскоре увидите, для каждого твита с *действительным* местоположением функция `get_geocodes` добавляет в словарь твитов в списке `tweets` два новых ключа — `'latitude'` и `'longitude'`. Функция использует координаты из твита, возвращенные OpenMapQuest.

Вывод статистики некорректных данных местоположения

Когда на экране появится следующее приглашение `In []`, вы сможете проверить процент твитов с некорректными данными местоположения:

```
In [15]: bad_locations
Out[15]: 7

In [16]: print(f'{bad_locations / counts["locations"]:.1%}')
14.0%
```

В данном случае из 50 твитов с данными местоположения 7 (14%) имели недействительные данные местоположения.

Очистка данных

До нанесения местоположения твита на карту воспользуемся коллекцией Pandas `DataFrame` и выполним очистку данных. При создании `DataFrame` на базе списка `tweets` коллекция будет содержать значение `NaN` в свойствах `'latitude'` и `'longitude'` любого твита, не имеющего действительного ме-

стоположения. Подобные строки можно удалить вызовом *метода* `dropna` коллекции `DataFrame`:

```
In [17]: import pandas as pd

In [18]: df = pd.DataFrame(tweets)

In [19]: df = df.dropna()
```

Создание объекта Map

Создадим объект folium `Map`, на который будут наноситься данные местоположения твитов:

```
In [20]: import folium

In [21]: usmap = folium.Map(location=[39.8283, -98.5795],
    ...:                    tiles='Stamen Terrain',
    ...:                    zoom_start=5, detect_retina=True)
    ...:
```

Ключевой аргумент `location` задает последовательность с широтой и долготой центральной точки карты. Приведенные значения соответствуют географическому центру США (http://bit.ly/CenterOfTheUS). Возможно, у некоторых твитов данные местоположения будут находиться за границами США. В этом случае они не будут видны изначально при открытии карты. Вы можете изменять масштаб при помощи кнопок `+` и `–` в левом верхнем углу карты или же панорамировать карту, перетаскивая изображение мышью, для просмотра любой точки мира.

Ключевой аргумент `zoom_start` задает исходный масштаб карты; при меньших значениях на карте помещается большая часть мира, а при больших — меньшая. В нашей системе при масштабе 5 выводится вся континентальная часть США. Ключевой аргумент `detect_retina` позволяет folium обнаруживать экраны высокого разрешения. В этом случае folium запрашивает с OpenStreetMap.org карты высокого разрешения, а уровень масштаба изменяется соответствующим образом.

Создание временных окон для местоположения твитов

На следующем этапе переберем `DataFrame` и добавим на объект `Map` объекты folium `Popup`, содержащие текст каждого твита. В данном случае мы используем метод `itertuples` для создания кортежа на базе каждой строки `DataFrame`. Каждый кортеж содержит свойство для каждого столбца `DataFrame`:

```
In [22]: for t in df.itertuples():
    ...:     text = ': '.join([t.screen_name, t.text])
    ...:     popup = folium.Popup(text, parse_html=True)
    ...:     marker = folium.Marker((t.latitude, t.longitude),
    ...:                            popup=popup)
    ...:     marker.add_to(usmap)
    ...:
```

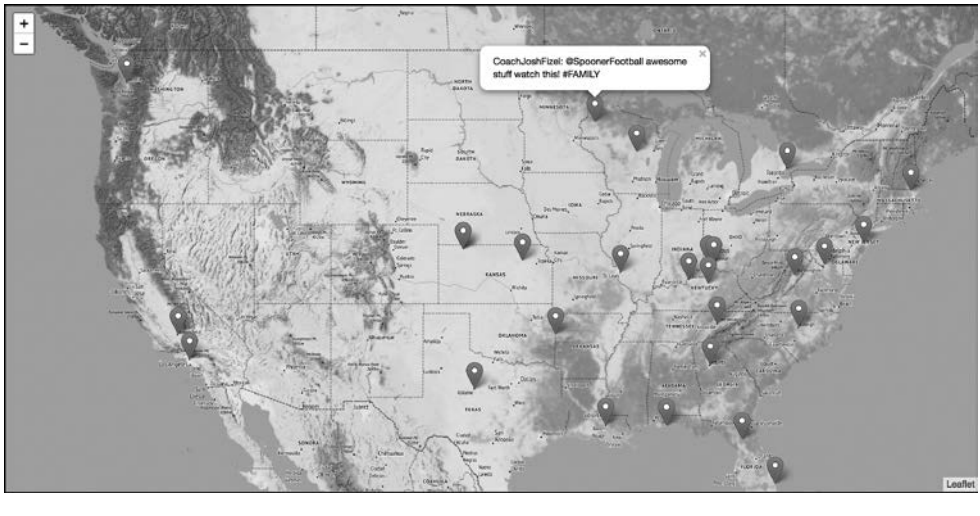
Сначала создается строка (`text`) с экранным именем пользователя `screen_name` и текстом твита (`text`), разделенными двоеточием. Эта строка будет выводиться на карте при щелчке на соответствующем маркере. Вторая команда создает объект folium `Popup` для вывода текста. Третья команда создает объект folium `Marker` с использованием кортежа для определения широты и долготы `маркера`. Ключевой аргумент `popup` связывает объект `Popup` для твита с новым объектом `Marker`. Наконец, последняя команда вызывает *метод* `add_to` объекта `Marker`, чтобы задать объект `Map`, на котором должен отображаться `маркер`.

Сохранение карты

Последним шагом становится вызов метода `save` объекта `Map` для сохранения карты в HTML-файле; двойной щелчок на этом файле откроет его в браузере:

```
In [23]: usmap.save('tweet_map.html')
```

Ниже приведена полученная карта (на вашей карте расположение маркеров будет другим):



12.15.2. Вспомогательные функции tweetutilities.py

В этом разделе будут представлены вспомогательные функции `get_tweet_content` и `get_geo_codes`, использованные в сеансе IPython из предыдущего раздела. В каждом случае номера строк начинаются с 1 для удобства обсуждения. Обе функции определяются в файле tweetutilities.py, включенном в каталог примеров ch12.

Вспомогательная функция get_tweet_content

Функция `get_tweet_content` получает объект `Status` (tweet) и создает словарь с экранным именем твита (строка 4), `текстом` (строки 7–10) и местоположением (строки 12–13). Местоположение включается только в том случае, если ключевой аргумент `location` равен `True`. Для текста твита попытаемся использовать свойство `full_text` объекта `extended_tweet`. Если оно недоступно, то используется свойство `text`:

```
 1 def get_tweet_content(tweet, location=False):
 2     """Возвращает словарь с данными из твита (объект Status)."""
 3     fields = {}
 4     fields['screen_name'] = tweet.user.screen_name
 5
 6     # Получение текста твита
 7     try:
 8         fields['text'] = tweet.extended_tweet.full_text
 9     except:
10         fields['text'] = tweet.text
11
12     if location:
13         fields['location'] = tweet.user.location
14
15     return fields
```

Вспомогательная функция get_geocodes

Функция `get_geocodes` получает список словарей, содержащих твиты и геокоды их местоположений. Если геокодирование для твита прошло успешно, то функция добавляет широту и долготу в словарь твита в `tweet_list`. Для выполнения этого кода необходим класс `OpenMapQuest` из модуля `geopy`, который импортируется в файл tweetutilities.py следующим образом:

```
from geopy import OpenMapQuest
```

```
1 def get_geocodes(tweet_list):
2     """Получает широту и долготу для местоположения каждого твита.
```

```
 3     Возвращает количество твитов с недействительными данными местоположения."""
 4     print('Getting coordinates for tweet locations...')
 5     geo = OpenMapQuest(api_key=keys.mapquest_key)  # Геокодер
 6     bad_locations = 0
 7
 8     for tweet in tweet_list:
 9         processed = False
10         delay = .1  # Используется, если происходит тайм-аут OpenMapQuest
11         while not processed:
12             try:  # Получить координаты для tweet['location']
13                 geo_location = geo.geocode(tweet['location'])
14                 processed = True
15             except:  # Тайм-аут, сделать паузу перед повторной попыткой
16                 print('OpenMapQuest service timed out. Waiting.')
17                 time.sleep(delay)
18                 delay += .1
19
20         if geo_location:
21             tweet['latitude'] = geo_location.latitude
22             tweet['longitude'] = geo_location.longitude
23         else:
24             bad_locations += 1  # Значение tweet['location'] недействительно
25
26     print('Done geocoding')
27     return bad_locations
```

Функция работает следующим образом:

- Строка 5 создает объект `OpenMapQuest`, используемый для геокодирования местоположений. Ключевой аргумент `api_key` загружается из файла keys.py, который был отредактирован ранее.
- Строка 6 инициализирует переменную `bad_locations`, используемую для отслеживания количества недействительных местоположений в собранных объектах твитов.
- В цикле строки 9–18 пытаются выполнить геокодирование местоположения текущего твита. Иногда сервис геокодирования OpenMapQuest отказывает по тайм-ауту, что указывает на его временную недоступность. Также это может произойти при выдаче слишком большого количества запросов. Цикл `while` продолжает выполняться, пока переменная `processed` равна `False`. При каждой итерации цикл вызывает *метод* `geocode` объекта `OpenMapQuest` с передачей строки местоположения твита в аргументе. В случае успеха `processed` присваивается значение `True` и цикл завершается. В противном случае строки 16–18 выводят сообщение о тайм-ауте, делается пауза продолжительностью `delay` секунд и задержка увеличива-

ется на случай возникновения следующего тайм-аута. Строка 17 вызывает метод `sleep` модуля `time` стандартной библиотеки Python для приостановки выполнения кода.

- После завершения цикла `while` строки 20–24 проверяют, были ли возвращены данные местоположения, и если да, то они добавляются в словарь твита. В противном случае строка 24 увеличивает счетчик `bad_locations`.
- Наконец, функция выводит сообщение о завершении геокодирования и возвращает значение `bad_locations`.

12.15.3. Класс LocationListener

Класс `LocationListener` выполняет многие операции, продемонстрированные в предшествующих примерах работы с потоками, поэтому мы сосредоточимся на отдельных строках класса:

```
 1 # locationlistener.py
 2 """Получает твиты, соответствующие искомой строке, и сохраняет список
 3 словарей с экранным именем/текстом/местоположением каждого твита."""
 4 import tweepy
 5 from tweetutilities import get_tweet_content
 6
 7 class LocationListener(tweepy.StreamListener):
 8     """Обрабатывает входной поток твитов для получения данных местоположения."""
 9
10     def __init__(self, api, counts_dict, tweets_list, topic, limit=10):
11         """Настройка LocationListener."""
12         self.tweets_list = tweets_list
13         self.counts_dict = counts_dict
14         self.topic = topic
15         self.TWEET_LIMIT = limit
16         super().__init__(api)  # Вызов версии суперкласса
17
18     def on_status(self, status):
19         """Вызывается, когда Twitter отправляет вам новый твит."""
20         # Получить экранное имя, текст и местоположение каждого твита.
21         tweet_data = get_tweet_content(status, location=True)
22
23         # Игнорировать ретвиты и твиты, не содержащие темы.
24         if (tweet_data['text'].startswith('RT') or
25             self.topic.lower() not in tweet_data['text'].lower()):
26             return
27
28         self.counts_dict['total_tweets'] += 1  # Исходный твит
29
```

```
30          # Игнорировать твиты без данных местоположения.
31          if not status.user.location:
32              return
33
34          self.counts_dict['locations'] += 1  # Твит с местоположением.
35          self.tweets_list.append(tweet_data)  # Сохранить твит.
36          print(f'{status.user.screen_name}: {tweet_data["text"]}\n')
37
38          # При достижении TWEET_LIMIT вернуть False, чтобы завершить работу
            # с потоком
39          return self.counts_dict['locations'] != self.TWEET_LIMIT
```

В этом случае метод `__init__` получает словарь `counts`, используемый для отслеживания общего количества обработанных твитов, и список `tweet_list`, в котором хранятся словари, возвращаемые вспомогательной функцией `get_tweet_content`.

Метод `on_status`:

- Вызывает `get_tweet_content` для получения экранного имени и местоположения каждого твита.
- Игнорирует твит, если он является ретвитом или его текст не содержит искомой темы — при подсчете используются только исходные твиты, содержащие искомую строку.
- Увеличивает значение ключа `'total_tweets'` в словаре `counts` на 1 для подсчета количества исходных твитов.
- Игнорирует твиты без данных местоположения.
- Увеличивает значение ключа `'locations'` в словаре `counts` на 1 для подсчета количества твитов с данными местоположения.
- Присоединяет к `tweets_list` словарь `tweet_data`, возвращенный вызовом `get_tweet_content`.
- Выводит экранное имя и текст твита, показывая, что приложение не висит.
- Проверяет, было ли достигнуто ограничение `TWEET_LIMIT`, и если да, то возвращает `False` для завершения работы с потоком.

12.16. Способы хранения твитов

Предназначенные для анализа твиты обычно сохраняются в следующих форматах:

- CSV-файлы — формат CSV был представлен в главе 9.
- Коллекции Pandas `DataFrame` в памяти — CSV-файлы легко загружаются в `DataFrame` для очистки и обработки.
- Базы данных SQL, такие как MySQL, — бесплатная реляционная система управления базами данных (РСУБД) с открытым кодом.
- Базы данных NoSQL — Twitter возвращает твиты в форме документов JSON и результаты будет хранить в документной базе данных JSON NoSQL (например, MongoDB). Обычно Tweepy скрывает работу с JSON от разработчика. Если вы предпочитаете работать с JSON напрямую, то используйте методы, описанные в главе 16, когда мы займемся изучением библиотеки PyMongo.

12.17. Twitter и временные ряды

Временной ряд представляет собой серию значений с временными метками. Примеры временных рядов — котировки на момент закрытия операций на бирже, ежедневные измерения температуры в заданной местности, количество ежемесячно появляющихся рабочих мест в США, квартальная прибыль компании и т. д. Твиты отлично подходят для анализа временных рядов, потому что они снабжены временными метками. В главе 14 метод простой линейной регрессии будет использоваться для прогнозирования временных рядов. Мы, кроме того, вернемся к временным рядам в главе 15 при обсуждении рекуррентных нейронных сетей.

12.18. Итоги

Эта глава была посвящена глубокому анализу данных Twitter — пожалуй, самой открытой и доступной из всех социальных сетей и одним из самых распространенных источников больших данных. Вы создали учетную запись разработчика Twitter и подключились к Twitter с регистрационными данными своей учетной записи. Мы обсудили ограничения частоты использования Twitter и некоторые дополнительные правила, а также важность их соблюдения.

Далее описано представление твита в формате JSON. Мы использовали Tweepy — один из самых популярных клиентов Twitter API — для аутентификации Twitter и обращения к различным API. Было показано, что твиты,

возвращаемые Twitter API, наряду с текстом твита содержат большое количество метаданных. Вы узнали, как определить подписчиков учетной записи и ее друзей и как получить последние твиты пользователя.

Объект Tweepy `Cursor` был использован для удобного получения последовательных страниц результатов от различных Twitter API. Мы использовали Twitter Search API для загрузки последних твитов, удовлетворяющих заданному критерию. Twitter Streaming API был использован для подключения к потоку твитов в процессе их появления, а Twitter Trends API — для определения актуальных тем для различных мест; на основании информации о темах было построено словарное облако.

Библиотека tweet-preprocessor была использована для очистки и предварительной обработки твитов в ходе подготовки их к анализу, после чего мы провели анализ эмоциональной окраски твитов. При помощи библиотеки folium была построена интерактивная карта местоположения твитов с возможностью просмотра твитов для конкретного места. Также мы рассмотрели стандартные способы хранения твитов и отметили, что твиты представляют собой естественную форму данных временных рядов. В следующей главе будет представлен суперкомпьютер IBM Watson и его возможности когнитивных вычислений.

13

IBM Watson и когнитивные вычисления

В этой главе...

- Диапазон сервисов Watson и использование уровня Lite для бесплатного знакомства с ними.
- Демонстрация сервисов Watson.
- Концепция когнитивных вычислений и их интеграция в ваши приложения.
- Регистрация учетной записи IBM Cloud и получение регистрационных данных для использования различных сервисов.
- Установка Watson Developer Cloud Python SDK для взаимодействия с сервисами Watson.
- Разработка приложения-переводчика Python с построением гибрида сервисов Watson Speech to Text, Language Translator и Text to Speech.
- Дополнительные ресурсы, упрощающие самостоятельную разработку приложений Watson.

13.1. Введение: IBM Watson и когнитивные вычисления

В главе 1 были рассмотрены некоторые ключевые достижения IBM в области искусственного интеллекта, включая победу над двумя сильнейшими игроками Jeopardy! в матче с призовым фондом в миллион долларов. Суперкомпьютер Watson выиграл соревнование, а компания IBM пожертвовала призовые деньги на благотворительность. Для поиска ответов суперкомпьютер Watson одновременно выполнял сотни алгоритмов языкового анализа для поиска правильных ответов в 200 миллионах страниц контента (включая всю «Википедию»), для хранения которых требовалось 4 терабайта пространства[1,2]. Исследователи IBM обучали Watson с использованием методов машинного обучения (рассматривается в следующей главе[3]) и обучения с подкреплением.

На ранней стадии работы над книгой мы признали стремительно растущую важность Watson и поэтому установили оповещения Google Alerts для Watson и сопутствующих тем. По этим оповещениям, рассылкам и блогам, которые мы отслеживали, были собраны свыше 900 статей, документов и видеороликов, относящихся к Watson. Мы проанализировали много конкурирующих сервисов и пришли к выводу, что политика Watson в стиле «кредитка необязательна» и бесплатный уровень сервиса *Lite*[4] наиболее удобны для разработчиков, желающих даром поэкспериментировать с сервисами Watson.

IBM Watson — облачная платформа когнитивных вычислений, применяемая в широком спектре реальных сценариев. Системы когнитивных вычислений моделируют возможности распознавания закономерностей и принятия решений человеческого мозга для «обучения» в ходе потребления больших объемов данных[5,6,7]. Мы приведем сводку широкого спектра веб-сервисов Watson, практические примеры ее использования и продемонстрируем многие возможности платформы. В таблице на следующей странице представлены некоторые примеры использования Watson в организациях.

---

1 https://www.techrepublic.com/article/ibm-watson-the-inside-story-of-how-the-jeopardy-winning-supercomputer-was-born-and-what-it-wants-to-do-next/.

2 https://en.wikipedia.org/wiki/Watson_(computer).

3 https://www.aaai.org/Magazine/Watson/watson.php, AI Magazine, осень 2010.

4 Всегда проверяйте новейшую версию условий на сайте IBM, так как условия и сервисы могут изменяться.

5 http://whatis.techtarget.com/definition/cognitive-computing.

6 https://en.wikipedia.org/wiki/Cognitive_computing.

7 https://www.forbes.com/sites/bernardmarr/2016/03/23/what-everyone-should-know-about-cognitive-computing.

Watson предлагает замечательный набор функциональных возможностей, которые могут быть встроены в приложения. В этой главе мы создадим учетную запись IBM Cloud[1] и воспользуемся уровнем *Lite* и демонстрационными примерами IBM Watson для экспериментов с различными веб-сервисами: переводом естественного языка, преобразованием речи в текст и текста в речь, пониманием естественных языков, чат-ботами, анализом текста на тональность и распознаванием визуальных объектов в графике и видео. Далее будет приведен краткий обзор других сервисов и инструментов Watson.

Примеры практического применения Watson	
Автономные автомобили	Музыка
Адресная реклама	Обработка естественного языка
Анализ голоса	Обработка изображений
Анализ эмоциональной окраски и настроения	Образование
Безопасность рабочего места	Перевод на другие языки
Виртуальная реальность	Поддержка клиентов
Выявление виртуального запугивания	Понимание естественного языка
Выявление вредоносных программ	Предотвращение мошенничества
Выявление угроз	Прогнозирование погоды
Генетика	Прогнозное техническое обслуживание
Диалоговый интерфейс	Профилактика преступности
Дополненная реальность	Разработка лекарств
Дополненный интеллект	Распознавание лиц
Здравоохранение	Распознавание объектов
Искусственный интеллект	Рекомендации продуктов
Когнитивные вычисления	Роботы и беспилотные аппараты
Компьютерные игры	Спорт
Личные помощники	Субтитры
Машинное обучение	Умные дома
Медицинская визуализация	Управление цепочкой поставок
Медицинская диагностика и лечение	Финансы
	Чат-боты
	IoT (интернет вещей)

[1] Технология IBM Cloud ранее называлась Bluemix. Во многих URL-адресах этой главы все еще встречается название «bluemix».

Для программного доступа к сервисам Watson из кода Python следует установить пакет Watson Developer Cloud Python Software Development Kit (SDK). Затем в качестве практического примера мы разработаем приложение-переводчик, для чего построим гибрид нескольких сервисов Watson. Приложение позволяет англоязычным и испаноязычным пользователям устно общаться друг с другом, невзирая на языковые барьеры. Для этого аудиозаписи на английском (испанском) языке переводятся в текст, который переводится на второй язык, после чего на основании переведенного текста приложение синтезирует английское и испанское аудио.

Сервисы Watson образуют динамичный, непрерывно расширяющийся набор функциональных возможностей. В то время, когда мы работали над книгой, появлялись новые сервисы, а существующие сервисы неоднократно обновлялись и/или удалялись. Описание сервисов Watson и выполняемых действий было точным на момент написания книги. Обновления будут публиковаться по мере необходимости на странице книги на сайте www.deitel.com.

13.2. Учетная запись IBM Cloud и консоль Cloud

Для работы с сервисами Watson уровня Lite вам понадобится бесплатная учетная запись IBM Cloud. На веб-странице с описанием каждого сервиса перечислены его возможности на разных уровнях и указано, какие возможности предоставляет каждый уровень. Хотя возможности сервисов уровня Lite ограниченны, обычно они достаточны для того, чтобы вы познакомились с функциями Watson и начали разрабатывать приложения. Ограничения постоянно изменяются, и вместо упоминания их в тексте приводятся ссылки на веб-страницы сервисов. Кстати, во время написания книги компания IBM серьезно ослабила ограничения на многие сервисы. Платные уровни доступны для использования в коммерческих приложениях. Чтобы создать бесплатную учетную запись IBM Cloud, выполните инструкции по адресу:

https://console.bluemix.net/docs/services/watson/index.html#about

Вы получите сообщение по электронной почте. Выполните содержащиеся в нем инструкции для подтверждения учетной записи. После этого вы получите доступ к консоли IBM Cloud и сможете выйти на *панель управления Watson* по адресу:

https://console.bluemix.net/developer/watson/dashboard

Панель управления позволяет:

- Просмотреть список сервисов Watson.
- Подключиться к сервисам, для использования которых вы уже зарегистрированы.
- Просмотреть различные ресурсы для разработчиков, включая документацию Watson, SDK и различные ресурсы для изучения Watson.
- Просмотреть приложения, созданные вами с использованием Watson.

После регистрации вы получите данные для использования различных сервисов Watson. Для просмотра и управления списком сервисов и ваших регистрационных данных используется *панель управления IBM Cloud* по адресу (этот список можно открыть и щелчком по ссылке Existing Services на панели управления Watson):

https://console.bluemix.net/dashboard/apps

13.3. Сервисы Watson

В этом разделе приведен обзор многих сервисов Watson и ссылки на их подробные описания. Непременно запустите демонстрационные приложения, чтобы увидеть сервисы в действии. Ссылки на документации сервисов Watson и справочник API доступны по адресу:

https://console.bluemix.net/developer/watson/documentation

В сносках приводятся ссылки на подробные описания всех сервисов. Когда вы будете готовы использовать тот или иной сервис, щелкните по кнопке Create на ее описании, чтобы создать свои регистрационные данные.

Watson Assistant

Сервис Watson Assistant[1] помогает строить чат-ботов и виртуальных помощников, упрощающих взаимодействие пользователей с текстом на естественном языке. IBM предоставляет веб-интерфейс, при помощи которого можно *обучить* сервис Watson Assistant для конкретных сценариев, относящихся к вашему приложению. Например, метеорологический чат-бот можно научить

[1] https://console.bluemix.net/catalog/services/watson-assistant-formerly-conversation.

отвечать на запросы типа: «Покажи прогноз погоды в Нью-Йорке». В сервисе клиентской поддержки можно построить чат-бот, отвечающий на вопросы клиентов и при необходимости направляющий их в нужный отдел. Примеры взаимодействий такого рода можно увидеть на сайте:

https://www.ibm.com/watson/services/conversation/demo/index.html#demo

Visual Recognition

Сервис Visual Recognition[1] позволяет приложениям находить и воспринимать информацию в форме графики и видео, включая цвета, объекты, лица, текст, еду и неподходящий контент. IBM предоставляет готовые модели (используемые в демонстрационном приложении сервиса), но вы также можете обучить и использовать собственную модель (см. главу 15). Опробуйте демонстрационное приложение с готовыми изображениями или загрузите собственную графику:

https://watson-visual-recognition-duo-dev.ng.bluemix.net/

Speech to Text

Сервис Speech to Text[2], используемый при построении приложения этой главы, преобразует речевые аудиофайлы в текстовую запись. Сервису можно передать ключевые слова для «прослушивания», а он сообщит, где эти ключевые слова были обнаружены, какова вероятность совпадения и где было совпадение в аудио. Сервис может различать голоса разных людей и использоваться для реализации голосовых приложений, преобразования живого аудио в текст и т. д. Опробуйте демоприложение с готовыми аудиоклипами или загрузите собственное аудио здесь:

https://speech-to-text-demo.ng.bluemix.net/

Text to Speech

Сервис Text to Speech[3], также используемый при построении приложения этой главы, синтезирует речь по тексту. В текст можно включать инструкции на языке *SSML* (*Speech Synthesis Markup Language*) для управления голосовыми

---

[1] https://console.bluemix.net/catalog/services/visual-recognition.

[2] https://console.bluemix.net/catalog/services/speech-to-text.

[3] https://console.bluemix.net/catalog/services/text-to-speech.

модуляциями, ритмом, тональностью и т. д. В настоящее время сервис поддерживает английский язык (США и Великобритания), французский, немецкий, итальянский, испанский, португальский и японский языки. Опробуйте демоприложение с готовым простым текстом, текстом с включениями SSML и вашим собственным текстом здесь:

https://text-to-speech-demo.ng.bluemix.net/

Language Translator

Сервис Language Translator[1], также используемый при построении приложения этой главы, решает две ключевые задачи:

- перевод текста на другие языки;
- распознавание текста, написанного на одном из более чем шестидесяти языков.

Перевод поддерживается между английским и множеством других языков, а также между другими языками. Попробуйте перевести текст на другие языки здесь:

https://language-translator-demo.ng.bluemix.net/

Natural Language Understanding

Сервис Natural Language Understanding[2] анализирует текст и выдает информацию с общей эмоциональной окраской и ключевыми словами, ранжированными по релевантности. Среди прочего, сервис может обнаруживать:

- людей, места, должности, организации, компании и количества;
- категории и концепции (спорт, правительство, политика и т. д.);
- части речи (например, глаголы).

Сервис также можно обучить с ориентацией на специфику конкретной отрасли, конкретного применения и т. д., с Watson Knowledge Studio (см. ниже). Опробуйте демоприложение с готовым простым текстом, скопированным текстом или со ссылкой на статью или документ в интернете здесь:

https://natural-language-understanding-demo.ng.bluemix.net/

---

[1] https://console.bluemix.net/catalog/services/language-translator.

[2] https://console.bluemix.net/catalog/services/natural-language-understanding.

Discovery

Сервис Watson Discovery[1] поддерживает ряд тех же возможностей, что и Natural Language Understanding, предоставляя также корпоративные средства хранения и управления документами. Например, организации могут использовать Watson Discovery для хранения всех своих текстовых документов и применять средства обработки естественного языка ко всей коллекции. Опробуйте демоприложение сервиса, предоставляющее возможность поиска компаний по текстовым новостям:

https://discovery-news-demo.ng.bluemix.net/

Personality Insights

Сервис Personality Insights[2] анализирует текст на предмет индивидуальных черт. Согласно описанию, сервис помогает «помочь получить представление о том, как и почему люди думают, действуют и чувствуют именно так, а не иначе. Этот сервис использует лингвистическую аналитику и теорию индивидуальности для получения информации об атрибутах личности по неструктурированному тексту автора». Полученная информация может использоваться для предоставления адресной рекламы людям, которые с наибольшей вероятностью приобретут эти продукты. Опробуйте следующее демоприложение с твитами от различных учетных записей Twitter, документами, встроенными в приложение, скопированными текстовыми документами или вашей собственной учетной записью Twitter здесь:

https://personality-insights-livedemo.ng.bluemix.net/

Tone Analyzer

Сервис Tone Analyzer[3] анализирует текст на тональность в трех категориях:

- эмоции — гнев, отвращение, страх, радость, печаль;
- социальные предрасположенности — открытость, добросовестность, доброжелательность и эмоциональный диапазон;
- стиль общения — аналитический, уверенный, осторожный.

---

[1] https://console.bluemix.net/catalog/services/discovery.
[2] https://console.bluemix.net/catalog/services/personality-insights.
[3] https://console.bluemix.net/catalog/services/tone-analyzer.

Анализ тональности проводится на уровне документа либо отдельных предложений. Опробуйте следующее демоприложение с примерами твитов, тестовым обзором продукта, сообщением электронной почты или предоставленным вами текстом здесь:

https://tone-analyzer-demo.ng.bluemix.net/

Natural Language Classifier

Сервис Natural Language Classifier[1] *обучается* на предложениях и фразах, специфичных для конкретной области применения, классифицируя предложения и фразы. Например, фразу «У меня возникли проблемы с вашим продуктом» можно отнести к категории «Техническая поддержка», а фразу «Мне прислали неправильный счет» — к категории «Выставление счетов». Обучив классификатор, вы сможете передавать сервису предложения и фразы, а затем использовать функциональность когнитивных вычислений Watson и ваш классификатор для получения оптимальных классификаций и вероятностей совпадения. Полученные классификации и вероятности могут использоваться для определения следующих действий приложения. Так, если в приложении технической поддержки пользователь обращается с вопросом о конкретном продукте, то вы можете воспользоваться сервисом Speech to Text для преобразования вопроса в текст, использовать сервис Natural Language Classifier для классификации текста, а затем передать обращение подходящему специалисту или отделу. Этот сервис *не предоставляет уровень Lite*. В следующем демонстрационном приложении введите вопрос о погоде — сервис сообщит, относится ваш вопрос к температуре или погодным условиям:

https://natural-language-classifier-demo.ng.bluemix.net/

Синхронная и асинхронная функциональность

Многие API, описанные в книге, являются *синхронными* — при вызове функции или метода программа дожидается, пока функция или метод вернет управление, перед переходом к следующей задаче. *Асинхронная* программа может запустить задачу, продолжить выполнение других действий, затем *получить уведомление* о том, что исходная задача завершилась и вернула свои результаты. Многие сервисы Watson предоставляют как синхронные, так и асинхронные API.

---

[1] https://console.bluemix.net/catalog/services/natural-language-classifier.

Демонстрационное приложение Speech to Text — хороший пример асинхронных API. Оно обрабатывает аудиоролик с разговором двух людей. В процессе перевода аудио в текст сервис возвращает промежуточные результаты, даже если он еще не смог различить говорящих. Эти промежуточные результаты выводятся параллельно с продолжением работы сервиса. Иногда в процессе определения говорящих демонстрационное приложение выводит сообщение «Detecting speakers». В конечном итоге сервис отправляет обновленные результаты, после чего приложение заменяет предыдущие результаты преобразования.

Асинхронные API с современными многоядерными компьютерами и компьютерными кластерами могут повысить быстродействие программ. Впрочем, программирование с использованием асинхронных API обычно сложнее программирования с синхронными. При описании установки Watson Developer Cloud Python SDK мы предоставим ссылку на примеры кода SDK на GitHub, в которых представлены синхронные и асинхронные версии ряда сервисов. За полной информацией обращайтесь к справочнику API конкретных сервисов.

13.4. Другие сервисы и инструменты

В этом разделе приведена сводка других сервисов и инструментов Watson.

Watson Studio

Watson Studio[1] — новый интерфейс Watson для создания и управления проектами Watson и взаимодействия с участниками команды этих проектов. Он позволяет добавлять данные, готовить данные для анализа, создавать документы Jupyter Notebook для взаимодействия с вашими данными, создавать и обучать модели, а также использовать функциональность глубокого обучения Watson. Watson Studio предоставляет однопользовательский уровень Lite. После того как вы настроите свой доступ к Watson Studio Lite щелчком на ссылке Create на веб-странице сервиса:

https://console.bluemix.net/catalog/services/data-science-experience,

вы сможете обратиться к Watson Studio по адресу:

https://dataplatform.cloud.ibm.com/

[1] https://console.bluemix.net/catalog/services/data-science-experience.

Watson Studio содержит ряд предварительно настроенных проектов[1]. Чтобы просмотреть их, щелкните на ссылке Create a project:

- Standard — «Работа с любыми активами. Сервисы для аналитических активов добавляются по мере необходимости».
- Data Science — «Анализ данных для выявления закономерностей и обмен результатами с другими».
- Visual Recognition — «Пометка и классификация визуального контента с использованием сервиса Watson Visual Recognition».
- Deep Learning — «Построение нейронных сетей и применение моделей глубокого обучения».
- Modeler — «Построение потоков моделирования для обучения моделей SPSS или проектирование нейронных сетей».
- Business Analytics — «Создание визуальных панелей управления на основе данных для ускорения извлечения информации».
- Data Engineering — «Объединение, очистка, анализ и формирование данных с использованием Data Refinery».
- Streams Flow — «Поглощение и анализ потоковых данных с использованием сервиса Streaming Analytics».

Knowledge Studio

Многие сервисы Watson работают с *предварительно определенными* моделями, но они также предоставляют возможность передачи пользовательских моделей, обученных для конкретных отраслей или применений. Сервис Watson *Knowledge Studio*[2] упрощает построение пользовательских моделей. Он позволяет корпоративным командам совместно работать над созданием и обучением новых моделей, которые затем могут использоваться различными сервисами Watson.

Machine Learning

Сервис *Watson Machine Learning*[3] позволяет подключать прогностическую функциональность через популярные библиотеки машинного обучения,

---

[1] https://dataplatform.cloud.ibm.com/.

[2] https://console.bluemix.net/catalog/services/knowledge-studio.

[3] https://console.bluemix.net/catalog/services/machine-learning.

включая Tensorflow, Keras, scikit-learn и др. В следующих двух главах будут использоваться scikit-learn и Keras.

Knowledge Catalog

Watson Knowledge Catalog[1,2] — средство корпоративного уровня для управления безопасностью, поиска и обмена данными вашей организации, обеспечивающее:

- централизованный доступ к локальным и облачным данным организации, а также к моделям машинного обучения;
- поддержку Watson Studio, посредством которой пользователи могут находить данные, работать с ними и использовать их в проектах машинного обучения;
- политики безопасности, гарантирующие, что доступ к данным будет предоставляться только доверенным пользователям;
- поддержку более 100 операций очистки и первичной обработки данных

  и т. д.

Cognos Analytics

Сервис IBM *Cognos Analytics*[3], имеющий 30-дневный пробный период, использует AI и методы машинного обучения для выявления и визуализации информации в данных без какого-либо программирования с вашей стороны. Он также предоставляет интерфейс естественного языка: пользователь задает вопросы, на которые Cognos Analytics отвечает на основании знаний, собранных в данных.

13.5. Watson Developer Cloud Python SDK

В этом разделе будут установлены модули для полнофункциональной реализации практического примера Watson из следующего раздела. Для удобства программирования IBM предоставляет *Watson Developer Cloud Python SDK* (пакет разработки программного обеспечения). Модуль `watson_developer_cloud`

---

[1] https://medium.com/ibm-watson/introducing-ibm-watson-knowledge-catalog-cf42c13032c1.

[2] https://dataplatform.cloud.ibm.com/docs/content/catalog/overview-wkc.html.

[3] https://www.ibm.com/products/cognos-analytics.

содержит классы, используемые для взаимодействия с сервисами Watson. Мы создадим объекты для каждого необходимого сервиса, после чего будем взаимодействовать с сервисом посредством вызова методов этих объектов. Для установки SDK[1] откройте приглашение Anaconda (Windows; с правами администратора), терминал (macOS/Linux) или командную оболочку (Linux), после чего выполните следующую команду[2]:

```
pip install --upgrade watson-developer-cloud
```

Модули, необходимые для записи и воспроизведения аудио

Вам также понадобятся два дополнительных модуля для записи аудио (PyAudio) и воспроизведения (PyDub). Чтобы установить их, введите следующие команды[3]:

```
pip install pyaudio
pip install pydub
```

Примеры SDK

На GitHub компания IBM предоставляет пример кода, демонстрирующий использование сервисов Watson на базе классов Watson Developer Cloud Python SDK. Пример можно найти по адресу:

https://github.com/watson-developer-cloud/python-sdk/tree/master/examples

13.6. Практический пример: приложение-переводчик

Допустим, вы путешествуете по испаноязычной стране, но не говорите на испанском языке и хотите поговорить с человеком, который не знает английского. Благодаря приложению-переводчику вы сможете говорить на англий-

---

[1] За подробными инструкциями по установке и советами по диагностике проблем обращайтесь по адресу https://github.com/watson-developer-cloud/python-sdk/blob/develop/README.md.

[2] Возможно, пользователям Windows придется установить средства сборки Microsoft C++ по адресу https://visualstudio.microsoft.com/visual-cpp-build-tools/, а затем установить модуль `watson-developer-cloud`.

[3] Возможно, пользователям Mac придется сначала выполнить команду `conda install -c conda-forge portaudio`.

ском языке; приложение переведет текст и воспроизведет его на испанском языке. Собеседник сможет ответить, а приложение переведет и воспроизведет ответ на английском языке. В реализации приложения-переводчика будут задействованы три ключевых сервиса IBM Watson[1]. Приложение позволит людям, говорящим на разных языках, общаться друг с другом практически в реальном времени. Подобное объединение сервисов называется *гибридизацией*. Приложение также использует простые средства работы с файлами, представленные в главе 9.

13.6.1. Перед запуском приложения

Для построения приложения будут использоваться (бесплатные) уровни Lite нескольких сервисов IBM. Прежде чем запускать приложение, не забудьте зарегистрироваться для создания учетной записи IBM Cloud, как обсуждалось ранее в этой главе, чтобы получить регистрационные данные для всех трех сервисов, используемых приложением. Когда у вас появятся регистрационные данные (см. ниже), загрузите их в файл keys.py (из каталога примеров ch13), который импортируется в данном примере. Никогда никому не передавайте ваши регистрационные данные.

В процессе настройки сервисов на странице регистрационных данных каждого сервиса также указывается URL-адрес сервиса. Это URL-адреса по умолчанию, используемые Watson Developer Cloud Python SDK, так что копировать их не нужно. В разделе 13.6.3 будет представлен сценарий SimpleLanguageTranslator.py и подробное описание кода.

Регистрация для сервиса Speech to Text

Приложение использует сервис Watson Speech to Text для перевода английских и испанских аудиофайлов в английский и испанский текст соответственно. Для взаимодействия с сервисом необходимо знать имя пользователя и пароль. Для этого:

- *Создайте экземпляр сервиса*: перейдите на страницу https://console.bluemix.net/catalog/services/speech-to-text и щелкните на кнопке Create в нижней части страницы. Кнопка автоматически генерирует ключ API и открывает учебное руководство по взаимодействию с сервисом Speech to Text.

---

[1] В будущем эти сервисы могут измениться. В таком случае мы опубликуем обновления на веб-странице книги по адресу http://www.deitel.com/books/IntroToPython.

- *Получите регистрационные данные сервиса*: чтобы просмотреть ключ API, щелкните на кнопке Manage в левом верхнем углу страницы. Щелкните на ссылке Show credentials справа от Credentials, скопируйте ключ API и вставьте его в переменную `speech_to_text_key` в файле keys.py из каталога примеров ch13.

Регистрация для сервиса Text to Speech

Приложение использует сервис Watson Text to Speech для синтеза речи по тексту. Для использования сервиса необходимо иметь имя пользователя и пароль. Для этого:

- *Создайте экземпляр сервиса*: перейдите на страницу https://console.bluemix.net/catalog/services/text-to-speech и щелкните на кнопке Create в нижней части страницы. Кнопка автоматически генерирует ключ API и открывает учебное руководство по взаимодействию с сервисом Text to Speech.
- *Получите регистрационные данные сервиса*: чтобы просмотреть ключ API, щелкните на кнопке Manage в левом верхнем углу страницы. Щелкните на ссылке Show credentials справа от Credentials, скопируйте ключ API и вставьте его в переменную `text_to_speech_key` в файле keys.py из каталога примеров ch13.

Регистрация для сервиса Language Translator

Приложение использует сервис Watson Language Translator для передачи текста Watson и получения текста, переведенного на другой язык. Для использования этого сервиса необходимо получить ключ API. Для этого:

- *Создайте экземпляр сервиса*: перейдите на страницу https://console.bluemix.net/catalog/services/language-translator и щелкните на кнопке Create в нижней части страницы. Кнопка автоматически генерирует ключ API и открывает страницу для управления экземпляром сервиса.
- *Получите регистрационные данные сервиса*: щелкните на ссылке Show credentials справа от Credentials, скопируйте ключ API и вставьте его в переменную `translate_key` в файле keys.py из каталога примеров ch13.

Получение регистрационных данных

Чтобы просмотреть регистрационные данные в любой момент, щелкните на вкладке сервиса по адресу:

https://console.bluemix.net/dashboard/apps

13.6.2. Пробный запуск приложения

После включения регистрационных данных в сценарий откройте приглашение Anaconda (Windows), терминал (macOS/Linux) или командную оболочку (Linux), после чего запустите сценарий[1] следующей командой из каталога ch13:

```
ipython SimpleLanguageTranslator.py
```

Обработка вопроса

Логика приложения состоит из 10 основных шагов, выделенных комментариями в коде. **Шаг 1** запрашивает и сохраняет вопрос — приложение выводит приглашение:

```
Press Enter then ask your question in English,
```

ожидая нажатия Enter. Когда клавиша нажата, приложение выводит сообщение:

```
Recording 5 seconds of audio
```

Пользователь произносит вслух свой вопрос — например, «Where is the closest bathroom?». Через 5 секунд приложение выводит следующее сообщение:

```
Recording complete
```

На **шаге 2** приложение взаимодействует с сервисом Watson Speech to Text для перевода записанного аудио в текст и выводит результат:

```
English: where is the closest bathroom
```

На **шаге 3** приложение при помощи сервиса Watson Language Translator переводит английский текст на испанский язык и выводит перевод, возвращенный Watson:

```
Spanish: ¿Dónde está el baño más cercano?
```

---

[1] Модуль `pydub.playback`, используемый в приложении, выдает предупреждение при запуске сценария. Это предупреждение относится к функциональности модуля, которую мы не используем, поэтому на него можно не обращать внимания. Чтобы избавиться от предупреждения, можно установить `ffmpeg` для Windows, macOS или Linux с сайта https://www.ffmpeg.org.

Шаг 4 передает испанский текст сервису Watson Text to Speech для преобразования в аудиофайл.

Шаг 5 воспроизводит полученный аудиофайл на испанском языке.

Обработка ответа

Теперь можно переходить к обработке ответа испаноговорящего пользователя.

На **шаге 6** выводится сообщение:

```
Press Enter then speak the Spanish answer
```

Приложение ожидает нажатия Enter. Когда клавиша нажата, приложение выводит сообщение:

```
Recording 5 seconds of audio
```

Собеседник произносит свой ответ на испанском языке. Мы не говорим на испанском, поэтому сервис Watson Text to Speech используется для *воспроизведения заранее записанного ответа* «El baño más cercano estáen el restaurante». Ответ воспроизводится достаточно громко, чтобы он мог быть записан с микрофона вашего компьютера. Мы предоставили заранее записанное аудио в файле SpokenResponse.wav в каталоге ch13. Если вы используете этот файл, то быстро воспроизведите его после нажатия Enter, потому что приложение ведет запись в течение всего 5 секунд[1]. Чтобы аудиофайл загружался и воспроизводился достаточно быстро, его стоит воспроизвести до нажатия Enter для начала записи. Через 5 секунд приложение выводит сообщение:

```
Recording complete
```

На **шаге 7** приложение взаимодействует с сервисом Watson Speech to Text для перевода испанской речи в текст и выводит результат:

```
Spanish response: el baño más cercano está en el restaurante
```

---

[1] Для простоты мы настроили приложение так, чтобы запись велась в течение 5 секунд. Для управления продолжительностью записи можно воспользоваться переменной `SECONDS` в функции `record_audio`. Можно создать систему записи, которая начинает запись при обнаружении звука и завершает после паузы определенной продолжительности, но это приведет к усложнению кода.

На **шаге 8** приложение взаимодействует с сервисом Watson Language Translator для перевода испанского текста на английский язык и выводит результат:

```
English response: The nearest bathroom is in the restaurant
```

Шаг 9 передает английский текст сервису Watson Text to Speech для преобразования в аудиофайл.

Шаг 10 воспроизводит полученный аудиофайл на английском языке.

13.6.3. Сценарий SimpleLanguageTranslator.py

В этом разделе будет представлен исходный код сценария SimpleLanguageTranslator.py, который мы разделили на небольшие фрагменты с последовательной нумерацией. При этом мы воспользуемся нисходящим методом, как было сделано в главе 3. Верхний уровень выглядит так:

Создание приложения-переводчика для общения собеседников, говорящих на английском и испанском языке.

Первое уточнение выглядит так:

Перевод вопроса, заданного на английском языке, в испанскую речь.

Перевод ответа на испанском языке в английскую речь.

Первая строка второй стадии разбивается на пять шагов:

Шаг 1: запрос и запись английской речи в аудиофайл.

Шаг 2: перевод английской речи в английский текст.

Шаг 3: перевод английского текста в испанский текст.

Шаг 4: синтез испанской речи по тексту и сохранение ее в аудиофайле.

Шаг 5: воспроизведение аудиофайла с испанской речью.

Вторая строка второй стадии также разбивается на пять шагов:

Шаг 6: запрос и запись испанской речи в аудиофайл.

Шаг 7: перевод испанской речи в испанский текст.

Шаг 8: перевод испанского текста в английский текст.

Шаг 9: синтез английского текста в английскую речь и сохранение ее в аудиофайле.

Шаг 10: воспроизведение аудиофайла с английской речью.

Методология нисходящей разработки наглядно демонстрирует все преимущества метода «разделяй и властвуй», позволяя сосредоточиться на меньших частях более серьезной проблемы.

В сценарии этого раздела мы реализуем 10 шагов, указанных при втором уточнении. **Шаги 2** и **7** используют сервис Watson Speech to Text, **шаги 3** и **8** — сервис Watson Language Translator и **шаги 4** и **9** — сервис Watson Text to Speech.

Импортирование классов Watson SDK

В строках 4–6 импортируются классы из модуля `watson_developer_cloud`, установленного с Watson Developer Cloud Python SDK. Каждый из этих классов использует регистрационные данные Watson, полученные ранее для взаимодействия с соответствующими сервисами Watson:

- Класс `SpeechToTextV1`[1] позволяет передать аудиофайл сервису Watson Speech to Text и получить документ JSON[2] с результатом преобразования речи в текст.
- Класс `LanguageTranslatorV3` позволяет передать текст сервису Watson Language Translator и получить документ JSON с переведенным текстом.
- Класс `TextToSpeechV1` позволяет передать текст сервису Watson Text to Speech и получить аудиофайл с текстом на заданном языке.

```
1 # SimpleLanguageTranslator.py
2 """Использование API IBM Watson Speech to Text, Language Translator и Text
      to Speech
3    для общения на английском и испанском языке."""
4 from watson_developer_cloud import SpeechToTextV1
5 from watson_developer_cloud import LanguageTranslatorV3
6 from watson_developer_cloud import TextToSpeechV1
```

[1] V1 в имени класса обозначает номер версии сервиса. В процессе обновления версий компания IBM добавляет новые классы в модуль `watson_developer_cloud` вместо изменения существующих классов. Это гарантирует, что существующие приложения сохранят работоспособность при обновлении сервисов. На момент написания книги сервисы Speech to Text и Text to Speech существовали в версии 1 (V1), а сервис Language Translator service — в версии 3 (V3).

[2] Формат JSON был описан в главе 12.

Другие импортируемые модули

В строке 7 импортируется файл keys.py с регистрационными данными Watson. Строки 8–11 импортируют модули, обеспечивающие функциональность обработки аудио нашего приложения:

- Модуль `pyaudio` позволяет записывать аудио с микрофона.
- Модули `pydub` и `pydub.playback` позволяют загружать и воспроизводить аудиофайлы.
- Модуль `wave` стандартной библиотеки Python позволяет сохранять файлы в формате WAV (Waveform Audio File Format) — популярном звуковом формате, разработанном компаниями Microsoft и IBM. Приложение использует модуль `wave` для сохранения записанного аудио в файле .wav, который отправляется сервису Watson Speech to Text для преобразования в текст.

```
 7 import keys  # Содержит ключи API для обращения к сервисам Watson

 8 import pyaudio  # Используется для записи с микрофона
 9 import pydub  # Используется для загрузки файла WAV
10 import pydub.playback  # Используется для воспроизведения файла WAV
11 import wave  # Используется для сохранения файла WAV
12
```

Основная программа: функция run_translator

Рассмотрим основную часть программы, определенную в функции `run_translator` (строки 13–54), которая вызывает функции, определяемые далее в сценарии. Для целей нашего обсуждения функция `run_translator` была разбита на 10 шагов. На **шаге 1** (строки 15–17) выводится сообщение на английском языке с предложением нажать Enter и произнести вопрос. Функция `record_audio` записывает аудио в течение 5 секунд, после чего сохраняет его в файле english.wav:

```
13 def run_translator():
14     """Вызывает функции, взаимодействующие с сервисами Watson."""
15     # Шаг 1: запрос и запись английской речи в аудиофайл.
16     input('Press Enter then ask your question in English')
17     record_audio('english.wav')
18
```

На **шаге 2** вызывается функция `speech_to_text`, которой передается файл english.wav для преобразования в текст, с использованием *предопределенной*

модели `'en-US_BroadbandModel'`[1]. Затем приложение выводит полученный текст:

```
19     # Шаг 2: перевод английской речи в английский текст.
20     english = speech_to_text(
21         file_name='english.wav', model_id='en-US_BroadbandModel')
22     print('English:', english)
23
```

На **шаге 3** вызывается функция `translate`, которой для перевода передается текст с **шага 2**. Сервис Language Translator используется для перевода текста с использованием *предопределенной* модели `'en-es'` для перевода с английского (`en`) языка на испанский (`es`), после чего выводится испанский перевод:

```
24     # Шаг 3: перевод английского текста в испанский текст.
25     spanish = translate(text_to_translate=english, model='en-es')
26     print('Spanish:', spanish)
27
```

На **шаге 4** вызывается функция `text_to_speech`, которой передается испанский текст с **шага 3**, чтобы сервис Text to Speech зачитал этот текст с использованием голоса `'es-US_SofiaVoice'`. Также указывается файл для сохранения аудио:

```
28     # Шаг 4: синтез испанской речи по тексту и сохранение ее в аудиофайле.
29     text_to_speech(text_to_speak=spanish, voice_to_use='es-US_SofiaVoice',
30         file_name='spanish.wav')
31
```

На **шаге 5** вызывается функция `play_audio` для воспроизведения файла `'spanish.wav'`, содержащего испанское аудио для текста, полученного на **шаге 3**.

```
32     # Шаг 5: воспроизведение аудиофайла с испанской речью.
33     play_audio(file_name='spanish.wav')
34
```

Наконец, **шаги 6–10** повторяют то, что делалось в **шагах 1–5**, но уже для перевода речи на испанском языке в речь на английском языке. **Шаг 6** записывает аудио на *испанском* языке.

---

[1] Для большинства языков сервис Watson Speech to Text поддерживает *широковещательные* и *узковещательные* модели. Термины относятся к качеству звука: для аудиоматериалов, записанных при 16 кГц и выше, IBM рекомендует использовать широковещательные модели. В данном приложении аудио записывалось на частоте 44,1 кГц.

Шаг 7 преобразует аудио на испанском языке в испанский текст с использованием сервиса предопределенной модели `'es-ES_BroadbandModel'` сервиса Speech to Text.

Шаг 8 переводит испанский текст в английский текст с использованием модели `'es-en'` сервиса Language Translator.

Шаг 9 синтезирует английское аудио с использованием голоса `'en-US_AllisonVoice'` сервиса Text to Speech. **Шаг 10** воспроизводит аудио на английском языке.

```
35     # Шаг 6: запрос и запись испанской речи в аудиофайл.
36     input('Press Enter then speak the Spanish answer')
37     record_audio('spanishresponse.wav')
38
39     # Шаг 7: перевод испанской речи в испанский текст.
40     spanish = speech_to_text(
41         file_name='spanishresponse.wav', model_id='es-ES_BroadbandModel')
42     print('Spanish response:', spanish)
43
44     # Шаг 8: перевод испанского текста в английский текст.
45     english = translate(text_to_translate=spanish, model='es-en')
46     print('English response:', english)
47
48     # Шаг 9: синтез английского текста в английскую речь и сохранение
       # ее в аудиофайле.
49     text_to_speech(text_to_speak=english,
50         voice_to_use='en-US_AllisonVoice',
51         file_name='englishresponse.wav')
52
53     # Шаг 10: воспроизведение аудиофайла с английской речью.
54     play_audio(file_name='englishresponse.wav')
55
```

А теперь реализуем функции, вызываемые в коде **шагов 1–10**.

Функция speech_to_text

Чтобы обратиться к сервису Watson Speech to Text, функция `speech_to_text` (строки 56–87) создает объект `SpeechToTextV1` с именем `stt` (сокращение от «Speech To Text»); при этом в аргументе передается ключ API, созданный ранее. Команда `with` (строки 62–65) открывает аудиофайл, заданный параметром `file_name`, и присваивает полученный объект файла переменной `audio_file`. Режим открытия файла `'rb'` означает, что мы собираемся читать (`r`) двоичные данные (`b`) — аудиофайлы хранятся в виде набора байтов в дво-

ичном формате. Затем строки 64–65 используют *метод* `recognize` объекта `SpeechToTextV1` для обращения к сервису Speech to Text. Метод получает три ключевых аргумента:

- `audio` — файл (`audio_file`), передаваемый сервису Speech to Text;
- `content_type` — тип содержимого файла; `'audio/wav'` означает, что это аудиофайл, хранящийся в формате WAV[1];
- `model` определяет модель разговорного языка, которая будет использоваться сервисом для распознавания речи и преобразования ее в текст. Приложение использует предопределенные модели — либо `'en-US_BroadbandModel'` (для английского языка), либо `'es-ES_BroadbandModel'` (для испанского языка).

```
56 def speech_to_text(file_name, model_id):
57     """Использует сервис Watson Speech to Text для преобразования аудиофайла
          в текст."""
58     # Создать клиента Watson Speech to Text
59     stt = SpeechToTextV1(iam_apikey=keys.speech_to_text_key)
60
61     # Открыть аудиофайл
62     with open(file_name, 'rb') as audio_file:
63         # Передать файл Watson для преобразования
64         result = stt.recognize(audio=audio_file,
65             content_type='audio/wav', model=model_id).get_result()
66
67     # Получить список 'results'. Список может содержать промежуточные
68     # или окончательные результаты. Нас интересуют только окончательные
69     # результаты, поэтому список состоит из одного элемента.
70     results_list = result['results']
71
72     # Получить окончательный результат распознавания текста - единственный
       # элемент списка.
73     speech_recognition_result  = results_list[0]
74
75     # Получить список 'alternatives'. Список может содержать несколько
76     # альтернативных вариантов. Нам альтернативы не нужны, поэтому
77     # список состоит из одного элемента.
78     alternatives_list = speech_recognition_result['alternatives']
79
80     # Получить единственный альтернативный текст из alternatives_list.
81     first_alternative = alternatives_list[0]
```

[1] Типы аудиовизуального содержимого ранее назывались *типами MIME* (*Multipurpose Internet Mail Extensions*) — стандарт, определяющий форматы данных, которые могут использоваться программами для правильной интерпретации данных.

```
82
83     # Получить значение ключа 'transcript', содержащее результат
84     # преобразования аудио в текст.
85     transcript = first_alternative['transcript']
86
87     return transcript  # Вернуть текстовую запись аудио
88
```

Метод `recognize` возвращает объект `DetailedResponse`. Его метод `getResult` возвращает документ JSON с преобразованным текстом, который будет сохранен в `result`. Разметка JSON выглядит примерно так (ее конкретный вид зависит от заданного вопроса):

```
{
  "results": [                                              Строка 70
    {                                                       Строка 73
      "alternatives": [                                     Строка 78
        {                                                   Строка 81
          "confidence": 0.983,
          "transcript": "where is the closest bathroom " Строка 85
        }
      ],
      "final": true
    }
  ],
  "result_index": 0
}
```

Разметка JSON содержит *вложенные* словари и списки. Чтобы упростить понимание структуры данных, в строках 70–85 используются отдельные небольшие конструкции для «отделения» отдельных частей вплоть до получения преобразованного текста (`"where is the closest bathroom "`), который будет возвращен в итоге. Прямоугольники вокруг отдельных частей JSON и номера строк в этих прямоугольниках соответствуют командам в строках 70–85. Команды работают следующим образом:

- Строка 70 присваивает `results_list` список, связанный с ключом `'results'`:

  ```
  results_list = result['results']
  ```

  В зависимости от аргументов, переданных методу `recognize`, этот список может содержать промежуточные или окончательные результаты. Промежуточные результаты могут быть полезны, например, при преобразовании

в текст живого аудио (скажем, новостного репортажа). Мы запросили только окончательные результаты, поэтому список состоит только из одного элемента[1].

- Строка 73 присваивает переменной `speech_recognition_result` итоговый результат распознавания речи — единственный элемент `results_list`:

```
speech_recognition_result  = results_list[0]
```

- Строка 78:

```
alternatives_list = speech_recognition_result['alternatives']
```

присваивает `alternatives_list` список, связанный с ключом `'alternatives'`. Этот список может содержать несколько альтернативных вариантов преобразования в зависимости от аргументов метода `recognize`. С переданными аргументами будет получен список из одного элемента.

- Строка 81 присваивает `first_alternative` единственный элемент `alternatives_list`:

```
first_alternative = alternatives_list[0]
```

- Строка 85 присваивает `transcript` значение, связанное с ключом `'transcript'`, которое содержит результат преобразования аудио в текст:

```
transcript = first_alternative['transcript']
```

- Наконец, строка 87 возвращает результат преобразования аудио в текст.

Строки 70–85 можно заменить более компактной записью:

```
return result['results'][0]['alternatives'][0]['transcript']
```

Однако мы предпочитаем более простые команды.

Функция translate

Чтобы обратиться к сервису Watson Language Translator, функция `translate` (строки 89–111) сначала создает объект `LanguageTranslatorV3` с именем `language_translator`; в аргументах передается версия сервиса (`'2018-`

[1] За подробной информацией об аргументах метода `recognize` и подробным описанием ответов JSON обращайтесь по адресу https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/python.html?python#recognize-sessionless.

`05-31'`[1]), созданный ранее ключ API и URL-адрес сервиса. В строках 93–94 *метод* `translate` объекта `LanguageTranslatorV3` используется для обращения к сервису Language Translator с передачей двух ключевых аргументов:

- `text` — строка для перевода на другой язык;
- `model_id` — предопределенная модель, используемая сервисом Language Translator для понимания исходного текста и перевода его на нужный язык. В этом приложении будет использоваться одна из *предопределенных* моделей IBM — `'en-es'` (с английского на испанский) или `'es-en'` (с испанского на английский).

```
89 def translate(text_to_translate, model):
90     """Использует сервис Watson Language Translator для перевода с английского
91        на испанский (en-es) или наоборот (es-en) по правилам модели."""
92     # Создать клиента Watson Translator
93     language_translator = LanguageTranslatorV3(version='2018-05-31',
94         iam_apikey=keys.translate_key)
95
96     # Выполнить перевод
97     translated_text = language_translator.translate(
98         text=text_to_translate, model_id=model).get_result()
99
100     # Получить список 'translations'. Если аргумент text метода translate
101     # содержит несколько строк, список будет содержать несколько элементов.
102     # Мы передали одну строку, поэтому в списке один элемент.
103     translations_list = translated_text['translations']
104
105     # Получить единственный элемент translations_list
106     first_translation = translations_list[0]
107
108     # Получить значение для ключа 'translation', то есть переведенный текст.
109     translation = first_translation['translation']
110
111     return translation  # Вернуть переведенную строку
112
```

Метод возвращает объект `DetailedResponse`. Метод `getResult` этого объекта возвращает документ JSON следующего вида:

---

[1] По данным справочника API сервиса Language Translator, `'2018-05-31'` является строкой текущей версии на момент написания книги. IBM изменяет строку версии только при внесении изменений API, не обладающих обратной совместимостью. Но даже в этом случае сервис ответит на ваши вызовы, используя версию API, заданную в строке версии. За дополнительной информацией обращайтесь по адресу https://www.ibm.com/watson/developercloud/language-translator/api/v3/python.html?python#versioning.

```
{
  "translations": [                                        Строка 103
    {                                                      Строка 106
      "translation": "¿Dónde está el baño más cercano?"    Строка 109
    }
  ],
  "word_count": 5,
  "character_count": 30
}
```

Разметка JSON, которую вы получите в ответе, зависит от заданного вопроса; как и в предыдущем случае, она содержит вложенные словари и списки. В строках 103–109 небольшие конструкции используются для выбора переведенного текста `"¿Dónde está el baño más cercano?"`. Прямоугольники вокруг отдельных частей JSON и номера строк в этих прямоугольниках соответствуют командам в строках 103–109. Команды работают следующим образом:

- Строка 103 получает список `'translations'`:

  ```
  translations_list = translated_text['translations']
  ```

  Если аргумент `text` метода `translate` содержит несколько строк, то список будет содержать несколько элементов. Мы передали только одну строку, поэтому список содержит только один элемент.

- Строка 106 получает единственный элемент `translations_list`:

  ```
  first_translation = translations_list[0]
  ```

- Строка 109 получает значение, связанное с ключом `'translation'`, то есть переведенный текст:

  ```
  translation = first_translation['translation']
  ```

- Строка 111 возвращает переведенную строку.

Строки 103–109 можно заменить более компактной записью:

```
return translated_text['translations'][0]['translation']
```

Но как и в предыдущем случае, мы предпочитаем более простые команды.

Функция text_to_speech

Чтобы обратиться к сервису Watson Text to Speech, функция `text_to_speech` (строки 113–122) создает объект `TextToSpeechV1` с именем `tts` (сокращение

от «Text To Speech»); при этом в аргументе передается ключ API, созданный ранее. Команда `with` открывает аудиофайл, заданный параметром `file_name`, и присваивает объект файла переменной `audio_file`. Режим открытия файла `'wb'` открывает файл для записи (`w`) в двоичном формате (`b`). В файл будет записано содержимое аудиоданных, возвращенных сервисом Speech to Text.

```
113 def text_to_speech(text_to_speak, voice_to_use, file_name):
114     """Использует сервис Watson Text to Speech для преобразования текста
115        в речь и сохранения результата в файле WAV."""
116     # Создать клиента Text to Speech
117     tts = TextToSpeechV1(iam_apikey=keys.text_to_speech_key)
118
119     # Открыть файл и записать синтезированный аудиоконтент в файл
120     with open(file_name, 'wb') as audio_file:
121         audio_file.write(tts.synthesize(text_to_speak,
122             accept='audio/wav', voice=voice_to_use).get_result().content)
123
```

В строках 121–122 вызываются два метода. Сначала приложение обращается к сервису Speech to Text вызовом *метода* `synthesize` объекта `TextToSpeechV1`; методу передаются три аргумента:

- `text_to_speak` — строка с произносимым текстом;
- `accept` — ключевой аргумент с типом аудиоданных, которые должен вернуть сервис Speech to Text; как и прежде, `'audio/wav'` обозначает аудиофайл в формате WAV;
- `voice` — ключевой аргумент, определяющий один из предопределенных голосов сервиса Speech to Text. В нашем приложении для английской речи будет использоваться голос `'en-US_AllisonVoice'`, а для испанской — голос `'es-US_SofiaVoice'`. Watson предоставляет много мужских и женских голосов для разных языков[1].

Объект ответа `DetailedResponse` содержит аудиофайл с синтезированной речью, для получения которого можно воспользоваться вызовом `get_result`. Обратимся к атрибуту `content` полученного файла, чтобы получить байты аудиоданных и передать их методу `write` объекта `audio_file` для сохранения в файле .wav.

---

[1] Полный список доступен по адресу https://www.ibm.com/watson/developercloud/text-to-speech/api/v1/python.html?python#get-voice. Попробуйте поэкспериментировать с другими голосами.

Функция record_audio

Модуль pyaudio позволяет записывать аудио с микрофона. Функция record_audio (строки 124–154) определяет несколько констант (строки 126–130) для настройки потока аудиоданных, поступающего с микрофона вашего компьютера. Мы воспользовались настройками из электронной документации модуля pyaudio:

- FRAME_RATE: 44100 кадров соответствуют частоте 44,1 кГц, стандартной для аудиоматериалов с CD-качеством.
- CHUNK: 1024 — количество кадров, передаваемых программе за один раз.
- FORMAT: pyaudio.paInt16 — размер каждого кадра (в данном случае 16 бит, то есть 2-байтовые целые числа).
- CHANNELS: 2 — количество точек данных на кадр.
- SECONDS: 5 — продолжительность записи аудио в приложении (в секундах).

```
124 def record_audio(file_name):
125     """Использует pyaudio для записи 5 секунд аудио в файл WAV."""
126     FRAME_RATE = 44100  # Количество кадров в секунду
127     CHUNK = 1024  # Количество кадров, читаемых за один раз
128     FORMAT = pyaudio.paInt16  # Каждый кадр - 16-разрядное (2-байтовое) целое
                                  # число
129     CHANNELS = 2  # 2 точки данных на кадр
130     SECONDS = 5  # Общее время записи
131
132     recorder = pyaudio.PyAudio()  # Открывает/закрывает аудиопотоки
133
134     # Настройка и открытие аудиопотока для записи (input=True)
135     audio_stream = recorder.open(format=FORMAT, channels=CHANNELS,
136         rate=FRAME_RATE, input=True, frames_per_buffer=CHUNK)
137     audio_frames = []  # Для хранения низкоуровневого ввода с микрофона
138     print('Recording 5 seconds of audio')
139
140     # Прочитать 5 секунд аудио блоками с размером CHUNK
141     for i in range(0, int(FRAME_RATE * SECONDS / CHUNK)):
142         audio_frames.append(audio_stream.read(CHUNK))
143
144     print('Recording complete')
145     audio_stream.stop_stream()  # Остановить запись
146     audio_stream.close()
147     recorder.terminate()  # Освободить ресурсы, используемые PyAudio
148
149     # Сохранить audio_frames в файле WAV
150     with wave.open(file_name, 'wb') as output_file:
151         output_file.setnchannels(CHANNELS)
```

```
152         output_file.setsampwidth(recorder.get_sample_size(FORMAT))
153         output_file.setframerate(FRAME_RATE)
154         output_file.writeframes(b''.join(audio_frames))
155
```

Строка 132 создает объект `PyAudio`, от которого мы будем получать входной поток для записи аудио с микрофона. В строках 135–136 *метод* `open` объекта `PyAudio` используется для открытия входного потока, параметры которого определяются константами `FORMAT`, `CHANNELS`, `FRAME_RATE` и `CHUNK`. Передача ключевого аргумента `input` со значением `True` означает, что поток будет использоваться для *получения* входных аудиоданных. Метод `open` возвращает объект `Stream` модуля `pyaudio` для взаимодействия с потоком.

В строках 141–142 *метод* `read` объекта `Stream` используется для получения 1024 (то есть `CHUNK`) кадров из входного потока, которые присоединяются к списку `audio_frames`. Чтобы определить общее количество итераций цикла, необходимых для производства 5 секунд аудио при `CHUNK` кадров за раз, мы умножаем `FRAME_RATE` на `SECONDS`, а затем делим результат на `CHUNK`. После того как чтение данных завершится, строка 145 вызывает *метод* `stop_stream` объекта `Stream` для завершения записи, строка 146 закрывает объект `Stream` вызовом *метода* `close` объекта `Stream`, а строка 147 вызывает *метод* `terminate` объекта `PyAudio` для освобождения аудиоресурсов, задействованных в управлении аудиопотоком.

Команда `with` в строках 150–154 использует функцию `open` модуля `wave` для открытия файла WAV, заданного `file_name` для записи в двоичном формате (`'wb'`). Строки 151–153 задают количество каналов файла WAV, размер данных (полученный *методом* `get_sample_size` объекта `PyAudio`) и частоту дискретизации. Затем строка 154 записывает аудиоданные в файл. Выражение `b''.join(audio_frames)` осуществляет конкатенацию всех байтов кадров в *строку байтов*. Присоединение в начало строки префикса `b` показывает, что это строка байтов, а не строка символов.

Функция play_audio

Чтобы воспроизвести аудиофайлы, возвращенные сервисом Watson Text to Speech, воспользуемся функциональностью модулей `pydub` и `pydub.playback`. Сначала строка 158 использует *метод* `from_wav` класса `AudioSegment` для загрузки файла WAV. Метод возвращает новый объект `AudioSegment`, представляющий аудиофайл. Чтобы воспроизвести `AudioSegment`, строка 159 вызывает *функцию* `play` модуля `pydub.playback` и передает `AudioSegment` в аргументе.

```
156 def play_audio(file_name):
157     """Использует модуль pydub (pip install pydub) для воспроизведения
          файла WAV."""
158     sound = pydub.AudioSegment.from_wav(file_name)
159     pydub.playback.play(sound)
160
```

Выполнение функции run_translator

Функция `run_translator` вызывается при выполнении SimpleLanguageTranslator.py в виде сценария:

```
161 if __name__ == '__main__':
162     run_translator()
```

Хочется надеяться, что выбранный нами метод нисходящей разработки с этим (довольно большим) сценарием помог вам разобраться в его логике. Многие шаги четко соответствуют ключевым сервисам Watson, что позволяет разработчику быстро построить мощное гибридное приложение.

13.7. Ресурсы Watson

IBM предоставляет разработчикам доступ к широкому спектру ресурсов для ознакомления с сервисами и их применения при построении приложений.

Документация сервисов Watson

Документация сервисов Watson доступна по адресу:

https://console.bluemix.net/developer/watson/documentation

Для каждого сервиса доступна документация и ссылки на справочники API. В документацию каждого сервиса обычно включаются:

- учебник для начинающих;
- видеообзор сервиса;
- ссылка на демонстрационное приложение сервиса;
- ссылки на более конкретные инструкции и учебные документы;
- примеры приложений;

- дополнительные ресурсы (более подробные учебные руководства, видеоролики, посты в блогах и т. д.).

В справочнике API каждого сервиса приведены все подробности взаимодействия с сервисом на разных языках, включая Python. Щелкните на вкладке Python, чтобы просмотреть документацию, относящуюся к Python, и соответствующие примеры кода для Watson Developer Cloud Python SDK. В справочнике API описаны все параметры обращения к сервису, разновидности ответов, которые он может вернуть, примеры ответов и т. д.

Watson SDK

Для разработки сценария этой главы использовался пакет Watson Developer Cloud Python SDK. Также существуют SDK для многих других языков и платформ. Полный список доступен по адресу:

https://console.bluemix.net/developer/watson/sdks-and-tools

Образовательные ресурсы

На странице Learning Resources

https://console.bluemix.net/developer/watson/learning-resources

приведены ссылки на следующие ресурсы:

- Сообщения в блогах о функциональности Watson, а также об использовании Watson и AI в отрасли.
- Репозиторий Watson на GitHub (средства разработчика, SDK и примеры кода).
- Канал Watson на YouTube (см. ниже).
- Паттерны, которые IBM называет «ориентирами для решения сложных задач из области программирования». Некоторые паттерны реализованы на Python, но, возможно, другие паттерны пригодятся вам при проектировании и реализации приложений на языке Python.

Видеоролики о Watson

Канал Watson на YouTube:

https://www.youtube.com/user/IBMWatsonSolutions/

содержит сотни видеороликов, демонстрирующих различные аспекты использования Watson. Также на канале доступны видеообзоры с примерами использования Watson.

Публикации IBM Redbook

В следующих публикациях IBM Redbook приведены подробные описания сервисов IBM Cloud и Watson, которые помогут вам повысить уровень владения Watson:

- Основы разработки приложений для IBM Cloud:

  http://www.redbooks.ibm.com/abstracts/sg248374.html

- Построение когнитивных приложений с использованием сервисов IBM Watson: том 1, «**Getting Started**»: http://www.redbooks.ibm.com/abstracts/sg248387.html
- Построение когнитивных приложений с использованием сервисов IBM Watson: том 2, «**Conversation**» (теперь называется Watson Assistant): http://www.redbooks.ibm.com/abstracts/sg248394.html
- Построение когнитивных приложений с использованием сервисов IBM Watson: том 3, «**Visual Recognition**»: http://www.redbooks.ibm.com/abstracts/sg248393.html
- Построение когнитивных приложений с использованием сервисов IBM Watson: том 4, «**Natural Language Classifier**»: http://www.redbooks.ibm.com/abstracts/sg248391.html
- Построение когнитивных приложений с использованием сервисов IBM Watson: том 5, «**Language Translator**»: http://www.redbooks.ibm.com/abstracts/sg248392.html
- Построение когнитивных приложений с использованием сервисов IBM Watson: том 6, «**Speech to Text and Text to Speech**»: http://www.redbooks.ibm.com/abstracts/sg248388.html
- Построение когнитивных приложений с использованием сервисов IBM Watson: том 7, «**Natural Language Understanding**»: http://www.redbooks.ibm.com/abstracts/sg248398.html

13.8. Итоги

В этой главе представлена платформа когнитивных вычислений IBM Watson и приведен обзор широкого спектра ее сервисов. Вы узнали, что Watson предоставляет многие интересные возможности, которые могут интегрироваться в ваши приложения. Для обучения и экспериментов IBM предоставляет бесплатные уровни доступа Lite. Чтобы воспользоваться ими, необходимо создать учетную запись IBM Cloud. Мы использовали демонстрационные приложения Watson для экспериментов с различными сервисами: переводом естественного языка, преобразованием речи в текст и текста в речь, пониманием естественного языка, чат-ботами, анализом текста на тональность и распознаванием визуальных объектов в графике и видео.

Мы установили пакет Watson Developer Cloud Python SDK для программного доступа к сервисам Watson из кода Python. В приложении-переводчике несколько сервисов Watson объединены в гибридное приложение, при помощи которого англоязычные и испаноязычные пользователи могут легко общаться друг с другом. Записи аудио на английском и испанском языке преобразовывались в текст, текст переводился на другой язык, а затем английская и испанская речь синтезировалась из переведенного текста. В завершающей части главы описаны различные ресурсы Watson, включая документацию, блоги, репозиторий Watson на GitHub, канал Watson на YouTube, паттерны, реализованные на Python (и других языках), и документы IBM Redbook.

14

Машинное обучение: классификация, регрессия и кластеризация

В этой главе...

- Использование scikit-learn с популярными наборами данных для проведения исследований из области машинного обучения.
- Визуализация и исследование данных средствами Seaborn и Matplotlib.
- Машинное обучение с учителем на примере классификации методом *k* ближайших соседей и линейной регрессии.
- Выполнение множественной классификации с набором данных Digits.
- Разделение набора данных на обучающий, тестовый и проверочный наборы.
- Настройка гиперпараметров модели с использованием *k*-проходной перекрестной проверки.
- Измерение эффективности модели.
- Вывод матрицы несоответствий с попаданиями и промахами классификационных прогнозов.
- Выполнение множественной линейной регрессии с набором данных California Housing.
- Выполнение пространственной свертки с использованием PCA и t-SNE с наборами данных Iris и Digits с целью подготовки их для двумерных визуализаций.
- Выполнение машинного обучения без учителя кластеризацией методом *k* средних с набором данных Iris.

14.1. Введение в машинное обучение

В этой и следующей главах рассматривается машинное обучение — одна из самых интересных и перспективных областей искусственного интеллекта. Вы научитесь быстро решать сложные и интересные задачи, за которые еще несколько лет назад не решился бы взяться не только новичок, но и самый опытный программист. Машинное обучение — большая и сложная тема, с которой связано множество неочевидных нюансов. В этой главе мы постараемся в доступной форме, выборочно ввести читателей в наиболее простые методы машинного обучения и их практическое применение.

Что такое машинное обучение?

Для начала зададимся вопросом: действительно ли машины (то есть наши компьютеры) способны к обучению? В этой и следующей главе мы покажем, как происходит это волшебство. Есть ли у нового стиля разработки приложений свой «секретный ингредиент»? Да, это данные — много данных. Вместо того чтобы программировать экспертные знания в своих приложениях, мы программируем приложения так, чтобы они учились на данных. Мы приведем множество примеров кода Python, которые строят работающие модели машинного обучения, использующие их для формирования на удивление точных прогнозов.

Прогнозирование

Как было бы здорово, если бы мы могли повысить точность прогнозов погоды для спасения жизней, сведения к минимуму числа жертв и ущерба для имущества! Или повысить точность диагностики рака и программы лечения для спасения жизней либо бизнес-прогнозов для максимизации прибыли и защиты рабочих мест... А как насчет выявления мошенничества при покупках по кредитным картам или обращениях за страховыми выплатами? Как насчет прогнозирования оттока клиентов или новых цен на недвижимость, кассовой прибыли новых фильмов, ожидаемого дохода от новых продуктов и сервисов? Прогнозирования оптимальных стратегий для тренеров и игроков, которые позволят им выигрывать больше игр и чемпионатов? Между тем уже сейчас благодаря машинному обучению такие прогнозы строятся каждый день.

Области применения машинного обучения

В табл. 14.1 перечислены некоторые популярные области применения машинного обучения.

Таблица 14.1. Некоторые популярные области применения машинного обучения

Автономные машины	Обнаружение объектов в сценах
Анализ эмоциональной окраски (например, классификация рецензий на фильмы на отрицательные, положительные и нейтральные)	Перевод естественных языков (с английского на испанский, с французского на японский и т. д.)
Выявление аномалий	Прогнозирование временных рядов — например, предсказание будущих котировок акций и прогнозы погоды
Выявление закономерностей в данных	Прогнозирование нарушений выплат ипотечных кредитов
Выявление попыток мошенничества с кредитными картами	Прогнозирование оттока клиентов
Выявление попыток страхового мошенничества	Распознавание голоса
Глубокий анализ данных в социальных сетях (Facebook, Twitter, LinkedIn)	Распознавание лиц
Диагностическая медицина	Распознавание образов и классификация изображений
Исследование данных	Распознавание рукописного текста
Классификация новостей: спорт, финансы, политика и т. д.	Рекомендательные системы («тем, кто купил этот продукт, также понравились...»)
Классификация электронной почты и выделение спама	Сжатие данных
Маркетинг: деление клиентов на группы	Фильтрация спама
Обнаружение вторжений в компьютерные системы	Чат-боты

14.1.1. Scikit-learn

В этой главе будет использоваться *популярная библиотека машинного обучения scikit-learn*. Библиотека scikit-learn, также называемая sklearn, предоставляет наиболее эффективные алгоритмы машинного обучения, удобно упакованные в форме *оценщиков* (estimators). Все оценщики инкапсулированы, поэтому подробности и математическое обоснование работы всех этих алгоритмов не видны разработчику. И вас это не должно беспокоить — человек может вести машину, не зная всех подробностей работы двигателя, системы передачи, системы торможения или системы рулевого управления. Представьте, как вы входите в лифт и выбираете нужный этаж или включаете телевизор и выбираете канал. Разбираетесь ли вы во всех подробностях того, как работает это оборудование или, скажем, как функционирует программное обеспечение вашего смартфона?

Со scikit-learn и небольшим объемом кода Python можно быстро создать мощные модели для анализа данных, извлечения закономерностей из данных и, что самое важное, построения прогнозов. Мы будем использовать scikit-learn для *обучения* моделей на подмножестве данных с последующим *тестированием* для проверки того, как работает каждая модель. После того как ваши модели пройдут обучение, мы применим их для построения прогнозов на основании данных, которые им еще не встречались. Результаты часто поражают. Внезапно ваш компьютер, который использовался в основном для всяких рутинных задач, начинает проявлять зачатки интеллекта.

Scikit-learn содержит инструменты, автоматизирующие процессы обучения и тестирования моделей. И хотя вы можете задать параметры для настройки моделей с возможным ростом их эффективности, в этой главе мы обычно используем *настройки* моделей *по умолчанию*, добиваясь при этом впечатляющих результатов.

Также существуют такие инструменты, как auto-sklearn (https://automl.github.io/auto-sklearn), автоматизирующие многие задачи, решаемые при помощи scikit-learn.

Какого оценщика scikit-learn следует выбрать для проекта

Трудно заранее определить, какие модели лучше всего подойдут для ваших данных, поэтому обычно аналитик опробует много моделей и выбирает ту, которая покажет наилучшие результаты. Как вы вскоре увидите, scikit-learn упрощает эту задачу. Популярный подход заключается в запуске многих моделей и выборе наилучшего варианта(-ов). Как же оценить, какая модель показывает наилучшие результаты?

Для этого нужно поэкспериментировать со множеством разных моделей с разными видами наборов данных. Обычно знать подробности сложных математических алгоритмов оценщиков sklearn не требуется, но с обретением опыта вы начнете представлять, какие алгоритмы лучше подходят для определенных типов задач и наборов данных. Впрочем, даже располагая определенным опытом, вряд ли вам удастся интуитивно угадать наилучшую модель для каждого нового набора данных. По этой причине scikit-learn помогает легко «опробовать их все». Создание и использование каждой модели занимает всего несколько строк кода. Модели выдают информацию о своей эффективности, позволяющей сравнить результаты и выбрать модель(-и) с лучшей эффективностью.

14.1.2. Типы машинного обучения

В этом разделе рассматриваются две основные разновидности машинного обучения — *машинное обучение с учителем*, которое работает с *помеченными данными*, и *машинное обучение без учителя*, которое работает с *непомеченными данными*.

Например, если разрабатываемое приложение должно распознавать собак и кошек на изображениях, то вы будете обучать модели на множестве фотографий собак (с пометкой «собака») и фотографий кошек (с пометкой «кошка»). Если ваша модель эффективна, то она сможет распознать непомеченные фотографиями собак и кошек, ранее никогда модели не встречавшиеся. Чем больше фотографий использовано для обучения, тем больше вероятность того, что модель точно определит, на каких новых фотографиях изображены собаки, а на каких — кошки. В эпоху больших данных и огромных недорогих компьютерных мощностей с теми методами, о которых мы собираемся рассказать, вы сможете строить довольно точные модели.

Какую пользу могут принести непомеченные данные? В интернете продается огромное количество книг. Продавцы хранят огромные объемы (непомеченных) данных о покупке книг. Они быстро заметили, что люди, покупающие определенные книги, с большой вероятностью будут приобретать другие книги по тем же или схожим темам. Это привело к появлению *рекомендательных систем*. В поисках нужной книги на сайте продавца вы с большой вероятностью будете видеть рекомендации вроде: «Люди, которые купили эту книгу, также купили эти книги». В наши дни рекомендательные системы играют важную роль, способствуя достижению максимальных продаж любых продуктов.

Машинное обучение с учителем

Машинное обучение с учителем делится на две категории — *классификацию* и *регрессию*. Модели проходят обучение на наборах данных, состоящих из строк и столбцов. Каждая строка представляет *точку* данных, а каждый столбец — некую *характеристику* этой точки. В машинном обучении с учителем с каждой точкой данных связывается метка, называемая *целевой меткой* (например, «dog» или «cat»). Она определяет то значение, которое должно прогнозироваться для новых данных, передаваемых вашим моделям.

Наборы данных

Мы будем работать с «игрушечными» наборами данных, состоящими из небольшого количества точек данных и ограниченного набора характеристик.

Также будут использованы несколько полноценных, реальных наборов данных — один состоит из нескольких тысяч точек данных, а в другом их количество достигает десятков тысяч. Заметим, что в мире больших данных наборы нередко содержат миллионы и миллиарды точек данных, и даже больше. Для исследований в области data science существует огромное число бесплатных и свободно распространяемых наборов данных. Такие библиотеки, как scikit-learn, включают популярные наборы данных для экспериментов, а также предоставляют средства для загрузки данных из различных репозиториев (например, openml.org). Правительственные учреждения, коммерческие и другие организации по всему миру предоставляют наборы данных по широкому спектру областей. Мы будем работать с популярными бесплатными наборами данных с применением различных средств машинного обучения.

Классификация

Для анализа набора данных Digits, включенного в поставку scikit-learn, будет применен один из простейших классификационных алгоритмов — *метод k ближайших соседей*. Классификационные алгоритмы прогнозируют дискретные классы (категории), к которым относятся точки данных. При бинарной классификации используются два класса: например, «спам» или «не спам» в приложении классификации электронной почты. В задачах множественной классификации используется более двух классов — например, 10 классов (от 0 до 9) в наборе данных Digits. Схема классификации для описаний фильмов может пытаться классифицировать их по жанру: «приключения», «фэнтези», «романтика», «исторический» и т. д.

Регрессия

Регрессионные модели прогнозируют *непрерывный вывод* — например, прогнозируемую температуру в анализе *временных рядов* из раздела «Введение в data science» главы 10. В этой главе мы вернемся к примеру *простой линейной регрессии*, но на этот раз реализуем его с использованием оценщика `LinearRegression` из scikit-learn. Затем оценщик `LinearRegression` будет использован для выполнения *множественной линейной регрессии* с набором данных California Housing, включенным в поставку scikit-learn. В этом примере будет прогнозироваться медианная стоимость дома в квартале по данным переписи США с учетом следующих характеристик: среднего количества комнат, медианного возраста дома, среднего количества спален и медианного дохода. Оценщик `LinearRegression` по умолчанию использует все числовые

характеристики набора данных для формирования более сложных прогнозов, чем это возможно с простой линейной регрессией с одним признаком.

Машинное обучение без учителя

Перейдем к рассмотрению машинного обучения без учителя с алгоритмами *кластеризации*. Мы воспользуемся методом *снижения размерности* признакового пространства (с применением оценщика scikit-learn `TSNE`) для *сжатия* 64 признаков набора данных Digits до двух в целях визуализации. Это позволит увидеть, как хорошо «группируются» данные Digits — наборы данных с рукописными цифрами наподобие тех, что должны распознаваться компьютерами в почтовых отделениях для отправки писем по указанным почтовым индексам. Речь идет о сложной задаче из области распознавания образов, если учесть неповторимость человеческого почерка. Тем не менее эта модель кластеризации будет построена всего в нескольких строках кода, а достигнутый результат окажется весьма впечатляющим. И все это не требует от вас знания внутреннего устройства алгоритма кластеризации. В этом проявляется вся элегантность объектно-базированного программирования. Другой пример удобного объектно-базированного программирования рассматривается в следующей главе, когда мы займемся построением мощных моделей глубокого обучения с использованием библиотеки Keras.

Кластеризация методом *k* средних и набор данных Iris

Мы представим простейший алгоритм машинного обучения без учителя — *кластеризацию методом k средних* и воспользуемся им для набора данных Iris, также включенного в поставку scikit-learn. Снижение размерности признакового пространства (с оценщиком `PCA` из scikit-learn) сжимает четыре признака набора данных Iris до двух с целью визуализации. Будет продемонстрирована кластеризация трех образцов Iris по набору данных и графическое представление *центроида* каждого кластера (то есть центральной точки кластера). Наконец, мы применим несколько оценщиков кластеризации для сравнения их эффективности по разбиению точек набора данных Iris на три кластера.

Обычно аналитик задает желательное количество моделей *k*. Метод *k* средних перебирает данные, стараясь разделить их на заданное количество кластеров. Как и многие алгоритмы машинного обучения, метод *k* средних работает по *итеративному* принципу и в конечном итоге сходится к кластерам в заданном количестве.

Кластеризация методом *k* средних может выявить сходство в непомеченных данных. Этот факт может помочь в назначении меток данным, чтобы оценщики для обучения с учителем смогли обработать его. С учетом того, насколько монотонен и ненадежен процесс назначения меток непомеченным данным (причем подавляющее большинство мировых данных не имеет меток), машинное обучение без учителя играет важную роль.

Большие данные и большие вычислительные мощности компьютеров

Объем доступных данных в наши дни уже огромен, причем он продолжает расти в экспоненциальном темпе. Только за последние годы было произведено столько же данных, сколько появилось до этого момента от начала цивилизации. Мы часто говорим о больших данных, но прилагательное «большой» недостаточно наглядно описывает, насколько огромен их объем. Когда-то люди говорили: «Я тону в данных и не знаю, что с ними делать». С появлением машинного обучения мы теперь говорим: «Затопите меня большими данными, и я воспользуюсь технологиями машинного обучения, чтобы извлечь из них информацию и сделать прогнозы».

Все это происходит в то время, когда вычислительная мощность компьютеров *стремительно растет*, а компьютерная память и дисковое пространство *увеличиваются* в объемах при значительном снижении стоимости. Все это позволяет нам взглянуть на методологию поиска решения под другим углом. Теперь мы можем программировать компьютеры так, чтобы они *обучались* на данных, притом в колоссальных объемах. На первый план выходит прогнозирование на основе данных.

14.1.3. Наборы данных, включенные в поставку scikit-learn

В табл. 14.2 перечислены наборы данных, включенные в поставку scikit-learn[1]. Также предоставляется возможность загрузки наборов данных из других источников, включая 20 000+ наборов данных, доступных на сайте openml.org.

[1] http://scikit-learn.org/stable/datasets/index.html.

Таблица 14.2. Наборы данных, включенные в поставку scikit-learn

«Игрушечные» наборы данных	*«Реальные» наборы данных*
Цены на дома в Бостоне	Лица Оливетти
Ирисы	Тексты 20 новостных групп
Диабет	Помеченные лица для распознавания
Оптическое распознавание рукописных цифр	Типы лесопосадок
Linnerrud	RCV1
Распознавание вин	Kidcup 99
Диагностика рака груди (Висконсин)	California Housing

14.1.4. Последовательность действий в типичном исследовании data science

Далее будут выполнены все основные шаги типичного практического сценария машинного обучения:

- загрузка набора данных;
- исследование данных с использованием pandas и визуализаций;
- преобразование данных (нечисловых данных в числовые, потому что scikit-learn требуются числовые данные; в главе 14 будут использоваться «готовые» наборы данных, но мы еще вернемся к этой теме в главе 15);
- разбиение данных для обучения и тестирования;
- создание модели;
- обучение и тестирование модели;
- настройка параметров модели и оценка ее точности;
- формирование прогнозов на основании «живых» данных, которые еще неизвестны модели.

В разделах «Введение в data science» глав 7 и 8 обсуждается решение проблемы отсутствующих и ошибочных значений средствами pandas. Эти операции играют важную роль при очистке данных перед их применением в машинном обучении.

14.2. Практический пример: классификация методом *k* ближайших соседей и набор данных Digits, часть 1

Чтобы почта обрабатывалась эффективно, а каждое письмо передавалось по правильному адресу, компьютеры почтовой службы должны сканировать рукописные имена, адреса и почтовые индексы, распознавая цифры и буквы. Как будет показано в этой главе, благодаря мощным библиотекам, таким как scikit-learn, даже начинающий программист способен справиться с подобной задачей из области машинного обучения. В следующей главе еще более мощная функциональность распознавания образов будет использоваться при представлении технологии глубокого обучения для сверточных нейронных сетей.

Задачи классификации

В этом разделе будет рассмотрена задача *классификации* в области машинного обучения с учителем, где требуется спрогнозировать класс[1], к которому относится образец. Например, если у вас имеются изображения собак и кошек, то каждое изображение должно классифицироваться как «собака» или «кошка». Подобная задача называется *бинарной*, поскольку в ней задействованы всего *два* класса.

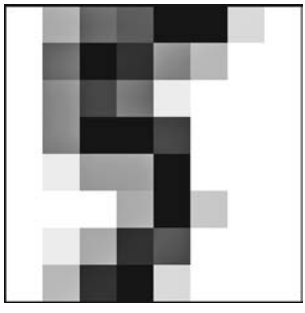
Воспользуемся *набором данных Digits*[2], входящим в поставку scikit-learn. Набор состоит из изображений 8 × 8 пикселов и представляет 1797 рукописных цифр (от 0 до 9). Требуется определить, какую цифру представляет изображение. Так как существует 10 возможных цифр (классов), данная задача является *задачей множественной классификации*. Для обучения модели используются *помеченные данные* — класс каждой цифры известен заранее. В этом примере для распознавания рукописных цифр будет применен один из простейших алгоритмов классификации — *метод k ближайших соседей* (*k*-NN).

Следующая визуализация цифры 5 в низком разрешении была получена в результате вывода одной цифры в виде матрицы 8 × 8. Вскоре мы покажем, как выводить такие изображения средствами Matplotlib:

---

[1] В данном случае под термином «класс» понимается «категория», а не концепция класса в языке Python.

[2] http://scikit-learn.org/stable/datasets/index.html#optical-recognition-of-handwritten-digits-dataset.



Исследователи создали изображения этого набора данных на основе базы данных MNIST с десятками тысяч изображений 32 × 32 пиксела, полученных в начале 1990-х. С современными камерами и сканерами высокого разрешения такие изображения можно записать с более высоким качеством.

Наш подход

Описание этого примера занимает два раздела. Начнем с основных этапов реализации задач машинного обучения:

- Выбор данных для обучения модели.
- Загрузка и анализ данных.
- Разбиение данных для обучения и тестирования.
- Выбор и построение модели.
- Обучение модели.
- Формирование прогнозов.

Как вы вскоре увидите, в scikit-learn каждый из этих шагов занимает лишь несколько строк кода. В следующем разделе мы:

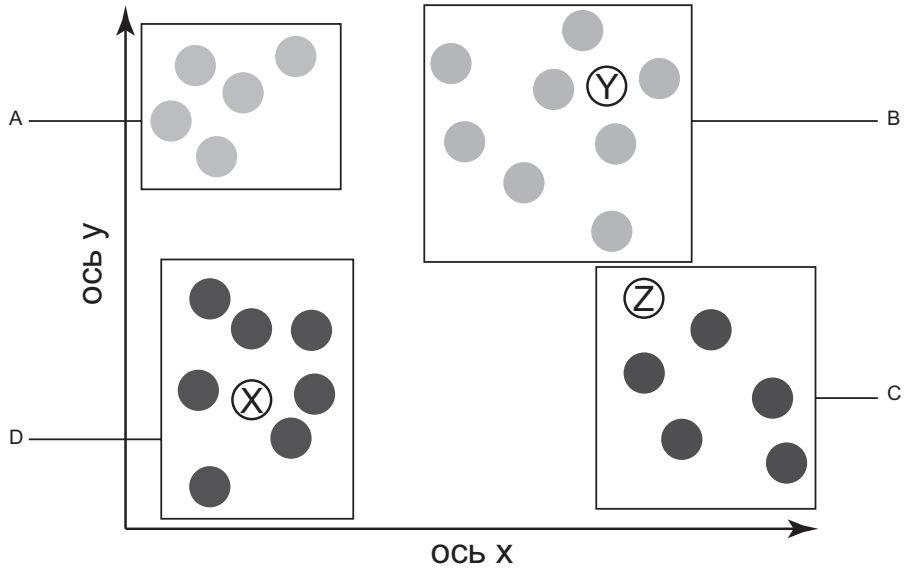
- проведем оценку результатов;
- настроим параметры модели;
- обработаем несколько классификационных моделей для выбора наилучшей модели(-ей).

Для визуализации данных будут использоваться библиотеки Matplotlib и Seaborn, поэтому IPython следует запустить с поддержкой Matplotlib:

```
ipython --matplotlib
```

14.2.1. Алгоритм *k* ближайших соседей

Scikit-learn поддерживает много алгоритмов классификации, включая простейший алгоритм *k ближайших соседей* (*k-NN*). Этот алгоритм пытается спрогнозировать класс тестового образца, анализируя *k* обучающих образцов, расположенных ближе всего (по расстоянию) к тестовому образцу. Для примера возьмем следующую диаграмму, на которой заполненные точки представляют четыре класса — A, B, C и D. В контексте нашего обсуждения эти буквы будут использоваться как имена классов:



Требуется спрогнозировать классы, к которым принадлежат новые образцы X, Y и Z. Будем считать, что прогнозы должны формироваться по *трем* ближайшим соседям каждого образца — *k* равно 3 в алгоритме *k* ближайших соседей:

- Все три ближайших соседа образца X являются точками класса D, поэтому модель прогнозирует, что X относится к классу D.
- Все три ближайших соседа образца Y являются точками класса B, поэтому модель прогнозирует, что Y относится к классу B.
- Для Z выбор не очевиден, потому что образец находится *между* точками B *и* C. Из трех ближайших соседей один принадлежит классу B, а два — клас-

су C. В алгоритме *k* ближайших соседей побеждает класс с большинством «голосов». Из-за двух голосов C против одного голоса B мы прогнозируем, что `Z` относится к классу C. Выбор нечетного значения *k* в алгоритме *k*-NN предотвращает «ничьи» и гарантирует, что количество голосов никогда не будет равным.

Гиперпараметры и настройка гиперпараметров

В области машинного обучения *модель* реализует алгоритм машинного обучения. В терминологии scikit-learn модели называются *оценщиками*. Существуют два типа параметров машинного обучения:

- вычисляемые оценщиком в ходе своего обучения на основании предоставленных вами данных;
- задаваемые заранее при создании объекта оценщика scikit-learn, представляющего модель.

Параметры, задаваемые заранее, называются *гиперпараметрами*.

В алгоритме *k* ближайших соседей *k* является гиперпараметром. Для простоты мы используем значения гиперпараметров *по умолчанию* для scikit-learn. В реальном исследовании из области машинного обучения желательно поэкспериментировать с разными значениями *k* для получения наилучших возможных моделей для ваших исследований. Этот процесс называется *настройкой гиперпараметров*. Позднее мы используем настройку гиперпараметров для выбора значения *k*, которое позволяет алгоритму *k* ближайших соседей выдать лучшие прогнозы для набора данных Digits. Scikit-learn также содержит средства *автоматической* настройки гиперпараметров.

14.2.2. Загрузка набора данных

Функция `load_digits` из *модуля* `sklearn.datasets` возвращает объект scikit-learn `Bunch`, содержащий данные цифр и информацию о наборе данных Digits (так называемые *метаданные*):

```
In [1]: from sklearn.datasets import load_digits

In [2]: digits = load_digits()
```

`Bunch` представляет собой подкласс `dict`, содержащий дополнительные атрибуты для взаимодействия с набором данных.

Вывод описания

Набор данных Digits, входящий в поставку scikit-learn, является подмножеством *набора данных рукописных цифр UCI* (*Калифорнийский университет в Ирвайне*) *ML*:

http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

Исходный набор данных UCI содержит 5620 образцов — 3823 для обучения и 1797 для тестирования. Версия набора данных, поставляемая со scikit-learn, содержит только 1797 *тестовых образцов*. *Атрибут* `DESCR` объекта `Bunch` содержит описание набора данных. Согласно описанию набора данных Digits[1], каждый образец содержит 64 признака (`Number of Attributes`), представляющие изображение 8 × 8 со значениями пикселов в диапазоне 0–16 (`Attribute Information`). Набор данных *не содержит отсутствующих значений* (`Missing Attribute Values`). Создается впечатление, что 64 признака — это много, но необходимо иметь в виду, что реальные наборы данных иногда содержат сотни, тысячи и даже миллионы признаков.

```
In [3]: print(digits.DESCR)
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range
     0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
http://archive.ics.uci.edu/ml/datasets/
   Optical+Recognition+of+Handwritten+Digits
...
```

Проверка атрибутов data и target

Атрибуты `data` и `target` объекта `Bunch` представляют собой массивы `NumPy`:

- *Массив* `data` содержит 1797 образца (изображения цифр), каждый из которых несет 64 признака со значениями в диапазоне 0–16, представляющие

[1] Ключевая информация выделена жирным шрифтом.

интенсивности пикселов. С Matplotlib можно визуализировать интенсивности в оттенках серого от белого (0) до черного (16):

- *Массив* `target` содержит метки изображений, то есть классы, указывающие, какую цифру представляет каждое изображение. Массив называется `target`, потому что при прогнозировании вы стремитесь «попасть в цель» с выбором значений. Чтобы увидеть метки образцов в наборе данных, выведем значения `target` каждого 100-го образца:

```
In [4]: digits.target[::100]
Out[4]: array([0, 4, 1, 7, 4, 8, 2, 2, 4, 4, 1, 9, 7, 3, 2, 1, 2, 5])
```

Количество образцов и признаков (на один образец) подтверждается при помощи атрибута `shape` массива `data`, который показывает, что набор данных состоит из 1797 строк (образцов) и 64 столбцов (признаков):

```
In [5]: digits.data.shape
Out[5]: (1797, 64)
```

Размеры массива `target` подтверждают, что количество целевых значений соответствует количеству образцов:

```
In [6]: digits.target.shape
Out[6]: (1797,)
```

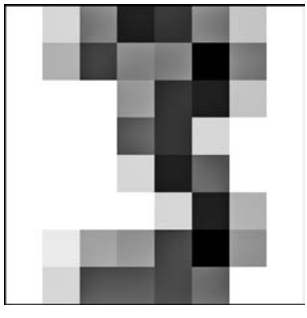
Пример изображения цифры

Все изображения двумерны — они обладают шириной и высотой в пикселах. Объект `Bunch`, возвращаемый `load_digits`, содержит атрибут `images` — массив, каждый элемент которого представляет собой двумерный массив 8×8 с интенсивностями пикселов изображения цифры. Хотя в исходном наборе данных каждый пиксел представлен целочисленным значением в диапазоне 0–16, scikit-learn хранит эти значения в виде *значений с плавающей точкой* (тип NumPy `float64`). Например, двумерный массив, представляющий изображение образца с индексом 13, выглядит так:

```
In [7]: digits.images[13]
Out[7]:
array([[ 0.,  2.,  9., 15., 14.,  9.,  3.,  0.],
       [ 0.,  4., 13.,  8.,  9., 16.,  8.,  0.],
```

```
[ 0.,  0.,  0.,  6., 14., 15.,  3.,  0.],
[ 0.,  0.,  0., 11., 14.,  2.,  0.,  0.],
[ 0.,  0.,  0.,  2., 15., 11.,  0.,  0.],
[ 0.,  0.,  0.,  0.,  2., 15.,  4.,  0.],
[ 0.,  1.,  5.,  6., 13., 16.,  6.,  0.],
[ 0.,  2., 12., 12., 13., 11.,  0.,  0.]])
```

Ниже показано изображение, представленное этим двумерным массивом — вскоре мы приведем код вывода этого изображения:



Подготовка данных для использования со scikit-learn

Алгоритмы машинного обучения scikit-learn требуют, чтобы образцы хранились в *двумерном массиве значений с плавающей точкой* (или коллекции, *сходной с* двумерным *массивом*, например списком списков или коллекцией pandas `DataFrame`):

- каждая строка представляет один *образец*;
- каждый столбец заданной строки представляет один *признак* этого образца.

Для представления каждого образца в виде одной строки данных многомерные данные (например, двумерный массив `image` из фрагмента [7]) должны быть *преобразованы* в одномерный массив.

Если вы работаете с данными, содержащими *категорийные признаки* (обычно представленные в виде строк — скажем, `'spam'` и `'not-spam'`), то вам также придется провести *предварительную обработку* этих признаков и преобразовать их в числовые значения (так называемое *прямое унитарное кодирование* будет рассматриваться в следующей главе). Модуль `sklearn.preprocessing` библиотеки Scikit-learn предоставляет функциональность для преобразования категорийных данных в числовые. Набор данных Digits не содержит категорийных признаков.

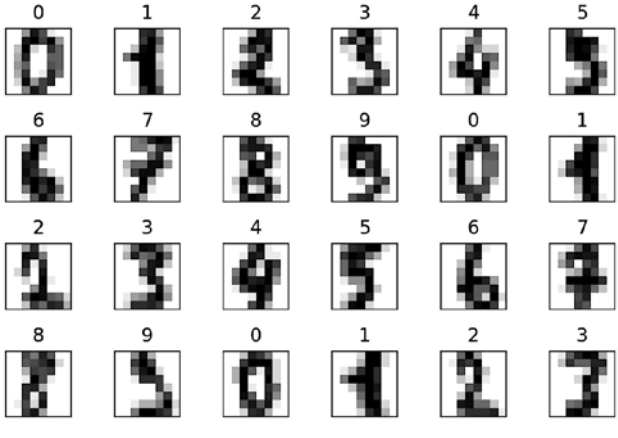
Для вашего удобства функция `load_digits` возвращает предварительно обработанные данные, готовые к машинному обучению. Набор данных Digits является числовым, поэтому `load_digits` просто сглаживает двумерный массив в одномерный массив. Например, массив 8 × 8 `digits.images[13]` из фрагмента [7] соответствует массиву 1 × 64 `digits.data[13]` следующего вида:

```
In [8]: digits.data[13]
Out[8]:
array([ 0.,  2.,  9., 15., 14.,  9.,  3.,  0.,  0.,  4., 13.,  8.,  9.,
       16.,  8.,  0.,  0.,  0.,  0.,  6., 14., 15.,  3.,  0.,  0.,  0.,
        0., 11., 14.,  2.,  0.,  0.,  0.,  0.,  0.,  2., 15., 11.,  0.,
        0.,  0.,  0.,  0.,  0.,  2., 15.,  4.,  0.,  0.,  1.,  5.,  6.,
       13., 16.,  6.,  0.,  0.,  2., 12., 12., 13., 11.,  0.,  0.])
```

В этом одномерном массиве первые восемь элементов содержат элементы строки 0 двумерного массива, следующие восемь элементов — элементы строки 1 двумерного массива, и т. д.

14.2.3. Визуализация данных

Всегда старайтесь поближе познакомиться со своими данными. Этот процесс называется *исследованием данных*. Например, изображения цифр (с тем чтобы составить представление об их внешнем виде) можно просто вывести функцией `implot` библиотеки Matplotlib. На следующей иллюстрации изображены первые 24 изображения набора данных. Чтобы понять, насколько трудна задача распознавания рукописных цифр, посмотрите, как сильно *различаются* изображения цифры 3 в первой, третьей и четвертой строке, и взгляните на изображения цифры 2 в первой, третьей и четвертой строке.

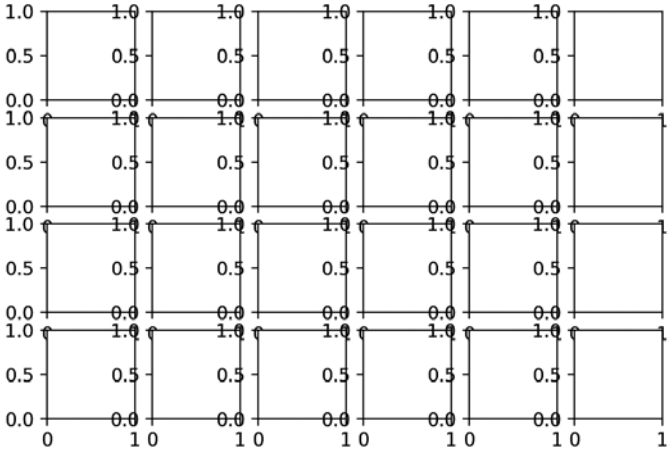


Создание диаграммы

А теперь рассмотрим код, которым были выведены эти 24 изображения цифр. Следующий вызов функции `subplots` создает эту диаграмму 6 × 4 (размер задается ключевым аргументом `figsize(6, 4)`), которая состоит из 24 поддиаграмм, расположенных в 4 строки (`nrows`=4) и 6 столбцов (`ncols`=6). Каждая поддиаграмма имеет собственный объект `Axes`, который используется для вывода одного изображения цифры:

```
In [9]: import matplotlib.pyplot as plt

In [10]: figure, axes = plt.subplots(nrows=4, ncols=6, figsize=(6, 4))
```

Функция `subplots` возвращает объекты `Axes` в двумерном массиве NumPy. Изначально диаграмма выглядит так, как показано ниже, — на ней выводятся деления (которые мы вскоре уберем) на осях *x* и *y* каждой поддиаграммы:



Вывод изображений и удаление меток осей

Затем команда `for` в сочетании со встроенной функцией `zip` будет использоваться для параллельного перебора всех 24 объектов `Axes`, первых 24 изображений в `digits.images` и первых 24 значений в `digits.target`:

```
In [11]: for item in zip(axes.ravel(), digits.images, digits.target):
    ...:     axes, image, target = item
    ...:     axes.imshow(image, cmap=plt.cm.gray_r)
    ...:     axes.set_xticks([])  # Удаление делений на оси x
    ...:     axes.set_yticks([])  # Удаление делений на оси y
```

```
...:     axes.set_title(target)
...: plt.tight_layout()
...:
...:
```

Напомним, метод массивов NumPy `ravel` создает *одномерное представление* многомерного массива, а функция `zip` — кортежи, содержащие элементы всех аргументов `zip` с одинаковыми индексами. Аргумент с наименьшим количеством элементов определяет количество возвращаемых кортежей. Каждая итерация цикла:

- распаковывает один кортеж на три переменные, представляющие объект `Axes`, изображение и целевое значение;
- вызывает метод `imshow` объекта `Axes` для вывода одного изображения. Ключевой аргумент `cmap=plt.cm.gray_r` определяет цвета, выводимые в изображении. Значение `plt.cm.gray_r` представляет собой *цветовую карту* — группу часто выбираемых цветов, хорошо сочетающихся друг с другом. С этой конкретной цветовой картой пикселы изображения выводятся в оттенках серого: 0 соответствует белому цвету, 16 — черному, а промежуточные значения — оттенкам серого с возрастанием темного. Названия цветовых карт Matplotlib приведены на странице https://matplotlib.org/examples/color/colormaps_reference.html. К ним можно обращаться через объект `plt.cm` bkb или в строковом виде `'gray_r'`;
- вызывает методы `set_xticks` и `set_yticks` объекта `Axes` с пустыми списками, чтобы указать, что оси *x* и *y* должны выводиться без делений;
- вызывает метод `set_title` объекта `Axes` для вывода `целевого` значения над изображением, то есть фактического значения, представляемого изображением.

После цикла вызов метода `tight_layout` удаляет лишние поля у верхнего, правого, нижнего и левого края `Figure`, с тем чтобы строки и столбцы цифровых изображений заполняли большую площадь диаграммы.

14.2.4. Разбиение данных для обучения и тестирования

Обучение моделей машинного обучения обычно производится на подмножестве набора данных. Как правило, чем больше данных доступно для обучения, тем качественнее обучается модель. Важно зарезервировать часть данных для тестирования, чтобы вы могли оценить эффективность модели на данных, которые

ей пока неизвестны. Когда вы будете уверены в том, что модель работает эффективно, ее можно будет использовать для прогнозирования на новых данных.

Сначала данные разбиваются на два поднабора: *обучающий* и *тестовый*. Функция `train_test_split` из модуля `sklearn.model_selection` осуществляет *случайную* перестановку данных, а затем разбивает образцы в массиве `data` и целевые значения в массиве `target` на обучающий и тестовый набор. Это гарантирует, что обучающий и тестовый наборы обладают сходными характеристиками. Случайная перестановка и разбиение выполняются для вашего удобства объектом `ShuffleSplit` из модуля `sklearn.model_selection`. Функция `train_test_split` возвращает кортеж из четырех элементов, в котором два первых элемента содержат *образцы*, разделенные на обучающий и тестовый набор, а два последних — соответствующие *целевые значения*, также разделенные на обучающий и тестовый набор. По общепринятым соглашениям буква верхнего регистра `X` используется для представления образцов, а буква `y` нижнего регистра — для представления целевых значений:

```
In [12]: from sklearn.model_selection import train_test_split

In [13]: X_train, X_test, y_train, y_test = train_test_split(
   ...:     digits.data, digits.target, random_state=11)
   ...:
```

Предполагается, что *классы* данных *сбалансированы*, то есть образцы равномерно распределены между классами. К слову, все классификационные наборы, входящие в поставку scikit-learn, обладают этим свойством. Несбалансированность классов может привести к ошибочным результатам.

В главе 4 было показано, как *инициализировать* генератор случайных чисел для получения *воспроизводимых* результатов. В исследованиях в области машинного обучения это позволяет другим пользователям проверить ваши результаты, так как они могут работать с *теми же* случайно выбранными данными. Функция `train_test_split` предоставляет ключевой аргумент `random_state` для *воспроизводимости* результатов. Если в будущем тот же код будет выполняться с *тем же* значением инициализации, `train_test_split` выберет *те же* данные для обучающего и тестового наборов. Значение инициализации в нашем примере (`11`) было выбрано произвольно.

Размеры обучающего и тестового наборов

Взглянув на размеры наборов `X_train` и `X_test`, мы видим, что *по умолчанию* `train_test_split` резервирует 75% данных для обучения и 25% для тестирования:

```
In [14]: X_train.shape
Out[14]: (1347, 64)

In [15]: X_test.shape
Out[15]: (450, 64)
```

Чтобы использовать *другое* соотношение, можно задать размеры тестового и обучающего набора при помощи ключевых аргументов `test_size` и `train_size` функции `train_test_split`. Используйте значения с плавающей точкой в диапазоне от `0.0` до `1.0` для определения процентной доли каждого набора в данных. Целочисленные значения задают точное количество образцов. Если один из этих ключевых аргументов задается при вызове, то второй вычисляется автоматически. Например, команда

```
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target, random_state=11, test_size=0.20)
```

сообщает, что 20% данных предназначены для тестирования, поэтому значение `train_size` вычисляется равным `0.80`.

14.2.5. Создание модели

Оценщик `KNeighborsClassifier` (модуль `sklearn.neighbors`) реализует алгоритм *k* ближайших соседей. Сначала создается объект оценщика `KNeighborsClassifier`:

```
In [16]: from sklearn.neighbors import KNeighborsClassifier

In [17]: knn = KNeighborsClassifier()
```

Чтобы создать оценщика, достаточно создать объект. Внутренние подробности того, как этот объект реализует алгоритм *k* ближайших соседей, скрыты в самом объекте. Вам остается просто вызывать методы этого объекта. В этом заключается суть *объектно-базированного программирования* Python.

14.2.6. Обучение модели

Затем вызывается *метод* `fit` объекта `KNeighborsClassifier`, который загружает обучающий набор образцов (`X_train`) и обучающий набор целевых значений (`y_train`) в оценщике:

```
In [18]: knn.fit(X=X_train, y=y_train)
Out[18]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

Для большинства оценщиков scikit-learn метод `fit` загружает данные в оценщика, а затем использует эти данные для выполнения «за кулисами» сложных вычислений, в ходе которых происходит извлечение информации и обучение модели. Метод `fit` объекта `KNeighborsClassifier` просто загружает данные в оценщике, потому что алгоритм k-NN не имеет исходного процесса обучения. Данный оценщик называется *отложенным*, потому что он выполняет свою работу только тогда, когда он используется для построения прогнозов. В этой и в следующей главе мы будем использовать множество моделей, имеющих значительные фазы обучения. В реальных приложениях машинного обучения обучение моделей может занимать минуты, часы, дни и даже месяцы, но (см. далее) специализированное высокопроизводительное оборудование — графические процессоры (GPU) и тензорные процессоры (TPU) — могут значительно сократить время обучения модели.

Как видно из вывода фрагмента [18], метод `fit` возвращает оценщика, поэтому IPython выводит его строковое представление, включающее настройки *по умолчанию*. Значение `n_neighbors` соответствует k в алгоритме k ближайших соседей. По умолчанию `KNeighborsClassifier` ищет пятерых ближайших соседей для построения своих прогнозов. Для простоты мы используем оценки оценщика по умолчанию. Для `KNeighborsClassifier` они описаны по адресу:

http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Рассмотрение большинства этих настроек выходит за рамки книги. В части 2 примера мы поговорим о том, как выбрать лучшее значение для `n_neighbors`.

14.2.7. Прогнозирование классов для рукописных цифр

Итак, после загрузки данных в `KNeighborsClassifier` эти данные могут использоваться с тестовыми образцами для построения прогнозов. При вызове *метода* `predict` оценщика с передачей `X_test` в аргументе возвращает массив, содержащий прогнозируемый класс каждого тестового изображения:

```
In [19]: predicted = knn.predict(X=X_test)

In [20]: expected = y_test
```

Сравним прогнозируемые цифры с ожидаемыми для первых 20 тестовых образцов:

```
In [21]: predicted[:20]
Out[21]: array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 5, 6])

In [22]: expected[:20]
Out[22]: array([0, 4, 9, 9, 3, 1, 4, 1, 5, 0, 4, 9, 4, 1, 5, 3, 3, 8, 3, 6])
```

Как видим, среди первых 20 элементов массивов `predicted` и `expected` не совпадают только значения с индексом 18. Здесь ожидалась цифра 3, но модель предсказала 5.

Воспользуемся трансформацией списка для нахождения *всех* ошибочных прогнозов для *всего* тестового набора, то есть тех случаев, в которых значения из массивов `predicted` и `expected` *не совпадают*:

```
In [23]: wrong = [(p, e) for (p, e) in zip(predicted, expected) if p != e]

In [24]: wrong
Out[24]:
[(5, 3),
 (8, 9),
 (4, 9),
 (7, 3),
 (7, 4),
 (2, 8),
 (9, 8),
 (3, 8),
 (3, 8),
 (1, 8)]
```

Трансформация списка использует `zip` для создания кортежей, содержащих соответствующие элементы `predicted` и `expected`. Кортеж включается в результат только в том случае, если его значение `p` (прогнозируемое значение) и `e` (ожидаемое значение) различны, то есть спрогнозированное значение было неправильным. В этом примере оценщик неправильно спрогнозировал только 10 из 450 тестовых образцов. Таким образом, точность прогнозирования для этого оценщика составила впечатляющую величину 97,78% даже при том, что мы использовали только параметры оценщика по умолчанию.

14.3. Практический пример: классификация методом *k* ближайших соседей и набор данных Digits, часть 2

В этом разделе мы продолжим работу над примером с классификацией цифр и сделаем следующее:

- оценим точность оценщика для классификации методом *k*-NN;
- выполним несколько оценщиков и сравним их результаты для выбора наилучшего варианта(-ов);
- продемонстрируем настройку гиперпараметра *k* метода k-NN для достижения оптимальной эффективности `KNeighborsClassifier`.

14.3.1. Метрики точности модели

После того как модель пройдет обучение и тестирование, желательно оценить ее точность. В этом разделе будут рассмотрены два способа оценки точности — метод `score` оценщика и *матрица несоответствий*.

Метод score оценщика

Каждый оценщик содержит метод `score`, который возвращает оценку результатов, показанных с тестовыми данными, переданными в аргументах. Для классификационных оценщиков метод возвращает *точность прогнозирования* для тестовых данных:

```
In [25]: print(f'{knn.score(X_test, y_test):.2%}')
97.78%
```

Оценщик `kNeighborsClassifier` со своим значением *k* по умолчанию (то есть `n_neighbors=5`) достигает точности прогнозирования 97,78%. Вскоре мы проведем настройку гиперпараметра, чтобы попытаться определить оптимальное значение *k* и добиться еще более высокой точности.

Матрица несоответствий

Другой способ проверки точности классификационного оценщика основан на использовании *матрицы несоответствий*, содержащей информацию

о правильно и неправильно спрогнозированных значениях (также называемых *попаданиями* и *промахами*) для заданного класса. Вызовите функцию `confusion_matrix` из *модуля* `sklearn.metrics` и передайте в аргументах классы `expected` и `predicted`:

```
In [26]: from sklearn.metrics import confusion_matrix

In [27]: confusion = confusion_matrix(y_true=expected, y_pred=predicted)
```

Ключевой аргумент `y_true` задает фактические классы тестовых образцов. Люди просмотрели изображения в наборе данных и пометили их конкретными классами (цифры). Ключевой аргумент `y_pred` определяет прогнозируемые цифры для этих тестовых изображений.

Ниже приведена матрица несоответствий, полученная по итогам предшествующего вывода. Правильные прогнозы находятся на *главной диагонали*, проходящей от левого верхнего до правого нижнего угла. Ненулевые значения, не находящиеся на главной диагонали, обозначают ошибочные прогнозы:

```
In [28]: confusion
Out[28]:
array([[45,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 45,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 54,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 42,  0,  1,  0,  1,  0,  0],
       [ 0,  0,  0,  0, 49,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0, 38,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 42,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 45,  0,  0],
       [ 0,  1,  1,  2,  0,  0,  0,  0, 39,  1],
       [ 0,  0,  0,  0,  1,  0,  0,  0,  1, 41]])
```

Каждая строка представляет один класс, то есть одну из цифр от 0 до 9. Столбцы обозначают количество тестовых образцов, классифицированных в соответствующий класс. Например, строка 0:

```
[45,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

представляет класс цифры 0. Столбцы представляют 10 возможных целевых классов 0–9. Так как мы работаем с цифрами, классы (0–9) и индексы строк и столбцов (0–9) совпадают. По данным строки, 0, 45 тестового образца были классифицированы как цифра 0, но *ни один* из тестовых образцов не был ошибочно классифицирован как одна из цифр 1–9. Таким образом, все 100% цифр 0 были спрогнозированы правильно.

Теперь возьмем строку 8, представляющую результат для цифры 8:

```
[ 0,  1,  1,  2,  0,  0,  0,  0, 39,  1]
```

- `1` в столбце с индексом 1 означает, что одна цифра 8 была *неправильно* классифицирована как 1.
- `1` в столбце с индексом 2 означает, что одна цифра 8 была *неправильно* классифицирована как 2.
- `2` в столбце с индексом 3 означает, что две цифры 8 были *неправильно* классифицированы как 3.
- `39` в столбце с индексом 8 означает, что 39 цифр 8 были *правильно* классифицированы как 8.
- 1 в столбце с индексом 9 означает, что одна цифра 8 была *неправильно* классифицирована как 9.

Таким образом, алгоритм правильно спрогнозировал 88,63% (39 из 44) всех цифр 8. Позднее было показано, что общая точность прогнозирования этого оценщика составляла 97,78%. Более низкая точность прогнозирования для цифры 8 означает, что она из-за своей формы труднее распознается, чем другие цифры.

Отчет по классификации

Модуль `sklearn.metrics` также предоставляет функцию `classification_report`, которая выводит таблицу *метрик классификации*[1], основанных на ожидаемых и прогнозируемых значениях:

```
In [29]: from sklearn.metrics import classification_report

In [30]: names = [str(digit) for digit in digits.target_names]

In [31]: print(classification_report(expected, predicted,
    ...:       target_names=names))
    ...:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        45
           1       0.98      1.00      0.99        45
           2       0.98      1.00      0.99        54
           3       0.95      0.95      0.95        44
```

[1] http://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-and-f-measures.

```
              4       0.98      0.98      0.98        50
              5       0.97      1.00      0.99        38
              6       1.00      1.00      1.00        42
              7       0.96      1.00      0.98        45
              8       0.97      0.89      0.93        44
              9       0.98      0.95      0.96        43
      micro avg       0.98      0.98      0.98       450
      macro avg       0.98      0.98      0.98       450
   weighted avg       0.98      0.98      0.98       450
```

В этом отчете:

- *precision* — точность, то есть общее количество точных прогнозов для заданной цифры, разделенное на общее количество прогнозов для этой цифры. Точность можно проверить по столбцам матрицы несоответствий. Например, взглянув на столбец с индексом 7, вы увидите значение 1 в строках 3 и 4: это означает, что одна цифра 3 и одна цифра 4 были ошибочно классифицированы как 7. Значение 45 в строке 7 показывает, что 45 изображений были правильно классифицированы как 7. Таким образом, *точность* для цифры 7 составляет 45/47, или 0,96;
- *recall* — отклик, то есть общее количество правильных прогнозов для заданной цифры, разделенное на общее количество образцов, которые должны были прогнозироваться как эта цифра. Отклик можно проверить по строкам матрицы несоответствий. Например, в строке с индексом 8 встречаются три значения 1 и значение 2; это означает, что некоторые цифры 8 были ошибочно классифицированы как другие цифры, а также значение 39, которое показывает, что 39 изображений были классифицированы правильно. Таким образом, *отклик* для цифры 8 составляет 39/44, или 0,89;
- *f1-score* — среднее значение *точности* и *отклика*;
- *support* — количество образцов с заданным ожидаемым значением. Например, 50 образцов были снабжены меткой 4, а 38 образцов — меткой 5.

Подробности о средних значениях в нижней части отчета можно найти здесь:

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

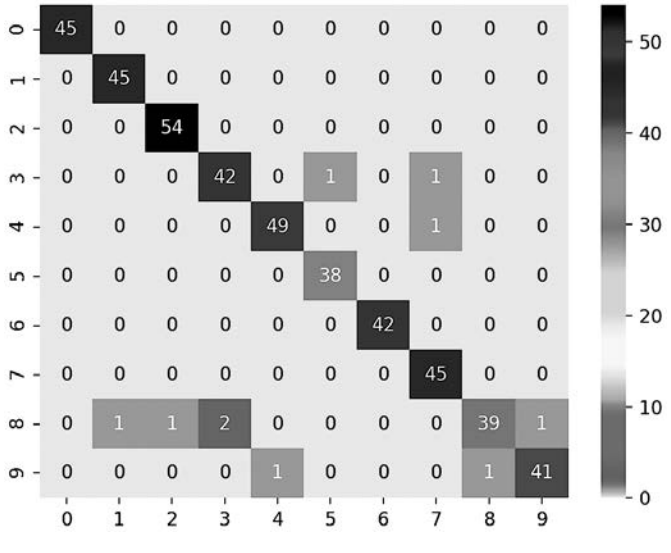
Визуализация матрицы несоответствий

На *тепловой карте* значения представлены цветами; обычно более высоким значениям соответствуют более интенсивные цвета. Функции построения диаграмм Seaborn работают с двумерными данными. При использовании pandas

`DataFrame` в качестве источника данных Seaborn автоматически помечает свои визуализации по именам столбцов и индексам строк. Преобразуем матрицу несоответствий в коллекцию `DataFrame`, а затем построим ее визуальное представление:

```
In [32]: import pandas as pd

In [33]: confusion_df = pd.DataFrame(confusion, index=range(10),
    ...:     columns=range(10))
    ...:

In [34]: import seaborn as sns

In [35]: axes = sns.heatmap(confusion_df, annot=True,
    ...:                    cmap='nipy_spectral_r')
    ...:
```

Функция `heatmap` библиотеки Seaborn строит тепловую карту по заданной коллекции `DataFrame`. Ключевой аргумент `annot=True` (сокращение от «annotation») выводит справа от диаграммы цветную полосу, которая обозначает соответствие между значениями и цветами цветовой карты. Ключевой аргумент `cmap='nipy_spectral_r'` определяет используемую цветовую карту. При выводе матрицы несоответствий в форме цветовой карты главная диагональ и ошибочные прогнозы хорошо выделяются на общем фоне.



14.3.2. K-проходная перекрестная проверка

K-проходная перекрестная проверка позволяет использовать все данные *как для* обучения, *так и для* тестирования. Повторное обучение и тестирование модели с разными частями набора данных помогают лучше понять, как модель справляется с прогнозированием для новых данных. Набор данных разбивается на k *частей* равного размера (параметр k в данном случае никак не связан с k из алгоритма k ближайших соседей). После этого модель повторно обучается на $k - 1$ частях и тестируется на оставшейся части. Для примера возьмем $k = 10$ с нумерацией частей от 1 до 10. Со всеми частями будут выполнены 10 последовательных циклов обучения и тестирования:

- Сначала выполняется обучение на частях 1–9, а затем тестирование с частью 10.
- Затем выполняется обучение на частях 1–8 и 10, а затем тестирование с частью 9.
- Затем выполняется обучение на частях 1–7 и 9–10, а затем тестирование с частью 8.

Цикл обучения и тестирования продолжается до тех пор, пока каждая часть не будет использована для тестирования модели.

Класс KFold

Библиотека scikit-learn предоставляет класс `KFold` и *функцию* `cross_val_score` (из модуля `sklearn.model_selection`) для выполнения описанных выше циклов обучения и тестирования. Выполним k-проходную перекрестную проверку с набором данных Digits и оценщиком `KNeighborsClassifier`, созданным ранее. Начнем с создания объекта `KFold`:

```
In [36]: from sklearn.model_selection import KFold

In [37]: kfold = KFold(n_splits=10, random_state=11, shuffle=True)
```

Ключевые аргументы:

- `n_splits=10` — количество частей;
- `random_state=11` — значение инициализации генератора случайных чисел для обеспечения *воспроизводимости* результатов;

- `shuffle=True` — объект `KFold` выполняет случайную перестановку данных перед разбиением их на части. Этот шаг особенно важен, если образцы могут быть сгруппированы или упорядочены. Например, набор данных *Iris*, который будет использован позднее в этой главе, содержит 150 образцов трех разновидностей ирисов: первые пятьдесят относятся к *Iris setosa*, следующие пятьдесят — к *Iris versicolor*, а последние 5 пятьдесят 0 — к *Iris virginica*. Если не переставить образцы, то может оказаться, что в обучающих данных нет ни одного образца конкретного вида *ирисов*, а тестовые данные состоят из данных одного вида.

Использование объекта KFold с функцией cross_val_score

Затем воспользуемся функцией `cross_val_score` для обучения и тестирования модели:

```
In [38]: from sklearn.model_selection import cross_val_score

In [39]: scores = cross_val_score(estimator=knn, X=digits.data,
    ...:     y=digits.target, cv=kfold)
    ...:
```

Ключевые аргументы:

- `estimator=knn` — оценщик, который вы хотите проверить;
- `X=digits.data` — образцы, используемые для обучения и тестирования;
- `y=digits.target` — прогнозы целевых значений для образцов;
- `cv=kfold` — генератор перекрестной проверки, определяющий способ разбиения образцов и целевых значений для обучения и тестирования.

Функция `cross_val_score` возвращает массив показателей точности — по одной для каждой части. Как видно из следующего вывода, модель была достаточно точной. *Наименьший* показатель точности составил `0,97777778` (97,78%), а в одном случае при прогнозировании всей части была достигнута 100-процентная точность:

```
In [40]: scores
Out[40]:
array([0.97777778, 0.99444444, 0.98888889, 0.97777778, 0.98888889,
       0.99444444, 0.97777778, 0.98882682, 1.        , 0.98324022])
```

Располагая частичными показателями точности, можно получить общее представление о точности модели. Для этого можно вычислить средний показатель

точности и стандартное отклонение по 10 показателям точности (или другому выбранному вами количеству частей):

```
In [41]: print(f'Mean accuracy: {scores.mean():.2%}')
Mean accuracy: 98.72%

In [42]: print(f'Accuracy standard deviation: {scores.std():.2%}')
Accuracy standard deviation: 0.75%
```

В среднем модель обеспечивала точность 98,72%, то есть даже больше, чем в предыдущем варианте, когда 75% данных использовалось для обучения, а 25% — для тестирования.

14.3.3. Выполнение нескольких моделей для поиска наилучшей

Трудно заранее определить, какая модель машинного обучения будет оптимальной для конкретного набора данных, особенно если подробности их работы скрыты от пользователя. И хотя `KNeighborsClassifier` прогнозирует изображения цифр с высокой точностью, может оказаться, что другие оценщики scikit-learn работают еще точнее. Scikit-learn предоставляет много моделей, позволяющих быстро провести обучение и тестирование данных. Это позволяет запустить *несколько разных моделей* и определить, какая из них лучше подходит для конкретного практического сценария.

Воспользуемся методами из предыдущего раздела для сравнения нескольких классификационных оценщиков — `KNeighborsClassifier`, `SVC` и `GaussianNB` (существуют и другие). И хотя оценщики `SVC` и `GaussianNB` ранее не описывались, scikit-learn позволяет легко опробовать их с настройками по умолчанию[1]. Импортируем двух других оценщиков:

```
In [43]: from sklearn.svm import SVC

In [44]: from sklearn.naive_bayes import GaussianNB
```

Теперь необходимо создать оценщиков. Следующий словарь содержит пары «ключ-значение» для существующего оценщика `KNeighborsClassifier`, созданного ранее, а также новых оценщиков `SVC` и `GaussianNB`:

---

[1] Чтобы избежать предупреждения в текущей версии scikit-learn на момент написания книги (версия 0.20), мы передали один ключевой аргумент при создании оценщика SVC. Значение этого аргумента будет использоваться по умолчанию, начиная с scikit-learn версии 0.22.

```
In [45]: estimators = {
    ...:     'KNeighborsClassifier': knn,
    ...:     'SVC': SVC(gamma='scale'),
    ...:     'GaussianNB': GaussianNB()}
    ...:
```

После этого можно переходить к выполнению модели:

```
In [46]: for estimator_name, estimator_object in estimators.items():
    ...:     kfold = KFold(n_splits=10, random_state=11, shuffle=True)
    ...:     scores = cross_val_score(estimator=estimator_object,
    ...:         X=digits.data, y=digits.target, cv=kfold)
    ...:     print(f'{estimator_name:>20}: ' +
    ...:           f'mean accuracy={scores.mean():.2%}; ' +
    ...:           f'standard deviation={scores.std():.2%}')
    ...:
KNeighborsClassifier: mean accuracy=98.72%; standard deviation=0.75%
                 SVC: mean accuracy=99.00%; standard deviation=0.85%
          GaussianNB: mean accuracy=84.48%; standard deviation=3.47%
```

Цикл перебирает элементы словаря `estimators` и для каждой пары «ключ-значение» выполняет следующие операции:

- распаковывает ключ в `estimator_name`, а значение — в `estimator_object`;
- создает объект `KFold`, осуществляющий случайную перестановку данных и формирующий 10 частей. В данном случае ключевой аргумент `random_state` особенно важен — он гарантирует, что все оценщики будут работать с идентичными частями (чтобы эффективность сравнивалась по одним исходным данным);
- оценивает текущий объект `estimator_object` с использованием `cross_val_score`;
- выводит имя оценщика, за которым следует математическое ожидание и стандартное отклонение для оценок точности, вычисленных для всех 10 частей.

Судя по результатам, оценщик `SVC` обеспечивает лучшую точность — по крайней мере, с настройками по умолчанию. Возможно, настройка некоторых параметров позволит добиться еще более точных результатов.

Точности оценщиков `KNeighborsClassifier` и `SVC` почти идентичны, поэтому стоит провести настройку гиперпараметров каждого оценщика для выбора лучшего варианта.

Диаграмма оценщиков scikit-learn

В документации scikit-learn приведена полезная диаграмма для выбора правильного оценщика в зависимости от размера и типа данных, а также поставленной задачи машинного обучения:

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

14.3.4. Настройка гиперпараметров

Ранее в этом разделе мы упоминали, что k в алгоритме k ближайших соседей является гиперпараметром алгоритма. Гиперпараметры задаются *до* того, как алгоритм начнет использоваться для обучения модели. В реальных исследованиях в процессе настройки гиперпараметров должны быть выбраны значения гиперпараметров, которые обеспечивают лучшие возможные прогнозы.

Чтобы определить лучшее значение k в алгоритме k-NN, поэкспериментируйте с разными значениями k и сравните эффективность оценщика в каждом варианте. Для этого можно воспользоваться теми же методами, что и при сравнении оценщиков. Следующий цикл создает объект `KNeighborsClassifiers` с нечетными значениями k от 1 до 19 (как упоминалось ранее, нечетные значения k в k-NN предотвращают неоднозначные ситуации с «ничейными» результатами) и выполняет k-проходную перекрестную проверку для каждого варианта. Как видно из оценок точности и стандартных отклонений, при значении $k = 1$ достигается наибольшая точность прогнозирования для набора данных Digits. Рост значений k ведет к снижению точности:

```
In [47]: for k in range(1, 20, 2):
    ...:     kfold = KFold(n_splits=10, random_state=11, shuffle=True)
    ...:     knn = KNeighborsClassifier(n_neighbors=k)
    ...:     scores = cross_val_score(estimator=knn,
    ...:         X=digits.data, y=digits.target, cv=kfold)
    ...:     print(f'k={k:<2}; mean accuracy={scores.mean():.2%}; ' +
    ...:           f'standard deviation={scores.std():.2%}')
    ...:
k=1 ; mean accuracy=98.83%; standard deviation=0.58%
k=3 ; mean accuracy=98.78%; standard deviation=0.78%
k=5 ; mean accuracy=98.72%; standard deviation=0.75%
k=7 ; mean accuracy=98.44%; standard deviation=0.96%
k=9 ; mean accuracy=98.39%; standard deviation=0.80%
k=11; mean accuracy=98.39%; standard deviation=0.80%
k=13; mean accuracy=97.89%; standard deviation=0.89%
k=15; mean accuracy=97.89%; standard deviation=1.02%
k=17; mean accuracy=97.50%; standard deviation=1.00%
k=19; mean accuracy=97.66%; standard deviation=0.96%
```

При проведении машинного обучения, особенно с переходом к большим данным и глубокому обучению, исследователь должен знать и свои данные, и свои инструменты. Например, с ростом k время обработки стремительно возрастает, потому что метод k-NN должен выполнить больше вычислений для нахождения ближайших соседей. Применение функции `cross_validate` позволяет провести перекрестную проверку и выполнить хронометраж результатов.

14.4. Практический пример: временные ряды и простая линейная регрессия

В предыдущем разделе продемонстрирована классификация, в которой каждый образец был связан с одним из *дискретных* классов. В этом разделе продолжится обсуждение простой линейной регрессии — простейшего из регрессионных алгоритмов (см. в этой связи также раздел «Введение в data science» главы 10). Напомним, что для заданной коллекции числовых значений, представляющих независимую и зависимую переменную, простая линейная регрессия описывает отношения этих переменных прямой линией, называемой *регрессионной прямой*.

Ранее простая линейная регрессия была использована в примере с временными рядами средней январской температуры в Нью-Йорке за период с 1895 по 2018 год. Кроме того, в этом примере использовалась функция `regplot` библиотеки Seaborn для построения диаграммы разброса данных с соответствующей регрессионной прямой, а функция `linregress` модуля `scipy.stats` — для вычисления угла наклона регрессионной прямой и точки пересечения с осью. Полученные значения были использованы для прогнозирования будущих и оценки прошлых температур. В этом разделе рассматриваются:

- Использование *оценщика scikit-learn* для повторной реализации простой линейной регрессии, продемонстрированной в главе 10.
- Использование функции `scatterplot` библиотеки Seaborn для графического вывода данных и функции `plot` библиотеки Matplotlib для вывода регрессионной прямой.
- Использование значений угла наклона и точки пересечения, вычисленных оценщиком scikit-learn, для построения прогнозов.

Позднее мы рассмотрим *множественную линейную регрессию* (которая называется также *линейной регрессией*).

Для вашего удобства мы разместили данные в каталоге примеров ch14 в файле ave_hi_nyc_jan_1895-2018.csv. Как и прежде, IPython следует запускать с ключом `--matplotlib`:

```
ipython --matplotlib
```

Загрузка средних температур в коллекцию DataFrame

Загрузите данные из файла ave_hi_nyc_jan_1895-2018.csv, переименуйте столбец `'Value'` в `'Temperature'`, удалите `01` в конце каждого значения даты и выведите несколько образцов данных:

```
In [1]: import pandas as pd

In [2]: nyc = pd.read_csv('ave_hi_nyc_jan_1895-2018.csv')

In [3]: nyc.columns = ['Date', 'Temperature', 'Anomaly']

In [4]: nyc.Date = nyc.Date.floordiv(100)

In [5]: nyc.head(3)
Out[5]:
   Date  Temperature  Anomaly
0  1895         34.2     -3.2
1  1896         34.7     -2.7
2  1897         35.5     -1.9
```

Разбиение данных для обучения и тестирования

В этом примере будет использоваться оценщик `LinearRegression` из `sklearn.linear_model`. По умолчанию он использует *все* числовые признаки в наборе данных, выполняя *множественную линейную регрессию* (см. следующий раздел). Выполним *простую линейную регрессию*, используя *один* признак как независимую переменную. В наборе данных необходимо выбрать один признак (`Date`) из набора данных.

При выборе одного столбца в двумерном `DataFrame` результат представляет собой *одномерную* коллекцию `Series`. Однако оценщики scikit-learn требуют, чтобы в качестве обучающих и тестовых данных использовались *двумерные массивы* (или двумерные структуры, *сходные с массивами*, например списки списков или коллекции pandas `DataFrame`). Чтобы использовать одномерные массивы с оценщиком, необходимо преобразовать их из одномерного массива с *n* элементами в двумерный массив с *n строками* и одним *столбцом*.

Как и прежде, данные будут разбиты на обучающий и тестовый наборы. И снова ключевой аргумент `random_state` используется для обеспечения воспроизводимости результатов:

```
In [6]: from sklearn.model_selection import train_test_split

In [7]: X_train, X_test, y_train, y_test = train_test_split(
   ...:     nyc.Date.values.reshape(-1, 1), nyc.Temperature.values,
   ...:     random_state=11)
   ...:
```

Выражение `nyc.Date` возвращает коллекцию `Series` для столбца `Date`, атрибут которой `values` возвращает массив NumPy со значениями коллекции. Для преобразования одномерного массива в двумерный вызовем *метод* `reshape` массива. Обычно в двух аргументах передается точное количество строк и столбцов, но первый аргумент `-1` означает, что метод `reshape` должен *вычислить* количество строк на основании количества столбцов (1) и количества элементов (124) в массиве. Преобразованный массив содержит только один столбец, поэтому `reshape` делает вывод, что количество строк равно 124: разместить 124 элемента в один столбец можно, только распределив их по 124 строкам.

Для проверки пропорции обучающих тестовых данных (75% к 25%) запросим размеры `X_train` и `X_test`:

```
In [8]: X_train.shape
Out[8]: (93, 1)

In [9]: X_test.shape
Out[9]: (31, 1)
```

Обучение модели

В scikit-learn нет отдельного класса для простой линейной регрессии, потому что простая линейная регрессия является частным случаем множественной линейной регрессии, поэтому мы воспользуемся оценщиком `LinearRegression`:

```
In [10]: from sklearn.linear_model import LinearRegression

In [11]: linear_regression = LinearRegression()

In [12]: linear_regression.fit(X=X_train, y=y_train)
Out[12]:
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

После обучения оценщика `fit` возвращает оценщика и IPython выводит строковое представление. Описания настроек по умолчанию доступны по адресу:

http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Чтобы найти регрессионную прямую с наилучшей подгонкой к данным, оценщик `LinearRegression` в итеративном режиме регулирует укол наклона и точку пересечения для минимизации суммы квадратов расстояний точек данных от линии (об определении значений параметров см. раздел «Введение в data science» главы 10).

После этого значения угла наклона и точки пересечения с осью, используемые в формуле $y = mx + b$, могут использоваться для прогнозирования. Угол наклона хранится в атрибуте `coeff_` оценщика (m в формуле), а точка пересечения — в атрибуте `intercept_` (b в формуле):

```
In [13]: linear_regression.coef_
Out[13]: array([0.01939167])

In [14]: linear_regression.intercept_
Out[14]: -0.30779820252656265
```

Позднее эти значения будут использованы для вывода регрессионной прямой и прогнозирования для конкретных дат.

Тестирование модели

Протестируем модель по данным из `X_test` и проверим прогнозы по набору данных, выводя прогнозируемые и ожидаемые значения для каждого пятого элемента (о том, как оценить точность модели, см. раздел 14.5.8):

```
In [15]: predicted = linear_regression.predict(X_test)

In [16]: expected = y_test

In [17]: for p, e in zip(predicted[::5], expected[::5]):
    ...:     print(f'predicted: {p:.2f}, expected: {e:.2f}')
    ...:
predicted: 37.86, expected: 31.70
predicted: 38.69, expected: 34.80
predicted: 37.00, expected: 39.40
predicted: 37.25, expected: 45.70
predicted: 38.05, expected: 32.30
predicted: 37.64, expected: 33.80
predicted: 36.94, expected: 39.70
```

Прогнозирование будущих температур и оценка прошлых температур

Воспользуемся полученными значениями угла наклона и точки пересечения для прогнозирования средней температуры в январе 2019 года, а также оценки средней температуры в январе 1890 года. Лямбда-выражение в следующем фрагменте реализует формулу:

$$y = mx + b$$

Значение `coef_` используется вместо m, а значение `intercept_` — вместо b.

```
In [18]: predict = (lambda x: linear_regression.coef_ * x +
    ...:                      linear_regression.intercept_)
    ...:

In [19]: predict(2019)
Out[19]: array([38.84399018])

In [20]: predict(1890)
Out[20]: array([36.34246432])
```

Визуализация набора данных с регрессионной прямой

Теперь построим диаграмму разброса данных при помощи функции `scatterplot` библиотеки Seaborn и функции `plot` библиотеки Matplotlib. Для вывода точек данных воспользуемся методом `scatterplot` с коллекцией `DataFrame` с именем `nyc` :

```
In [21]: import seaborn as sns

In [22]: axes = sns.scatterplot(data=nyc, x='Date', y='Temperature',
    ...:     hue='Temperature', palette='winter', legend=False)
    ...:
```

Ключевые аргументы:

- `data` — коллекция `DataFrame` (`nyc`) с выводимыми данными;
- `x` и `y` — имена столбцов `nyc`, которые являются источником данных по осям x и y соответственно. В данном случае `x` содержит имя столбца `'Date'`, а `y` — `'Temperature'`. Соответствующие значения столбцов образуют пары координат x-y, наносимые на диаграмму;

- `hue` — столбец, данные которого используются для определения цветов точек (`'Temperature'`). В нашем примере цвет особой роли не играет, но мы хотели сделать диаграмму более привлекательной;
- `palette` — цветовая карта Matplotlib, по которой выбираются цвета точек;
- `legend=False` — на диаграмме разброса данных не должны выводиться условные обозначения. По умолчанию используется значение `True`, но в нашем примере условные обозначения не нужны.

Как и в главе 10, изменим масштаб оси *y*, чтобы при выводе регрессионной прямой линейность отношения была более очевидной:

```
In [23]: axes.set_ylim(10, 70)
Out[23]: (10, 70)
```

Перейдем к выводу регрессионной прямой. Начнем с создания массива, содержащего минимальные и максимальные значения даты из `nyc.Date`. Они станут координатами *x* начальной и конечной точек регрессионной прямой:

```
In [24]: import numpy as np

In [25]: x = np.array([min(nyc.Date.values), max(nyc.Date.values)])
```

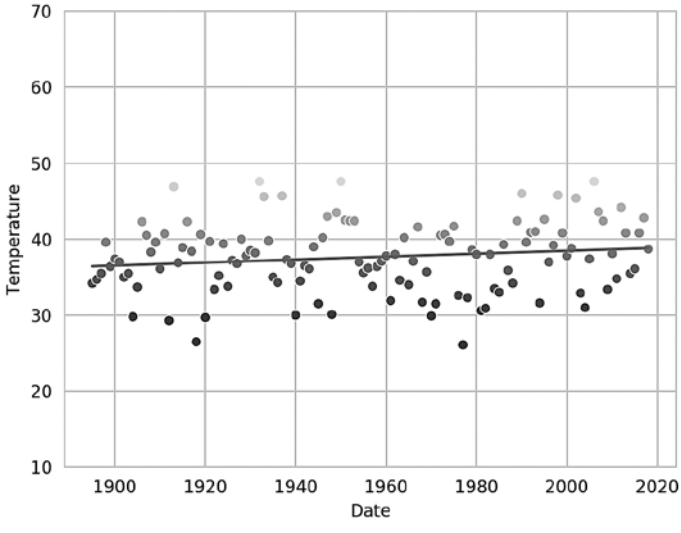
В результате передачи `predict` массива `x` во фрагменте [26] будет получен массив соответствующих прогнозируемых значений, которые будут использоваться в качестве координат *y*:

```
In [26]: y = predict(x)
```

Наконец, функция `plot` библиотеки Matplotlib рисует линию по массивам `x` и `y`, представляющим координаты *x* и *y* точек соответственно:

```
In [27]: import matplotlib.pyplot as plt

In [28]: line = plt.plot(x, y)
```

Полученная диаграмма разброса данных с регрессионной прямой изображена на следующей диаграмме. Она практически идентична той, что приведена в разделе «Введение в data science» главы 10.



Чрезмерная/недостаточная подгонка

При создании модели нужно в первую очередь стремиться к тому, чтобы эта модель выдавала точные прогнозы для данных, которые ей пока неизвестны. Две распространенные проблемы, препятствующие точному прогнозированию, — *чрезмерная* и *недостаточная подгонка*:

- *Недостаточная подгонка* происходит в том случае, если модель слишком проста для построения прогнозов на основании тренировочных данных. Например, линейная модель (скажем, простая линейная регрессия) используется в задаче, которая на самом деле требует нелинейной модели. Например, температуры существенно изменяются на протяжении четырех времен года. Если вы попытаетесь создать обобщенную модель, которая может прогнозировать температуры круглый год, модель простой линейной регрессии приведет к недостаточной подгонке данных.
- *Чрезмерная подгонка* происходит при излишней сложности модели. Крайний случай такого рода — модель, запоминающая свои обучающие данные. Такое решение приемлемо, если новые данные будут *полностью* совпадать с обучающими, но обычно это не так. При построении прогнозов на основании модели с чрезмерной подгонкой для новых данных, совпадающих с обучающими, будут сделаны идеально точные прогнозы, но такая модель не будет знать, что делать с данными, которые ей еще не встречались.

Дополнительную информацию о чрезмерной и недостаточной подгонке можно найти здесь:

https://ru.wikipedia.org/wiki/Переобучение

https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

14.5. Практический пример: множественная линейная регрессия с набором данных California Housing

В разделе «Введение в data science» главы 10 простая линейная регрессия выполнена на основе небольшого набора погодных данных с использованием pandas, функции `regplot` библиотеки Seaborn и функции `linregress` модуля `stats` библиотеки SciPy. В предыдущем разделе этот пример был реализован заново на базе оценщика `LinearRegression` библиотеки scikit-learn, функции `scatterplot` библиотеки Seaborn и функции `plot` библиотеки Matplotlib.

Теперь реализуем линейную регрессию для гораздо более объемного реального набора данных *California Housing*[1], входящего в поставку scikit-learn. Выполним *множественную линейную регрессию*, использующую (см. ниже) все имеющиеся числовые признаки для построения более сложных прогнозов цен, чем при использовании одного признака или подмножества признаков. Как и прежде, scikit-learn выполнит за вас большую часть работы — `LinearRegression` выполняет множественную линейную регрессию по умолчанию.

Для визуализации данных будут использоваться Matplotlib и Seaborn, поэтому IPython следует запускать с поддержкой Matplotlib:

```
ipython --matplotlib
```

14.5.1. Загрузка набора данных

Согласно описанию набора данных California Housing Prices в scikit-learn, «этот набор данных был сформирован по данным переписи 1990 года в США, одна строка данных представляет переписную группу кварталов — наименьшую гео-

[1] http://lib.stat.cmu.edu/datasets. Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions, Statistics and Probability Letters, 33 (1997) 291–297. Данные переданы в архив StatLib Datasets Archive Келли Пейс (Kelley Pace) (kpace@unix1.sncc.lsu.edu). [9 ноября 1999 г.].

графическую единицу, для которой Федеральная служба государственной статистики США публикует образцы данных (население типичной группы кварталов составляет от 600 до 3000 человек)». Набор данных содержит 20 640 образцов (по одному на группу кварталов), каждый из которых обладает восемью признаками:

- Медианный доход — в десятках тысяч долларов, то есть значение `8.37` соответствует 83 700 долларов.
- Медианный возраст дома — в наборе данных максимальное значение этого признака равно 52.
- Среднее количество комнат.
- Среднее количество спален.
- Население квартала.
- Средняя населенность дома.
- Широта блока домов.
- Долгота блока домов.

С каждым образцом также связывается соответствующая медианная стоимость дома (в сотнях тысяч USD), так что `3.55` представляет сумму 355 000 долларов. В наборе данных максимальное значение этого признака равно 5, что соответствует 500 000 долларов. Уместно предположить, что чем больше в доме спален, чем больше в нем комнат, и что чем выше доход владельца, тем выше будет оценочная стоимость дома. Объединение этих признаков в прогнозы с большей вероятностью обеспечит и более точные прогнозы.

Загрузка данных

Загрузим набор данных и исследуем его. Функция `fetch_california_housing` из модуля `sklearn.datasets` возвращает объект `Bunch` с данными и другой информацией о наборе данных:

```
In [1]: from sklearn.datasets import fetch_california_housing

In [2]: california = fetch_california_housing()
```

Вывод описания набора данных

Загрузим описание набора данных. Информация `DESCR` включает:

- `Количество экземпляров` — набор данных содержит 20 640 образцов.

- `Количество атрибутов` — восемь признаков (атрибутов) на образец.
- `Информация об атрибутах` — описания признаков.
- `Отсутствующие значения атрибутов` — в этом наборе данных отсутствующих значений нет.

Согласно описанию, целевой переменной в этом наборе данных является *медианная оценочная стоимость дома* — это значение, которое мы будем пытаться прогнозировать с использованием множественной линейной регрессии:

```
In [3]: print(california.DESCR)
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and
        the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/
...
```

И снова атрибуты `data` и `target` объекта `Bunch` включают массивы NumPy, содержащие 20 640 образцов и соответствующие им целевые значения. Можно проверить количество образцов (строк) и признаков (столбцов) по атрибуту `shape` массива `data`, который показывает, что данные состоят из 20 640 строк и восьми столбцов:

```
In [4]: california.data.shape
Out[4]: (20640, 8)
```

Аналогичным образом можно убедиться в том, что количество целевых значений, то есть медианных оценочных стоимостей дома, соответствует количеству образцов: обратимся к атрибуту `shape` массива `target`:

```
In [5]: california.target.shape
Out[5]: (20640,)
```

Атрибут `feature_names` объекта `Bunch` содержит имена, соответствующие каждому столбцу в массиве `data`:

```
In [6]: california.feature_names
Out[6]:
['MedInc',
 'HouseAge',
 'AveRooms',
 'AveBedrms',
 'Population',
 'AveOccup',
 'Latitude',
 'Longitude']
```

14.5.2. Исследование данных средствами Pandas

Воспользуемся коллекцией pandas `DataFrame` для дальнейшего исследования данных (в следующем разделе коллекция `DataFrame` будет использоваться с Seaborn для визуализации данных). Начнем с импортирования pandas и настройки некоторых параметров:

```
In [7]: import pandas as pd

In [8]: pd.set_option('precision', 4)

In [9]: pd.set_option('max_columns', 9)

In [10]: pd.set_option('display.width', None)
```

В этих вызовах `set_option`:

- `'precision'` — максимальное количество цифр, выводимых в дробной части;
- `'max_columns'` — максимальное количество столбцов, выводимых в строковом представлении `DataFrame`. По умолчанию, если pandas не может разместить все столбцы слева направо, то средние столбцы усекаются, а вместо них выводится многоточие (`...`). Параметр `'max_columns'` раз-

решает выводить все столбцы в многострочном формате. Коллекция `DataFrame` состоит из девяти столбцов — восемь признаков набора данных `california.data` и дополнительный столбец для целевой медианной оценочной стоимости дома (`california.target`);

- `'display.width'` — задает ширину (в символах) области приглашения командной строки (Windows), терминала (macOS/Linux) или командной оболочки (Linux). Значение `None` приказывает pandas автоматически определять ширину вывода при форматировании строковых представлений `Series` и `DataFrame`.

Затем создадим коллекцию `DataFrame` по данным `Bunch`, массивам `target` и `feature_names`. Первый фрагмент (см. ниже) создает исходную коллекцию `DataFrame` по данным из `california.data` и именам столбцов, определяемым атрибутом `california.feature_names`. Вторая команда добавляет столбец для медианной стоимости дома, хранящейся в `california.target`:

```
In [11]: california_df = pd.DataFrame(california.data,
    ...:                              columns=california.feature_names)
    ...:

In [12]: california_df['MedHouseValue'] = pd.Series(california.target)
```

Для просмотра небольшого подмножества данных можно воспользоваться функцией `head`. Обратите внимание: pandas выводит первые шесть столбцов `DataFrame`, после чего пропускает строку и выводит остальные столбцы. Знак \ справа от имени заголовка столбца `"AveOccup"` означает, что в выводе остаются столбцы, которые будут выведены ниже. Знак \ появляется только в том случае, если окну, в котором работает IPython, не хватает ширины для вывода всех столбцов слева направо:

```
In [13]: california_df.head()
Out[13]:
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  \
0  8.3252      41.0    6.9841     1.0238       322.0    2.5556
1  8.3014      21.0    6.2381     0.9719      2401.0    2.1098
2  7.2574      52.0    8.2881     1.0734       496.0    2.8023
3  5.6431      52.0    5.8174     1.0731       558.0    2.5479
4  3.8462      52.0    6.2819     1.0811       565.0    2.1815

   Latitude  Longitude  MedHouseValue
0     37.88    -122.23          4.526
1     37.86    -122.22          3.585
2     37.85    -122.24          3.521
3     37.85    -122.25          3.413
4     37.85    -122.25          3.422
```

Чтобы составить общее представление о данных в каждом столбце, вычислим сводную статистику `DataFrame`. Учтите, что значения среднего дохода и стоимости дома (в сотнях тысяч USD) относятся к 1990-м годам; сейчас эти значения существенно выше:

```
In [14]: california_df.describe()
Out[14]:
              MedInc     HouseAge     AveRooms    AveBedrms   Population     \
count   20640.0000   20640.0000   20640.0000   20640.0000   20640.0000
mean        3.8707      28.6395       5.4290       1.0967    1425.4767
std         1.8998      12.5856       2.4742       0.4739    1132.4621
min         0.4999       1.0000       0.8462       0.3333       3.0000
25%         2.5634      18.0000       4.4407       1.0061     787.0000
50%         3.5348      29.0000       5.2291       1.0488    1166.0000
75%         4.7432      37.0000       6.0524       1.0995    1725.0000
max        15.0001      52.0000     141.9091      34.0667   35682.0000

           AveOccup     Latitude    Longitude   MedHouseValue
count   20640.0000   20640.0000   20640.0000      20640.0000
mean        3.0707      35.6319    -119.5697          2.0686
std        10.3860       2.1360       2.0035          1.1540
min         0.6923      32.5400    -124.3500          0.1500
25%         2.4297      33.9300    -121.8000          1.1960
50%         2.8181      34.2600    -118.4900          1.7970
75%         3.2823      37.7100    -118.0100          2.6472
max      1243.3333      41.9500    -114.3100          5.0000
```

14.5.3. Визуализация признаков

Для исследования данных будет полезно визуализировать данные и вывести на диаграмме связь целевого значения с *каждым* признаком, в данном случае — чтобы увидеть, как медианная оценочная стоимость относится к каждому признаку. Воспользуемся методом `sample` коллекции `DataFrame` для случайного выбора 10% из 20 640 образцов для построения диаграммы:

```
In [15]: sample_df = california_df.sample(frac=0.1, random_state=17)
```

Ключевой аргумент `frac` определяет долю отбираемых данных (`0.1` соответствует 10%), а ключевой аргумент `random_state` инициализирует генератор случайных чисел. Целочисленное значение инициализации (`17`), выбранное произвольно, обеспечивает *воспроизводимость* результатов. Каждый раз, когда вы используете *фиксированное* значение инициализации, метод `sample` выбирает *одно и то же* случайное подмножество строк `DataFrame`. После этого при нанесении данных на диаграмму будут получены *одни и те же* результаты.

После этого библиотеки Matplotlib и Seaborn используются для построения диаграмм разброса данных для каждого из восьми признаков. Отметим, что такие диаграммы могут строить обе библиотеки. Диаграммы Seaborn выглядят лучше и требуют меньшего объема кода, поэтому для построения диаграмм будет использоваться Seaborn. Импортируем обе библиотеки и воспользуемся функцией `set` библиотеки Seaborn для увеличения шрифтов каждой диаграммы в два раза по сравнению с размером по умолчанию:

```
In [16]: import matplotlib.pyplot as plt

In [17]: import seaborn as sns

In [18]: sns.set(font_scale=2)

In [19]: sns.set_style('whitegrid')
```

В следующем фрагменте выводятся диаграммы разброса данных[1]. Ось *x* каждой диаграммы представляет один признак, а ось *y* — медианную стоимость дома (`california.target`). Таким образом, диаграмма показывает, как каждый признак и стоимость дома связаны друг с другом. Каждая диаграмма разброса данных выводится в отдельном окне. Окна отображаются в порядке перечисления признаков во фрагменте [6]; окно, выведенное последним, находится на первом плане:

```
In [20]: for feature in california.feature_names:
    ...:     plt.figure(figsize=(16, 9))
    ...:     sns.scatterplot(data=sample_df, x=feature,
    ...:                     y='MedHouseValue', hue='MedHouseValue',
    ...:                     palette='cool', legend=False)
    ...:
```

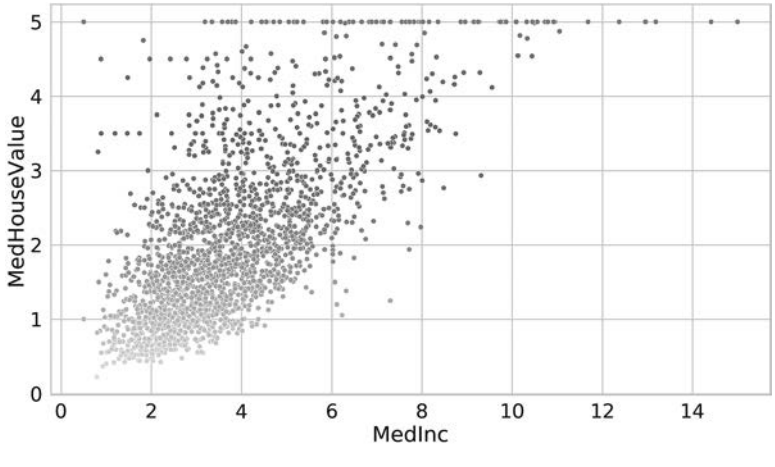
Для каждого имени признака этот фрагмент сначала создает объект `Figure` библиотеки matplotlib размером 16 × 9 дюймов — мы выводим множество точек данных, поэтому придется использовать окно большего размера. Если размер окна больше размера экрана, то Matplotlib подгоняет `Figure` по размеру экрана. Seaborn использует текущий объект `Figure` для вывода диаграммы разброса данных. Если вы не создадите объект `Figure` заранее, то Seaborn создаст его автоматически. Мы создали `Figure`, чтобы вывести большое окно для диаграммы разброса данных с более 2000 точек.

Затем фрагмент создает диаграмму разброса данных Seaborn, у которой ось *x* представляет текущий признак, ось *y* — `'MedHouseValue'` (*медианная оценочная*

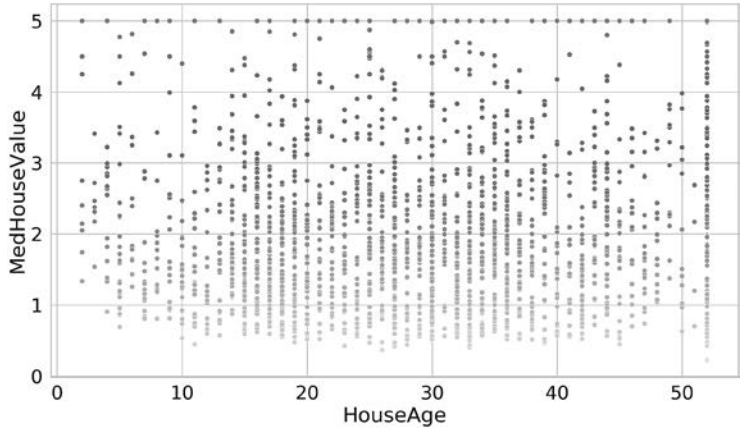
[1] При выполнении этого кода в IPython каждое окно будет открываться поверх предыдущего. После закрытия окна на экране появится то окно, которое находилось позади него.

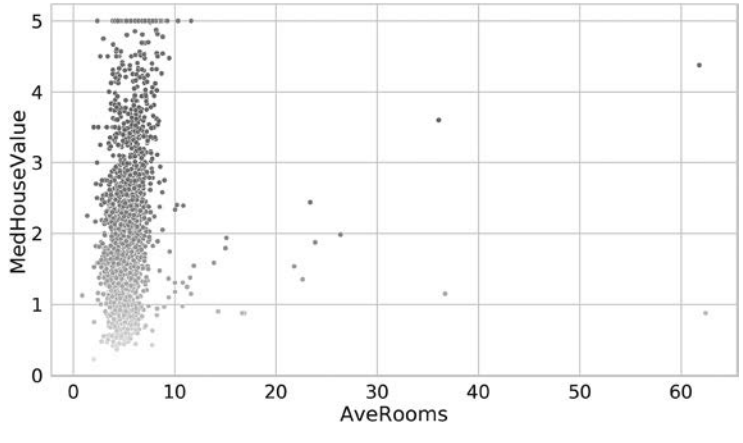
стоимость дома), а `'MedHouseValue'` определяет цвета точек (`hue`). Некоторые моменты, на которые стоит обратить внимание на этих диаграммах:

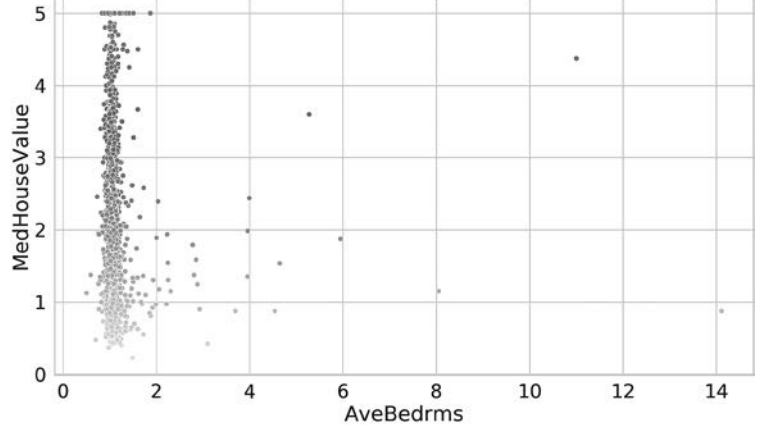
- На каждой из диаграмм с широтой и долготой наблюдаются две области со значительной плотностью точек. Если вы проведете поиск в интернете для значений широты и долготы, в которых наблюдается скопление точек, то увидите, что они представляют окрестности Лос-Анджелеса и Сан-Франциско, для которых характерны высокие цены на недвижимость.
- На каждой диаграмме присутствует горизонтальная линия точек для значения 5 на оси *y*, соответствующего медианной стоимости 500 000 долларов. Наивысшим значением стоимости дома, которое можно было выбрать в переписи 1990 года, было «500 000 долларов и выше»[1]. Таким образом, любая группа кварталов с медианной стоимостью более 500 000 долларов регистрировалась в наборе данных как значение 5. Возможность выявления подобных характеристик — веский довод для проведения исследования данных и визуализации.
- На диаграмме возраста `HouseAge` присутствует вертикальная линия точек для значения 52 на оси *x*. Наибольший возраст дома, который можно было выбрать в переписи 1990 года, равен 52. Таким образом, любая группа кварталов с медианным возрастом дома свыше 52 регистрировалась в наборе данных как значение 52.

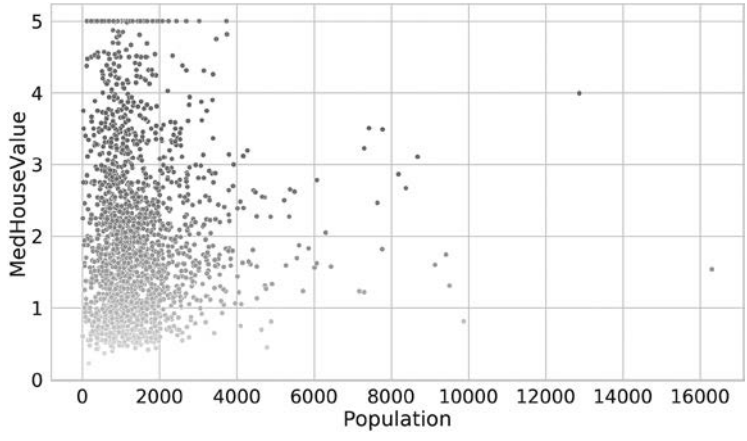


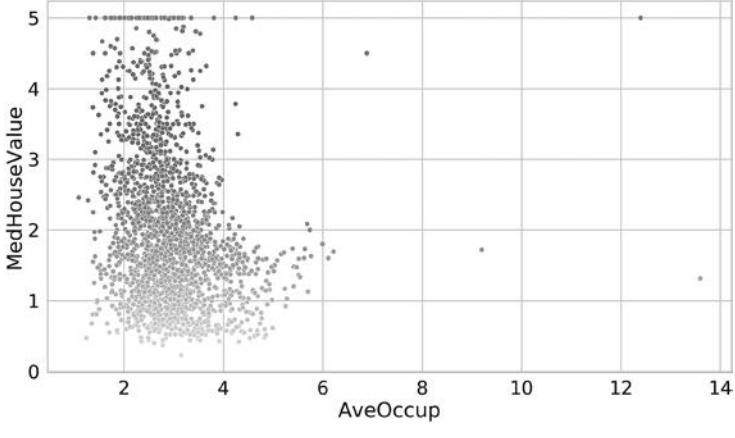
[1] https://www.census.gov/prod/1/90dec/cph4/appdxe.pdf.

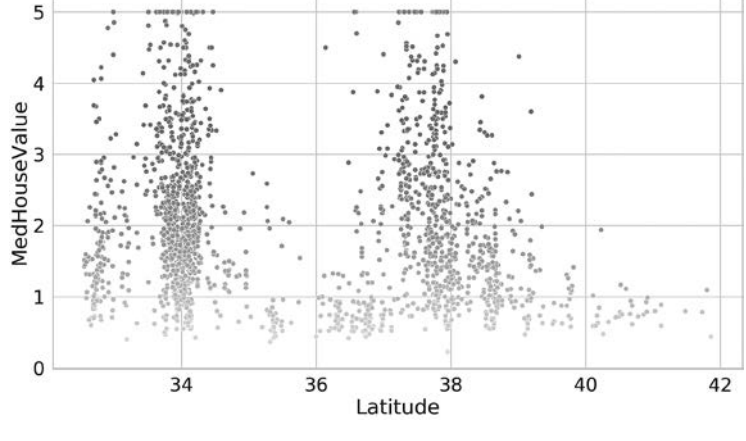

MedHouseValue
HouseAge
0
10
20
30
40
50
0
1
2
3
4
5

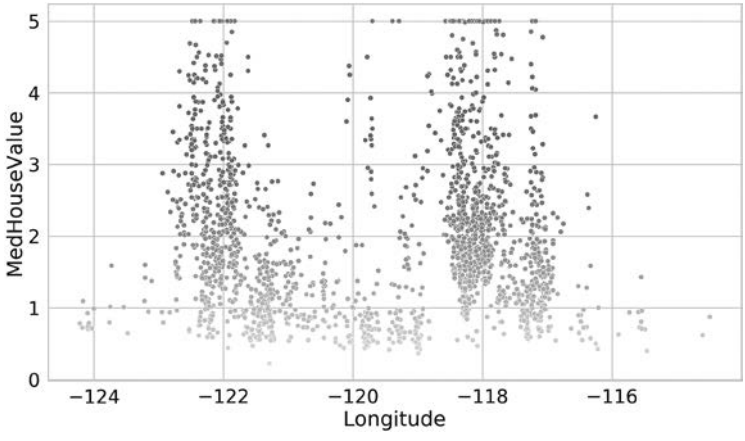

MedHouseValue
AveRooms
0
10
20
30
40
50
60
0
1
2
3
4
5


MedHouseValue
AveBedrms
0
2
4
6
8
10
12
14
0
1
2
3
4
5


MedHouseValue
Population
0
1
2
3
4
5
0
2000
4000
6000
8000
10000
12000
14000
16000


MedHouseValue
AveOccup
0
1
2
3
4
5
2
4
6
8
10
12
14


MedHouseValue
Latitude
0
1
2
3
4
5
34
36
38
40
42



14.5.4. Разбиение данных для обучения и тестирования

Как и прежде, чтобы подготовиться к обучению и тестированию модели, разобьем данные на обучающий и тренировочный наборы при помощи функции `train_test_split`, а затем проверим их размеры:

```
In [21]: from sklearn.model_selection import train_test_split

In [22]: X_train, X_test, y_train, y_test = train_test_split(
    ...:     california.data, california.target, random_state=11)
    ...:

In [23]: X_train.shape
Out[23]: (15480, 8)

In [24]: X_test.shape
Out[24]: (5160, 8)
```

Ключевой аргумент `random_state` функции `train_test_split` использовался для инициализации генератора случайных чисел с целью обеспечения воспроизводимости.

14.5.5. Обучение модели

На следующем шаге будет выполнено обучение модели. По умолчанию оценщик `LinearRegression` использует *все* признаки массива набора данных для

выполнения множественной линейной регрессии. Если все признаки являются *категорийными*, а не числовыми, то происходит ошибка. Если набор данных содержит категорийные данные, следует провести предварительную обработку категорийных признаков в числовые (как это будет сделано в следующей главе) либо исключить категорийные признаки из процесса обучения. К преимуществам наборов данных, входящих в поставку scikit-learn, следует отнести то, что они уже имеют правильный формат для применения машинного обучения с моделями scikit-learn.

Как было показано в двух предыдущих фрагментах, каждый из наборов `X_train` и `X_test` содержит восемь столбцов — по одному на признак. Создадим оценщика `LinearRegression` и вызовем его метод `fit` для обучения оценщика с данными `X_train` и `y_train`:

```
In [25]: from sklearn.linear_model import LinearRegression

In [26]: linear_regression = LinearRegression()

In [27]: linear_regression.fit(X=X_train, y=y_train)
Out[27]:
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Множественная линейная регрессия выдает для каждого признака отдельный коэффициент (хранится в `coeff_`) и отдельную точку пересечения (хранится в `intercept_`):

```
In [28]: for i, name in enumerate(california.feature_names):
    ...:     print(f'{name:>10}: {linear_regression.coef_[i]}')
    ...:
    MedInc: 0.4377030215382206
  HouseAge: 0.009216834565797713
  AveRooms: -0.10732526637360985
 AveBedrms: 0.611713307391811
Population: -5.756822009298454e-06
  AveOccup: -0.0033845664657163703
  Latitude: -0.419481860964907
 Longitude: -0.4337713349874016

In [29]: linear_regression.intercept_
Out[29]: -36.88295065605547
```

Для положительных коэффициентов медианная стоимость дома *возрастает* с *ростом* признака. Для отрицательных коэффициентов медианная стоимость дома *убывает* с *ростом* признака. Учтите, что коэффициент для заселенности

имеет *отрицательный показатель степени* (`e-06`), так что значение коэффициента в действительности равно `-0.00000575682200929845 4`. Величина близка к нулю, то есть заселенность группы кварталов практически не влияет на значение медианной стоимости. Полученные значения можно использовать со следующим уравнением для прогнозирования:

$$y = m_1x_1 + m_2x_2 + ... + m_nx_n + b,$$

где $m_1, m_2, ..., m_n$ — коэффициенты признаков;

b — точка пересечения;

$x_1, x_2, ..., x_n$ — значения признаков (то есть значения независимых переменных);

y — прогнозируемое значение (то есть зависимая переменная).

14.5.6. Тестирование модели

Теперь протестируем модель вызовом метода `predict` оценщика, передавая тестовые образцы в аргументе. Как и в предыдущих примерах, массив прогнозов хранится в `predicted`, а массив ожидаемых значений — в `expected`:

```
In [30]: predicted = linear_regression.predict(X_test)

In [31]: expected = y_test
```

Рассмотрим первые пять прогнозов и соответствующие значения из `expected`:

```
In [32]: predicted[:5]
Out[32]: array([1.25396876, 2.34693107, 2.03794745, 1.8701254,
2.53608339])

In [33]: expected[:5]
Out[33]: array([0.762, 1.732, 1.125, 1.37, 1.856])
```

В задаче классификации прогнозы представляли собой дискретные классы, совпадающие с существующими классами в наборе данных. При регрессии получить точные прогнозы сложнее из-за непрерывности вывода. Каждое возможное значение $x_1, x_2 ... x_n$ в формуле

$$y = m_1x_1 + m_2x_2 + ... m_nx_n + b$$

прогнозирует некоторое значение.

14.5.7. Визуализация ожидаемых и прогнозируемых цен

Сравним ожидаемую медианную ценность дома с прогнозируемой для тестовых данных. Создадим коллекцию `DataFrame` со столбцами для ожидаемых и прогнозируемых значений:

```
In [34]: df = pd.DataFrame()

In [35]: df['Expected'] = pd.Series(expected)

In [36]: df['Predicted'] = pd.Series(predicted)
```

Построим визуализацию данных в форме диаграммы разброса данных, у которой ось *x* представляет ожидаемую (целевую) стоимость, а ось *y* — прогнозируемую:

```
In [37]: figure = plt.figure(figsize=(9, 9))

In [38]: axes = sns.scatterplot(data=df, x='Expected', y='Predicted',
    ...:     hue='Predicted', palette='cool', legend=False)
    ...:
```

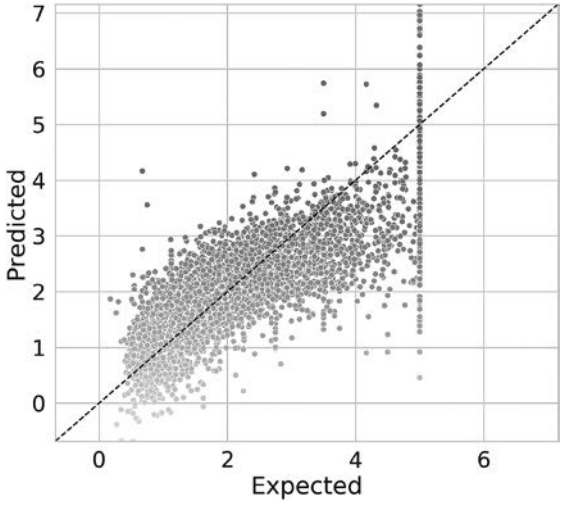
Затем установим ограничения по осям *x* и *y*, чтобы масштаб обеих осей был одинаковым:

```
In [39]: start = min(expected.min(), predicted.min())

In [40]: end = max(expected.max(), predicted.max())

In [41]: axes.set_xlim(start, end)
Out[41]: (-0.6830978604144491, 7.155719818496834)

In [42]: axes.set_ylim(start, end)
Out[42]: (-0.6830978604144491, 7.155719818496834)
```

Теперь построим линию, которая представляет *идеальные прогнозы* (обратите внимание: это *не* регрессионная прямая). Следующий фрагмент выводит линию от левого нижнего угла диаграммы (`start`, `start`) до правого верхнего угла (`end`, `end`). Третий аргумент (`'k--'`) обозначает стиль линии. Буква `k` представляет черный цвет, а `--` обозначает, что выводимая линия должна быть пунктирной:

```
In [43]: line = plt.plot([start, end], [start, end], 'k--')
```

Если бы каждое прогнозируемое значение совпадало с ожидаемым, то все точки лежали бы на пунктирной линии. Как видно из следующей диаграммы, с возрастанием ожидаемой медианной стоимости большая часть прогнозируемых точек оказывается ниже линии. Таким образом, с ростом *ожидаемой* медианной стоимости *прогнозы* модели оказываются заниженными.



14.5.8. Метрики регрессионной модели

Scikit-learn предоставляет множество метрических функций для оценки того, насколько качественно оценщики прогнозируют результаты, и для сравнения оценщиков и выбора лучшего(-их) для вашего конкретного исследования. Эти метрики зависят от типа оценщика. Например, функции `confusion_matrix` и `classification_report` модуля `sklearn.metrics`, использованные в примере с классификацией набора данных Digits, принадлежат к числу метрических функций, предназначенных для оценки *классификационных* оценщиков.

К числу других метрик регрессионных оценщиков принадлежит *коэффициент детерминации* модели, также называемый *коэффициентом* R^2. Чтобы вычислить коэффициент R^2 оценщика, вызовите функцию `r2_score` модуля `sklearn.metrics` с массивами, представляющими ожидаемые и прогнозируемые результаты:

```
In [44]: from sklearn import metrics

In [45]: metrics.r2_score(expected, predicted)
Out[45]: 0.6008983115964333
```

Значения R^2 лежат в диапазоне от 0,0 до 1,0 (1,0 — лучшее значение). Коэффициент R^2 = 1,0 означает, что оценщик идеально прогнозирует значение зависимой переменной по заданным значениям независимой(-ых) переменной(-ых). Коэффициент R^2 = 0,0 означает, что модель не способна выдать прогноз с какой-либо точностью на основании значений независимых переменных.

Другая распространенная метрика регрессионных моделей — *среднеквадратичная ошибка* — вычисляется следующим образом:

- вычисляется разность между каждым ожидаемым и прогнозируемым значением;
- каждая разность возводится в квадрат;
- вычисляется среднее значение всех квадратов.

Чтобы вычислить среднеквадратичную ошибку, вызовите функцию `mean_squared_error` (из модуля `sklearn.metrics`) с массивами, представляющими ожидаемые и прогнозируемые результаты:

```
In [46]: metrics.mean_squared_error(expected, predicted)
Out[46]: 0.5350149774449119
```

При сравнении оценщиков по метрике среднеквадратичной ошибки тот оценщик, у которого это значение находится ближе всего к 0, обеспечивает наилучшую подгонку для ваших данных. В следующем разделе мы выполним несколько регрессионных оценщиков с набором данных California Housing. За списком метрических функций scikit-learn из категории оценщиков обращайтесь по адресу:

https://scikit-learn.org/stable/modules/model_evaluation.html

14.5.9. Выбор лучшей модели

Как и в примере с классификацией, опробуем несколько оценщиков и проверим, не дает ли какой-либо из них лучшие результаты, чем оценщик `LinearRegression`. В этом примере используется уже созданный оценщик `linear_regression`, а также регрессионные оценщики `ElasticNet`, `Lasso` и `Ridge` (все они принадлежат модулю `sklearn.linear_model`). За дополнительной информацией об этих оценщиках обращайтесь по адресу:

https://scikit-learn.org/stable/modules/linear_model.html

```
In [47]: from sklearn.linear_model import ElasticNet, Lasso, Ridge

In [48]: estimators = {
    ...:     'LinearRegression': linear_regression,
    ...:     'ElasticNet': ElasticNet(),
    ...:     'Lasso': Lasso(),
    ...:     'Ridge': Ridge()
    ...: }
```

Оценщики будут выполняться с применением *k*-проходной перекрестной проверки с объектом `KFold` и функцией `cross_val_score`. Здесь `cross_val_score` передается дополнительный ключевой аргумент `scoring='r2'`, который означает, что функция должна выдать коэффициенты R^2 для каждой части, — и снова `1.0` является наилучшим значением. Похоже, `LinearRegression` и `Ridge` оказываются наилучшими моделями для этого набора данных:

```
In [49]: from sklearn.model_selection import KFold, cross_val_score

In [50]: for estimator_name, estimator_object in estimators.items():
    ...:     kfold = KFold(n_splits=10, random_state=11, shuffle=True)
    ...:     scores = cross_val_score(estimator=estimator_object,
    ...:         X=california.data, y=california.target, cv=kfold,
    ...:         scoring='r2')
    ...:     print(f'{estimator_name:>16}: ' +
    ...:           f'mean of r2 scores={scores.mean():.3f}')
    ...:
LinearRegression: mean of r2 scores=0.599
      ElasticNet: mean of r2 scores=0.423
           Lasso: mean of r2 scores=0.285
           Ridge: mean of r2 scores=0.599
```

14.6. Практический пример: машинное обучение без учителя, часть 1 — понижение размерности

В процессе обсуждения data science мы всегда подчеркивали то, насколько важно хорошо знать данные. *Машинное обучение без учителя* и визуализация помогут вам в этом, способствуя выявлению закономерностей и отношений между непомеченными образцами.

Для таких наборов данных, как одномерные временные ряды, использованных ранее в этой главе, визуализация данных выполняется достаточно просто. При использовании двух переменных — дата и температура — данные были выведены в двумерной форме, при этом каждая ось представляла одну переменную. При использовании Matplotlib, Seaborn и других библиотек визуали-

зации можно выводить наборы данных с тремя переменными на трехмерных диаграммах. Но как вывести данные с бóльшим количеством измерений? Например, в наборе данных Digits каждый образец имеет 64 признака и целевое значение. Между тем в больших данных образцы могут иметь сотни, тысячи и даже миллионы признаков.

Чтобы представить набор данных со множеством признаков (то есть множеством измерений), сначала сократим его размерность до двух или трех. Для этого будет использоваться метод машинного обучения без учителя, называемый *понижением размерности*. При выводе полученной информации на диаграмме могут быть выявлены закономерности в данных, которые помогут выбрать наиболее подходящие алгоритмы машинного обучения. Например, если визуализация содержит кластеры точек данных, это может указывать на наличие обособленных классов информации в наборе; в такой ситуации классификационный алгоритм может оказаться уместным. Конечно, сначала необходимо определить класс образцов в каждом кластере. Для этого может потребоваться анализ образцов, входящих в кластер, и выявление сходства между ними.

Понижение размерности также служит другим целям. Обучение оценщиков на больших данных со значительным количеством измерений может занять часы, дни, недели и даже более. Кроме того, человеку трудно представить себе данные с большим количеством измерений — это называется «*проклятием размерности*». Если данные содержат сильно коррелированные признаки, то некоторые из них могут быть исключены посредством понижения размерности для повышения эффективности обучения. Это, однако, может привести к потере точности модели.

Вспомним, что набор данных Digits уже помечен 10 классами, соответствующими цифрам 0–9. Проигнорируем эти метки и воспользуемся понижением размерности, чтобы свести признаки набора данных к двум измерениям для визуализации результатов.

Загрузка набора данных Digits

Запустите IPython следующей командой:

```
ipython --matplotlib
```

и загрузите набор данных:

```
In [1]: from sklearn.datasets import load_digits

In [2]: digits = load_digits()
```

Создание оценщика TSNE для понижения размерности

Теперь воспользуемся *оценщиком* `TSNE` (из модуля `sklearn.manifold`) для выполнения понижения размерности. Этот оценщик использует алгоритм, называемый методом нелинейного понижения размерности и виртуализации многомерных переменных (t-SNE)[1] для анализа признаков набора данных и сведения их к заданному числу измерений. Сначала мы попытались воспользоваться популярным оценщиком PCA (анализ главных компонент, Principal Components Analysis), но полученные результаты нас не устроили, поэтому мы переключились на `TSNE` (оценщик PCA продемонстрирован позднее).

Создадим объект `TSNE` для сведения признаков набора данных к двум измерениям, на что указывает ключевой аргумент `n_components`. Как и в случае с другими оценщиками, упоминавшимися ранее, использован ключевой аргумент `random_state`, обеспечивающий воспроизводимость «последовательности вывода» при выводе кластеров цифр:

```
In [3]: from sklearn.manifold import TSNE

In [4]: tsne = TSNE(n_components=2, random_state=11)
```

Сведение признаков набора данных Digits к двум измерениям

Понижение размерности в scikit-learn обычно состоит из двух шагов — обучения оценщика на наборе данных и последующего использования оценщика для преобразования данных к заданному количеству измерений. Эти шаги могут выполняться по отдельности методами `fit` и `transform` оценщика `TSNE` или же одной командой с использованием метода `fit_transform`:[2]

```
In [5]: reduced_data = tsne.fit_transform(digits.data)
```

Метод `fit_transform` оценщика `TSNE` какое-то время обучает оценщика, а затем выполняет понижение размерности. В нашей системе это заняло около 20 секунд. Когда метод завершает свою операцию, он возвращает массив с таким же количеством строк, что и у `digits.data`, но только двумя столбцами. Чтобы убедиться в этом, достаточно проверить размеры `reduced_data`:

```
In [6]: reduced_data.shape
Out[6]: (1797, 2)
```

---

[1] Подробности устройства алгоритма выходят за рамки этой книги. За дополнительной информацией обращайтесь по адресу https://scikit-learn.org/stable/modules/manifold.html#t-sne.

[2] Каждый вызов `fit_transform` выполняет обучение оценщика. Если вы хотите многократно использовать оценщика для сокращения размерности образцов, вызовите `fit` для однократного обучения оценщика, а затем вызывайте `transform` для понижения размерности Этот прием будет использован с `PCA` позднее в этой главе.

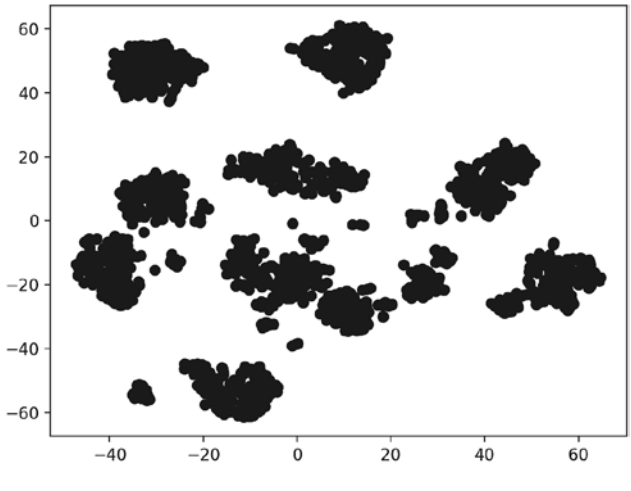
Визуализация данных с пониженной размерностью

Итак, исходный набор данных был сведен до двух измерений, и мы можем построить диаграмму разброса данных. На этот раз вместо функции `scatterplot` библиотеки Seaborn будет использоваться *функция* `scatter` библиотеки Matplotlib, потому что она возвращает коллекцию элементов, нанесенных на диаграмму. Вскоре она будет использована на второй диаграмме разброса:

```
In [7]: import matplotlib.pyplot as plt

In [8]: dots = plt.scatter(reduced_data[:, 0], reduced_data[:, 1],
   ...:                    c='black')
   ...:
```

Первые два аргумента `scatter` содержат столбцы `reduced_data` (0 и 1) с данными по осям *x* и *y*. Ключевой аргумент `c='black'` задает цвет точек. Мы не пометили оси, потому что они не соответствуют конкретным признакам исходного набора данных. Новые признаки, сгенерированные оценщиком `TSNE`, могут заметно отличаться от исходных признаков набора данных.

Следующая диаграмма выводит полученную диаграмму разброса. Очевидно, на диаграмме присутствуют *кластеры* взаимосвязанных точек данных, хотя основных кластеров одиннадцать, а не десять. Также существуют «свободные» точки данных, которые не являются частью конкретных кластеров. На основании предшествующего изучения набора данных Digits эта картина выглядит разумно из-за трудностей с классификацией некоторых цифр.



Визуализация данных после понижения размерности с разными цветами

Хотя на предыдущей диаграмме видны скопления точек, мы не знаем, представляют ли все точки в каждом кластере одну и ту же цифру. Если они представляют разные цифры, то от кластеров особой пользы не будет. Воспользуемся известными целевыми значениями в наборе данных Digits и раскрасим все точки, чтобы было видно, действительно ли кластеры представляют конкретные цифры:

```
In [9]: dots = plt.scatter(reduced_data[:, 0], reduced_data[:, 1],
   ...:     c=digits.target, cmap=plt.cm.get_cmap('nipy_spectral_r', 10))
   ...:
   ...:
```

В этом случае ключевой аргумент `c=digits.target` функции `scatter` указывает, что целевые значения определяют цвета точек. Добавленный ключевой аргумент

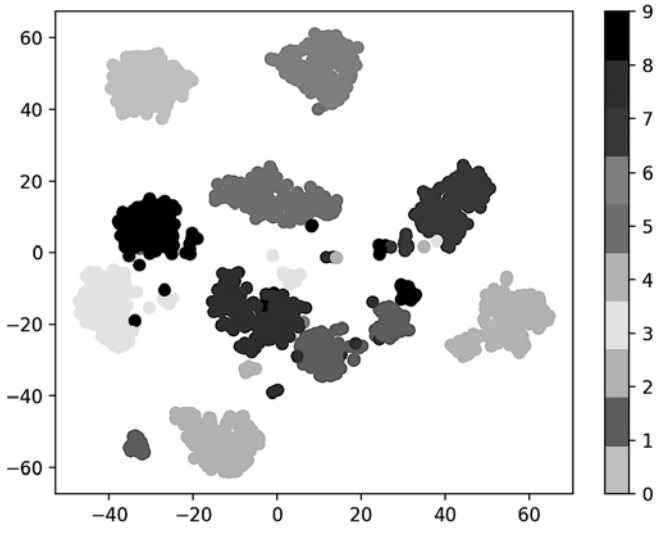
```
cmap=plt.cm.get_cmap('nipy_spectral_r', 10)
```

задает цветовую карту, используемую для выбора цветов точек. В данном случае известно, что окрашиваются 10 цифр, поэтому воспользуемся методом `get_cmap` объекта `cm` библиотеки Matplotlib (из модуля `matplotlib.pyplot`) для загрузки цветовой карты (`'nipy_spectral_r'`) и выбора 10 четко различающихся цветов из цветовой карты.

Следующая команда добавляет справа от диаграммы цветную полоску для расшифровки цветов, чтобы вы видели, какую цифру представляет каждый цвет:

```
In [10]: colorbar = plt.colorbar(dots)
```

На диаграмме хорошо видны 10 кластеров, соответствующих цифрам 0–9. И снова остаются меньшие группы точек, не входящие ни в один кластер. На основании этого можно решить, что метод обучения с учителем, такой как метод k ближайших соседей, хорошо подойдет для этих данных. Для эксперимента также можно воспользоваться классом `Axes3D` библиотеки Matplotlib, предоставляющим оси x, y и z для построения пространственных диаграмм.



14.7. Практический пример: машинное обучение без учителя, часть 2 — кластеризация методом k средних

В этом разделе будет представлен, пожалуй, самый простой из алгоритмов машинного обучения без учителя — *кластеризация методом k средних*. Алгоритм анализирует *непомеченные* образцы и пытается объединить их в кластеры. Поясним, что *k* в «методе *k* средних» представляет количество кластеров, на которые предполагается разбить данные.

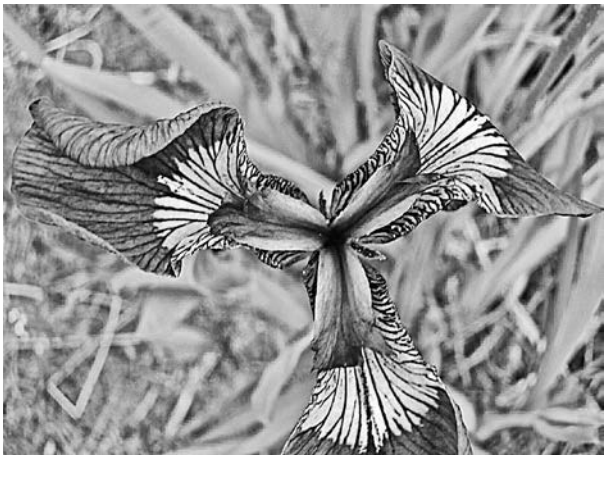
Алгоритм распределяет образцы на заранее заданное количество кластеров, используя метрики расстояния, сходные с метриками алгоритма кластеризации *k* ближайших соседей. Каждый кластер группируется вокруг *центроида* — центральной точки кластера. Изначально алгоритм выбирает *k* случайных центроидов среди образцов набора данных, после чего остальные образцы распределяются по кластерам с ближайшим центроидом. Далее выполняется итеративный пересчет центроидов, а образцы перераспределяются по кластерам, пока для всех кластеров расстояние от заданного центроида до образцов, входящих в его кластер, не будет минимизировано. В результате выполнения алгоритма формируется одномерный массив меток, обозначающих кластер,

к которому относится каждый образец, а также двумерный массив центроидов, представляющих центр каждого кластера.

Набор данных Iris

Поработаем с популярным *набором данных Iris*[1], входящим в поставку scikit-learn. Этот набор часто анализируется при классификации и кластеризации. И хотя набор данных помечен, мы не будем использовать эти метки, чтобы продемонстрировать кластеризацию. Затем метки будут использованы для определения того, насколько хорошо алгоритм *k* средних выполняет кластеризацию образцов.

Набор данных Iris относится к «игрушечным» наборам данных, поскольку состоит только из 150 образцов и четырех признаков. Набор данных описывает 50 образцов трех видов цветов ириса — *Iris setosa*, *Iris versicolor* и *Iris virginica* (см. фотографии ниже). Признаки образцов: длина наружной доли околоцветника (sepal length), ширина наружной доли околоцветника (sepal width), длина внутренней доли околоцветника (petal length) и ширина внутренней доли околоцветника (petal width), измеряемые в сантиметрах.



Iris setosa. Credit: Courtesy of Nation Park services

[1] Fisher, R.A., «The use of multiple measurements in taxonomic problems», Annual Eugenics, 7, Part II, 179–188 (1936); также «Contributions to Mathematical Statistics» (John Wiley, NY, 1950).

Iris versicolor. Credit: Courtesy of Jefficus

Iris virginica. Credit: Christer T Johansson

14.7.1. Загрузка набора данных Iris

Запустите IPython командой `ipython --matplotlib`, после чего воспользуйтесь *функцией* `load_iris` модуля `sklearn.datasets` для получения объекта `Bunch` с набором данных:

```
In [1]: from sklearn.datasets import load_iris

In [2]: iris = load_iris()
```

Атрибут `DESCR` объекта `Bunch` показывает, что набор данных состоит из 150 образцов (`Number of Instances`), каждый из которых обладает четырьмя признаками (`Number of Attributes`). В наборе данных нет отсутствующих значений. Образцы классифицируются целыми числами 0, 1 и 2, представляющими *Iris setosa*, *Iris versicolor* и *Iris virginica* соответственно. *Проигнорируем* метки и поручим определение классов образцов алгоритму кластеризации методом *k* средних. Ключевая информация `DESCR` выделена жирным шрифтом:

```
In [3]: print(iris.DESCR)
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
...
```

Проверка количества образцов, признаков и целевых значений

Количество образцов и признаков можно узнать из атрибута `shape` массива `data`, а количество целевых значений — из атрибута `shape` массива `target`:

```
In [4]: iris.data.shape
Out[4]: (150, 4)

In [5]: iris.target.shape
Out[5]: (150,)
```

Массив `target_names` содержит имена числовых меток массива. Выражение `target — dtype='<U10'` означает, что его элементами являются строки длиной не более 10 символов:

```
In [6]: iris.target_names
Out[6]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Массив `feature_names` содержит список строковых имен для каждого столбца в массиве `data`:

```
In [7]: iris.feature_names
Out[7]:
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

14.7.2. Исследование набора данных Iris: описательная статистика в Pandas

Используем коллекцию `DataFrame` для исследования набора данных Iris. Как и в случае с набором данных California Housing, зададим параметры pandas для форматирования столбцового вывода:

```
In [8]: import pandas as pd

In [9]: pd.set_option('max_columns', 5)

In [10]: pd.set_option('display.width', None)
```

Создадим коллекцию `DataFrame` с содержимым массива `data`, используя содержимое массива `feature_names` как имена столбцов:

```
In [11]: iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

Затем добавим столбец с названием вида для каждого из образцов. Трансформация списка в следующем фрагменте использует каждое значение в массиве `target` для поиска соответствующего названия в массиве `target_names`:

```
In [12]: iris_df['species'] = [iris.target_names[i] for i in iris.target]
```

Воспользуемся pandas для идентификации нескольких образцов. Как и прежде, если pandas выводит \ справа от имени столбца, это означает, что в выводе остаются столбцы, которые будут выведены ниже:

```
In [13]: iris_df.head()
Out[13]:
   sepal length (cm)  sepal width (cm)  petal length (cm)  \
0                5.1               3.5                1.4
1                4.9               3.0                1.4
2                4.7               3.2                1.3
3                4.6               3.1                1.5
4                5.0               3.6                1.4
   petal width (cm) species
0               0.2  setosa
1               0.2  setosa
2               0.2  setosa
3               0.2  setosa
4               0.2  setosa
```

Вычислим некоторые показатели описательной статистики для числовых столбцов:

```
In [14]: pd.set_option('precision', 2)

In [15]: iris_df.describe()
Out[15]:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count             150.00            150.00             150.00
mean                5.84              3.06               3.76
std                 0.83              0.44               1.77
min                 4.30              2.00               1.00
25%                 5.10              2.80               1.60
50%                 5.80              3.00               4.35
75%                 6.40              3.30               5.10
max                 7.90              4.40               6.90
```

```
       petal width (cm)
count            150.00
mean               1.20
std                0.76
min                0.10
25%                0.30
50%                1.30
75%                1.80
max                2.50
```

Вызов метода `describe` для столбца `'species'` подтверждает, что он содержит три уникальных значения. Нам заранее известно, что данные состоят из трех классов, к которым относятся образцы, хотя в машинном обучении без учителя это и не всегда так.

```
In [16]: iris_df['species'].describe()
Out[16]:
count        150
unique         3
top       setosa
freq          50
Name: species, dtype: object
```

14.7.3. Визуализация набора данных функцией pairplot

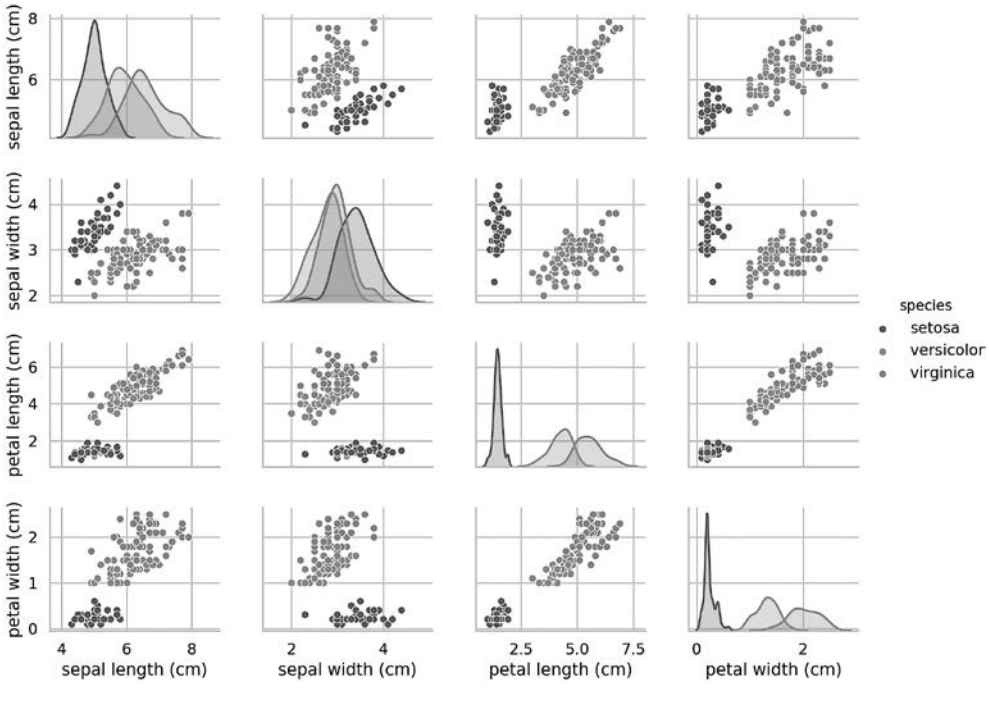
Проведем визуализацию признаков в этом наборе данных. Один из способов извлечь информацию о ваших данных — посмотреть, как признаки связаны друг с другом. Набор данных имеет четыре признака. Мы не сможем построить диаграмму соответствия одного признака с тремя другими на одной диаграмме. Тем не менее можно построить диаграмму, на которой будет представлено соответствие между двумя признаками. Фрагмент [20] использует функцию `pairplot` библиотеки Seaborn для создания таблицы диаграмм, на которых каждый признак сопоставляется с одним из других признаков:

```
In [17]: import seaborn as sns

In [18]: sns.set(font_scale=1.1)

In [19]: sns.set_style('whitegrid')

In [20]: grid = sns.pairplot(data=iris_df, vars=iris_df.columns[0:4],
    ...:     hue='species')
    ...:
```

Ключевые аргументы:

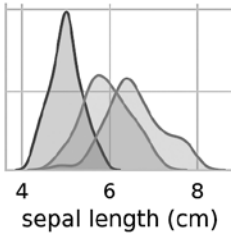
- `data` — коллекция `DataFrame`[1] с набором данных, наносимым на диаграмму;
- `vars` — последовательность с именами переменных, наносимых на диаграмму. Для коллекции `DataFrame` она содержит имена столбцов. В данном случае используются первые четыре столбца `DataFrame`, представляющие длину (ширину) наружной доли околоцветника и длину (ширину) внутренней доли околоцветника соответственно;
- `hue` — столбец коллекции `DataFrame`, используемый для определения цветов данных, наносимых на диаграмму. В данном случае данные окрашиваются в зависимости от вида ирисов.

Предыдущий вызов `pairplot` строит следующую таблицу диаграмм 4 × 4:



[1] Также может использоваться двумерный массив или список.

Диаграммы на диагонали, ведущей из левого верхнего в правый нижний угол, показывают *распределение* признака, выведенного в этом столбце, с диапазоном значений (слева направо) и количеством образцов с этими значениями (сверху вниз). Возьмем распределение длины наружной доли околоцветника:



Самая высокая закрашенная область указывает, что диапазон значений длины наружной доли околоцветника (по оси x) для вида *Iris setosa* составляет приблизительно 4–6 см, а у большинства образцов *Iris setosa* значения лежат в середине этого диапазона (приблизительно 5 см). Крайняя правая закрашенная область указывает, что диапазон значений длины наружной доли околоцветника (по оси x) для вида *Iris virginica* составляет приблизительно 4–8,5 см, а у большинства образцов *Iris virginica* значения лежат между 6 и 7 см.

На других диаграммах в столбце представлены диаграммы разброса данных других признаков относительно признака по оси x. В первом столбце на первых трех диаграммах по оси y представлены ширина наружной доли околоцветника, длина внутренней доли околоцветника и ширина внутренней доли околоцветника соответственно, а на оси x — длина наружной доли околоцветника.

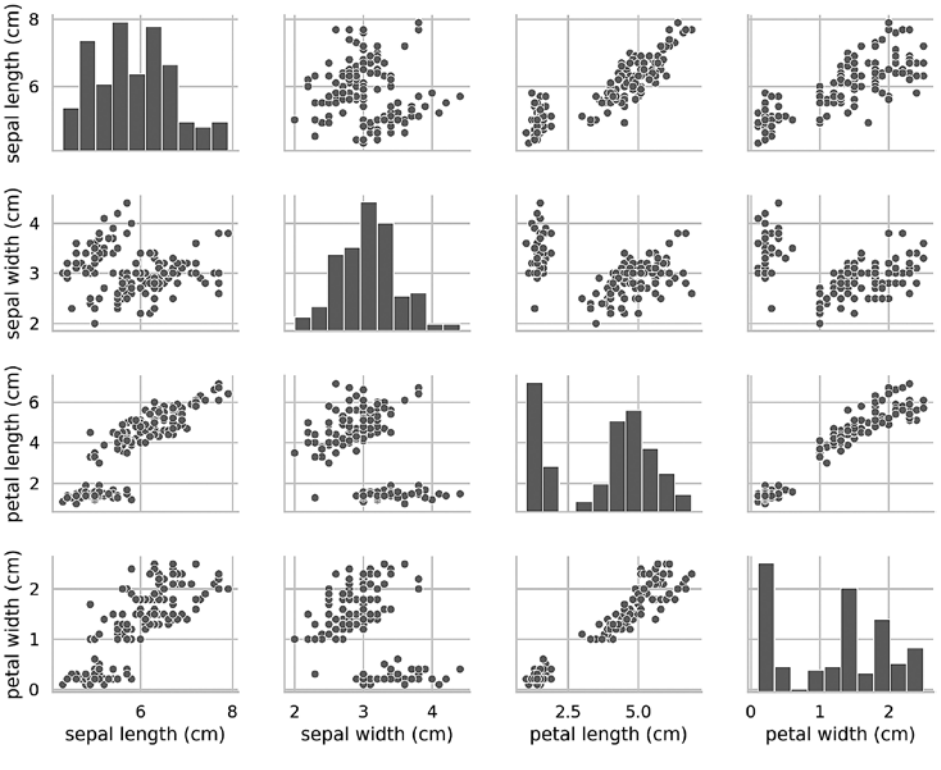
При выполнении этого кода на экране появляется цветное изображение, показывающее отношения между разными видами ирисов на уровне отдельных признаков. Интересно, что на всех диаграммах синие точки *Iris setosa* четко отделяются от оранжевых и зеленых точек других видов; это говорит о том, что *Iris setosa* действительно является отдельным классом. Также можно заметить, что других два вида иногда можно перепутать, на что указывают перекрывающиеся оранжевые и зеленые точки. Например, по диаграмме ширины и длины наружной доли околоцветника видно, что точки *Iris versicolor* и *Iris virginica* смешиваются. Это говорит о том, что если доступны только измерения наружной доли околоцветника, то различить эти два вида будет сложно.

Вывод результатов pairplot в одном цвете

Если убрать ключевой аргумент `hue`, то функция `pairplot` использует только один цвет для вывода всех данных, потому что она не знает, как различать виды при выводе:

```
In [21]: grid = sns.pairplot(data=iris_df, vars=iris_df.columns[0:4])
```

Как видно из следующей диаграммы, в данном случае диаграммы на диагонали представляют собой гистограммы с распределениями всех значений этого признака независимо от вида. При изучении диаграмм может показаться, что существуют всего *два* кластера, хотя мы знаем, что в наборе содержатся данные *трех* видов. Если количество кластеров неизвестно заранее, то можно обратиться к *эксперту предметной области*, хорошо знакомому с данными. Эксперт может знать, что в наборе данных присутствуют три вида; эта информация может пригодиться при проведении машинного обучения с данными.



Диаграммы `pairplot` хорошо работают при *малом количестве признаков* или подмножестве признаков, чтобы количество строк и столбцов было ограниченным, и при относительно небольшом количестве образцов, чтобы были видны точки данных. С ростом количества признаков и образцов диаграммы разброса данных становятся слишком мелкими для чтения данных. В больших наборах данных можно нанести на диаграмму подмножество признаков и, возможно, случайно выбранное подмножество образцов, чтобы получить некоторое представление о данных.

14.7.4. Использование оценщика KMeans

В этом разделе будет применена кластеризация методом *k* средних с *оценщиком* `KMeans` библиотеки scikit-learn (из модуля `sklearn.cluster`) для распределения образцов набора данных Iris по кластерам. Как и с другими оценщиками, которыми мы пользовались ранее, оценщик `KMeans` скрывает от вас сложные математические подробности реализации алгоритма, чтобы им было удобно пользоваться.

Создание оценщика

Создадим объект `KMeans`:

```
In [22]: from sklearn.cluster import KMeans

In [23]: kmeans = KMeans(n_clusters=3, random_state=11)
```

Ключевой аргумент `n_clusters` определяет гиперпараметр *k* алгоритма кластеризации методом *k* средних, необходимый `KMeans` для вычисления кластеров и пометки образцов. В процессе обучения оценщика `KMeans` алгоритм вычисляет для каждого кластера центроид, представляющий центральную точку данных этого кластера.

По умолчанию значение параметра `n_clusters` равно 8. Нередко для выбора подходящего значения *k* приходится полагаться на знания экспертов предметной области. Тем не менее настройка гиперпараметра позволяет получить оценку оптимального значения *k*, как это будет сделано позднее. В данном случае мы знаем, что набор содержит данные трех видов, поэтому для проверки эффективности `KMeans` при пометке образцов ирисов используется значение `n_clusters=3`. И снова ключевое слово `random_state` используется для обеспечения воспроизводимости результатов.

Подгонка модели

Затем происходит обучение оценщика, для чего вызывается метод `fit` объекта `KMeans` и выполняется алгоритм *k* средних, рассмотренный ранее:

```
In [24]: kmeans.fit(iris.data)
Out[24]:
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=11, tol=0.0001, verbose=0)
```

Как и у других оценщиков, метод `fit` возвращает объект оценщика, а IPython выводит его строковое представление. Список аргументов по умолчанию для `KMeans` можно просмотреть по адресу:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Когда обучение завершено, объект `KMeans` содержит:

- Массив `labels_` со значениями от 0 до `n_clusters - 1` (в данном случае 0–2), обозначающими кластеры, к которым принадлежат образцы.
- Массив `cluster_centers_`, каждая строка которого представляет центроид.

Сравнение меток кластеров с целевыми значениями из набора данных

Так как набор Iris размечен, мы можем проверить значения из его массива `target`, чтобы составить представление о том, насколько эффективно алгоритм *k* средних выполняет кластеризацию для трех видов ирисов. С непомеченными данными вам придется обратиться к эксперту предметной области, который поможет оценить разумность прогнозируемых классов.

В этом наборе данных первые 50 образцов относятся к виду *Iris setosa*, следующие 50 — к виду *Iris versicolor*, а последние 50 — к виду *Iris virginica*. Массив `target` набора данных Iris представляет образцы со значениями 0–2. Если оценщик `KMeans` выбрал кластеры идеально, то все группы из 50 элементов массива `labels_` оценщика будут обладать разными метками. При изучении приведенных ниже результатов следует заметить, что оценщик `KMeans` использует значения от 0 до $k - 1$ для пометки кластеров, но эти значения не связаны со значениями из массива `target` набора данных Iris.

Сегментация поможет оценить результаты кластеризации для каждой группы из 50 образцов ирисов. Следующий фрагмент показывает, что первые 50 образцов были объединены в кластер 1:

```
In [25]: print(kmeans.labels_[0:50])
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Следующие 50 образцов должны быть помещены во второй кластер. Фрагмент показывает, что большинство из них попало в кластер 0, но два образца были отнесены к кластеру 2:

```
In [26]: print(kmeans.labels_[50:100])
[0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Аналогичным образом последние 50 образцов должны быть помещены в третий кластер. Следующий фрагмент показывает, что многие из этих образцов попали в кластер 2, но 14 из них были помещены в кластер 0; алгоритм решил, что они принадлежат к другому кластеру:

```
In [27]: print(kmeans.labels_[100:150])
[2 0 2 2 2 2 0 2 2 2 2 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2
 2 2 0 2 2 2 0 2 2 2 0 2 2 0]
```

Результаты этих трех фрагментов подтверждают то, что мы уже видели на диаграммах `pairplot` ранее в этом разделе, — *Iris setosa* выделяется в обособленный класс, а *Iris versicolor* можно спутать с *Iris virginica*.

14.7.5. Понижение размерности методом анализа главных компонент

Затем воспользуемся *оценщиком* `PCA` (из модуля `sklearn.decomposition`) для выполнения понижения размерности. Этот оценщик использует алгоритм, называемый анализом главных компонент[1], для анализа признаков набора данных и их сведения к заданному количеству измерений. Для набора данных

[1] Подробности алгоритма выходят за рамки книги. За дополнительной информацией обращайтесь по адресу: https://scikit-learn.org/stable/modules/decomposition.html#pca.

Iris мы сначала применили оценщик TSNE, упоминавшийся ранее, но полученные результаты нас не устроили. По этой причине мы перешли на PCA для следующей демонстрации.

Создание объекта PCA

Как и оценщик TSNE, оценщик PCA использует ключевой аргумент n_components для определения количества измерений:

```
In [28]: from sklearn.decomposition import PCA

In [29]: pca = PCA(n_components=2, random_state=11)
```

Преобразование признаков набора данных Iris к двум измерениям

Обучим оценщика и получим данные сокращенной размерности вызовом методов fit и transform оценщика PCA:

```
In [30]: pca.fit(iris.data)
Out[30]:
PCA(copy=True, iterated_power='auto', n_components=2, random_state=11,
  svd_solver='auto', tol=0.0, whiten=False)

In [31]: iris_pca = pca.transform(iris.data)
```

Когда метод завершит свою задачу, он возвращает массив с таким же количеством строк, как у iris.data, но только с двумя столбцами. Чтобы убедиться в этом, проверим атрибут shape объекта iris_pca:

```
In [32]: iris_pca.shape
Out[32]: (150, 2)
```

Обратите внимание: на этот раз мы вызываем методы fit и transform оценщика PCA вместо метода fit_transform, как при использовании оценщика TSNE. В этом примере мы *повторно используем* обученного оценщика (полученного вызовом fit) для выполнения второго преобразования с целью сокращения центроидов кластеров от четырех измерений до двух. Это позволит нам нанести на диаграмму местоположения центроидов для каждого кластера.

Визуализация данных с пониженной размерностью

Теперь, когда исходный набор данных был сокращен до двух измерений, воспользуемся диаграммой разброса данных для вывода. Для этого применим функцию `scatterplot` библиотеки Seaborn. Начнем с преобразования данных с пониженной размерностью в `DataFrame` и добавления столбца `species`, который будет использоваться для определения цветов точек:

```
In [33]: iris_pca_df = pd.DataFrame(iris_pca,
    ...:                            columns=['Component1', 'Component2'])
    ...:

In [34]: iris_pca_df['species'] = iris_df.species
```

Данные выводятся на диаграмме вызовом scatterplot:

```
In [35]: axes = sns.scatterplot(data=iris_pca_df, x='Component1',
    ...:     y='Component2', hue='species', legend='brief',
    ...:     palette='cool')
    ...:
```

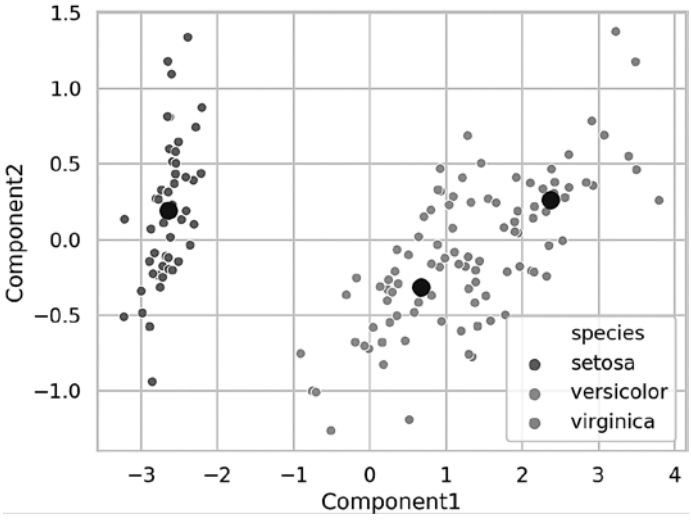
Каждый центроид из массива `cluster_centers_` объекта `KMeans` имеет *такое же* количество признаков, как и исходный набор данных (4 — в данном случае). Чтобы нанести на диаграмму центроиды, необходимо понизить их размерность. Центроид можно рассматривать как «усредненный» образец в кластере. Таким образом, каждый центроид должен быть преобразован тем же оценщиком `PCA`, который использовался для понижения размерности других образцов в этом кластере:

```
In [36]: iris_centers = pca.transform(kmeans.cluster_centers_)
```

Теперь нанесем центроиды трех кластеров в виде черных точек большего размера. Вместо того чтобы преобразовывать массив `iris_centers` в `DataFrame`, используем функцию `scatter` библиотеки Matplotlib для нанесения на диаграмму трех центроидов:

```
In [37]: import matplotlib.pyplot as plt

In [38]: dots = plt.scatter(iris_centers[:,0], iris_centers[:,1],
    ...:                    s=100, c='k')
    ...:
```

Ключевой аргумент `s=100` задает размер точек на диаграмме, а ключевой аргумент `c='k'` указывает, что точки должны выводиться черным цветом.



14.7.6. Выбор оптимального оценщика для кластеризации

Как и в примерах классификации и регрессии, выполним несколько алгоритмов кластеризации и посмотрим, как они справляются с кластеризацией цветов ириса. Попытаемся провести кластеризацию образцов набора данных Iris с объектом `kmeans`, который был создан ранее[1], а также с объектами scikit-learn `DBSCAN`, `MeanShift`, `SpectralClustering` и `AgglomerativeClustering`. Как и в случае с `KMeans`, количество кластеров для оценщиков `SpectralClustering` и `AgglomerativeClustering` задается заранее:

```
In [39]: from sklearn.cluster import DBSCAN, MeanShift,\
    ...:     SpectralClustering, AgglomerativeClustering

In [40]: estimators = {
    ...:     'KMeans': kmeans,
    ...:     'DBSCAN': DBSCAN(),
    ...:     'MeanShift': MeanShift(),
    ...:     'SpectralClustering': SpectralClustering(n_clusters=3),
```

[1] Здесь `KMeans` применяется к *небольшому* набору данных Iris. Если вы столкнетесь с проблемами быстродействия, используя `KMeans` с большими наборами данных, рассмотрите возможность использования `MiniBatchKMeans`. В документации scikit-learn сказано, что `MiniBatchKMeans` быстрее работает с большими наборами данных практически с тем же качеством результатов.

```
    ...:     'AgglomerativeClustering':
    ...:         AgglomerativeClustering(n_clusters=3)
    ...: }
```

Каждая итерация следующего цикла вызывает метод `fit` одного оценщика с аргументом `iris.data`, после чего использует функцию `unique` библиотеки NumPy для получения меток и численности кластеров для трех групп по 50 образцов, после чего выводит результаты. Вспомните, что для оценщиков `DBSCAN` и `MeanShift` количество кластеров *не* указывалось заранее. Интересно, что `DBSCAN` правильно прогнозирует наличие трех кластеров (с метками –1, 0 и 1), хотя и помещает 84 из 100 образцов *Iris virginica* и *Iris versicolor* в один кластер. С другой стороны, оценщик `MeanShift` прогнозирует всего два кластера (с метками 0 и 1) и помещает 99 из 100 образцов *Iris virginica* и *Iris versicolor* в один кластер:

```
In [41]: import numpy as np

In [42]: for name, estimator in estimators.items():
    ...:     estimator.fit(iris.data)
    ...:     print(f'\n{name}:')
    ...:     for i in range(0, 101, 50):
    ...:         labels, counts = np.unique(
    ...:             estimator.labels_[i:i+50], return_counts=True)
    ...:         print(f'{i}-{i+50}:')
    ...:         for label, count in zip(labels, counts):
    ...:             print(f'   label={label}, count={count}')
    ...:

KMeans:
0-50:
   label=1, count=50
50-100:
   label=0, count=48
   label=2, count=2
100-150:
   label=0, count=14
   label=2, count=36

DBSCAN:
0-50:
   label=-1, count=1
   label=0, count=49
50-100:
   label=-1, count=6
```

```
   label=1, count=44
100-150:
   label=-1, count=10
   label=1, count=40

MeanShift:
0-50:
   label=1, count=50
50-100:
   label=0, count=49
   label=1, count=1
100-150:
   label=0, count=50

SpectralClustering:
0-50:
   label=2, count=50
50-100:
   label=1, count=50
100-150:
   label=0, count=35
   label=1, count=15

AgglomerativeClustering:
0-50:
   label=1, count=50
50-100:
   label=0, count=49
   label=2, count=1
100-150:
   label=0, count=15
   label=2, count=35
```

Хотя эти алгоритмы помечают каждый образец, метки всего лишь отражают принадлежность к определенному кластеру. Что делать с информацией о кластерах, когда она у вас появится? Если вы стремитесь к тому, чтобы использовать данные в машинном обучении с учителем, то обычно анализируете образцы в каждом кластере, пытаетесь определить, как они связаны друг с другом, и помечаете их соответствующим образом. Как будет показано в следующей главе, обучение без учителя часто применяется в приложениях глубокого обучения. Примерами непомеченных данных, обрабатываемых средствами обучения без учителя, могут послужить твиты из Twitter, сообщения в Facebook, видеоролики, фотографии, новостные статьи, обзоры на продукты от потребителей, отзывы о фильмах и т. д.

14.8. Итоги

В этой главе началось ваше знакомство с машинным обучением с использованием популярной библиотеки scikit-learn. Вы узнали, что машинное обучение делится на два типа: машинное обучение с учителем, работающее с помеченными данными, и машинное обучение без учителя, работающее с непомеченными данными. В этой главе снова широко применялась визуализация средствами Matplotlib и Seaborn, особенно для целей изучения данных. Вы узнали, как scikit-learn упаковывает алгоритмы машинного обучения в объекты-оценщики. Каждый оценщик инкапсулирован, что позволяет быстро создавать модели с минимальным объемом кода, даже если вам неизвестны все тонкости работы этих алгоритмов.

Сначала было рассмотрено машинное обучение с учителем для классификации, затем регрессия. Мы воспользовались одним из простейших алгоритмов классификации, методом *k* ближайших соседей, для анализа набора данных Digits, входящего в поставку scikit-learn. Было показано, что алгоритмы классификации прогнозируют классы, к которым относятся образцы. При бинарной классификации используются два класса (например, «спам» или «не спам»), а при множественной — более чем два класса (например, 10 классов в наборе данных Digits).

Были рассмотрены основные этапы типичного исследования в области машинного обучения, включая загрузку набора данных, изучение данных средствами pandas и визуализацию, разбиение данных для обучения и тестирования, создание модели, обучение модели и построение прогнозов. Вы узнали, почему данные разбиваются на обучающий и тестовый наборы и то, как оценить точность классификационного оценщика при помощи матрицы несоответствий и классификационного отчета.

Мы упоминали о том, что трудно заранее определить, какая модель(-и) покажет наилучшие результаты с вашими данными, поэтому обычно исследователь опробует много моделей и выбирает ту, которая показала лучшие результаты. Мы показали, что исследователь может легко выполнить несколько оценщиков. Также настройка гиперпараметра с *k*-проходной перекрестной проверкой использовалась для выбора лучшего значения *k* в алгоритме k-NN.

Затем мы вернулись к временным рядам и примеру линейной регрессии из раздела «Введение в data science» главы 10; на этот раз он реализован с использованием оценщика `LinearRegression` из scikit-learn. Оценщик `LinearRegression` использован для выполнения множественной линейной регрессии с набором

данных California Housing, входящим в поставку scikit-learn. Вы узнали, что оценщик `LinearRegression` по умолчанию использует все числовые признаки в наборе данных для построения более сложных прогнозов, чем возможно при простой линейной регрессии. И снова мы выполнили несколько оценщиков scikit-learn, чтобы сравнить их эффективность и выбрать лучший вариант.

Затем мы представили машинное обучение без учителя, которое обычно реализуется с применением алгоритмов кластеризации. Был описан метод понижения размерности (с оценщиком `TSNE` библиотеки scikit-learn), который был применен для сокращения 64 признаков набора данных Digits до двух при создании визуализации. Это позволило наглядно увидеть кластеризацию данных цифр.

Далее мы описали один из простейших алгоритмов машинного обучения без учителя, алгоритм кластеризации методом *k* средних. Кластеризация продемонстрирована на примере набора данных Iris, также входящего в поставку scikit-learn. Посредством понижения размерности (с оценщиком `PCA` библиотеки scikit-learnestimator) четыре признака набора данных Iris сокращены до двух, чтобы выполнить визуализацию и продемонстрировать кластеризацию трех видов ирисов из набора данных, а также положения центроидов. Наконец, мы выполнили несколько оценщиков кластеризации, сравнив их эффективность по разметке образцов набора данных Iris на три кластера.

В следующей главе наше изучение технологий машинного обучения продолжится в области глубинного обучения. Мы рассмотрим некоторые интересные и непростые задачи.

15

Глубокое обучение

В этой главе:

- Концепция нейронных сетей и ее применение при глубоком обучении.
- Создание нейронных сетей Keras.
- Уровни Keras, функции активации, функции потерь и оптимизаторы.
- Применение сверточной нейронной сети Keras (CNN), обученной на наборе данных MNIST, для распознавания рукописных цифр.
- Применение рекуррентной нейронной сети Keras (RNN), обученной на наборе данных IMDb, для выполнения бинарной классификации положительных и отрицательных отзывов о фильмах.
- Применение TensorBoard для визуализации хода глубокого обучения сетей.
- Предварительно обученные нейронные сети, поставляемые с Keras.
- Полезность моделей, обученных на наборе данных ImageNet, в приложениях распознавания образов.

15.1. Введение

Одной из самых интересных областей искусственного интеллекта (AI) является *глубокое обучение* — перспективное направление машинного обучения, которое позволило добиться впечатляющих результатов в распознавании образов и многих других областях за последние несколько лет. Благодаря доступности больших данных, значительным вычислительным мощностям, повышению скорости интернета и достижениям в области программного и аппаратного оборудования параллельных вычислений, появляется все больше организаций и физических лиц, которые берутся за ресурсоемкие решения в области глубокого обучения.

Keras и TensorFlow

В предыдущей главе библиотека scikit-learn позволяла вам легко определять модели машинного обучения всего в одной команде. Модели глубокого обучения требуют более сложной подготовки, в которой чаще всего задействовано несколько разных объектов, называемых *уровнями* (layers). Мы будем строить модели глубокого обучения на базе библиотеки *Keras*, предоставляющей удобный интерфейс к библиотеке Google *TensorFlow* — самой популярной библиотеке глубокого обучения[1]. Франсуа Шолле (François Chollet) из команды Google Mind разработал Keras, чтобы сделать средства глубокого обучения более доступными. Его книга «*Deep Learning with Python*» обязательна к прочтению[2]. В настоящее время в компании Google ведутся тысячи проектов TensorFlow и Keras, и это количество стремительно растет[3,4].

Модели

Модели глубокого обучения сложны, а для понимания их внутренних механизмов необходима серьезная математическая подготовка. Как и в предыдущих главах, мы постараемся избежать сложной теории, ограничившись объяснени-

---

[1] Keras также предоставляет удобный интерфейс к инструментарию Microsoft CNTK и библиотеке *Theano*, разработанной в Монреальском университете (разработка которой была прекращена в 2017 году). Среди других популярных фреймворков глубокого обучения также стоит упомянуть Caffe (http://caffe.berkeleyvision.org/), Apache MXNet (https://mxnet.apache.org/) и PyTorch (https://pytorch.org/).

[2] *Шолле Франсуа*. Глубокое обучение на Python. — СПб.: Питер, 2020. — 400 с.: ил.

[3] http://theweek.com/speedreads/654463/google-more-than-1000-artificial-intelligence-projects-works.

[4] https://www.zdnet.com/article/google-says-exponential-growth-of-ai-is-changing-nature-of-compute/.

ями. Для глубокого обучения Keras играет ту же роль, что и scikit-learn — для машинного обучения. Каждая библиотека инкапсулирует сложные вычисления, и разработчикам остается только определять и параметризовать объекты, а также выполнять с ними нужные операции. Благодаря Keras вы строите свои модели из готовых компонентов и быстро параметризуете эти компоненты в соответствии со своими уникальными требованиями. В этой книге мы называли такую методологию *объектно-базированным программированием*.

Эксперименты с вашими моделями

Машинное обучение и глубокое обучение — области скорее эмпирические, нежели теоретические. Вы экспериментируете со многими моделями и настраиваете их, пока не найдете модели, лучше всего подходящие для ваших приложений. Keras упрощает подобные эксперименты.

Размеры наборов данных

Глубокое обучение хорошо работает при больших объемах данных, но может быть эффективным при меньших объемах данных в сочетании с такими методами, как перенос обучения[1,2] и приращение данных.[3,4] Перенос обучения использует существующие знания, полученные от ранее обученной модели, как фундамент для создания новой модели. Приращение данных добавляет в набор новые данные, выведенные на основании существующих данных. Например, в наборе данных графических изображений изображения можно поворачивать, чтобы модель могла узнать об объектах в разных ориентациях. Тем не менее в общем случае чем больше у вас данных, тем лучше вы сможете обучить модель глубокого обучения.

Вычислительные мощности

Глубокое обучение может требовать значительных вычислительных мощностей. Обучение сложных моделей на наборах больших данных может занять часы, дни и более. Модели, представленные в этой главе, обучаются от

---

[1] https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751.

[2] https://medium.com/nanonets/nanonets-how-to-use-deep-learning-when-you-have-limited-data-f68c0b512cab.

[3] https://towardsdatascience.com/data-augmentation-and-images-7aca9bd0dbe8.

[4] https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced.

нескольких минут до часа на компьютерах с обычным процессором. Все, что потребуется, — это сколько-нибудь современный персональный компьютер. В этой главе мы обсудим специальное высокопроизводительное оборудование: графические процессоры (GPU) и тензорные процессоры (TPU), разработанные NVIDIA и Google для удовлетворения высоких потребностей современных приложений глубокого обучения.

Наборы данных

В поставку Keras включены некоторые популярные наборы данных. Два таких набора задействованы в примерах этой главы. Для каждого из этих наборов в интернете можно найти много других исследований Keras, в том числе и проекты с другими подходами. Так, в главе 14 использовался набор данных Digits библиотеки scikit-learn из 1797 изображений рукописных цифр, выбранных из гораздо более обширного набора данных MNIST (60 000 обучающих и 10 000 тестовых изображений[1]). В этой главе мы будем работать с полным набором данных MNIST и построим *сверточную нейронную сеть* (CNN) Keras, которая достигнет высокой эффективности в распознавании изображений цифр из тестового набора. Сверточные нейронные сети особенно хорошо подходят для задач распознавания образов, например распознавания рукописных цифр и букв или распознавания объектов (включая лица) в изображениях и видеороликах. Поработаем мы и с *рекуррентной нейронной сетью* Keras. В этом примере будет рассматриваться анализ эмоциональной окраски в наборе данных с обзорами фильмов на IMDb, где отзывы в обучающем и тестовом наборе помечаются как положительные или отрицательные.

Будущее глубокого обучения

Более новые автоматизированные средства глубокого обучения еще больше упрощают построение решений. К их числу относятся: Auto-Keras[2] от лаборатории DATA Техасского университета A&M, Baidu EZDL[3] и Google AutoML[4].

---

[1] «The MNIST Database». MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges. http://yann.lecun.com/exdb/mnist/.

[2] https://autokeras.com/.

[3] https://ai.baidu.com/ezdl/.

[4] https://cloud.google.com/automl/.

15.1.1. Практическое применение глубокого обучения

Глубокое обучение используется во многих практических областях, в числе которых:

- Компьютерные игры.
- Распознавание образов: распознавание объектов, выявление закономерностей, распознавание лиц.
- Автономные автомобили.
- Робототехника.
- Повышение качества обслуживания.
- Чат-боты.
- Медицинская диагностика.
- Поиск в Google.
- Автоматизированное генерирование подписей к изображениям и субтитров.
- Улучшение разрешения графических изображений.
- Распознавание речи.
- Перевод на другие языки.
- Прогнозирование результатов выборов.
- Прогнозирование землетрясений и погодных условий.
- Сервис Google Sunroof для проверки возможности размещения солнечных батарей на крыше.
- Генерирующие приложения — генерирование новых изображений, обработка существующих изображений в стиле конкретного художника, раскрашивание черно-белых изображений и видео, создание музыки, создание текста (книги, поэзия) и многое другое.

15.1.2. Демонстрационные приложения глубокого обучения

Опробуйте следующие четыре демонстрационных приложения и поищите в интернете информацию о многих других, включая практическое применение в областях, перечисленных в предыдущем разделе:

- DeepArt.io — преобразует фотографию в картину применением художественного стиля к фотографии: https://deepart.io/.
- DeepWarp Demo — анализирует фотографию человека и заставляет его глаза перемещаться в разных направлениях: https://sites.skoltech.ru/sites/compvision_wiki/static_pages/projects/deepwarp/.
- Image-to-Image Demo — преобразует штриховой рисунок в картину: https://affinelayer.com/pixsrv/.
- Google Translate Mobile App (загрузите из магазина приложений на смартфон) — переводит текст на фотографии на другой язык (например, можно сделать фотографию знака или ресторанного меню на испанском языке и перевести текст на английский).

15.1.3. Ресурсы Keras

Вот некоторые источники информации, которые могут вам пригодиться в ходе знакомства с глубоким обучением:

- Чтобы получить ответы на вопросы, посетите slack-канал команды Keras по адресу https://kerasteam.slack.com.
- За статьями и учебниками обращайтесь по адресу https://blog.keras.io.
- Документация Keras доступна по адресу http://keras.io.
- Если вы ищете темы для курсовой или дипломной работы или даже диссертации, то посетите сайт arXiv по адресу https://arXiv.org. Здесь люди публикуют свои исследовательские работы и получают быструю обратную связь от коллег. На этом сайте вы получите доступ к материалам новейших исследований.

15.2. Встроенные наборы данных Keras

Ниже перечислены некоторые наборы данных Keras (из модуля `tensorflow.keras.datasets`[1]) для экспериментов с глубоким обучением. Мы используем пару этих наборов в примерах главы:

[1] В автономной библиотеке Keras имена модулей начинаются с префикса keras вместо tensorflow.keras.

База данных рукописных цифр MNIST[1] — используется для классификации рукописных изображений цифр. Набор данных содержит изображения цифр размера 28 × 28 в оттенках серого, снабженные метками от 0 до 9; 60 000 изображений используются для обучения, а 10 000 — для тестирования. Этот набор будет использован в разделе 15.6, когда мы займемся изучением сверточных нейронных сетей.

База данных модных товаров Fashion-MNIST[2] — используется для классификации изображений одежды. Набор данных содержит изображения модных товаров размера 28 × 28 в оттенках серого, снабженные метками 10 категорий[3]; 60 000 изображений используются для обучения, а 10 000 — для тестирования. Модель, построенную для использования с MNIST, можно повторно использовать с Fashion-MNIST, изменив лишь несколько команд.

Обзоры фильмов на IMDb[4] — это набор, используемый для анализа эмоциональной окраски; он содержит обзоры, помеченные как положительные (1) или отрицательные (0); 25 000 обзоров предназначены для обучения и 25 000 — для тестирования. Этот набор данных используется в разделе 15.9 при изучении рекуррентных нейронных сетей.

CIFAR10[5] — используется для классификации малых изображений. Набор содержит цветные изображения 32 × 32, помеченные 10 категориями; 50 000 изображений предназначены для обучения и 10 000 — для тестирования.

CIFAR100[6] — этот набор данных, также используемый для классификации малых изображений, содержит цветные изображения 32 × 32, помеченные 100 категориями; 50 000 изображений предназначены для обучения и 10 000 — для тестирования.

---

[1] «The MNIST Database». MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges. http://yann.lecun.com/exdb/mnist/.

[2] Han Xiao and Kashif Rasul and Roland Vollgraf, Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, arXiv, cs.LG/1708.07747.

[3] https://keras.io/datasets/#fashion-mnist-database-of-fashion-articles.

[4] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

[5] https://www.cs.toronto.edu/~kriz/cifar.html.

[6] https://www.cs.toronto.edu/~kriz/cifar.html.

15.3. Нестандартные среды Anaconda

Прежде чем запускать примеры этой главы, необходимо установить используемые библиотеки. В примерах этой главы мы будем использовать версию Keras из библиотеки глубокого обучения TensorFlow[1]. На момент написания книги TensorFlow еще не поддерживает Python 3.7. Следовательно, для выполнения примеров этой главы потребуется Python 3.6.x. Мы покажем, как настроить *специализированную среду* для работы с Keras и TensorFlow.

Нестандартные среды в Anaconda

Дистрибутив Anaconda Python позволяет легко создавать нестандартные *среды* — конфигурации, в которых можно устанавливать разные библиотеки и разные версии библиотек. Они помогут обеспечить *воспроизводимость*, если ваш код зависит от конкретных версий Python или библиотек[2].

Среда по умолчанию в Anaconda называется *базовой средой*. Она создается автоматически при установке Anaconda. Все библиотеки Python, поставляемые с Anaconda, устанавливаются в базовой среде; все дополнительные устанавливаемые библиотеки также размещаются в ней, если явно не указано обратное. Нестандартные среды предоставляют средства управления конкретными библиотеками, которые должны устанавливаться для конкретных задач.

Создание среды Anaconda

Среда создается *командой* `conda create`. Создадим среду TensorFlow и присвоим ей имя `tf_env` (выберите любое имя по своему усмотрению). Выпол-

---

[1] Также существует автономная версия, которая позволяет выбрать между TensorFlow, Microsoft CNTK или библиотекой *Theano* Монреальского университета (разработка которой завершилась в 2017 году).

[2] В следующей главе будет представлена технология Docker как еще один механизм обеспечения воспроизводимости и удобное средство для формирования сложных сред на вашем локальном компьютере.

ните следующую команду в терминале, командной оболочке или командном приглашении Anaconda[1,2]:

```
conda create -n tf_env tensorflow anaconda ipython jupyterlab
   scikit-learn matplotlib seaborn h5py pydot graphviz
```

Команда определит зависимости перечисленных библиотек, после чего выведет все библиотеки, которые будут установлены в новой среде. Зависимостей много, поэтому перечисление может занять несколько минут. Когда появится приглашение:

```
Proceed ([y]/n)?
```

нажмите клавишу Enter, чтобы создать среду и установить библиотеки[3].

Активация альтернативной среды Anaconda

Чтобы использовать нестандартную среду, выполните *команду* `conda activate`:

```
conda activate tf_env
```

Команда действует только на текущий терминал, командную оболочку или приглашение командной строки Anaconda. Когда вы активируете нестандартную среду и устанавливаете новые библиотеки, они становятся частью активированной, но не базовой среды. Если вы открыли отдельные терминалы, командные оболочки или приглашения Anaconda, то по умолчанию они будут использовать базовую среду Anaconda.

Деактивация альтернативной среды Anaconda

Завершив работу с нестандартной средой, вернитесь к базовой среде в текущем терминале, командной оболочке или приглашении Anaconda следующей командой:

```
conda deactivate
```

---

[1] Пользователи Windows должны запустить приглашение командной строки Anaconda с правами администратора.

[2] Если ваш компьютер оснащен графическим процессором NVIDIA, совместимым с TensorFlow, вы можете заменить библиотеку tensorflow на tensorflow-gpu для улучшения быстродействия. За дополнительной информацией обращайтесь по адресу https://www.tensorflow.org/install/gpu. Некоторые графические процессоры AMD также могут использоваться с TensorFlow: http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/.

[3] При создании нестандартной среды команда conda устанавливает Python 3.6.7 — новейшую версию Python, совместимую с библиотекой tensorflow.

Документы Jupyter Notebook и JupyterLab

Примеры этой главы предоставляются только в форме документов Jupyter Notebook, упрощающих эксперименты с примерами. Вы можете настроить параметры и повторно выполнить документы Notebook. Для примеров этой главы JupyterLab следует запускать из папки ch15 (см. раздел 1.5.3).

15.4. Нейронные сети

Глубокое обучение — разновидность машинного обучения, использующая нейронные сети для обучения. *Искусственная нейронная сеть* (или просто нейронная сеть) представляет собой программную конструкцию, которая работает примерно по тем же принципам, по каким работает наш мозг. Управление нашей биологической нервной системой осуществляется по *нейронам*[1], которые взаимодействуют друг с другом по каналам, называемым *синапсами*[2]. По мере обучения конкретные нейроны для выполнения конкретной задачи (например, ходьба) взаимодействуют друг с другом более эффективно. В дальнейшем эти нейроны *активизируются* каждый раз, когда вы собираетесь идти[3].

Искусственные нейроны

В нейронной сети взаимосвязанные *искусственные нейроны* моделируют работу нейронов человеческого мозга для обучения сети. В процессе обучения связи между конкретными нейронами укрепляются для достижения желаемого результата. В *глубоком обучении с учителем*, которое используется в этой главе, мы стремимся спрогнозировать целевые метки, прилагаемые к образцам данных. Для этого проводится обучение модели обобщенной нейронной сети, которая затем может использоваться для прогнозирования на основании неизвестных данных[4].

Диаграмма искусственной нейронной сети

На следующей диаграмме изображена *трехуровневая* нейронная сеть. Каждый кружок представляет нейрон, а строки между ними моделируют синапсы.

---

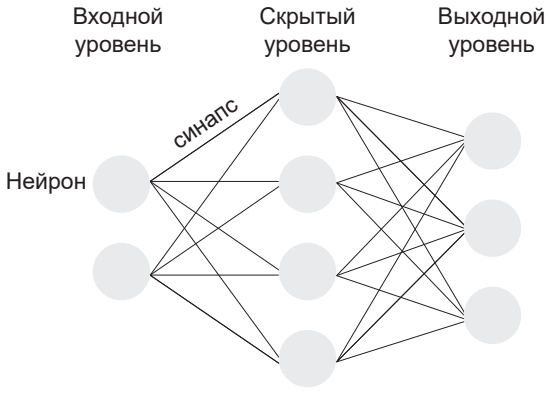
[1] https://ru.wikipedia.org/wiki/Нейрон.

[2] https://ru.wikipedia.org/wiki/Синапс.

[3] https://www.sciencenewsforstudents.org/article/learning-rewires-brain.

[4] Как и в области машинного обучения, можно создать сеть глубокого обучения *без учителя*, но эта тема выходит за рамки книги.

Выход одного нейрона становится входом другого нейрона, отсюда и термин «нейронная сеть». На этой конкретной диаграмме показана *полносвязная сеть* — каждый нейрон в заданном уровне соединяется со *всеми* нейронами следующего уровня:



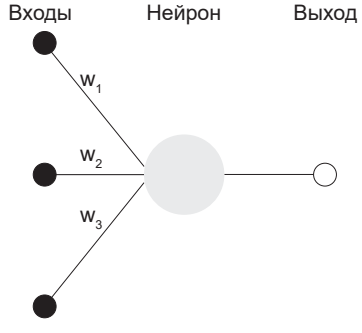
Обучение как итеративный процесс

Когда вы были ребенком, вы не научились ходить в один момент. Вы изучали этот процесс *со временем*, повторяя его снова и снова. В вашем мозге формировались меньшие компоненты движений, из которых состоит ходьба: вы учились стоять, удерживать равновесие, для того чтобы оставаться в вертикальном положении, поднимать ногу и перемещать ее вперед, и т. д. При этом вы получали обратную связь от своей среды. Когда вы делали успешный шаг, ваши родители улыбались и хлопали, а когда падали — могли удариться и ощутить боль.

Обучение нейронных сетей также происходит по итеративному принципу. Каждая итерация называется *эпохой* (epoch); каждый образец в обучающем наборе данных обрабатывается по одному разу. «Правильного» количества эпох не существует. Это гиперпараметр, который вам, возможно, придется настраивать с учетом обучающих данных и модели. Входными данными сети являются признаки обучающих образцов. Одни уровни изучают новые признаки по выходу предыдущих уровней, другие интерпретируют эти признаки для прогнозирования.

Как искусственные нейроны принимают решение об активации синапсов

В фазе обучения сеть вычисляет значения, называемые *весами*, для каждой связи между нейронами одного уровня и нейронами следующего уровня. На уровне отдельных нейронов каждое входное значение умножается на вес соответствующей связи, после чего сумма взвешенных входов передается *функции активации* нейрона. Вывод функции определяет, какие нейроны должны активироваться на основании входных данных — по аналогии с тем, как нейроны в мозге передают информацию в ответ на входные импульсы, поступающие от глаз, носа, ушей и т. д. На следующей диаграмме изображен нейрон с тремя входами (черные точки) и одним выводом (незакрашенный круг), который будет передан всем нейронам (или их части) следующего уровня в зависимости от типов уровней нейронной сети:



Значения w_1, w_2 и w_3 называются весами. В новой модели, которая обучается «с нуля», эти значения инициализируются моделью случайным образом. В процессе обучения сеть пытается минимизировать пропорцию ошибок между прогнозируемыми метками модели и фактическими метками образцов. Доля ошибок называется *потерями*, а формула для определения величины потерь называется *функцией потерь*. В процессе обучения сеть определяет величину, вносимую каждым нейроном в общие потери, после чего проходит по уровням в обратном направлении и регулирует веса, стараясь минимизировать потери. Этот прием называется *обратным распространением*. Оптимизация весов происходит постепенно — чаще всего в процессе, называемом *градиентным спуском*.

15.5. Тензоры

Библиотеки глубокого обучения обычно работают с данными в форме *тензоров*. Тензор по сути представляет собой многомерный массив. Такие библиотеки, как TensorFlow, упаковывают данные в один или несколько массивов, которые используются для выполнения математических вычислений, обеспечивающих обучение нейронной сети. Тензоры могут быть довольно большими, и количество измерений растет с повышением насыщенности данных (например, изображения, аудио и видео обладают большей насыщенностью, чем текст). Шолле обсуждает типы тензоров, часто встречающиеся в глубоком обучении[1]:

- *0D- (0-мерный) тензор* — отдельное значение, также называемое *скаляром*.
- *1D-тензор* — аналог одномерного массива, также называемого *вектором*. 1D-тензор может представлять последовательность, например почасовые показания температуры от датчика или слова одного обзора кинофильма.
- *2D-тензор* — аналог двумерного массива, также называемого *матрицей*. 2D-тензор может представлять изображение в оттенках серого, в котором два измерения тензора представляют ширину и высоту изображения в пикселах, а значение каждого элемента — интенсивность этого пиксела.
- *3D-тензор* — аналог трехмерного массива, который может использоваться для представления цветного изображения. Первые два измерения представляют ширину и высоту изображения в пикселах, а *глубина* каждой точки — составляющие RGB (интенсивности красной, зеленой и синей составляющей) цвета заданного пиксела. 3D-тензор также может представлять *коллекцию* 2D-тензоров, содержащих изображения в оттенках серого.
- *4D-тензор* — может использоваться для представления *коллекции* цветных изображений в 3D-тензорах. Также может использоваться для представления одного видеоролика. Каждый кадр видео фактически представляет цветное изображение.

---

[1] Chollet, François. *Deep Learning with Python*. Section 2.2. Shelter Island, NY: Manning Publications, 2018.

- *5D-тензор* — может использоваться для представления коллекции 4D-тензоров, содержащих видеоролики.

Форма тензора представляется в виде кортежа значений, в котором количество элементов задает количество измерений тензора, а каждое значение в кортеже — размер соответствующего измерения тензора.

Предположим, мы создаем сеть глубокого обучения для выявления и отслеживания объектов в видео 4K (высокое разрешение) с 30 кадрами в секунду. Каждый кадр видео 4K имеет размер 3840 × 2160 пикселов. Теперь предположим, что пикселы представлены красной, зеленой и синей составляющими цвета. Таким образом, *каждый кадр* будет представлен 3D-тензором, содержащим 24 883 200 элементов (3840 × 2160 × 3), а каждый видеоролик будет представлен 4D-тензором, содержащим последовательность кадров. Если продолжительность видео превышает минуту, то *тензор* будет содержать не менее 44 789 760 000 элементов!

Между тем каждую минуту на YouTube загружаются более 600 часов видео[1]. Таким образом, всего за одну минуту на Google может сформироваться тензор, содержащий 1 612 431 360 000 000 элементов для моделей глубокого обучения, бесспорно, это *большие данные*. Как вы вскоре увидите, тензоры быстро разрастаются *до гигантских размеров*, поэтому так важно эффективно работать с ними. Это — одна из ключевых причин, по которым большая часть глубокого обучения выполняется на графических процессорах (GPU). Недавно компания Google создала тензорные процессоры (TPU) специально для операций с тензорами. Они в разы превосходят GPU по скорости.

Высокопроизводительные процессоры

Для реальных задач глубокого обучения необходимы мощные процессоры, потому что размер тензоров может быть огромным, а операции с большими тензорами могут создать непосильную нагрузку на процессор. Перечислим процессоры, чаще всего используемые для глубокого обучения:

- NVIDIA GPU — изначально разработанные такими компаниями, как NVIDIA, для компьютерных игр, графические процессоры значительно превосходят традиционные процессоры при обработке больших объемов

[1] https://www.inc.com/tom-popomaronis/youtube-analyzed-trillions-of-data-points-in-2018-revealing-5-eye-opening-behavioral-statistics.html.

данных. Это позволяет разработчикам обучать, проверять и тестировать модели глубокого обучения с большей эффективностью, а следовательно, экспериментировать с большим количеством таких моделей. Графические процессоры оптимизированы для математических матричных операций, часто выполняемых с тензорами; это важный аспект внутренней реализации глубокого обучения. Процессоры NVIDIA Volta Tensor Core специально проектировались для глубокого обучения[1,2]. Многие NVIDIA GPU совместимы с TensorFlow (а следовательно, и с Keras) и могут повысить эффективность моделей глубокого обучения[3].

- Google TPU — понимая, что глубокое обучение играет важнейшую роль в ее будущем, компания Google разработала тензорные процессоры (TPU), которые в настоящее время используются в облачном сервисе Cloud TPU, способном «обеспечить производительность до 11,5 *петафлопса* в одном блоке[4]» (то есть 11,5 квадриллиона операций с плавающей точкой в секунду). Кроме того, TPU проектировались с расчетом на исключительную эффективность энергопотребления. Данный фактор крайне важен для таких компаний, как Google, уже имеющих гигантские вычислительные кластеры, которые растут с экспоненциальной скоростью и потребляют колоссальные объемы энергии.

15.6. Сверточные нейронные сети для распознавания образов; множественная классификация с набором данных MNIST

В главе 14 мы классифицировали рукописные цифры по изображениям низкого разрешения 8 × 8 пикселов из набора данных Digits, входящего в поставку scikit-learn. Набор данных базируется на подмножестве набора данных рукописных цифр MNIST с более высоким разрешением. В этой главе набор MNIST будет использоваться для исследования глубокого обучения на базе *сверточной нейронной сети*[5] (*CNN*). Сверточные нейронные сети часто приме-

---

1 https://www.nvidia.com/en-us/data-center/tensorcore/.
2 https://devblogs.nvidia.com/tensor-core-ai-performance-milestones/.
3 https://www.tensorflow.org/install/gpu.
4 https://cloud.google.com/tpu/.
5 https://en.wikipedia.org/wiki/Convolutional_neural_network.

няются в приложениях распознавания образов, например при распознавании рукописных цифр и символов, а также распознавании объектов в графических изображениях и видео. Кроме того, они могут применяться и в других классах приложений, например в обработке естественных языков и рекомендательных системах.

Набор данных Digits содержит всего 1797 образцов, тогда как набор MNIST содержит 70 000 помеченных образцов с изображениями цифр — 60 000 для обучения и 10 000 для тестирования. Каждый образец представляет собой изображение 28 × 28 пикселов (всего 784 признака) в оттенках серого, представленное в виде массива NumPy. Каждый пиксел является значением от 0 до 255, представляющим интенсивность (оттенок) этого пиксела — набор данных Digits использует менее точные оттенки со значениями от 0 до 16. Метки MNIST являются целыми значениями от 0 до 9 и обозначают цифры, представляемые каждым изображением.

Модель машинного обучения, использованная в предыдущей главе, выдавала на выходе прогнозируемый класс изображения цифры — целое число в диапазоне 0–9. Модель сверточной нейронной сети, которую мы построим, будет выполнять *вероятностную классификацию*[1]. Для каждого изображения цифры модель будет выдавать *массив* из 10 вероятностей, каждая из которых обозначает вероятность принадлежности цифры к классам от 0 до 9. Класс с *наивысшей* вероятностью становится прогнозируемым значением.

Воспроизводимость результатов в Keras и глубокое обучение

В этой книге уже неоднократно говорилось о важности *воспроизводимости* результатов. В области глубокого обучения обеспечить воспроизводимость сложнее, потому что библиотеки интенсивно используют параллельные операции с плавающей точкой. При разных запусках операции могут выполняться в разном порядке, что может привести к различающимся результатам. Получение воспроизводимых результатов в Keras требует особого сочетания настроек среды и конфигурации кода, описанных в Keras FAQ:

https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development

[1] https://en.wikipedia.org/wiki/Probabilistic_classification.

Базовая нейронная сеть Keras

Нейронная сеть Keras состоит из следующих компонентов:

- *Сеть* (также называемая *моделью*) — серия *уровней* с нейронами, используемыми для обучения на образцах. Нейроны каждого уровня получают входные данные, обрабатывают их (с использованием *функции активации*) и генерируют выходные данные. Эти данные подаются в сеть через *входной уровень*, определяющий размерность данных образца. За ним следуют *скрытые уровни* нейронов, реализующие обучение, и *выходной уровень*, генерирующий прогнозы. Чем больше уровней объединяет модель, тем глубже сеть — отсюда и термин «глубокое обучение».
- *Функция потерь* — вычисляет метрику эффективности прогнозирования сетью целевых значений. Более низкие значения соответствуют более качественным прогнозам.
- *Оптимизатор* — стремится минимизировать значения, сгенерированные функцией потерь, для оптимизации сети с целью повышения качества прогнозов.

Запуск JupyterLab

Этот раздел предполагает, что вы активировали среду Anaconda `tf_env`, созданную в разделе 15.3, после чего запустили JupyterLab из каталога `ch15`. Либо откройте файл `MNIST_CNN.ipynb` в JupyterLab и выполните готовый код в ячейках, либо создайте новый документ Notebook и введите код самостоятельно. При желании вы можете работать в командной строке IPython; тем не менее размещение кода в Jupyter Notebook существенно упрощает *повторное выполнение* кода примеров этой главы.

Напомним, что вы можете сбросить документ Jupyter Notebook и удалить его вывод командой Restart Kernel and Clear All Outputs... из меню JupyterLab Kernel. Команда завершает выполнение документа и стирает его вывод. Например, это можно сделать, если ваша модель работает недостаточно хорошо и вы хотите опробовать другие гиперпараметры или изменить структуру нейронной сети[1]. После этого можно снова выполнить документ Notebook по одной ячейке за раз или выполнить весь документ командой Run All из меню JupyterLab Run.

---

[1] Мы обнаружили, что для очистки вывода иногда нам приходилось выполнять эту команду дважды.

15.6.1. Загрузка набора данных MNIST

Импортируем *модуль* `tensorflow.keras.datasets.mnist`, чтобы загрузить набор данных:

```
[1]: from tensorflow.keras.datasets import mnist
```

Обратите внимание: так как мы используем версию Keras, встроенную в TensorFlow, имена модулей Keras начинаются с префикса `"tensorflow."`. В автономной версии Keras имена модулей начинаются с `"keras."`, поэтому будет использоваться имя `keras.datasets`. Keras использует *TensorFlow* для выполнения моделей глубокого обучения.

Функция `load_data` модуля `mnist` загружает обучающие и тестовые наборы MNIST:

```
[2]: (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

При вызове `load_data` набор данных MNIST будет загружен в вашей системе. Функция возвращает кортеж из двух элементов, содержащих обучающие и тестовые наборы. Каждый элемент сам по себе является кортежем, содержащим образцы и метки соответственно.

15.6.2. Исследование данных

Познакомимся поближе с данными, перед тем как работать с ними. Сначала проверим размеры изображений обучающего набора (`X_train`), меток обучающего набора (`y_train`), изображений тестового набора (`X_test`) и меток тестового набора (`y_test`):

```
[3]: X_train.shape
[3]: (60000, 28, 28)

[4]: y_train.shape
[4]: (60000,)

[5]: X_test.shape
[5]: (10000, 28, 28)

[6]: y_test.shape
[6]: (10000,)
```

Из размеров `X_train` и `X_test` видно, что изображения имеют более высокое разрешение, чем изображения из набора данных Digits библиотеки scikit-learn (8×8).

Визуализация цифр

Выведем некоторые изображения цифр. Подключите Matplotlib в документе Notebook, импортируйте Matplotlib и Seaborn, а затем выберите масштаб шрифта:

```
[7]: %matplotlib inline

[8]: import matplotlib.pyplot as plt

[9]: import seaborn as sns

[10]: sns.set(font_scale=2)
```

Магическая команда IPython

```
%matplotlib inline
```

означает, что графика на базе Matplotlib должна выводиться *в документе Notebook*, а не в отдельных окнах. За описаниями других магических команд IPython, которые могут использоваться в документах Jupyter Notebook, обращайтесь по адресу:

https://ipython.readthedocs.io/en/stable/interactive/magics.html

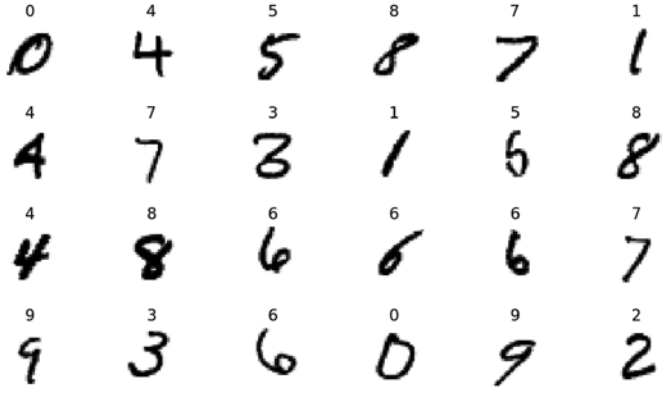
Затем мы выведем случайно выбранный набор из 24 изображений обучающего набора MNIST. Вспомните, о чем говорилось в главе 5: при индексировании массива NumPy можно передать последовательность индексов, чтобы выбрать только элементы массива с этими индексами. Здесь мы воспользуемся этой возможностью, чтобы выбрать элементы с одинаковыми индексами в массивах `X_train` и `y_train`. Это гарантирует, что для каждого случайно выбранного изображения будет выбрана правильная метка.

Функция `choice` библиотеки NumPy (из модуля `numpy.random`) случайным образом выбирает количество элементов, заданное вторым аргументом (`24`) из массива значений, заданного первым аргументом (в данном случае массив, содержащий диапазон индексов `X_train`). Функция возвращает массив, содержащий выбранные значения, который мы сохраняем в переменной `index`. Выражения `X_train[index]` и `y_train[index]` используют `index` для получения соответствующих элементов из обоих массивов. Остальные ячейки в коде визуализации из примера предыдущей главы:

```
[11]: import numpy as np
      index = np.random.choice(np.arange(len(X_train)), 24, replace=False)
```

```
figure, axes = plt.subplots(nrows=4, ncols=6, figsize=(16, 9))
for item in zip(axes.ravel(), X_train[index], y_train[index]):
    axes, image, target = item
    axes.imshow(image, cmap=plt.cm.gray_r)
    axes.set_xticks([])  # remove x-axis tick marks
    axes.set_yticks([])  # remove y-axis tick marks
    axes.set_title(target)
plt.tight_layout()
```

Из следующего вывода видно, что изображения цифр MNIST имеют более высокое разрешение, чем изображения из набора данных Digits библиотеки scikit-learn.



Присмотревшись к цифрам, вы поймете, почему распознавание рукописных цифр стало такой трудной задачей:

- Одни люди пишут «открытые» цифры 4 (как в первой и третьей строке), другие используют «закрытый» вариант написания (как во второй строке). И хотя все разновидности «четверки» обладают сходными характеристики, все они отличаются друг от друга.
- Цифра 3 во второй строке выглядит странно — она напоминает гибрид 6 и 7. Сравните с намного более четко прописанной цифрой 3 в четвертой строке.
- Цифру 5 во второй строке легко можно принять за 6.
- Кроме того, люди пишут цифры под разными углами, как видно из четырех цифр 6 в третьей и четвертой строках — две цифры располагаются прямо, одна наклонена влево, а другая вправо.

Если выполнить приведенный фрагмент несколько раз, то каждый раз будут выводиться случайно выбранные цифры[1]. Вероятно, вы увидите, что без меток над каждой цифрой вам будет нелегко распознать некоторые цифры. Вскоре вы увидите, насколько точно наша первая сверточная нейронная сеть будет распознавать цифры в тестовом наборе MNIST.

15.6.3. Подготовка данных

Как упоминалось в главе 14, упакованные наборы данных scikit-learn были предварительно обработаны по размерам нужных моделей. В реальных исследованиях вам обычно придется заниматься подготовкой данных самостоятельно (частично или полностью). Набор данных MNIST требует определенной подготовки для использования в сверточной нейронной сети Keras.

Переформатирование изображений

Сверточные нейронные сети Keras должны получать входные массивы NumPy, у которых каждый образец имеет следующий формат:

(*ширина*, *высота*, *каналы*)

В наборе MNIST *ширина* и *высота* каждого изображения равны 28 пикселам, при этом каждый пиксел имеет один *канал* (оттенок серого от 0 до 255), так что формат каждого образца будет выглядеть так:

```
(28, 28, 1)
```

Полноцветные изображения со значениями RGB (красный/зеленый/синий) каждого пиксела будут иметь три *канала* — по одному для красной, зеленой и синей составляющей.

По мере того как нейронная сеть обучается на основании изображений, она может создать еще много каналов. Вместо оттенков цвета каналы могут представлять более сложные признаки, например контуры, кривые и линии, что в конечном итоге позволит сети распознавать цифры на основании этих дополнительных признаков и их сочетаний.

---

[1] При многократном выполнении кода ячейки номер фрагмента рядом с ячейкой будет каждый раз увеличиваться, как это происходит в IPython в командной строке.

Преобразуем 60 000 обучающих и 10 000 тестовых изображений к правильным размерам для использования в нашей сверточной нейронной сети. Напомним, что метод `reshape` массива NumPy получает кортеж, представляющий новую конфигурацию массива:

```
[12]: X_train = X_train.reshape((60000, 28, 28, 1))

[13]: X_train.shape
[13]: (60000, 28, 28, 1)

[14]: X_test = X_test.reshape((10000, 28, 28, 1))

[15]: X_test.shape
[15]: (10000, 28, 28, 1)
```

Нормализация данных изображений

Числовые признаки в образцах данных могут иметь сильно различающиеся диапазоны значений. Сети глубокого обучения лучше работают с данными, масштабированными в диапазоне от 0,0 до 1,0 или в диапазоне с математическим ожиданием 0,0 и стандартным отклонением 1,0[1]. Преобразование данных в одну из этих форм называется *нормализацией*.

В MNIST каждый пиксел представлен целым числом в диапазоне 0–255. Следующие команды преобразуют значения в 32-разрядные (4-байтовые) числа с плавающей точкой при помощи метода `astype` массива NumPy, после чего каждый элемент полученного массива делится на 255 с получением нормализованных значений в диапазоне 0,0–1,0:

```
[16]: X_train = X_train.astype('float32') / 255

[17]: X_test = X_test.astype('float32') / 255
```

Прямое унитарное кодирование: преобразование меток из целых чисел в категорийные данные

Как упоминалось ранее, прогноз сверточной нейронной сети для каждой цифры будет представлять собой массив из 10 вероятностей, обозначающих

[1] S. Ioffe and Szegedy, C. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». https://arxiv.org/abs/1502.03167

принадлежность цифры к одному из классов от 0 до 9. При оценке точности модели Keras сравнивает прогнозы модели с метками. Для этого Keras требует, чтобы значения имели одинаковую размерность, но метка MNIST для каждой цифры представляет собой целое значение в диапазоне 0–9. Следовательно, метки необходимо преобразовать в *категорийные данные*, то есть массивы категорий, соответствующие формату прогнозов. Для этого будет использоваться процесс, называемый *прямым унитарным кодированием* [1]: данные преобразуются в массивы значений `1.0` и `0.0`, в котором только один элемент равен `1.0`, а остальные равны `0.0`. Для MNIST в результате прямого унитарного кодирования будут получены 10-элементные массивы, представляющие категории от 0 до 9. Прямое унитарное кодирование также может применяться к другим типам данных.

Мы точно знаем, к какой категории относится каждая цифра, так что категорийное представление метки цифры будет состоять из `1.0` в элементе с индексом цифры и `0.0` во всех остальных элементах (напомним, во внутренней реализации Keras используются числа с плавающей точкой). Таким образом, категорийное представление 7 будет выглядеть так:

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]
```

а представление 3 — так:

```
[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Модуль `tensorflow.keras.utils` предоставляет функцию `to_categorical` для выполнения прямого унитарного кодирования. Функция подсчитывает уникальные категории, а затем для каждого закодированного элемента создает массив соответствующей длины со значением `1.0` в нужной позиции. Преобразуем `y_train` и `y_test` из одномерных массивов со значениями 0–9 в двумерные массивы категорийных данных. После этого строки массивов будут выглядеть так, как показано выше. Фрагмент [21] выводит категорийные данные одного образца для цифры 5 (напомним, что NumPy выводит точку, но не выводит завершающие нули в значениях с плавающей точкой):

```
[18]: from tensorflow.keras.utils import to_categorical

[19]: y_train = to_categorical(y_train)
```

---

[1] Этот термин происходит от некоторых цифровых схем, у которых в группе битов может быть установлен только один бит (то есть только один бит может иметь значение 1). https://en.wikipedia.org/wiki/One-hot.

```
[20]: y_train.shape
[20]: (60000, 10)

[21]: y_train[0]
[21]: array([ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
dtype=float32)
[22]: y_test = to_categorical(y_test)

[23]: y_test.shape
[23]: (10000, 10)
```

15.6.4. Создание нейронной сети

Итак, данные подготовлены, и мы можем переходить к настройке сверточной нейронной сети. Начнем с *модели* Keras `Sequential` из *модуля* `tensorflow.keras.models`:

```
[24]: from tensorflow.keras.models import Sequential

[25]: cnn = Sequential()
```

Полученная сеть будет выполнять свои уровни последовательно — выход одного уровня станет входом другого; такие сети также называются *сетями прямого распространения*. Как будет показано при обсуждении *рекуррентных нейронных сетей*, не все нейронные сети работают по этому принципу.

Добавление уровней в сеть

Типичная сверточная нейронная сеть состоит из нескольких уровней — *входного уровня*, получающего обучающие образцы *скрытых уровней*, которые обучаются по образцам, и *выходного уровня*, генерирующего вероятности прогнозов. Теперь создадим простейшую сверточную сеть. Импортируем из *модуля* `tensorflow.keras.layers` классы уровней, которые будут использоваться в этом примере:

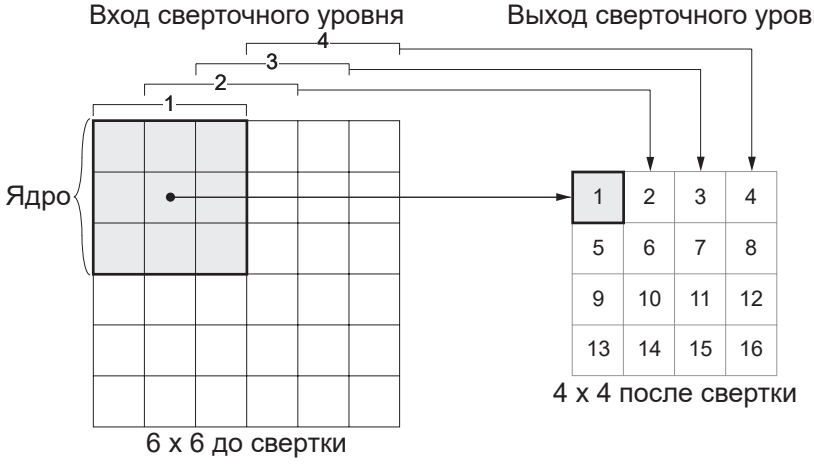
```
[26]: from tensorflow.keras.layers import Conv2D, Dense, Flatten,
      MaxPooling2D
```

Все разновидности уровней рассматриваются ниже.

Свертка

Наша сеть будет начинаться со *сверточного уровня*, который использует отношения между пикселами, расположенными поблизости друг к другу, для выявления полезных признаков (или закономерностей) для малых областей каждого образца. Эти признаки станут входными данными последующих уровней.

Малые области, на которых обучается сверточная нейронная сеть, называются *ядрами* (kernels) или *участками* (patches). Рассмотрим свертку на примере изображения 6 × 6. На диаграмме слева закрашенный квадрат 3 × 3 представляет ядро — числа просто определяют порядок посещения и обработки ядер:



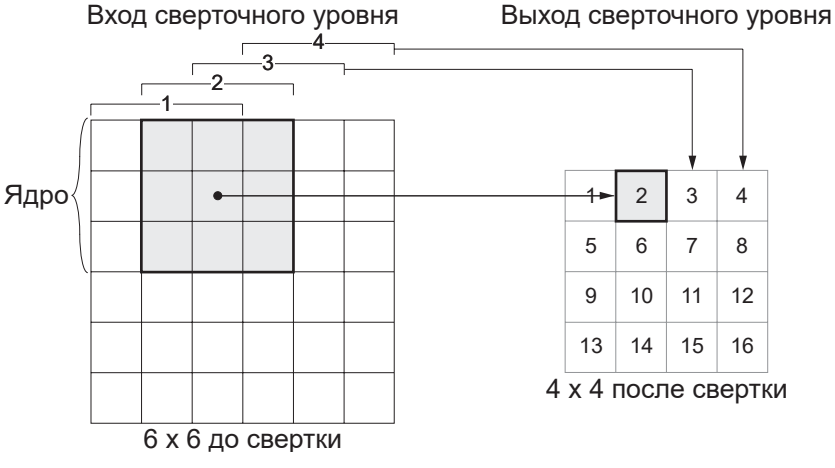
Ядро можно рассматривать как «скользящее окно», которое сверточный уровень перемещает по одному пикселу слева направо по изображению. Когда ядро достигает правого края, сверточный уровень смещает ядро на один пиксел вниз и повторяет процесс слева направо. Обычно используются размеры ядра 3 × 3[1], хотя нам встречались ядра 5 × 5 и 7 × 7 для изображений более высокого разрешения. Размер ядра является настраиваемым гиперпараметром.

В исходном состоянии ядро находится в левом верхнем углу исходного изображения — позиция ядра 1 (закрашенный квадрат) на входном уровне на диаграмме со с. 716. Сверточный уровень выполняет математические вычис-

[1] https://www.quora.com/How-can-I-decide-the-kernel-size-output-maps-and-layers-of-CNN.

ления, используя эти *девять* признаков для «обучения», после чего выводит *один* новый признак в позицию 1 выхода уровня. Рассматривая признаки, находящиеся по соседству друг с другом, сеть начинает распознавать такие признаки, как контуры, прямые линии и кривые.

Затем сверточный уровень смещает ядро на один пиксел вправо (величина смещения называется *шагом*) в позицию 2 входного уровня. Новая позиция *перекрывается* с двумя из трех столбцов предыдущей позиции, что позволяет сверточному уровню учиться по всем соприкасающимся признакам. Уровень обучается по девяти признакам ядра в позиции 2 и выводит один новый признак в позиции 2 на выходе:



Для изображения 6 × 6 и ядра 3 × 3 сверточный уровень повторяет процедуру еще два раза, чтобы получить признаки для позиций 3 и 4 на выходе уровня. Затем сверточный уровень смещает ядро на один пиксел и повторяет процесс слева направо для следующих четырех позиций ядра, формируя выход для позиций 5–8, затем 9–12 и, наконец, 13–16. Полный проход по изображению слева направо и сверху вниз называется *фильтром*. Для ядра 3 × 3 размеры фильтра (4 × 4 в приведенном примере) *на 2 меньше размеров входных данных* (6 × 6). Для каждого изображения MNIST 28 × 28 размер фильтра составит 26 × 26. При обработке небольших изображений (таких, как в MNIST) количество фильтров в сверточном уровне обычно равно 32 или 64, и каждый фильтр предоставляет разные результаты. Число фильтров зависит от размеров изображения: изображения с высоким разрешением имеют больше признаков, поэтому им требуется больше фильтров. Если изучить код, который

использовался командой Keras для построения заранее обученных сверточных нейронных сетей[1], вы увидите, что они используют в своих первых сверточных уровнях 64, 128 и даже 256 фильтров. На основании сверточных нейронных сетей и того факта, что изображения MNIST невелики, мы будем использовать 64 фильтра в первом сверточном уровне. Набор фильтров, производимых сверточным уровнем, называется *картой признаков*.

Последующие сверточные уровни объединяют признаки из предыдущих карт признаков для распознавания более крупных признаков и т. д. Например, если сеть предназначена для распознавания лиц, то ранние уровни могут распознавать линии, контуры и кривые, а последующие уровни могут объединять их в признаки более высокого уровня — глаза, брови, носы, уши и т. д. После того как сеть изучит некоторый признак, вследствие свертки она сможет распознать его в любом месте изображения. Это — одна из причин использования сверточных нейронных сетей для распознавания объектов в изображениях.

Добавление сверточного уровня

Добавим в модель сверточный уровень `Conv2D`:

```
[27]: cnn.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu',
                     input_shape=(28, 28, 1)))
```

Уровень `Conv2D` настраивается следующими аргументами:

- `filters=64` — количество фильтров в итоговой карте признаков;
- `kernel_size=(3, 3)` — размер ядра, используемого в каждом фильтре;
- `activation='relu'` — для получения вывода уровня используется функция активации `'relu'` (*Rectified Linear Unit*). `'relu'` — наиболее часто используемая функция активации в современных сетях глубокого обучения[2]; она просто вычисляется, что хорошо отражается на быстродействии[3]. Чаще всего для сверточных уровней рекомендуется использовать именно эту функцию[4].

---

[1] https://github.com/keras-team/keras-applications/tree/master/keras_applications.

[2] *Шолле Франсуа*. Глубокое обучение на Python. — СПб.: Питер, 2020. С. 97.

[3] https://towardsdatascience.com/exploring-activation-functions-for-neural-networks-73498da59b02.

[4] https://www.quora.com/How-should-I-choose-a-proper-activation-function-for-theneural-network.

Так как это первый уровень модели, мы также передаем аргумент `input_shape=(28, 28,1)` для определения размерности каждого образца. При этом автоматически создается входной уровень для загрузки образцов и передачи их уровню `Conv2D`, который является первым *скрытым* уровнем. В Keras каждый последующий уровень определяет свое значение `input_shape` по выходным данным предыдущего уровня, что упрощает *наложение* уровней.

Размерность вывода первого сверточного уровня

На предыдущем сверточном уровне входные образцы имеют формат $28 \times 28 \times 1$, то есть 784 признака для каждого. Мы задали для уровня 64 фильтра и размер ядра 3×3, так что вход каждого изображения имеет размеры $26 \times 26 \times 64$, что дает 43 264 признака в карте признаков — размерность существенно возрастает, а число огромно по сравнению с признаками, которые обрабатывались в моделях главы 14. Так как каждый уровень добавляет новые признаки, *степень размерности* полученных карт признаков значительно увеличивается. Это — одна из причин, по которым исследования в области глубокого обучения часто требуют колоссальных вычислительных мощностей.

Чрезмерная подгонка

Чрезмерная подгонка (см. главу 14) возникает, когда модель слишком сложна по сравнению с тем, что она моделирует. В исключительных случаях модель запоминает свои обучающие данные. Построенный вами прогноз на основании модели с чрезмерной подгонкой будет точным, если новые данные совпадают с обучающими, но модель при этом может работать неэффективно с неизвестными ей данными. Чрезмерная подгонка обычно возникает при глубоком обучении, когда размерность уровней становится слишком большой[1,2,3]. В результате сеть изучает признаки изображений цифр из обучающего набора, вместо того чтобы изучать *общие* признаки изображений цифр. Среди методов предотвращения чрезмерной подгонки можно выделить обучение для меньшего количества эпох, приращение данных, прореживание и L1/L2-регуляризацию[4,5] (о прореживании см. ниже).

---

[1] https://cs231n.github.io/convolutional-networks/.

[2] https://medium.com/@cxu24/why-dimensionality-reduction-is-important-dd60b5611543.

[3] https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a.

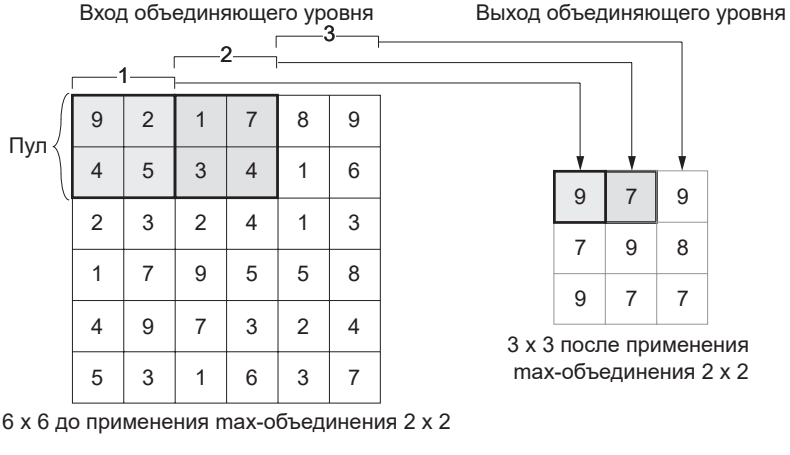
[4] https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d.

[5] https://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html.

Более высокая степень размерности увеличивает (иногда значительно) и время вычислений. Если же глубокое обучение выполняется на обычном процессоре (вместо графического или тензорного), то обучение может стать недопустимо медленным.

Добавление объединяющего уровня

Для предотвращения чрезмерной подгонки и сокращения времени вычислений за сверточным уровнем часто следует один или несколько уровней, *сокращающих степень размерности* выходных данных сверточного уровня. *Объединяющий уровень сжимает* результаты, отбрасывая признаки, вследствие чего модель становится более общей. Самый распространенный метод объединения — *max-объединение*; он анализирует квадрат признаков 2 × 2 и оставляет только максимальный признак. Чтобы понять, как работает объединение, вернемся к набору признаков 6 × 6. На следующей диаграмме числовые значения в квадрате 6 × 6 представляют признаки, которые требуется сжать, и квадрат 2 × 2 в позиции 1 представляет исходный пул признаков для анализа:



Уровень max-объединения сначала проверяет пул в позиции 1 на приведенной диаграмме, после чего выводит *максимальный* признак из этого пула (9) на диаграмме. В отличие от свертки, пулы *не перекрываются*. Пул смещается на свою ширину — для пула 2 × 2 *шаг* равен 2. Для второго пула, представленного

оранжевым квадратом 2 × 2, уровень выводит признак (7). Для третьего пула выводится признак (9). Когда пул достигнет правого края, объединяющий уровень смещает пул вниз на его высоту (2 строки), после чего продолжает смещение слева направо. Так как каждая группа из четырех признаков сокращается до 1, объединение 2 × 2 *сжимает* количество признаков на 75%. Добавим уровень `MaxPooling2D` в модель:

```
[28]: cnn.add(MaxPooling2D(pool_size=(2, 2)))
```

Выход предыдущего уровня сокращается с 26 × 26 × 64 до 13 × 13 × 64[1].

Хотя объединение часто применяется для борьбы с чрезмерной подгонкой, авторы некоторых исследований считают, что дополнительные сверточные уровни с большим размером шага ядра могут сократить степень размерности и предотвратить чрезмерную подгонку *без* потери признаков[2].

Добавление еще одного сверточного уровня и объединяющего уровня

Сверточные нейронные сети часто содержат много сверточных и объединяющих уровней. Сверточные сети Keras обычно удваивают количество фильтров в последующих сверточных уровнях, чтобы модель могла извлечь больше связей между признаками[3]. Добавим второй сверточный уровень со 128 фильтрами, за которым следует второй объединяющий уровень для сокращения степени размерности на 75%:

```
[29]: cnn.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))

[30]: cnn.add(MaxPooling2D(pool_size=(2, 2)))
```

Входом второго сверточного уровня является выход 13 × 13 × 64 первого объединяющего уровня. Таким образом, выход фрагмента [29] будет иметь размеры 11 × 11 × 128. Для нечетных размеров (например, 11 × 11) объединяющие уровни Keras по умолчанию используют округление *вниз* (в данном

---

[1] Другой метод сокращения чрезмерной подгонки заключается в добавлении уровней прореживания.

[2] Tobias, Jost, Dosovitskiy, Alexey, Brox, Thomas, Riedmiller, and Martin. «Striving for Simplicity: The All Convolutional Net.» 13 апреля 2015 г., https://arxiv.org/abs/1412.6806.

[3] https://github.com/keras-team/keras-applications/tree/master/keras_applications.

случае до 11 × 11), так что выход этого объединяющего уровня будет иметь размеры 5 × 5 × 128.

Одномерное преобразование результатов

На текущий момент выход предыдущего уровня является трехмерным (5 × 5 × 128), но на выходе модель должна выдавать *одномерный массив* 10 вероятностей классификации цифр. Для подготовки одномерных прогнозов сначала необходимо *преобразовать* трехмерный выход предыдущего уровня к одному измерению. Уровень Keras `Flatten` преобразует свой вход к одному измерению. В данном случае выход уровня `Flatten` будет иметь размеры 1 × 3200 (то есть `5*5*128`):

```
[31]: cnn.add(Flatten())
```

Добавление уровня Dense для сокращения количества признаков

Уровни, предшествующие уровню `Flatten`, изучали признаки цифр. Теперь необходимо взять все эти признаки и изучить отношения между ними, чтобы наша модель могла определить, какую цифру представляет то или иное изображение. Изучение отношений между признаками и проведение классификации достигается в полносвязных уровнях `Dense` наподобие изображенных на диаграмме нейронной сети ранее в этой главе. Следующий уровень `Dense` создает 128 нейронов (блоков), обучающихся на 3200 выходных значениях предыдущего уровня:

```
[32]: cnn.add(Dense(units=128, activation='relu'))
```

Многие сверточные сети содержат по крайней мере один уровень `Dense`. Сверточные сети, предназначенные для обработки более сложных графических наборов данных с изображениями более высокого разрешения вроде ImageNet — набора данных, содержащего более 14 миллионов изображений[1], — часто содержат несколько уровней `Dense`, обычно с 4096 нейронами. Такие конфигурации встречаются в некоторых предварительно обученных сверточных нейронных сетях Keras для ImageNet[2] (см. раздел 15.11).

---

[1] http://www.image-net.org.

[2] https://github.com/keras-team/keras-applications/tree/master/keras_applications.

Добавление еще одного уровня Dense для получения итогового результата

Последним уровнем нашей сети будет уровень `Dense`, который классифицирует свои входные данные по нейронам, представляющим классы от 0 до 9. *Функция активации* `softmax` преобразует значения 10 нейронов в классификационные вероятности. Нейрон, производящий наибольшую вероятность, представляет прогноз для заданного изображения:

```
[33]: cnn.add(Dense(units=10, activation='softmax'))
```

Вывод сводной информации модели

Метод `summary` модели выводит информацию об уровнях модели. В частности, здесь можно найти интересную информацию о размерах выходных данных разных уровней и количестве параметров. Параметрами являются *веса*, выведенные сетью в ходе обучения[1,2]. Сеть относительно невелика, однако ей пришлось изучить почти 500 000 параметров! И это — в случае с крошечными изображениями, занимающими меньше четверти размера значка на домашних экранах большинства современных смартфонов. Но представьте, сколько признаков придется изучить сети для обработки кадров видео 4K или изображений сверхвысокого разрешения, создаваемых современными цифровыми камерами! В графе `Output Shape` значение `None` означает лишь то, что модель не знает заранее, сколько обучающих образцов вы предоставите, — это станет известным лишь с началом обучения.

```
[34]: cnn.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 128)         0
```

[1] https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491.

[2] https://www.kdnuggets.com/2018/06/deep-learning-best-practices-weight-initialization.html.

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               409728
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 485,514
Trainable params: 485,514
Non-trainable params: 0
_________________________________________________________________
```

Обратите внимание и на отсутствие «необучаемых» (`Non-trainable`) параметров. По умолчанию Keras проводит обучение по *всем* параметрам, но обучение можно заблокировать для конкретных уровней; обычно это делается в процессе настройки и оптимизации сетей или при использовании параметров, полученных в ходе обучения другой модели, в новой модели (этот процесс называется *переносом обучения*[1]).

Визуализация структуры модели

Чтобы получить наглядное представление о модели, воспользуйтесь *функцией* `plot_model` из модуля `tensorflow.keras.utils`:

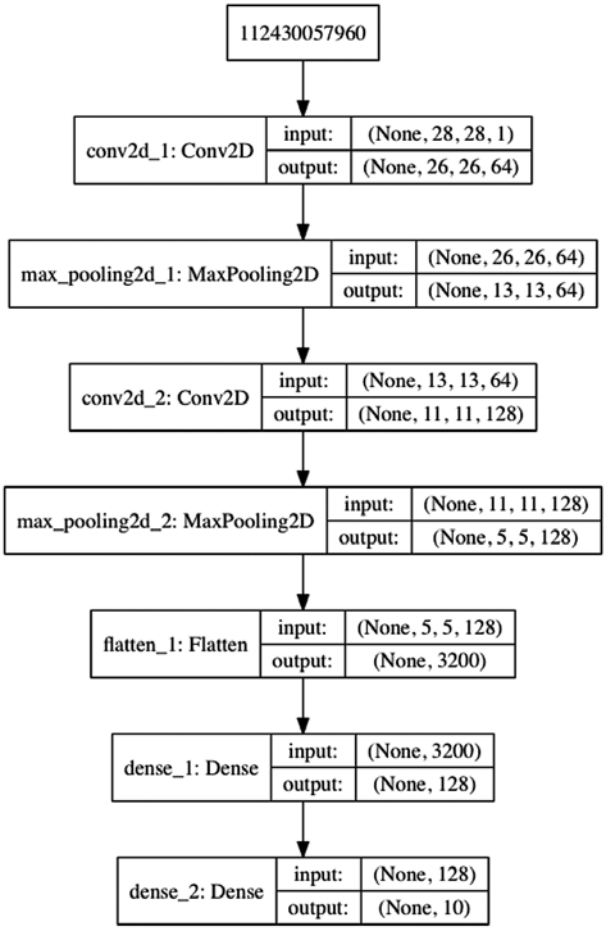
```
[35]: from tensorflow.keras.utils import plot_model
      from IPython.display import Image
      plot_model(cnn, to_file='convnet.png', show_shapes=True,
                 show_layer_names=True)
      Image(filename='convnet.png')
```

После сохранения результата в файле convnet.png мы используем *класс* `Image` модуля `IPython.display` для отображения результата в документе Notebook. Keras назначает имена уровней на диаграмме[2]:

---

[1] https://keras.io/getting-started/faq/#how-can-i-freeze-keras-layers.

[2] Узел с большим целочисленным значением 112430057960 в верхней части диаграммы появился из-за ошибки в текущей версии Keras. Узел представляет входной уровень и должен иметь обозначение «InputLayer».



Компиляция модели

После добавления всех уровней построение модели завершается вызовом *метода* `compile`:

```
[36]: cnn.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

Аргументы:

- `optimizer='adam'` — *оптимизатор*, используемый моделью для регулировки весов в нейронной сети в процессе обучения. Отметим, что существует много подходящих оптимизаторов[1], в частности, `'adam'` показывает неплохие результаты в широком спектре моделей[2,3].
- `loss='categorical_crossentropy'` — *функция потерь*, используемая оптимизатором в сетях множественной классификации (таких, как наша сверточная сеть), которая будет прогнозировать 10 классов. В процессе обучения нейронной сети оптимизатор пытается минимизировать значения, возвращаемые функцией потерь. Чем ниже потери, тем лучше справляется нейронная сеть с прогнозированием. Для бинарной классификации (которая будет использоваться позднее в этой главе) Keras предоставляет функцию `'binary_crossentropy'`, а для регрессии — `'mean_squared_error'`. За информацией о других функциях потерь обращайтесь по адресу:

  https://keras.io/losses/.

- `metrics=['accuracy']` — список *метрик*, которые будут производиться сетью, для того чтобы упростить вам оценку модели. Точность — популярная метрика, часто используемая в классификационных моделях. В этом примере мы воспользуемся метрикой точности для проверки процента правильных прогнозов. Список других метрик доступен по адресу:

  https://keras.io/metrics/.

15.6.5. Обучение и оценка модели

По аналогии с моделями scikit-learn обучим модель Keras вызовом ее *метода* `fit`:

- Как и в Scikit-learn, в первых двух аргументах передаются обучающие данные и категорийные метки целей.

---

[1] За описаниями других оптимизаторов Keras обращайтесь по адресу https://keras.io/optimizers/.

[2] https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2.

[3] https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f.

- Аргумент `epochs` сообщает, сколько раз модель должна обработать весь набор обучающих данных. Напомним, обучение нейронных сетей проходит в итеративном режиме.
- Аргумент `batch_size` задает количество образцов, обрабатываемых за один раз в каждой эпохе. Для большинства моделей выбирается степень 2 от 32 до 512. Более высокие значения могут привести к снижению точности модели[1]. Мы выбрали значение 64. Вы можете опробовать разные значения и определить, как они влияют на эффективность модели.
- В общем случае некоторые образцы должны использоваться для *проверки* модели. Если вы укажете проверочные данные, то после каждой эпохи модель использует их для построения прогнозов и вывода *проверочных значений потерь и точности*. Вы можете изучить эти модели для настройки ваших уровней и гиперпараметров метода `fit` или, возможно, для изменения структуры уровней модели. В данном случае мы использовали *аргумент* `validation_split` для обозначения того, что модель должна зарезервировать *последние* 10% (`0.1`) обучающих образцов для проверки[2], — в данном случае 6000 образцов будут использованы для проверки. Если у вас имеются отдельные проверочные данные, то аргумент `validation_data` (см. раздел 15.9) может использоваться для передачи кортежа с массивами образцов и целевых меток. В общем случае лучше использовать *случайно выбранные проверочные данные*. Для этой цели можно воспользоваться функцией `train_test_split` библиотеки scikit-learn (как это будет сделано позднее в этой главе), после чего передать случайно выбранные данные в аргументе `validation_data`.

В следующем выводе точность обучения (`acc`) и точность проверки (`val_acc`) выделены жирным шрифтом:

```
[37]: cnn.fit(X_train, y_train, epochs=5, batch_size=64,
              validation_split=0.1)
Train on 54000 samples, validate on 6000 samples
Epoch 1/5
54000/54000 [==============================] - 68s 1ms/step - loss:
```

[1] Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy and Ping Tak Peter Tang. «On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.» CoRR abs/1609.04836 (2016). https://arxiv.org/abs/1609.04836.

[2] https://keras.io/getting-started/faq/#how-is-the-validation-split-computed.

```
0.1407 - acc: 0.9580 - val_loss: 0.0452 - val_acc: 0.9867
Epoch 2/5
54000/54000 [==============================] - 64s 1ms/step - loss:
0.0426 - acc: 0.9867 - val_loss: 0.0409 - val_acc: 0.9878
Epoch 3/5
54000/54000 [==============================] - 69s 1ms/step - loss:
0.0299 - acc: 0.9902 - val_loss: 0.0325 - val_acc: 0.9912
Epoch 4/5
54000/54000 [==============================] - 70s 1ms/step - loss:
0.0197 - acc: 0.9935 - val_loss: 0.0335 - val_acc: 0.9903
Epoch 5/5
54000/54000 [==============================] - 63s 1ms/step - loss:
0.0155 - acc: 0.9948 - val_loss: 0.0297 - val_acc: 0.9927
[37]: <tensorflow.python.keras.callbacks.History at 0x7f105ba0ada0>
```

В разделе 15.7 рассматривается *TensorBoard* — инструмент TensorFlow для визуализации данных по моделям глубокого обучения. В частности, мы рассмотрим диаграммы, которые показывают, как точность обучения и проверки и значения потерь изменяются в разных эпохах. В разделе 15.8 будет продемонстрирована программа Андрея Карпати (Andrej Karpathy) ConvnetJS, которая обучает сверточные сети в браузере и динамически строит визуализацию выходов уровней, включая то, что «видит» каждый сверточный уровень в процессе обучения. Апробация моделей MNIST и CIFAR10 поможет вам лучше понять сложные механизмы работы нейронных сетей.

В процессе обучения метод `fit` выводит различную информацию: прогресс каждой эпохи, длительность выполнения эпохи (в нашем случае каждая занимала 63–70 секунд) и оценочные метрики для текущего прохода. В последней эпохе этой модели точность достигла 99,48% для обучающих образцов (`acc`) и 99,27% для проверочных образцов (`val_acc`). Впечатляющие показатели, особенно если учесть, что мы еще не пытались настраивать гиперпараметры или подбирать количество и типы уровней, что может улучшить (или ухудшить) результаты. Как и машинное обучение, глубокое обучение является эмпирической дисциплиной, в которой большое количество экспериментов приносит пользу.

Оценка модели

Теперь точность модели можно проверить на данных, неизвестных модели. Для этого вызовем *метод* `evaluate` модели, который выводит продолжитель-

ность обработки тестовых образцов (4 секунды и 366 микросекунд в данном случае):

```
[38]: loss, accuracy = cnn.evaluate(X_test, y_test)
10000/10000 [==============================] - 4s 366us/step

[39]: loss
[39]: 0.026809450998473768

[40]: accuracy
[40]: 0.9917
```

Из этого вывода следует, что наша модель сверточных нейронных сетей обеспечивает точность 99,17% при прогнозировании меток для незнакомых данных, хотя мы еще даже не пытались настраивать модель. Поиски в интернете позволяют отыскать модели, способные прогнозировать данные MNIST с почти 100-процентной точностью. Попробуйте поэкспериментировать с разными уровнями, типами уровней и их параметрами, наблюдая за тем, как эти изменения влияют на результаты.

Построение прогнозов

Метод `predict` модели прогнозирует классы изображений цифр из своего аргумента-массива (`X_test`):

```
[41]: predictions = cnn.predict(X_test)
```

Чтобы проверить, какой цифре соответствует первый образец, достаточно просмотреть `y_test[0]`:

```
[42]: y_test[0]
[42]: array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

Как видим, первый образец представляет цифру 7, потому что в категорийном представлении метки тестового образца в элементе с индексом 7 содержится значение `1.0` (напомним, представление создано *прямым унитарным кодированием*).

Проверим вероятности, возвращаемые методом `predict` для первого тестового образца:

```
[43]: for index, probability in enumerate(predictions[0]):
          print(f'{index}: {probability:.10%}')
0: 0.0000000201%
1: 0.0000001355%
2: 0.0000186951%
3: 0.0000015494%
4: 0.0000000003%
5: 0.0000000012%
6: 0.0000000000%
7: 99.9999761581%
8: 0.0000005577%
9: 0.0000011416%
```

Согласно этим результатам, `predictions[0]` показывает, что по мнению нашей модели образец представляет цифру 7 с *почти* 100-процентной вероятностью. Не все прогнозы обладают такой достоверностью.

Поиск неправильных прогнозов

На следующем шаге стоит просмотреть несколько *неправильно* спрогнозированных изображений, чтобы понять, с какими изображениями у нашей модели возникли трудности. Например, если модель всегда ошибается с распознаванием цифры 8, то, возможно, в обучающие данные стоит добавить больше образцов 8.

Прежде чем просматривать ошибочные прогнозы, необходимо их найти. Возьмем приведенную выше строку `predictions[0]`. Чтобы определить, был ли прогноз правильным, необходимо сравнить индекс наибольшей вероятности в `predictions[0]` с индексом элемента, содержащего `1.0` в `y_test[0]`. Если значения индексов совпадают, то прогноз правильный; в противном случае он ошибочен. Функция `argmax` библиотеки NumPy определяет индекс элемента с наибольшим значением в своем аргументе-массиве. Воспользуемся этой функцией для поиска ошибочных прогнозов. В следующем фрагменте `p` — массив прогнозируемых значений, а `e` — массив ожидаемых значений (ожидаемыми значениями являются метки тестовых изображений из набора данных):

```
[44]: images = X_test.reshape((10000, 28, 28))
      incorrect_predictions = []

      for i, (p, e) in enumerate(zip(predictions, y_test)):
```

```
    predicted, expected = np.argmax(p), np.argmax(e)

    if predicted != expected:
        incorrect_predictions.append(
            (i, images[i], predicted, expected))
```

В этом фрагменте мы сначала изменим размеры (28, 28, 1), необходимые Keras для обучения, до (28, 28), требуемых Matplotlib для вывода изображений. Затем список `incorrect_predictions` заполняется в цикле `for`. Мы упаковываем строки данных, представляющие каждый образец в массивах `predictions` и `y_test`, а затем перебираем их для сохранения индексов. Если результаты `argmax` для `p` и `e` различные, то это означает, что прогноз ошибочен, и к списку `incorrect_predictions` присоединяется кортеж с индексом образца, изображением, прогнозируемым и ожидаемым значением. Следующий фрагмент проверяет общее количество ошибочных прогнозов (из 10 000 изображений тестового набора):

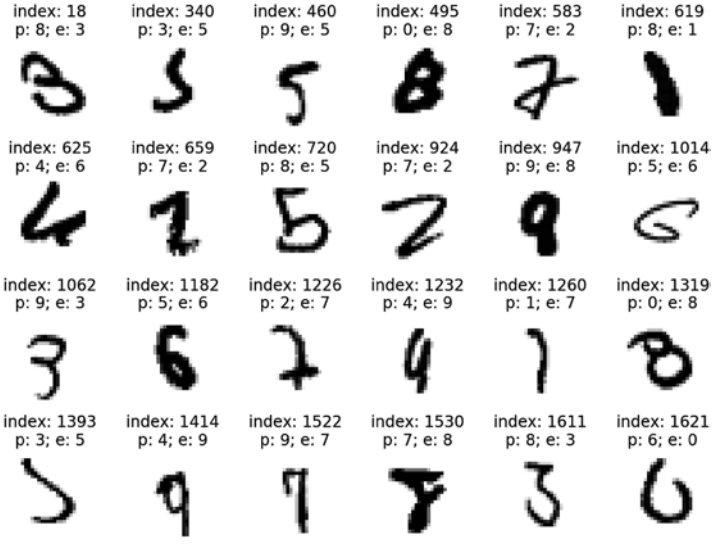
```
[45]: len(incorrect_predictions)
[45]: 83
```

Визуализация ошибочных предсказаний

Следующий фрагмент выводит 24 неправильных изображения с указанием индекса каждого изображения (`index`), прогнозируемого значения (`p`) и ожидаемого значения (`e`):

```
[46]: figure, axes = plt.subplots(nrows=4, ncols=6, figsize=(16, 12))
      for axes, item in zip(axes.ravel(), incorrect_predictions):
          index, image, predicted, expected = item
          axes.imshow(image, cmap=plt.cm.gray_r)
          axes.set_xticks([])  # remove x-axis tick marks
          axes.set_yticks([])  # remove y-axis tick marks
          axes.set_title(
              f'index: {index}\np: {predicted}; e: {expected}')
      plt.tight_layout()
```

Прежде чем просматривать ожидаемые значения, взгляните на каждое изображение и запишите, какой цифре, на ваш взгляд, оно соответствует — это важная составляющая изучения данных:



Вывод вероятностей для нескольких ошибочных прогнозов

Просмотрим вероятности нескольких ошибочных прогнозов. Следующая функция выводит вероятности заданного массива прогнозов:

```
[47]: def display_probabilities(prediction):
          for index, probability in enumerate(prediction):
              print(f'{index}: {probability:.10%}')
```

Хотя изображение 8 (с индексом 495) в первой строке вывода похоже на 8, у нашей модели возникли с ней затруднения. Как видно из следующего вывода, модель спрогнозировала это значение как 0, но также полагала, что с 16-процентной вероятностью это может быть 6, а с 23-процентной вероятностью — 8:

```
[48]: display_probabilities(predictions[495])
0: 59.7235262394%
1: 0.0000015465%
2: 0.8047289215%
3: 0.0001740813%
4: 0.0016636326%
5: 0.0030567855%
6: 16.1390662193%
7: 0.0000001781%
8: 23.3022540808%
9: 0.0255270657%
```

Изображение 2 (с индексом 583) в первой строке спрогнозировано как 7 с вероятностью 62,7%, но модель также считала, что с 36,4-процентной вероятностью это может быть цифра 2:

```
[49]: display_probabilities(predictions[583])
0: 0.0000003016%
1: 0.0000005715%
2: 36.4056706429%
3: 0.0176281916%
4: 0.0000561930%
5: 0.0000000003%
6: 0.0000000019%
7: 62.7455413342%
8: 0.8310816251%
9: 0.0000114385%
```

Изображение 6 (с индексом 625) в начале второй строки было спрогнозировано как 4, хотя этот прогноз был далек от полной уверенности. В данном случае вероятность 4 (51,6%) была лишь немного выше вероятности 6 (48,38%):

```
[50]: display_probabilities(predictions[625])
0: 0.0008245181%
1: 0.0000041209%
2: 0.0012774357%
3: 0.0000000009%
4: 51.6223073006%
5: 0.0000001779%
6: 48.3754962683%
7: 0.0000000085%
8: 0.0000048182%
9: 0.0000785786%
```

15.6.6. Сохранение и загрузка модели

Обучение моделей нейронных сетей может потребовать значительного времени. После того как вы спроектируете и протестируете подходящую модель, можно сохранить ее состояние. Это позволит вам загрузить ее в будущем для построения новых прогнозов. Иногда модели загружаются и проходят дальнейшее обучение для новых задач. Например, уровни нашей модели уже умеют распознавать такие признаки, как линии и кривые, которые также могут пригодиться для распознавания рукописных символов (как в наборе данных EMNIST). Таким образом, теоретически вы можете загрузить существующую модель и использовать ее как основу для построения более мощной модели.

Этот процесс называется *переносом обучения*[1,2] — знания существующей модели переносятся в новую модель. *Метод* `save` моделей Keras сохраняет архитектуру модели и информацию состояния в формате *HDF5* (*Hierarchical Data Format*). Такие файлы по умолчанию используют расширение .h5:

```
[51]: cnn.save('mnist_cnn.h5')
```

Сохраненную модель можно загрузить *функцией* `load_model` из модуля `tensorflow.keras.models`:

```
from tensorflow.keras.models import load_model
cnn = load_model('mnist_cnn.h5')
```

После этого вы сможете вызывать методы модели. Например, при появлении дополнительных данных можно вызвать метод `predict` для построения новых прогнозов для новых данных или же вызвать `fit`, чтобы начать обучение с дополнительными данными.

Keras предоставляет ряд дополнительных функций для сохранения и загрузки отдельных аспектов ваших моделей. За подробной информацией обращайтесь по адресу:

https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model

15.7. Визуализация процесса обучения нейронной сети в TensorBoard

Сети глубокого обучения невероятно сложны, а их внутренняя работа в основном остается скрытой, и разработчику трудно понять все происходящее. Это создает проблемы с тестированием, отладкой и обновлением моделей и алгоритмов. Глубокое обучение выявляет признаки, число которых может быть колоссальным и которые могут быть вам совершенно не очевидны.

Компания Google предоставляет *TensorBoard*[3,4] — инструмент для визуализации нейронных сетей, реализованный с использованием TensorFlow и Keras.

---

[1] https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751.

[2] https://medium.com/nanonets/nanonets-how-to-use-deep-learning-when-you-have-lim-ited-data-f68c0b512cab.

[3] https://github.com/tensorflow/tensorboard/blob/master/README.md.

[4] https://www.tensorflow.org/guide/summaries_and_tensorboard.

Подобно тому как на приборной панели автомобиля отображаются данные с датчиков (скорость, температура двигателя, количество оставшегося бензина), на *панели TensorBoard* отображаются данные модели глубокого обучения, дающие представление о том, как проходит обучение модели. Эти данные, возможно, помогут и в настройке ее гиперпараметров.

Запуск TensorBoard

TensorBoard отслеживает каталог в вашей системе и ищет в нем файлы с данными, которые будут отображаться в браузере. Создадим эту папку, запустим сервер TensorBoard и обратимся к нему из браузера. Выполните следующие действия:

1. Перейдите в каталог `ch15` в терминале, командной оболочке или приглашении Anaconda.
2. Активируйте нестандартную среду Anaconda `tf_env`:

```
conda activate tf_env
```

3. Создайте подкаталог с именем `logs`, в который модели глубокого обучения запишут информацию для визуализации средствами TensorBoard:

```
mkdir logs
```

4. Запустите TensorBoard:

```
tensorboard --logdir=logs
```

5. Теперь к TensorBoard можно обратиться в браузере по адресу:

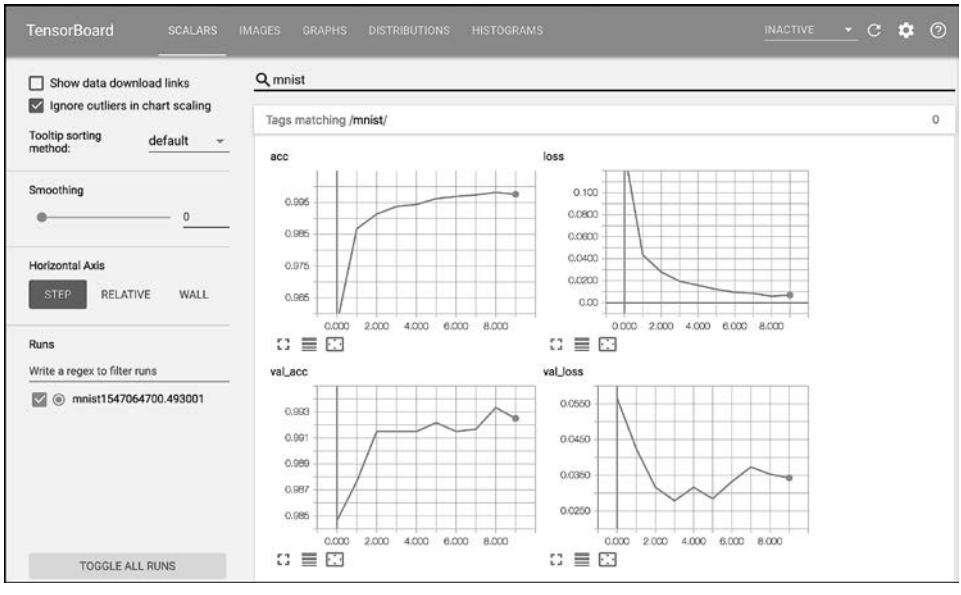
```
http://localhost:6006
```

Если подключиться к TensorBoard перед выполнением какой-либо модели, будет выведена страница с сообщением об отсутствии активных панелей для текущего набора данных[1].

Панель TensorBoard

TensorBoard отслеживает содержимое заданного каталога и файлы, выводимые моделью во время обучения. Обнаружив обновления, TensorBoard загружает данные на панель:

[1] В настоящее время TensorBoard не работает с браузером Microsoft Edge.



Данные можно просматривать как в процессе обучения, так и после его завершения. На панели, изображенной выше, показана вкладка TensorBoard SCALARS с диаграммами отдельных значений, изменяющихся во времени; в первой строке выводится точность обучения (`acc`) и потери обучения (`loss`), во второй — точность проверки (`val_acc`) и потери проверки (`val_loss`). На диаграммах представлена визуализация серии из 10 эпох для нашей сверточной нейронной сети MNIST, представленной в документе Notebook MNIST_CNN_TensorBoard.ipynb. Эпохи выводятся по осям x, начиная с 0 для первой эпохи. Значения точности и потерь выводятся по осям y. По виду графиков точностей обучения и проверки заметно, что результаты первых пяти эпох сходны с результатами серии из пяти эпох в предыдущем разделе.

Для серии из 10 эпох точность обучения продолжала улучшаться до девятой эпохи, а затем слегка снизилась. Возможно, в этой точке началось влияние чрезмерной подгонки модели, но чтобы быть уверенным в этом, необходимо продолжить обучение. Что касается точности проверки, то мы видим, что она сначала быстро возросла, а затем оставалась относительно неизменной на протяжении пяти эпох, после чего вновь выросла и уменьшилась. Потери обучения быстро падали, а затем непрерывно уменьшались до девятой эпохи с последующим незначительным ростом. Потери проверки быстро снижались, а затем попеременно то росли, то вновь снижались. Эту модель можно выпол-

нить с большим количеством эпох, чтобы узнать, улучшит ли это результаты, но по виду диаграмм похоже, что в шестой эпохе достигается хорошее сочетание точности обучения и проверки при минимальных потерях проверки.

Обычно такие диаграммы выстраиваются по вертикали на панели TensorBoard. Мы использовали поле поиска (над диаграммами) для вывода всех каталогов, имя которых содержит «mnist», — вскоре мы настроим этот аспект. TensorBoard может загрузить данные нескольких моделей одновременно, чтобы вы выбрали, какая модель должна визуализироваться. Это позволяет легко сравнить несколько разных моделей или несколько результатов выполнения одной и той же модели.

Копирование документа Notebook для сверточной сети MNIST

Чтобы создать новый документ Notebook в данном примере:

1. Щелкните правой кнопкой мыши на файле MNIST_CNN.ipynb на вкладке File Browser в JupyterLab и выберите команду Duplicate, чтобы создать копию документа.
2. Щелкните правой кнопкой мыши на новом документе с именем MNIST_CNN-Copy1.ipynb, выберите команду Rename, введите имя MNIST_CNN_TensorBoard.ipynb и нажмите Enter.

Откройте документ Notebook, сделав двойной щелчок на его имени.

Настройка Keras для записи журнальных файлов TensorBoard

Чтобы использовать TensorBoard, перед подгонкой модели следует настроить объект `TensorBoard` (модуль `tensorflow.keras.callbacks`), используемый моделью для записи данных в каталог, отслеживаемый TensorBoard. Этот объект называется в Keras *объектом обратного вызова* (callback). В документе Notebook щелкните слева от фрагмента, в котором вызывается метод `fit` модели, и нажмите клавишу «a» для добавления новой кодовой ячейки *над* текущей (клавишу «b» — для добавления ячейки ниже текущей). В новой ячейке введите следующий код для создания объекта `TensorBoard`:

```
from tensorflow.keras.callbacks import TensorBoard
import time

tensorboard_callback = TensorBoard(log_dir=f'./logs/mnist{time.time()}',
    histogram_freq=1, write_graph=True)
```

Аргументы:

- `log_dir` — имя каталога, в который будут записываться журнальные файлы модели. Запись `'./logs/'` означает, что новый каталог создается внутри текущего; далее следует имя `'mnist'` и текущее время. Такая схема выбора имени гарантирует, что при каждом новом выполнении документа Notebook будет создаваться отдельный каталог и вы сможете сравнить результаты нескольких выполнений в TensorBoard.
- `histogram_freq` — частота *в эпохах*, с которой Keras будет выводить данные в журнальные файлы модели. В данном случае данные будут записываться в журналы каждую эпоху.
- `write_graph` — при истинном значении этого аргумента будет выводиться граф модели. Его можно посмотреть на вкладке GRAPHS в TensorBoard.

Обновление вызова fit

Наконец, необходимо изменить исходный вызов метода `fit` во фрагменте 37. Для данного примера назначается количество эпох 10, и мы добавляем аргумент `callbacks` со списком объектов обратного вызова[1]:

```
cnn.fit(X_train, y_train, epochs=10, batch_size=64,
        validation_split=0.1, callbacks=[tensorboard_callback])
```

Теперь вы можете повторно выполнить документ Notebook командой Kernel ▸ Restart Kernel and Run All Cells в JupyterLab. После завершения первой эпохи в TensorBoard начнут появляться данные.

15.8. ConvnetJS: глубокое обучение и визуализация в браузере

В этом разделе рассматривается инструмент ConvnetJS на базе JavaScript, разработанный Андреем Карпати (Andrej Karpathy) и предназначенный для обучения и визуализации сверточных нейронных сетей в браузере[2]:

https://cs.stanford.edu/people/karpathy/convnetjs/

---

[1] Другие объекты обратного вызова Keras можно просмотреть по адресу https://keras.io/callbacks/.

[2] ConvnetJS также можно загрузить из репозитория GitHub по адресу https://github.com/karpathy/convnetjs.

Вы можете запускать готовые примеры сверточных нейронных сетей ConvnetJS или создавать собственные. Мы использовали ConvnetJS в разных браузерах для настольных компьютеров, планшетов и телефонов. Демонстрационное приложение ConvnetJS обучает сверточную нейронную сеть по набору данных MNIST из раздела 15.6. В нем используется панель с поддержкой прокрутки, которая динамически обновляется в процессе обучения модели и состоит из нескольких разделов.

Training Stats (статистика обучения)

Кнопка Pause позволяет приостановить процесс обучения и «зафиксировать» текущие визуализации на панели. Когда вы приостанавливаете демонстрационное приложение, надпись на кнопке заменяется текстом Resume. Повторный щелчок на кнопке продолжает обучение. В этом разделе выводится статистика обучения, включая точность обучения и проверки, а также график потерь обучения.

Instantiate a Network and Trainer (создание сети и обучающего кода)

В этом разделе находится код JavaScript, который создает сверточную нейронную сеть. По умолчанию сеть состоит примерно из тех же уровней, что и сверточная нейронная сеть из раздела 15.6. В документации ConvnetJS[1] описаны поддерживаемые типы уровней и возможности их настройки. Вы можете поэкспериментировать с разными конфигурациями уровней при помощи текстового поля и начать обучение обновленной сети, щелкнув на кнопке Change Network.

Network Visualization (визуализация сети)

В этом ключевом разделе выводится одно обучающее изображение и информация о том, как сеть обрабатывает это изображение на каждом уровне. Щелкните на кнопке Pause, чтобы просмотреть выход всех уровней для заданной цифры, — это поможет получить представление о том, какую информацию «видит» сеть в процессе обучения. Последний уровень сети выдает вероятностные классификации. Для него отображаются 10 квадратов — 9 черных и 1 белый, обозначающие прогнозируемый класс текущего изображения цифры.

---

[1] https://cs.stanford.edu/people/karpathy/convnetjs/docs.html.

Example Predictions on Test Set (примеры прогнозов для тестового набора)

В последнем разделе приводится случайная выборка изображений из тестового набора и три возможных класса с максимальными вероятностями для каждой цифры. Прогноз с наибольшей вероятностью выводится на зеленой полосе, а два других — на красных полосах. Длина каждой полосы является визуальным признаком вероятности этого класса.

15.9. Рекуррентные нейронные сети для последовательностей; анализ эмоциональной окраски с набором данных IMDb

В сверточной нейронной сети MNIST мы сосредоточились на *последовательном* наложении уровней сети. Также возможны модели со структурой, которая не является последовательной, как будет показано в этом разделе на примере *рекуррентных нейронных сетей*. В этом разделе набор данных с отзывами о фильмах IMDb (Internet Movie Database), входящий в поставку Keras[1], используется для выполнения *бинарной классификации*, то есть прогнозирования того, является эмоциональная окраска заданного отзыва положительной или отрицательной.

Мы будем использовать *рекуррентную нейронную сеть* (*RNN*), обрабатывающую последовательности данных, например временные ряды или текст в предложениях. Термин «рекуррентный» возник как реакция на наличие в нейронной сети *циклов*, в которых выход заданного уровня становится входом для того же уровня на следующем *временном шаге*. Во временных рядах временной шаг определяет следующую точку во времени. В текстовой последовательности «временным шагом» является следующее слово в последовательности слов. Наличие циклов в рекуррентных нейронных сетях позволяет им выявлять и запоминать отношения между данными в последовательностях. Для примера возьмем следующие предложения, которые использовались в главе 11. Предложение

```
The food is not good.
```

[1] Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher, "Learning Word Vectors for Sentiment Analysis," *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, июнь 2011 г. — Портленд (Орегон), США. Association for Computational Linguistics, с. 142–150. http://www.aclweb.org/anthology/P11-1015.

очевидно обладает отрицательной эмоциональной окраской. Аналогичным образом предложение

```
The movie was good.
```

обладает положительной эмоциональной окраской, хотя и не настолько положительной, как предложение

```
The movie was excellent!
```

В первом предложении слово «good» само по себе обладает положительной эмоциональной окраской. Однако если ему в предложении *предшествует* слово «not», эмоциональная окраска становится отрицательной. Рекуррентные сети учитывают отношения между более ранними и более поздними частями последовательности. В предыдущем примере слова, определяющие эмоциональную окраску, стоят в соседних позициях. Тем не менее при определении смысла слова иногда приходится учитывать контекст соседних с ним слов, которые могут быть разделены произвольным количеством других слов. В этом разделе мы используем уровень *долгой краткосрочной памяти* (*LSTM*), который делает нейронную сеть *рекуррентной* и оптимизируется для обучения на последовательностях, сходных с описанными выше. Рекуррентные нейронные сети применялись для решения многих задач, включая следующие[1,2,3]:

- прогнозируемый ввод текста — вывод следующих возможных слов во время ввода;
- анализ эмоциональной окраски;
- ответы на вопросы с прогнозируемыми лучшими ответами на базе корпуса текста;
- перевод на другие языки;
- автоматизированное генерирование субтитров в видео.

15.9.1. Загрузка набора данных IMDb

Набор данных с отзывами на фильмы IMDb, поставляемый с Keras, содержит 25 000 обучающих наборов и 25 000 тестовых образцов, каждый из которых помечен эмоциональной окраской: положительной (1) или отрицательной

[1] https://www.analyticsindiamag.com/overview-of-recurrent-neural-networks-and-their-applications/.
[2] https://en.wikipedia.org/wiki/Recurrent_neural_network#Applications.
[3] http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

(0). Импортируем *модуль* `tensorflow.keras.datasets.imdb`, чтобы загрузить набор данных:

```
[1]: from tensorflow.keras.datasets import imdb
```

Функция `load_data` модуля `imdb` возвращает обучающий и тестовый наборы IMDb. Набор данных содержит более 88 000 уникальных слов. Функция `load_data` позволяет задать количество уникальных слов, импортируемых в составе обучающих и тестовых данных. В данном случае были загружены только 10 000 наиболее часто встречающихся слов из-за ограниченного объема памяти в нашей системе и того факта, что мы (намеренно) выполняем обучение на основном процессоре вместо графического процессора (потому что для большинства читателей системы с GPU и TPU недоступны). Чем больше данных вы загрузите, тем больше времени займет обучение, но больший объем данных поможет построить лучшие модели:

```
[2]: number_of_words = 10000

[3]: (X_train, y_train), (X_test, y_test) = imdb.load_data(
         num_words=number_of_words)
```

Функция `load_data` возвращает кортеж из двух элементов, содержащих обучающий и тестовый наборы. Каждый элемент представляет собой кортеж, содержащий образцы и метки соответственно. В каждом отдельном отзыве `load_data` заменяет все слова, кроме 10 000 наиболее часто употребляемых в отзывах, заполнителем (см. ниже).

15.9.2. Исследование данных

Проверим размерность образцов обучающего набора (`X_train`), меток обучающего набора (`y_train`), образцов тестового набора (`X_test`) и меток тестового набора (`y_test`):

```
[4]: X_train.shape
[4]: (25000,)

[5]: y_train.shape
[5]: (25000,)

[6]: X_test.shape
[6]: (25000,)

[7]: y_test.shape
[7]: (25000,)
```

Одномерные массивы `y_train` и `y_test` содержат значения `1` и `0`, показывающие, является каждый отзыв положительным или отрицательным. На основании предшествующего вывода `X_train` и `X_test` также кажутся одномерными. Тем не менее их элементы в действительности представляют собой *списки* целых чисел, каждый из которых представляет содержимое одного отзыва, как показывает фрагмент [9][1]:

```
[8]: %pprint
[8]: Pretty printing has been turned OFF

[9]: X_train[123]
[9]: [1, 307, 5, 1301, 20, 1026, 2511, 87, 2775, 52, 116, 5, 31, 7, 4,
91, 1220, 102, 13, 28, 110, 11, 6, 137, 13, 115, 219, 141, 35, 221, 956,
54, 13, 16, 11, 2714, 61, 322, 423, 12, 38, 76, 59, 1803, 72, 8, 2, 23,
5, 967, 12, 38, 85, 62, 358, 99]
```

Модели глубокого обучения Keras требуют *числовых данных*, поэтому команда Keras заранее обработала набор данных IMDb.

Кодирование отзывов о фильмах

Так как отзывы о фильмах кодируются в числовом виде, для просмотра их исходного текста необходимо знать, какому слову соответствует каждое число. Набор данных IMDb в Keras предоставляет словарь, связывающий слова с их индексами. Значение, соответствующее каждому слову, представляет частоту его вхождения среди всех слов во всем наборе отзывов. Таким образом, слово с частотой 1 является наиболее частым (что было вычислено командой Keras на основании набора данных), слово с частотой 2 — вторым по частоте, и т. д.

Хотя значения в словаре начинаются с 1 (как наиболее часто используемого слова), в кодированных отзывах (таких, как показанное выше значение `X_train[123]`) индексы *смещаются на 3*. Таким образом, в любом отзыве, содержащем самое частое слово, все его вхождения слова заменяются значением 4. Keras резервирует значения 0, 1 и 2 в каждом закодированном отзыве для следующих целей:

- Значение 0 в отзывах представляет *дополнение*. Алгоритмы глубокого обучения Keras ожидают, что все обучающие образцы имеют одинако-

[1] Здесь мы используем магическую команду `%pprint` для отключения «красивой печати», чтобы вывод следующего фрагмента отображался горизонтально, а не вертикально для экономии места. Режим «красивой печати» можно снова включить повторным выполнением магической команды `%pprint`.

вые размеры, поэтому некоторые отзывы иногда приходится расширять до нужной длины, а другие — укорачивать. Отзывы, которые необходимо расширить, дополняются нулями.

- Значение 1 представляет маркер, который используется во внутренней работе Keras для обозначения начала текстовой последовательности для целей обучения.
- Значение 2 в отзыве представляет неизвестное слово — обычно слово, которое не было загружено, потому что функция `load_data` была вызвана с аргументом `num_words`. В этом случае в любом отзыве, содержащем слова с частотным показателем выше `num_words`, числовые значения этих слов заменяются на 2. Все это Keras делает автоматически при загрузке данных.

Так как числовые значения во всех отзывах смещены на 3, нам придется учитывать это обстоятельство при декодировании отзыва.

Декодирование отзывов о фильмах

Попробуем декодировать отзыв. Сначала получим словарь, связывающий слова с индексами, вызовом функции `get_word_index` из модуля `tensorflow.keras.datasets.imdb`:

```
[10]: word_to_index = imdb.get_word_index()
```

Слово `'great'` может присутствовать в положительных отзывах; посмотрим, присутствует ли оно в словаре:

```
[11]: word_to_index['great']
[11]: 84
```

Следовательно, `'great'` является 84-м по частоте словом в наборе данных. При попытке поиска по слову, отсутствующему в словаре, происходит исключение.

Чтобы преобразовать частотные показатели в слова, сначала инвертируем словарь `word_to_index` для поиска слов по их частотным показателям. Следующая трансформация словаря создает словарь с обратным направлением ассоциаций:

```
[12]: index_to_word = \
          {index: word for (word, index) in word_to_index.items()}
```

Напомним, метод `items` словаря используется для перебора кортежей пар «ключ-значение». Каждый кортеж распаковывается в переменные `word` и `index`, после чего в новом словаре создается новый элемент с использованием синтаксиса `index: word`.

Следующая трансформация списка получает 50 самых частых слов из нового словаря — напомним, что самому частому слову соответствует значение 1:

```
[13]: [index_to_word[i] for i in range(1, 51)]
[13]: ['the', 'and', 'a', 'of', 'to', 'is', 'br', 'in', 'it', 'i',
'this', 'that', 'was', 'as', 'for', 'with', 'movie', 'but', 'film', 'on',
'not', 'you', 'are', 'his', 'have', 'he', 'be', 'one', 'all', 'at', 'by',
'an', 'they', 'who', 'so', 'from', 'like', 'her', 'or', 'just', 'about',
"it's", 'out', 'has', 'if', 'some', 'there', 'what', 'good', 'more']
```

Обратите внимание: большинство слов в списке относится к категории *игнорируемых*. Игнорируемые слова можно оставить или удалить — все зависит от конкретного приложения. Например, если вы создаете приложение для прогнозируемого ввода текста, которое предлагает следующее слово в вводимой пользователем серии, то игнорируемые слова стоит оставить, чтобы они выводились среди прогнозируемых слов.

Теперь можно перейти к декодированию отзыва. Для получения значения, связанного с ключом, используется метод `get` с двумя аргументами словаря `index_to_word` вместо оператора `[]`. Если значение отсутствует в словаре, то метод `get` возвращает второй аргумент вместо выдачи исключения. Аргумент `i-3` учитывает существующее смещение в закодированных отзывах частотных показателей каждого отзыва. Когда в отзыве встречаются зарезервированные значения Keras 0–2, `get` возвращает `'?'`; в остальных случаях `get` возвращает слово с ключом `i-3` в словаре `index_to_word`:

```
[14]: ' '.join([index_to_word.get(i - 3, '?') for i in X_train[123]])
[14]: '? beautiful and touching movie rich colors great settings good
      acting and one of the most charming movies i have seen in a while i
      never saw such an interesting setting when i was in china my wife
      liked it so much she asked me to ? on and rate it so other would
      enjoy too'
```

Из массива `y_train` видно, что отзыв классифицируется как положительный:

```
[15]: y_train[123]
[15]: 1
```

15.9.3. Подготовка данных

Количество слов в отзывах непостоянно, а Keras требует, чтобы все образцы имели одинаковые размеры. Следовательно, данные необходимо подготовить. В данном случае необходимо ограничить каждый отзыв *постоянным* количеством слов. Одни обзоры приходится *дополнять* неинформативными данными, другие нуждаются в *усечении*. Вспомогательная функция `pad_sequences` (модуль `tensorflow.keras.preprocessing.sequence`) изменяет размеры образцов `X_train` (то есть строк данных) по количеству признаков, заданному аргументом `maxlen` (`200`), и возвращает двумерный массив:

```
[16]: words_per_review = 200

[17]: from tensorflow.keras.preprocessing.sequence import pad_sequences

[18]: X_train = pad_sequences(X_train, maxlen=words_per_review)
```

Если образец содержит больше признаков, то `pad_sequences` усекает его по заданной длине. Если образец содержит меньше признаков, то `pad_sequences` добавляет 0 в начало последовательности до заданной длины. Проверим новые размеры `X_train`:

```
[19]: X_train.shape
[19]: (25000, 200)
```

Также необходимо изменить размеры `X_test` для оценки модели в будущем:

```
[20]: X_test = pad_sequences(X_test, maxlen=words_per_review)

[21]: X_test.shape
[21]: (25000, 200)
```

Разбиение тестовых данных на проверочные и тестовые данные

В нашей сверточной нейронной сети аргумент `validation_split` метода `fit` используется для обозначения того, что 10% обучающих данных должны быть зарезервированы для проверки модели в процессе обучения. В нашем примере 25 000 тестовых образцов будут вручную разбиты на 20 000 тестовых образцов и 5000 проверочных образцов. Затем 5000 проверочных образцов передаются методу `fit` модели в аргументе `validation_data`. Для разбиения тестового на-

бора мы воспользуемся функцией `train_test_split` библиотеки scikit-learn, упоминавшейся в предыдущей главе:

```
[22]: from sklearn.model_selection import train_test_split
      X_test, X_val, y_test, y_val = train_test_split(
          X_test, y_test, random_state=11, test_size=0.20)
```

Теперь проверим разбиение по размерам `X_test` и `X_val`:

```
[23]: X_test.shape
[23]: (20000, 200)

[24]: X_val.shape
[24]: (5000, 200)
```

15.9.4. Создание нейронной сети

Перейдем к настройке рекуррентной нейронной сети. Вновь начнем с модели `Sequential`, в которую будут добавляться уровни, образующие сеть:

```
[25]: from tensorflow.keras.models import Sequential

[26]: rnn = Sequential()
```

Импортируем уровни, которые будут использоваться в модели:

```
[27]: from tensorflow.keras.layers import Dense, LSTM

[28]: from tensorflow.keras.layers.embeddings import Embedding
```

Добавление уровня векторного представления

Ранее прямое унитарное кодирование использовалось для *преобразования целочисленных* меток набора данных MNIST в *категорийные* данные. Результатом для каждой метки будет вектор, в котором все элементы, кроме одного, равны нулю. То же можно сделать со значениями индексов, представляющими слова. Однако в нашем примере обрабатываются 10 000 слов; это означает, что для представления всех слов понадобится массив 10 000 × 10 000. Получается, что массив содержит 100 000 000 элементов, из которых почти все равны нулю. Такой способ кодирования данных вряд ли можно назвать эффективным. А для обработки всех 88 000+ уникальных слов в наборе данных потребуется массив почти из *8 миллиардов* элементов!

Для *сокращения степени размерности* рекуррентная нейронная сеть, обрабатывающая текстовые последовательности, обычно начинается с *кодирующего уровня*, который преобразует каждое слово в более *компактное представление плотного вектора*. Векторы, производимые уровнем векторного представления, также сохраняют контекст слова, то есть отражают, как конкретное слово связано с соседними словами. Таким образом, кодирующий уровень позволяет рекуррентной сети изучать отношения между словами в обучающих данных.

Существуют и *предварительно определенные векторные представления*, например *Word2Vec* и *GloVe*. Вы можете загрузить их в нейронные сети для экономии времени обучения. Иногда они используются и для включения в модель информации об отношениях между словами при недостаточных объемах обучающих данных. Это может повысить точность модели, позволяя ей пользоваться ранее изученными отношениями между словами, вместо того чтобы пытаться изучать эти отношения при недостаточных объема данных.

Создадим уровень `векторного представления` (модуль `tensorflow.keras.layers`):

```
[29]: rnn.add(Embedding(input_dim=number_of_words, output_dim=128,
                        input_length=words_per_review))
```

Аргументы:

- `input_dim` — количество уникальных слов;
- `output_dim` — размер векторного представления каждого слова. Если вы загружаете готовые векторные представления[1] (такие, как *Word2Vec* и *GloVe*), этому аргументу следует присвоить размер загружаемых векторных представлений слов;
- `input_length=words_per_review` — количество слов в каждом входном образце.

Добавление уровня LSTM

Затем добавим уровень долгой краткосрочной памяти (LSTM):

```
[30]: rnn.add(LSTM(units=128, dropout=0.2, recurrent_dropout=0.2))
```

[1] https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html.

Аргументы:

- `units` — количество нейронов на уровне. Чем больше нейронов, тем больше может запомнить сеть. Простое эмпирическое правило: начните со значения между длиной обрабатываемых последовательностей (200 в данном примере) и количеством классов, которые вы пытаетесь прогнозировать (2 в данном примере[1]).
- `dropout` — процент нейронов, случайным образом блокируемых при обработке входа и выхода уровня. Прореживание, как и объединяющие уровни в сверточных нейронных сетях, — проверенный метод[2,3] для предотвращения чрезмерной подгонки. В Keras существует уровень прореживания `Dropout`, который вы можете добавлять в свои модели.
- `recurrent_dropout` — процент нейронов, случайным образом блокируемых при передаче выхода уровня обратно на уровень, чтобы сеть могла обучаться на известных ей данных.

Механика того, как уровень LSTM выполняет свою задачу, выходит за рамки этой книги. Шолле по этому поводу замечает: «Вам не обязательно понимать все о конкретной архитектуре ячейки LSTM; вы — человек, и это не ваша задача. Просто помните, для чего предназначена ячейка LSTM: предоставить возможность повторного внедрения прошлой информации в будущем»[4].

Добавление уровня уплотненного вывода

Наконец, необходимо взять вывод уровня LSTM и сократить его до одного результата — признака эмоциональной окраски отзыва (положительная или отрицательная); отсюда значение 1 аргумента `units`. Здесь используется *функция активации* `'sigmoid'`, которая считается предпочтительной для бинарной

---

[1] https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046.

[2] Yarin, Ghahramani, and Zoubin. «A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.» 05 октября 2016 г. https://arxiv.org/abs/1512.05287.

[3] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting.» *Journal of Machine Learning Research* 15 (14 июня 2014 г.): 1929–1958. http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

[4] *Шолле Франсуа*. Глубокое обучение на Python. — СПб.: Питер, 2020. С. 238.

классификации[1]. Она сводит произвольные значения до диапазона 0,0–1,0 с получением вероятности:

```
[31]: rnn.add(Dense(units=1, activation='sigmoid'))
```

Компиляция модели и вывод итоги

Затем модель нужно откомпилировать. В данном случае есть всего два возможных вывода, поэтому используется *функция потерь* `binary_crossentropy`:

```
[32]: rnn.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

Ниже приведено сводное описание модели. Хотя рекуррентная нейронная сеть содержит меньше уровней, чем наша сверточная нейронная сеть, она имеет почти втрое больше обучаемых параметров (весов сети), а рост количества параметров означает рост продолжительности обучения. Большое количество параметров возникает прежде всего от умножения количества слов в словаре (мы загрузили 10 000) на количество нейронов в выходе уровня векторного представления (128):

```
[33]: rnn.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 200, 128)          1280000
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               131584
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,411,713
Trainable params: 1,411,713
Non-trainable params: 0
```

15.9.5. Обучение и оценка модели

Проведем обучение модели[2]. Обратите внимание: для каждой следующей эпохи обучение модели занимает намного больше времени. Отчасти это обу-

[1] *Шолле Франсуа*. Глубокое обучение на Python. — СПб.: Питер, 2020. С. 144.

[2] На момент написания книги TensorFlow выводит предупреждение при выполнении этой команды. Это известный дефект TensorFlow; из публикаций на форумах следует, что на предупреждение можно не обращать внимания.

словлено большим количеством параметров (`weights`), на которых должна обучаться модель RNN. Мы выделили жирным шрифтом значения точности (`acc`) и точности проверки (`val_acc`) для удобочитаемости — они представляют процент обучающих образцов и процент образцов `validation_data`, правильно спрогнозированных моделью.

```
[34]: rnn.fit(X_train, y_train, epochs=10, batch_size=32,
              validation_data=(X_test, y_test))
Train on 25000 samples, validate on 5000 samples
Epoch 1/5
25000/25000 [==============================] - 299s 12ms/step - loss:
0.6574 - acc: 0.5868 - val_loss: 0.5582 - val_acc: 0.6964
Epoch 2/5
25000/25000 [==============================] - 298s 12ms/step - loss:
0.4577 - acc: 0.7786 - val_loss: 0.3546 - val_acc: 0.8448
Epoch 3/5
25000/25000 [==============================] - 296s 12ms/step - loss:
0.3277 - acc: 0.8594 - val_loss: 0.3207 - val_acc: 0.8614
Epoch 4/5
25000/25000 [==============================] - 307s 12ms/step - loss:
0.2675 - acc: 0.8864 - val_loss: 0.3056 - val_acc: 0.8700
Epoch 5/5
25000/25000 [==============================] - 310s 12ms/step - loss:
0.2217 - acc: 0.9083 - val_loss: 0.3264 - val_acc: 0.8704
[34]: <tensorflow.python.keras.callbacks.History object at 0xb3ba882e8>
```

Наконец, мы можем оценить результаты по тестовым данным. Функция `evaluate` возвращает значения потерь и точности. В данном случае точность модели составила 85,99%:

```
[35]: results = rnn.evaluate(X_test, y_test)
20000/20000 [==============================] - 42s 2ms/step

[36]: results
[36]: [0.3415240607559681, 0.8599]
```

Заметим, что точность модели кажется низкой по сравнению с результатами сверточной сети MNIST, но ведь и задача перед ней стояла намного сложнее. Поискав в интернете данные других исследований бинарной классификации эмоциональной окраски, вы найдете множество результатов менее 90%. Следовательно, наша маленькая рекуррентная нейронная сеть, состоящая всего из трех уровней, показала вполне неплохие результаты. Тем не менее попробуйте изучить модели в интернете и построить более эффективную модель.

15.10. Настройка моделей глубокого обучения

В разделе 15.9.5 в выводе метода `fit` как точность тестирования (85,99%), так и точность проверки (87,04%) были значительно ниже точности обучения 90,83%. Обычно такие расхождения происходят от чрезмерной подгонки, так что в нашей модели остается достаточно возможностей для усовершенствования[1,2]. Взглянув на вывод каждой эпохи, вы заметите, что точность как обучения, так и проверки продолжает возрастать. Напомним, что обучение на слишком большом количестве эпох может привести к чрезмерной подгонке, но, возможно, в нашем случае обучение все еще было недостаточным. Одной из возможных настроек гиперпараметра для этой модели могло бы стать повышение количества эпох.

Назовем некоторые переменные, влияющие на эффективность модели:

- больший или меньший объем данных для обучения;
- больший или меньший объем данных для тестирования;
- больший или меньший объем данных для проверки;
- большее или меньшее количество уровней;
- типы используемых уровней;
- порядок уровней.

В нашем примере рекуррентной нейронной сети IMDb можно было попытаться настроить следующие параметры:

- разные объемы обучающих данных — мы использовали только 10 000 наиболее часто используемых (в отзывах о фильмах) слов;
- разное количество слов в одном отзыве — мы использовали только 200;
- разное количество нейронов в уровнях;
- большее количество уровней;
- предварительная загрузка заранее обученных векторов вместо информации, которую уровень Embedded сможет извлечь «с нуля».

Время вычислений, необходимое для многократного обучения моделей, значительно, поэтому в глубоком обучении обычно вы не настраиваете ги-

---

[1] https://towardsdatascience.com/deep-learning-overfitting-846bf5b35e24.

[2] https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42.

перпараметры такими методами, как *k*-проходная перекрестная проверка или сеточный поиск[1]. Существует много разных методов оптимизации[2,3,4,5], но одной из самых перспективных областей является автоматизированное машинное обучение (AutoML). Например, библиотека AutoKeras[6] предназначена для автоматизации выбора лучших конфигураций моделей Keras. Среди множества других проектов в области автоматизированного машинного обучения можно выделить Google Cloud AutoML и Baidu EZDL.

15.11. Модели сверточных нейронных сетей с предварительным обучением на ImageNet

При глубоком обучении (вместо того чтобы начинать затратное обучение, проверку и тестирование заново для каждого проекта) вы можете использовать *предварительно обученные модели глубокого обучения для*:

- построения новых прогнозов;
- продолжения обучения на новых данных;
- переноса весов, полученных моделью в ходе обучения для аналогичной задачи, в новую модель — это называется *переносом обучения*.

Предварительно обученные модели сверточных нейронных сетей Keras

Keras поставляется со следующими предварительно обученными моделями сверточных сетей[7], прошедшими обучение на ImageNet[8] — растущем наборе данных из 14+ миллионов изображений:

---

1 https://www.quora.com/Is-cross-validation-heavily-used-in-deep-learning-or-is-it-too-expensive-to-be-used.

2 https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a.

3 https://medium.com/machine-learning-bites/deeplearning-series-deep-neural-networks-tuning-and-optimization-39250ff7786d.

4 https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html и https://flyyufelix.github.io/2016/10/08/fine-tuning-in-keras-part2.html.

5 https://towardsdatascience.com/a-comprehensive-guide-on-how-to-fine-tune-deep-neural-networks-using-keras-on-google-colab-free-daaaa0aced8f.

6 https://autokeras.com/.

7 https://keras.io/applications/.

8 http://www.image-net.org.

- Xception
- VGG16
- VGG19
- ResNet50
- Inception v3
- Inception-ResNet v2
- MobileNet v1
- DenseNet
- NASNet
- MobileNet v2

Повторное использование предварительно обученных моделей

Набор ImageNet слишком велик для эффективного обучения на большинстве компьютеров, поэтому многие разработчики предпочли бы начать с одной из меньших предварительно обученных моделей. Вы можете повторно использовать архитектуру каждой модели и обучить ее на новых данных или же повторно использовать предварительно обученные веса. Несколько простых примеров доступны по адресу:

https://keras.io/applications/

ImageNet Challenge

В завершающих проектах этой главы исследуются и используются некоторые из этих моделей, входящих в поставку Keras. Также будет рассмотрен конкурс *ImageNet Large Scale Visual Recognition Challenge* по оценке обнаружения объектов и моделей распознавания изображений[1]. Этот конкурс проходил с 2010 по 2017 год. В настоящее время ImageNet проводит на сайте Kaggle постоянный конкурс, который называется *ImageNet Object Localization Challenge*[2]. Задача

[1] http://www.image-net.org/challenges/LSVRC/.
[2] https://www.kaggle.com/c/imagenet-object-localization-challenge.

заключается в выявлении «всех объектов в пределах изображения для их классификации и разметки». ImageNet публикует текущий рейтинг лидеров раз в квартал.

Многое из того, что вы видели в главах, посвященных машинному обучению и глубокому обучению, составляет суть конкурса, представленного на сайте Kaggle. Не существует очевидного оптимального решения для многих задач машинного и глубокого обучения. На самом деле единственным ограничением является творческий потенциал разработчиков. На сайте Kaggle компании и организации финансируют конкурсы, в которых разработчики со всего мира ищут более эффективные решения задач, важных для их отрасли или организации. Иногда компании предоставляют призовой фонд, который достигал 1 000 000 долларов в знаменитом конкурсе Netflix. Компания Netflix хотела добиться 10% или более высокого прироста качества в своей модели определения того, понравился ли зрителям фильм, на основании их предшествующих оценок[1]. Результаты использовались для повышения качества рекомендаций. Даже если вы не победите в конкурсе Kaggle, участие в нем будет очень полезно — вы сможете получить практический опыт в работе над актуальными задачами.

15.12. Итоги

Итак, мы заглянули в будущее искусственного интеллекта. Глубокое обучение захватило воображение сообществ компьютерных технологий и data science. Возможно, это была самая важная глава, посвященная искусственному интеллекту.

Мы упомянули ключевые платформы глубокого обучения и выделили среди них Google TensorFlow как наиболее широко используемую. Вы узнали, почему библиотека Keras, предоставляющая удобный интерфейс к TensorFlow, стала настолько популярной.

Затем мы настроили нестандартную среду Anaconda для TensorFlow, Keras и JupyterLab, после чего воспользовались средой для реализации примеров Keras.

---

[1] https://netflixprize.com/rules.html.

Вы узнали, что такое тензоры и почему они играют такую важную роль в глубоком обучении. Были рассмотрены основные концепции нейронов и многоуровневых нейронных сетей для построения моделей глубокого обучения Keras, а также часто используемые разновидности уровней и возможности их упорядочения.

Далее были представлены сверточные нейронные сети, особенно хорошо подходящие для приложений распознавания изображений. Затем мы построили, обучили, проверили на корректность и протестировали сверточную сеть по базе данных изображений рукописных цифр MNIST, для которых была достигнута точность прогнозирования 99,17%. Это выдающийся результат, особенно если учесть, что он был достигнут с использованием базовой модели и без настройки гиперпараметров. Вы можете опробовать более сложные модели и настроить гиперпараметры для улучшения эффективности. В этом разделе были перечислены различные интересные задачи из области распознавания изображений.

Далее мы представили TensorBoard — инструмент визуализации процессов обучения и проверки нейронных сетей Keras и TensorFlow. Также был описан ConvnetJS — инструмент обучения и визуализации сверточных сетей на базе браузера, позволяющий заглянуть «за кулисы» процесса обучения.

После этого мы представили рекуррентные нейронные сети (RNN) для обработки последовательностей данных, таких как временные ряды или текст в предложениях. Мы использовали RNN с набором данных IMDb, содержащим отзывы о фильмах, для выполнения бинарной классификации эмоциональной окраски (то есть прогнозирования того, является каждый отзыв положительным или отрицательным). Была рассмотрена настройка моделей глубокого обучения и то, как высокопроизводительное оборудование (например, графические процессоры NVIDIA или тензорные процессоры Google) позволяет более широкому кругу разработчиков браться за серьезные исследования в области глубокого обучения.

С учетом того, насколько дорогостоящей и продолжительной оказывается процедура обучения моделей глубокого обучения, мы объяснили стратегию использования предварительно обученных моделей. Мы привели список различных моделей распознавания изображений на базе сверточных сетей Keras, которые были обучены на гигантском наборе данных ImageNet, и рассказали, как перенос обучения позволяет быстро и эффективно использовать эти модели для создания новых моделей. Поскольку глубокое обучение — большая и сложная тема, в этой главе мы сосредоточились на его основах.

В следующей главе будет представлена инфраструктура больших данных, поддерживающая различные технологии AI, рассмотренные в главах с 12-й по 15-ю. Будут рассмотрены платформы Hadoop и Spark для пакетной обработки больших данных, а также приложения потоковой передачи данных в реальном времени. Мы представим реляционные базы данных и язык SQL для обработки запросов — эти технологии многие годы занимали ведущее положение в области баз данных. Мы поговорим о проблемах, которые возникают из-за больших данных и плохо решаются реляционными базами данных, и познакомимся с базами данных NoSQL, которые проектировались для решения этих проблем. Книга завершается обсуждением концепции «интернета вещей» (IoT), который наверняка станет крупнейшим мировым источником больших данных и предоставит предпринимателям много возможностей для создания качественно новых видов коммерческой деятельности, способных оказать сильное влияние на человеческую жизнь.

16

Большие данные: Hadoop, Spark, NoSQL и IoT

В этой главе:

- Концепция больших данных и темпы роста.
- Работа с реляционной базой данных SQLite на языке SQL (Structured Query Language).
- Четыре основные разновидности баз данных NoSQL.
- Сохранение твитов в документной базе данных MongoDB и их визуализация на карте Folium.
- Технология Apache Hadoop и ее применение в приложениях пакетной обработки больших данных.
- Построение MapReduce-приложения на базе Hadoop в облачном сервисе Microsoft Azure HDInsight.
- Применение Apache Spark в высокопроизводительных приложениях, работающих с большими данными в реальном времени.
- Использование потоковой передачи Spark для обработки данных в формате мини-пакетов.
- «Интернет вещей» (IoT) и модель публикации/подписки.
- Публикация сообщений с моделируемого устройства, подключенного к интернету, и их визуализация на информационной панели.
- Подписка на «живой» Twitter PubNub и IoT-потоки, и визуализация данных.

16.1. Введение

В разделе 1.7 была представлена концепция больших данных. В этой главе будут рассмотрены популярные аппаратные и программные инфраструктуры для работы с большими данными. Кроме того, мы разработаем полноценные приложения на нескольких настольных и облачных платформах больших данных.

Базы данных

Базы данных (далее — БД, не путать с большими данными) формируют важнейшую инфраструктуру больших данных для хранения и обработки больших объемов создаваемых данных. Они также играют важную роль для безопасного и конфиденциального сопровождения этих данных, особенно в контексте еще более жестких законов защиты данных, таких как *Закон об ответственности и переносе данных о страховании здоровья граждан* (*HIPAA, Health Insurance Portability and Accountability Act*) в США и *Общий регламент защиты данных* (*GDPR, General Data Protection Regulation)* в Евросоюзе.

Начнем с изучения *реляционных БД*, хранящих *структурированные данные* в таблицах. Каждая строка данных в такой таблице состоит из определенного количества столбцов фиксированного размера. Для работы с реляционными БД используется *язык структурированных запросов SQL* (*Structured Query Language*).

Большая часть данных, производимых в наши дни, относится к *неструктурной категории* (например, содержимое постов Facebook, твитов Twitter) или *полуструктурированных данных* (например, документов JSON или XML). Скажем, Twitter обрабатывает содержимое каждого твита в полуструктурированный документ JSON, содержащий большое количество *метаданных* (см. главу 12). Реляционные БД не предназначены для неструктурированных и полуструктурированных данных в приложениях больших данных. По мере развития больших данных создавались новые разновидности БД для эффективной обработки таких данных. Мы рассмотрим четыре основные разновидности *БД NoSQL*: ключ-значение, документные, столбцовые и графовые. Кроме того, будут описаны *БД NewSQL*, сочетающие преимущества реляционных баз данных и баз данных NoSQL. Многие разработчики баз данных NoSQL и NewSQL упрощают освоение своих продуктов за счет предоставления бесплатных уровней и пробных версий, чаще всего в облачных средах, требующих минимальной установки и настройки. Это позволяет получить опыт работы с большими данными, перед тем как серьезно браться за дело.

Apache Hadoop

В наши дни данные нередко настолько велики, что не помещаются в одной системе. С ростом больших данных возникла необходимость в распределенном хранении данных и средствах параллельной обработки для повышения эффективности работы с ними. Это привело к появлению сложных технологий распределенной обработки и массового параллелизма между кластерами компьютеров, в которых система автоматически и корректно реализует все технические подробности, таких как Apache Hadoop. Мы обсудим ниже технологию Hadoop, ее архитектуру и способы применения в приложениях больших данных. Будет описан процесс настройки многоузлового кластера Hadoop с использованием облачного сервиса Microsoft Azure HDInsigh. Затем мы воспользуемся им для выполнения MapReduce-задания, реализованного на Python. Хотя сервис HDInsight не бесплатен, компания Microsoft предоставляет щедрый кредит для новых пользователей, что позволит вам выполнить примеры кода этой главы без лишних затрат.

Apache Spark

С ростом потребностей в обработке больших данных сообщество информационных технологий постоянно искало возможности повышения эффективности. Hadoop выполняет задачи, разбивая их на блоки, выполняющие большой объем операций ввода/вывода между многими компьютерами. Технология Spark была разработана для выполнения некоторых операций больших данных *в памяти* для улучшения быстродействия.

В этой главе будет рассмотрена система Apache Spark, ее архитектура и возможность применения в высокопроизводительных приложениях больших данных в реальном времени. Мы реализуем приложение Spark с использованием средств программирования в функциональном стиле «фильтр/отображение/свертка». Сначала приложение будет построено на базе стека Jupyter Docker, выполняемого локально на вашем настольном компьютере, а затем оно будет реализовано на базе многоузлового кластера Spark в облачном сервисе Microsoft Azure HDInsight. Далее будет представлен механизм потоковой передачи данных Spark для обработки потоковых данных в мини-пакетах. Потоковая передача Spark собирает данные в течение короткого интервала времени, заданного вами, а затем предоставляет собранный пакет данных для обработки. Мы реализуем потоковое приложение Spark для обработки твитов. В этом примере мы используем Spark SQL для запроса данных, хранящихся в коллекции Spark `DataFrame`; в отличие от коллекций `DataFrame` библиотеки

Pandas, они могут содержать данные, распределенные по многим компьютерам в кластере.

Интернет вещей

Глава завершается введением в «интернет вещей» (IoT) — это миллиарды устройств, постоянно производящих данные по всему миру. Будет описана *модель публикации/подписки*, которая используется IoT и другими типами приложений для связывания пользователей данных с поставщиками данных. Сначала без написания кода мы построим информационную панель на базе браузера с использованием Freeboard.io и прямой потоковой передачи от сервиса обмена сообщениями PubNub. Затем мы смоделируем подключенный к интернету термостат, публикующий сообщения в бесплатном сервисе обмена сообщениями Dweet.io, с использованием модуля Python Dweepy, а затем создадим визуализацию данных при помощи Freeboard.io. Наконец, мы построим клиент Python, который *подписывается* на живой поток данных от сервиса PubNub и строит динамическую визуализацию потока средствами Seaborn и `FuncAnimation` библиотеки Matplotlib.

Облачные сервисы и настольные программы для работы с большими данными

Разработчики облачных сервисов ставят на первый план *сервисно-ориентированную архитектуру* (*SOA*), в которой они предоставляют функциональность «как сервис», к которому подключаются приложения и используются в облаке. Вот некоторые типичные сервисы, предоставляемые разработчиками облачных платформ[1]:

Сокращения «как сервис» (обратите внимание на некоторые совпадения)	
Большие данные как сервис (BDaaS)	Платформа как сервис (PaaS)
Hadoop как сервис (HaaS)	Программное обеспечение как сервис (SaaS)
Оборудование как сервис (HaaS)	Пространство для хранения данных как сервис (SaaS)
Инфраструктура как сервис (IaaS)	Spark как сервис (SaaS)

[1] Другие сокращения «как сервис» перечислены в статьях https://en.wikipedia.org/wiki/Cloud_computing и https://en.wikipedia.org/wiki/As_a_service.

Поэкспериментируем в этой главе с несколькими облачными инструментами. В примерах будут задействованы следующие платформы:

- *Бесплатный* облачный кластер MongoDB Atlas.
- Многоузловой кластер Hadoop, работающий в облачном сервисе Microsoft Azure HDInsight, — для этого вы можете воспользоваться *кредитом*, который предоставляется для новой учетной записи Azure.
- *Бесплатный* одноузловой «кластер» Spark, работающий на вашем настольном компьютере с использованием контейнера в технологическом стеке Jupyter Docker.
- Многоузловой кластер Spark, также работающий в Microsoft Azure HDInsight, — для этого вы продолжите пользоваться своим *кредитом* новой учетной записи Azure.

Существует и много других вариантов, включая облачные сервисы от Amazon Web Services, Google Cloud и IBM Watson, бесплатные *настольные* версии платформ Hortonworks и Cloudera (которые также существуют в облачных платных версиях). Кроме того вы можете опробовать одноузловой кластер Spark, работающий на *бесплатной* облачной платформе Databricks Community Edition. Создатели Spark были основателями Databricks.

Во всех случаях проверяйте новейшие условия использования каждого задействованного сервиса. Некоторые сервисы требуют включения расчета по кредитной карте для использования своих кластеров. *Предупреждение:* **после того как вы выделите ресурсы кластеров Microsoft Azure HDInsight (или кластеров других разработчиков), вы начинаете нести расходы. Завершив работу с такими сервисами, как Microsoft Azure, обязательно удалите свой кластер(-ы) и другие ресурсы (например, дисковое пространство). Это продлит срок действия кредита новой учетной записи Azure.**

Процедуры установки и настройки зависят от конкретной платформы и изменяются со временем. Всегда следуйте новейшему описанию от разработчика. Если у вас появятся вопросы, то лучшим источником информации станет техническая поддержка разработчика и форумы. Также проверяйте такие сайты, как stackoverflow.com, — возможно, другие люди задавали вопросы о похожих проблемах и уже получили на них ответы от сообщества разработчиков.

Алгоритмы и данные

Алгоритмы и данные занимают центральное место в программировании Python. Начальные главы этой книги в основном были посвящены алгоритмам. Мы описали управляющие команды и обсуждали разработку алгоритмов. Данные были небольшими — преимущественно целые числа, числа с плавающей точкой и строки.

В главах 5–9 основное внимание уделялось *структурированию* данных в списках, кортежах, словарях, множествах, массивах и файлах.

Смысл данных

Но как насчет *смысла* данных? Можно ли воспользоваться данными для извлечения информации, которая повысит эффективность диагностики рака? Спасения жизней? Улучшения качества жизни пациентов? Борьбы с загрязнением окружающей среды? Экономии воды? Повышения урожайности? Снижения ущерба от опустошительных бурь и пожаров? Создания рабочих мест? Повышения прибыльности работы компании?

Во всех практических примерах data science из глав 11–15 центральное место занимал искусственный интеллект. В этой главе мы сосредоточимся на инфраструктуре больших данных, которая лежит в основе AI-решений. Так как объем данных, используемых с этими технологиями, продолжает экспоненциально расти, мы хотим обучаться на этих данных, причем делать это очень быстро. Для достижения этих целей используется сочетание сложных алгоритмов, оборудования, программного обеспечения и сетевых структур. Мы представили различные методы машинного обучения и убедились в том, что из данных действительно можно извлечь очень ценную информацию. При большем объеме данных, и особенно с большими данными, машинное обучение может стать еще более эффективным.

Источники больших данных

В следующих статьях и на сайтах содержатся ссылки на сотни бесплатных источников больших данных:

ИСТОЧНИКИ БОЛЬШИХ ДАННЫХ

«Awesome-Public-Datasets», GitHub.com, https://github.com/caesar0301/awesome-public-datasets.

«AWS Public Datasets», https://aws.amazon.com/public-datasets/.

«Big Data And AI: 30 Amazing (And Free) Public Data Sources For 2018», by B. Marr, https://www.forbes.com/sites/bernardmarr/2018/02/26/big-data-and-ai-30-amazing-and-free-public-data-sources-for-2018/.

«Datasets for Data Mining and Data Science», http://www.kdnuggets.com/datasets/index.html.

«Exploring Open Data Sets», https://datascience.berkeley.edu/open-data-sets/.

«Free Big Data Sources», Datamics, http://datamics.com/free-big-data-sources/.

Hadoop Illuminated, глава 16. Publicly Available Big Data Sets, http://hadoopilluminated.com/hadoop_illuminated/Public_Bigdata_Sets.html.

«List of Public Data Sources Fit for Machine Learning», https://blog.bigml.com/list-of-public-data-sources-fit-for-machine-learning/.

«Open Data», Wikipedia, https://en.wikipedia.org/wiki/Open_data.

«Open Data 500 Companies», http://www.opendata500.com/us/list/.

«Other Interesting Resources/Big Data and Analytics Educational Resources and Research», B. Marr, http://computing.derby.ac.uk/bigdatares/?page_id=223.

«6 Amazing Sources of Practice Data Sets», https://www.jigsawacademy.com/6-amazing-sources-of-practice-data-sets/.

«20 Big Data Repositories You Should Check Out», M. Krivanek, http://www.datasciencecentral.com/profiles/blogs/20-free-big-data-sources-everyone-should-check-out.

«70+ Websites to Get Large Data Repositories for Free», http://bigdata-madesimple.com/70-websites-to-get-large-data-repositories-for-free/.

«Ten Sources of Free Big Data on Internet», A. Brown, https://www.linkedin.com/pulse/ten-sources-free-big-data-internet-alan-brown.

«Top 20 Open Data Sources», https://www.linkedin.com/pulse/top-20-open-data-sources-zygimantas-jacikevicius.

«We're Setting Data, Code and APIs Free», NASA, https://open.nasa.gov/open-data/.

«Where Can I Find Large Datasets Open to the Public?» Quora, https://www.quora.com/Where-can-I-find-large-datasets-open-to-the-public.

16.2. Реляционные базы данных и язык структурированных запросов (SQL)

Базы данных играют важную роль, особенно при работе с большими данными. В главе 9 была продемонстрирована последовательная обработка текстовых файлов, в том числе при работе с данными из файлов CSV и работе с JSON.

Обе возможности чрезвычайно важны при обработке данных файла *полностью или частично*. С другой стороны, обработка транзакций требует быстрого поиска и, если потребуется. быстрого обновления *отдельных* элементов данных.

База данных представляет собой интегрированную коллекцию данных. *Система управления базами данных* (*СУБД*) предоставляет механизмы хранения и упорядочения данных способом, соответствующим формату БД. СУБД обеспечивает удобный доступ к данным и их хранение, при этом разработчику не приходится отвлекаться на внутреннее представление баз данных.

Реляционные системы управления базами данных (*РСУБД*) хранят данные в *таблицах* и определяют отношения между таблицами. Язык структурированных запросов (SQL) почти повсеместно используется с РСУБД для работы с данными и выполнения *запросов*, извлекающих информацию по заданному критерию. Среди популярных РСУБД с *открытым кодом* выделим SQLite, PostgreSQL, MariaDB и MySQL. Любой желающий может загрузить эти продукты и *свободно* пользоваться ими. Для перечисленных РСУБД существует поддержка Python. Мы будем использовать РСУБД SQLite, входящую в поставку Python. В число популярных коммерческих РСУБД входят Microsoft SQL Server, Oracle, Sybase и IBM Db2.

Таблицы, строки и столбцы

Реляционная БД — логическое табличное представление данных, которое позволяет обращаться к данным без учета их физической структуры. Таблицы состоят из строк, каждая из которых описывает отдельный объект. На следующей диаграмме изображена таблица `Employee`, которая может использоваться в отделе кадров:

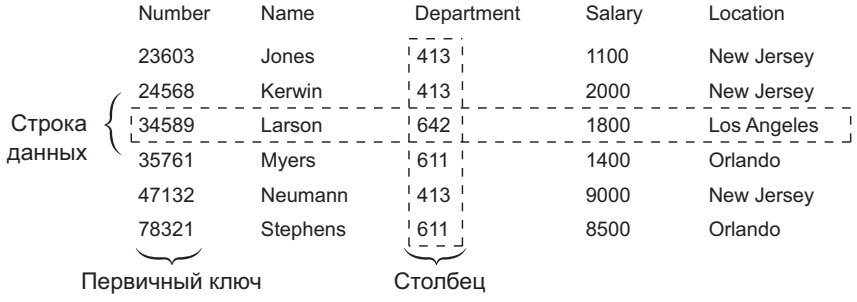


Таблица предназначена для хранения атрибутов работников — каждая ее строка представляет одного работника. Строки состоят из *столбцов*, содержащих значения атрибутов. Каждый столбец представляет отдельный атрибут данных.

Эта таблица состоит из шести строк. Столбец `Number` представляет *первичный ключ* — столбец (группу столбцов) со значением, *уникальным* для каждой строки (см. также ниже). Это гарантирует, что каждая строка однозначно идентифицируется своим первичным ключом. Примеры первичных ключей — номера социального страхования, табельные номера работников и номера деталей в системе складского учета. Значения каждого из этих атрибутов заведомо уникальны. В данном случае строки перечисляются по возрастанию уникального ключа, но они могут перечисляться и по убыванию или вообще без какого-либо определенного порядка. Итак, строки таблицы уникальны (по значению первичного ключа), но значения обычного столбца могут повторяться в разных строках. Например, три разные строки столбца `Department` таблицы `Employee` содержат число 413.

Выборка подмножества данных

Разных пользователей баз данных интересуют разные данные и разные отношения между данными. Многим пользователям достаточно небольшого подмножества строк и столбцов. Запросы указывают, какие подмножества данных должны выбираться из таблицы. Для определения запросов используется язык SQL, который вскоре будет описан более подробно. Например, вы можете выбрать данные из таблицы `Employee` для получения информации о местонахождении филиалов с сортировкой данных в порядке возрастания номеров отделов:

```
Department   Location

413          New Jersey
611          Orlando
642          Los Angeles
```

SQLite

В примерах кода из оставшейся части раздела 16.2 используется СУБД с открытым кодом SQLite, входящая в поставку Python, но в большинстве популярных СУБД реализована поддержка Python. Каждая СУБД обычно предоставляет модуль с поддержкой программного интерфейса Python *Database Application Programming Interface* (*DB-API*), определяющего общие имена объектов и методов для работы с любой БД.

16.2.1. База данных books

В этом разделе представлена БД books, содержащая информацию о нескольких книгах. Мы создадим эту БД в SQLite с использованием *модуля* `sqlite3` стандартной библиотеки Python при помощи сценария, находящегося в подкаталоге sql каталога примеров ch16, перейдя затем к описанию таблиц БД. Эта БД будет использована в сеансе IPython для представления различных концепций баз данных, включая операции *создания*, *чтения*, *обновления* и *удаления* данных (так называемые операции CRUD). При описании таблиц мы воспользуемся SQL и коллекциями `DataFrame` библиотеки Pandas для вывода содержимого каждой таблицы. Затем в нескольких ближайших разделах будут рассмотрены другие возможности SQL.

Создание базы данных books

В приглашении Anaconda, терминале или командной оболочке перейдите в подкаталог sql каталога примеров ch16. Следующая *команда* sqlite3 создает БД SQLite с именем books.db и выполняет сценарий SQL books.sql, который определяет таблицы БД и заполняет их:

```
sqlite3 books.db < books.sql
```

Синтаксис < указывает, что содержимое books.sql передается на вход команды `sqlite3`. После завершения команды БД готова к использованию. Создайте новый сеанс IPython.

Подключение к базе данных Python из Python

Для работы с БД из Python сначала вызовите *функцию* `connect` модуля `sqlite3`, чтобы подключиться к БД и подключить объект `Connection`:

```
In [1]: import sqlite3

In [2]: connection = sqlite3.connect('books.db')
```

Таблица authors

База данных состоит из трех таблиц — `authors`, `author_ISBN` и `titles`. Таблица `authors`, содержащая все данные авторов, состоит из трех столбцов:

- `id` — уникальный идентификатор автора. Этот целочисленный столбец определяется как *автоматически увеличиваемый* — для каждой строки, вставляемой в таблицу, SQLite увеличивает значение `id` на 1, чтобы каждая строка гарантированно имела уникальное значение этого столбца. Этот столбец является первичным ключом таблицы;
- `first` — имя автора (строка);
- `last` — фамилия автора (строка).

Просмотр содержимого таблицы authors

Используем запрос SQL с pandas для просмотра содержимого таблицы `authors`:

```
In [3]: import pandas as pd

In [4]: pd.options.display.max_columns = 10

In [5]: pd.read_sql('SELECT * FROM authors', connection,
   ...:             index_col=['id'])
   ...:
Out[5]:
        first    last
id
1        Paul  Deitel
2      Harvey  Deitel
3       Abbey  Deitel
4         Dan   Quirk
5   Alexander    Wald
```

Функция Pandas `read_sql` выполняет запрос SQL и возвращает коллекцию `DataFrame` с результатами запроса. Аргументы функции:

- строка, представляющая выполняемый запрос SQL;
- объект `Connection` базы данных SQLite;
- ключевой аргумент `index_col`, указывающий, какой столбец должен использоваться в качестве индекса строк `DataFrame` (столбец `id` в данном случае).

Как вы вскоре увидите, если аргумент `index_col` не указан, то слева от строк `DataFrame` выводятся значения индексов, начинающиеся с 0.

Запрос SQL `SELECT` получает строки и столбцы из одной и более таблиц в БД. В запросе:

```
SELECT * FROM authors
```

звездочка (`*`) представляет собой *универсальный символ*, показывающий, что запрос должен получить *все* столбцы из таблицы `authors` (о запросах `SELECT` см. ниже).

Таблица titles

Таблица `titles` содержит данные всех книг. Она состоит из четырех столбцов:

- `isbn` — код ISBN книги (строка) является первичным ключом таблицы. ISBN — сокращение от «International Standard Book Number», то есть «международный стандартный номер книги» — схема нумерации, которая позволяет издателю присвоить каждой книге уникальный идентификационный номер;
- `title` — название книги (строка);
- `edition` — номер издания книги (целое число);
- `copyright` — год регистрации авторского права книги (строка).

Воспользуемся SQL и pandas для просмотра содержимого таблицы `titles`:

```
In [6]: pd.read_sql('SELECT * FROM titles', connection)
Out[6]:
         isbn                              title  edition copyright
0  0135404673     Intro to Python for CS and DS        1      2020
1  0132151006     Internet & WWW How to Program        5      2012
2  0134743350               Java How to Program       11      2018
3  0133976890                  C How to Program        8      2016
4  0133406954  Visual Basic 2012 How to Program        6      2014
```

```
5  0134601548          Visual C# How to Program        6      2017
6  0136151574         Visual C++ How to Program        2      2008
7  0134448235               C++ How to Program       10      2017
8  0134444302           Android How to Program        3      2017
9  0134289366         Android 6 for Programmers       3      2016
```

Таблица author_ISBN

Таблица `author_ISBN` использует следующие столбцы для связывания авторов из таблицы `authors` с книгами из таблицы `titles`:

- `id` — идентификатор автора (целое число).
- `isbn` — код ISBN книги (строка).

Столбец `id` является *внешним ключом* — столбцом таблицы, который совпадает со столбцом *первичного ключа* другой таблицы, а именно столбцом `id` таблицы `authors`. Столбец `isbn` тоже является внешним ключом — он совпадает со столбцом первичного ключа `isbn` таблицы `titles`. База данных может содержать много таблиц. При проектировании БД проектировщик стремится *минимизировать дублирование* данных между таблицами. Для этого каждая таблица должна иметь конкретное наполнение, а внешние ключи — использоваться для связывания данных в *нескольких* таблицах. Первичные и внешние ключи назначаются при создании таблиц БД (в нашем случае в сценарии books.sql).

Комбинация столбцов `id` и `isbn` таблицы образует *составной первичный ключ*. Каждая ее строка *однозначно* связывает одного автора с кодом ISBN *одной* книги. Таблица содержит много записей, поэтому воспользуемся SQL и pandas для просмотра первых пяти строк:

```
In [7]: df = pd.read_sql('SELECT * FROM author_ISBN', connection)

In [8]: df.head()
Out[8]:
   id        isbn
0   1  0134289366
1   2  0134289366
2   5  0134289366
3   1  0135404673
4   2  0135404673
```

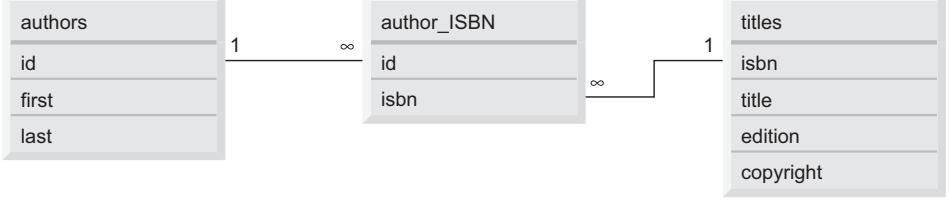
Каждое значение внешнего ключа должно присутствовать в строке другой таблицы как значение первичного ключа, чтобы СУБД могла проверить действительность значения внешнего ключа — это называется *правилом ссылочной целостности*. В частности, СУБД проверяет действительность значения

`id` конкретной строки `author_ISBN`, убеждаясь в том, что в таблице `authors` существует строка с таким значением первичного ключа.

Внешние ключи также позволяют выбирать *взаимосвязанные* данные из *нескольких* таблиц и *объединять* их. Между первичным ключом и соответствующим внешним ключом существует *отношение «один ко многим»* — один автор может написать много книг, хотя одна книга может быть написана несколькими авторами. Таким образом, внешний ключ может встречаться в своей таблице *многократно*, но только *один раз* (как первичный ключ) в другой таблице. Например, в БД `books` код ISBN `0134289366` встречается в нескольких строках таблицы `author_ISBN`, потому что книга имеет несколько авторов, но в `titles` встречается только один раз в качестве первичного ключа.

Диаграмма сущностей и связей (ER)

На следующей *диаграмме сущностей и связей* (*ER-диаграмма, Entity-Relationship*) БД `books` показаны *таблицы* базы данных и *отношения* между ними:



В первом разделе каждого блока указано имя таблицы, в остальных — имена ее столбцов. Курсивом выделены первичные ключи. *Первичный ключ таблицы однозначно идентифицирует каждую строку таблицы*. Каждая строка должна иметь определенное значение первичного ключа, и это значение должно быть уникальным в таблице — в этом заключается *правило ссылочной целостности*. И снова для таблицы `author_ISBN` первичный ключ образуется комбинацией обоих столбцов (составной первичный ключ).

Линии, соединяющие таблицы, представляют *отношения* между таблицами. Возьмем линию между таблицами `authors` и `author_ISBN`. На стороне `authors` стоит обозначение 1, а на стороне `author_ISBN` — знак бесконечности (∞). Так обозначаются *отношения* «*один ко многим*». Для *каждого* автора из таблицы `authors` может существовать *произвольное количество* кодов ISBN для книг, написанных этим автором, в таблице `author_ISBN`, то есть автор может написать

любое количество книг, а значение `id` автора может встречаться в нескольких строках таблицы `author_ISBN`. Линия отношения связывает столбец `id` в таблице `authors` (для которой `id` является первичным ключом) со столбцом `id` таблицы `author_ISBN` (для которой `id` является внешним ключом). Линия между таблицами соединяет первичный ключ с соответствующим внешним ключом.

Линия между таблицами `titles` и `author_ISBN` представляет *отношение «один ко многим»* — одна книга может быть написана многими авторами. Линия связывает первичный ключ `isbn` в таблице `titles` с соответствующим внешним ключом в таблице `author_ISBN`. Отношения на диаграмме сущностей и связей демонстрируют, что единственным предназначением таблицы `author_ISBN` является формирование *отношения «многие ко многим»* между таблицами `authors` и `titles` — автор может написать *много* книг, а книга может иметь *много* авторов.

Ключевые слова SQL

В следующем подразделе продолжится ваше знакомство с SQL в контексте БД `books`. Запросы SQL и команды, использующие ключевые слова SQL, описаны в табл. 16.1. Другие ключевые слова SQL выходят за рамки настоящего описания.

Таблица 16.1. Запросы SQL и команды, использующие ключевые слова SQL

Ключевое слово SQL	Описание
`SELECT`	Выбирает данные из одной или нескольких таблиц
`FROM`	Таблицы, задействованные в запросе (должно присутствовать в каждой команде `SELECT`)
`WHERE`	Критерий выбора, определяющий строки данных для операции выборки, удаления или обновления (может отсутствовать в команде SQL)
`GROUP BY`	Критерий группировки строк (может отсутствовать в команде SQL)
`ORDER BY`	Критерий упорядочения строк (может отсутствовать в команде SQL)
`INNER JOIN`	Слияние строк из нескольких таблиц
`INSERT`	Вставка строк в заданную таблицу
`UPDATE`	Обновление строк в заданной таблице
`DELETE`	Удаление строк из заданной таблицы

16.2.2. Запросы SELECT

Предыдущий раздел использует команды `SELECT` и универсальный символ * для получения всех столбцов из таблицы. Обычно программиста интересует только подмножество столбцов, особенно в больших данных, которые могут содержать десятки, сотни, тысячи и более столбцов. Чтобы извлечь отдельные столбцы, укажите список имен столбцов, разделенных запятыми. Допустим, вы хотите извлечь из таблицы `authors` только столбцы `first` и `last`:

```
In [9]: pd.read_sql('SELECT first, last FROM authors', connection)
Out[9]:
       first    last
0       Paul  Deitel
1     Harvey  Deitel
2      Abbey  Deitel
3        Dan   Quirk
4  Alexander    Wald
```

16.2.3. Секция WHERE

Нередко из БД выбирается часть строк данных, удовлетворяющих определенным *критериям*, особенно в области больших данных, в которых БД может содержать миллионы и даже миллиарды строк. Выбираются только строки, удовлетворяющие критерию выбора (который формально называется *предикатом*). *Условие* `WHERE` определяет критерий выбора. Выберем поля `title`, `edition` и `copyright` для всех книг, у которых год регистрации авторского права (`copyright`) больше `2016`. Строковые значения в запросах SQL заключаются в одинарные апострофы (`'`) — например, `'2016'`:

```
In [10]: pd.read_sql("""SELECT title, edition, copyright
    ...:                FROM titles
    ...:                WHERE copyright > '2016'""", connection)
Out[10]:
                          title  edition copyright
0  Intro to Python for CS and DS        1      2020
1            Java How to Program       11      2018
2       Visual C# How to Program        6      2017
3             C++ How to Program       10      2017
4         Android How to Program        3      2017
```

Поиск по шаблону: нуль и более символов

Условие `WHERE` может содержать операторы `<`, `>`, `<=`, `>=`, `=`, `<>` (не равно) и `LIKE`. Оператор `LIKE` используется для *поиска по шаблону* — поиска строк, удовлет-

воряющих заданному условию. Шаблон, содержащий знак % (*процент*), ищет все строки, содержащие *0 и более* символов в позиции символа %. Например, найдем всех авторов, у которых фамилия начинается с буквы D:

```
In [11]: pd.read_sql("""SELECT id, first, last
    ...:                FROM authors
    ...:                WHERE last LIKE 'D%'""",
    ...:             connection, index_col=['id'])
    ...:
Out[11]:
     first    last
id
1     Paul  Deitel
2   Harvey  Deitel
3    Abbey  Deitel
```

Поиск по шаблону: любой символ

Символ подчеркивания (_) в строке-шаблоне обозначает один символ в указанной позиции. Выберем строки всех авторов, имена которых начинаются с произвольного символа, за которым следует буква b, после чего следует любое количество дополнительных символов (обозначаемое символом %):

```
In [12]: pd.read_sql("""SELECT id, first, last
    ...:                FROM authors
    ...:                WHERE first LIKE '_b%'""",
    ...:             connection, index_col=['id'])
    ...:
Out[12]:
    first    last
id
3   Abbey  Deitel
```

16.2.4. Условие ORDER BY

Условие ORDER BY сортирует результаты запроса по возрастанию (от меньших значений к большим) или по убыванию (от больших значений к меньшим); способ сортировки задается ключевым словом ASC или DESC соответственно. По умолчанию используется сортировка по возрастанию, так что ключевое слово ASC необязательно. Отсортируем названия книг по возрастанию:

```
In [13]: pd.read_sql('SELECT title FROM titles ORDER BY title ASC',
    ...:             connection)
Out[13]:
```

```
                            title
0         Android 6 for Programmers
1            Android How to Program
2                  C How to Program
3                C++ How to Program
4     Internet & WWW How to Program
5     Intro to Python for CS and DS
6                Java How to Program
7  Visual Basic 2012 How to Program
8           Visual C# How to Program
9         Visual C++ How to Program
```

Сортировка по нескольким столбцам

Чтобы отсортировать данные по нескольким столбцам, укажите разделенный запятыми список столбцов после ключевых слов `ORDER BY`. Отсортируем таблицу `authors` по фамилии, а затем по имени для авторов с одинаковыми фамилиями:

```
In [14]: pd.read_sql("""SELECT id, first, last
    ...:                FROM authors
    ...:                ORDER BY last, first""",
    ...:             connection, index_col=['id'])
    ...:
Out[14]:
        first    last
id
3       Abbey  Deitel
2      Harvey  Deitel
1        Paul  Deitel
4         Dan   Quirk
5   Alexander    Wald
```

Порядок сортировки может изменяться на уровне отдельных столбцов. Отсортируем данные `authors` по убыванию фамилии и по возрастанию имени для авторов с одинаковыми фамилиями:

```
In [15]: pd.read_sql("""SELECT id, first, last
    ...:                FROM authors
    ...:                ORDER BY last DESC, first ASC""",
    ...:             connection, index_col=['id'])
    ...:
Out[15]:
        first    last
id
5   Alexander    Wald
```

```
4         Dan   Quirk
3       Abbey  Deitel
2      Harvey  Deitel
1        Paul  Deitel
```

Объединение условий WHERE и ORDER BY

Условия `WHERE` и `ORDER BY` могут объединяться в одном запросе. Получим значения `isbn`, `title`, `edition` и `copyright` для каждой книги из таблицы `titles`, название которой завершается строкой `'How to Program'`, после чего отсортируем их по возрастанию `title`:

```
In [16]: pd.read_sql("""SELECT isbn, title, edition, copyright
    ...:                FROM titles
    ...:                WHERE title LIKE '%How to Program'
    ...:                ORDER BY title""", connection)
Out[16]:
         isbn                              title  edition copyright
0  0134444302            Android How to Program        3      2017
1  0133976890                  C How to Program        8      2016
2  0134448235                C++ How to Program       10      2017
3  0132151006     Internet & WWW How to Program        5      2012
4  0134743350               Java How to Program       11      2018
5  0133406954  Visual Basic 2012 How to Program        6      2014
6  0134601548          Visual C# How to Program        6      2017
7  0136151574         Visual C++ How to Program        2      2008
```

16.2.5. Слияние данных из нескольких таблиц: INNER JOIN

Вспомните, что таблица `author_ISBN` в составе БД `books` связывает авторов с соответствующими названиями книг. Если бы эта информация не была разделена по разным таблицам, то в каждую запись таблицы `titles` пришлось бы включать информацию об авторе. Это привело бы к хранению *повторяющейся* информации об авторах, написавших несколько книг.

Конструкция `INNER JOIN` позволяет объединить данные из нескольких таблиц. Построим список авторов с кодами ISBN книг, написанных каждым автором: запрос возвращает слишком много результатов, поэтому мы приводим только начало вывода:

```
In [17]: pd.read_sql("""SELECT first, last, isbn
    ...:                FROM authors
    ...:                INNER JOIN author_ISBN
```

```
    ...:                    ON authors.id = author_ISBN.id
    ...:                ORDER BY last, first""", connection).head()
Out[17]:
    first    last        isbn
0   Abbey  Deitel  0132151006
1   Abbey  Deitel  0133406954
2  Harvey  Deitel  0134289366
3  Harvey  Deitel  0135404673
4  Harvey  Deitel  0132151006
```

Условие `ON` конструкции `INNER JOIN` использует столбец первичного ключа одной таблицы и столбец внешнего ключа другой таблицы для определения того, какие строки следует объединять из каждой таблицы. Этот запрос объединяет столбцы `first` и `last` таблицы `authors` со столбцом `isbn` таблицы `author_ISBN` и сортирует результаты по возрастанию сначала `last`, а затем `first`.

Обратите внимание на синтаксис `authors.id` (*table_name.column_name*) в условии `ON`. Синтаксис *уточнения имен* необходим в том случае, если столбцы имеют одинаковые имена в обеих таблицах. Этот синтаксис может использоваться в любой команде SQL, для того чтобы различать одноименные столбцы в разных таблицах. В некоторых системах имена таблиц, уточненные именами баз данных, могут использоваться для выполнения запросов к другим базам данных. Как обычно, запрос может содержать условие `ORDER BY`.

16.2.6. Команда INSERT INTO

До настоящего момента мы обращались с запросами на выборку существующих данных. Иногда выполняются команды SQL, которые *изменяют* БД. Для этого мы воспользуемся объектом `sqlite3 Cursor`, для получения которого применим метод `cursor` объекта `Connection`:

```
In [18]: cursor = connection.cursor()
```

Метод `read_sql` библиотеки Pandas использует `Cursor` во внутренней реализации для выполнения запросов и обращения к строкам результатов.

Команда `INSERT INTO` вставляет строку в таблицу. Вставим в таблицу `authors` нового автора `Sue Red`; для этого вызовем метод `execute` объекта `Cursor`, который выполняет свой аргумент SQL и возвращает `Cursor`:

```
In [19]: cursor = cursor.execute("""INSERT INTO authors (first, last)
    ...:                            VALUES ('Sue', 'Red')""")
    ...:
```

За ключевыми словами SQL `INSERT INTO` следует таблица, в которую вставляется новая строка, и разделенный запятыми список имен столбцов в круглых скобках. За списком имен столбцов следует ключевое слово SQL `VALUES` и разделенный запятыми список значений в круглых скобках. Передаваемые значения должны соответствовать заданным именам столбцов по типу и порядку.

Значение столбца `id` не указывается, потому что это автоматически увеличиваемый столбец таблицы `authors` (см. сценарий books.sql, который создавал таблицу). Для каждой новой строки SQLite присваивает уникальное значение `id`, которое является следующим значением в автоматически увеличиваемой последовательности (1, 2, 3 и т. д.). В данном случае Sue Red присваивается идентификатор 6. Чтобы убедиться в этом, создадим запрос на выборку содержимого таблицы `authors`:

```
In [20]: pd.read_sql('SELECT id, first, last FROM authors',
    ...:             connection, index_col=['id'])
    ...:
Out[20]:
        first    last
id
1        Paul  Deitel
2      Harvey  Deitel
3       Abbey  Deitel
4         Dan   Quirk
5   Alexander    Wald
6         Sue     Red
```

Строки, содержащие внутренние одинарные кавычки

В SQL строки заключаются в одинарные кавычки (`'`). Если в строке присутствует внутренняя одинарная кавычка (например, `O'Malley`), то в этой позиции должны стоять *две* одинарные кавычки (`'O''Malley'`). Первая кавычка интерпретируется как служебный символ для экранирования второй. Если вы забудете экранировать символы одинарной кавычки в строке, которая является частью команды SQL, то получите сообщение о синтаксической ошибке SQL.

16.2.7. Команда UPDATE

Команда `UPDATE` обновляет существующие значения. Допустим, фамилия `Red` была введена в БД неправильно, и ее нужно заменить на `'Black'`:

```
In [21]: cursor = cursor.execute("""UPDATE authors SET last='Black'
    ...:                            WHERE last='Red' AND first='Sue'""")
```

За ключевым словом `UPDATE` следует таблица, содержимое которой требуется обновить, ключевое слово `SET` и разделенный запятыми список пар *имя_столбца = значение*, определяющий изменяемые столбцы и их новые значения. Если условие `WHERE` не указано, то изменения будут внесены в каждую строку. Условие `WHERE` в этом запросе указывает, что обновляться должны только те строки, которые содержат фамилию `'Red'` и имя `'Sue'`.

Конечно, в БД может упоминаться несколько людей с одинаковым именем и фамилией. Чтобы внести изменение только в одну строку, лучше использовать в условии `WHERE` уникальный первичный ключ. В данном примере это может выглядеть так:

```
WHERE id = 6
```

Для команд, изменяющих БД, атрибут `rowcount` объекта `Cursor` содержит целочисленное значение, представляющее количество измененных строк. Если значение равно 0, то изменения не вносились. Следующий фрагмент подтверждает, что команда `UPDATE` изменила только одну строку:

```
In [22]: cursor.rowcount
Out[22]: 1
```

Чтобы убедиться в том, что обновление прошло успешно, выведем содержимое таблицы:

```
In [23]: pd.read_sql('SELECT id, first, last FROM authors',
    ...:             connection, index_col=['id'])
    ...:
Out[23]:
        first    last
id
1        Paul  Deitel
2      Harvey  Deitel
3       Abbey  Deitel
4         Dan   Quirk
5   Alexander    Wald
6         Sue   Black
```

16.2.8. Команда DELETE FROM

Команда SQL `DELETE FROM` удаляет строки из таблицы. Удалим строку `Sue Black` из таблицы `authors` по идентификатору:

```
In [24]: cursor = cursor.execute('DELETE FROM authors WHERE id=6')

In [25]: cursor.rowcount
Out[25]: 1
```

Необязательное условие `WHERE` определяет удаляемые строки. Если условие `WHERE` отсутствует, то будут удалены все строки таблицы. Вот как выглядит таблица `authors` после выполнения операции `DELETE`:

```
In [26]: pd.read_sql('SELECT id, first, last FROM authors',
    ...:             connection, index_col=['id'])
    ...:
Out[26]:
        first    last
id
1        Paul  Deitel
2      Harvey  Deitel
3       Abbey  Deitel
4         Dan   Quirk
5   Alexander    Wald
```

Закрытие базы данных

После завершения работы с БД следует вызвать метод `close` объекта `Connection` для отключения:

```
connection.close()
```

SQL в больших данных

Важность SQL возрастает в области больших данных. Позднее в этой главе мы воспользуемся Spark SQL для выборки данных из коллекции Spark `DataFrame`, данные которой могут быть распределены по многим компьютерам в кластере Spark. Как вы вскоре увидите, Spark SQL имеет много общего с языком SQL.

16.3. Базы данных NoSQL и NewSQL: краткое введение

В течение десятилетий реляционные БД считались стандартом в области обработки данных. Однако они требуют *структурированных данных*, хорошо укладывающихся в формат прямоугольных таблиц. С увеличением размера данных, количества таблиц и отношений эффективно управлять подобными базами данных становится намного сложнее. В современном мире больших данных появились БД NoSQL и БД NewSQL, предназначенные для механизмов хранения данных и потребностей в обработке, с которыми не справляются традиционные реляционные БД. Большие данные требуют огромных баз данных, часто распределенных по многим компьютерным центрам по всему миру в огромных кластерах серийных компьютеров. По сведениям statista.com, в настоящее время в мире существует более 8 миллионов центров хранения и обработки данных[1].

Изначально термин NoSQL трактовался буквально, то есть как «без SQL». Но с ростом важности SQL в области больших данных — например, SQL для Hadoop и Spark SQL — принято считать, что термин NoSQL означает «не только SQL». Базы данных NoSQL предназначены для неструктурированных данных (фотографии, видео и тексты на естественном языке в сообщениях электронной почты, текстовых сообщениях и публикациях в социальных сетях), а также полуструктурированных данных вроде документов JSON и XML. Полуструктурированные данные часто представляют собой неструктурированные данные с включением дополнительной информации (*метаданных*). Например, видео YouTube представляет собой неструктурированные данные, но YouTube также хранит для каждого видео ролика метаданные: кто опубликовал ролик, дата публикации, название, описание, теги для упрощения поиска, настройки конфиденциальности и др., — все эти данные возвращаются в данных JSON функциями YouTube API. Метаданные добавляют структуру в неструктурированные видеоданные, в результате чего те становятся полуструктурированными.

В нескольких ближайших подразделах приведен обзор основных разновидностей баз данных NoSQL — базы данных «ключ-значение», документные, столбцовые и графовые. Кроме того, будет приведен обзор баз данных NewSQL, объединяющих функциональность реляционных БД и БД NoSQL. В разделе 16.4 представлен пример, в котором мы будем сохранять и обраба-

[1] https://www.statista.com/statistics/500458/worldwide-datacenter-and-it-sites/.

тывать большое количество объектов твитов в формате JSON в документной БД NoSQL. Затем данные будут отображены в интерактивной визуализации, отображаемой на карте Folium.

16.3.1. Базы данных NoSQL «ключ-значение»

Как и словари Python, *базы данных «ключ-значение»*[1] предназначены для хранения пар «ключ-значение», но они оптимизированы для распределенных систем и обработки больших данных. Для надежности обычно применяется репликация данных по нескольким узлам кластера. Некоторые из них (такие, как БД Redis) по соображениям эффективности реализуются в памяти, другие хранят данные на диске (например, HBase) и работают на базе распределенной файловой системы Hadoop HDFS. Среди популярных баз данных «ключ-значение» можно выделить Amazon DynamoDB, Google Cloud Datastore и Couchbase. БД DynamoDB и БД Couchbase — *многомодельные базы данных*, которые также поддерживают работу с документами. HBase также является столбцовой базой данных.

16.3.2. Документные базы данных NoSQL

В *документной базе данных*[2] хранятся полуструктурированные данные (например, документы JSON и XML). Обычно в документных базах данных создаются индексы для отдельных атрибутов, позволяющие более эффективно находить документы и работать с ними. Допустим, вы храните документы JSON, производимые устройствами IoT, и каждый документ содержит атрибут типа. Вы можете добавить индекс для этого атрибута, чтобы иметь возможность фильтровать документы по типу. Без индексов вы тоже сможете выполнять эту операцию, но она будет выполняться медленнее, потому что БД придется проводить поиск по каждому документу для получения нужного атрибута.

Самая популярная документная БД (и самая популярная база данных NoSQL вообще[3]) — *MongoDB*. Ее название состоит из букв, входящих в слово «humongous» (то есть «огромный»). В этом примере мы сохраним в БД MongoDB большое количество твитов для обработки. Напомним, Twitter

---

[1] https://en.wikipedia.org/wiki/Key-value_database.

[2] https://en.wikipedia.org/wiki/Document-oriented_database.

[3] https://db-engines.com/en/ranking.

API возвращает твиты в формате JSON, что позволяет сохранить их напрямую в БД MongoDB. После получения твитов будет построена их сводка с использованием коллекции `DataFrame` библиотеки Pandas и карты Folium. Другие популярные документные базы данных — Amazon DynamoDB (также является базой данных «ключ-значение»), Microsoft Azure Cosmos DB и Apache CouchDB.

16.3.3. Столбцовые базы данных NoSQL

Одной из типичных операций, выполняемых с использованием РСУБД, является получение значения конкретного столбца для каждой строки. Так как данные упорядочены по строкам, запрос на выборку данных конкретного столбца может выполняться неэффективно. СУБД должна найти каждую подходящую строку, каждый нужный столбец, отбросив остальную информацию, не относящуюся к запросу. *Столбцовая база данных*[1,2], также называемая *столбцово-ориентированной*, похожа на РСУБД, но структурированные данные хранятся по столбцам, а не по строкам. Так как все элементы столбцов хранятся вместе, выборка всех данных заданного столбца будет выполняться более эффективно.

Возьмем таблицу `authors` в базе данных `books`:

```
        first    last
id
1        Paul  Deitel
2      Harvey  Deitel
3       Abbey  Deitel
4         Dan   Quirk
5   Alexander    Wald
```

В РСУБД все данные строки хранятся вместе. Если рассматривать каждую строку как кортеж Python, то строки будут представлены в виде `(1, 'Paul', 'Deitel')`, `(2, 'Harvey', 'Deitel')` и т. д. В столбцовой БД все значения заданного столбца будут храниться вместе: `(1, 2, 3, 4, 5)`, `('Paul', 'Harvey', 'Abbey', 'Dan', 'Alexander')` и `('Deitel', 'Deitel', 'Deitel', 'Quirk', 'Wald')`. Элементы каждого столбца хранятся в порядке строк, так что значение с заданным индексом в каждом столбце принадлежат одной строке. Самые популярные столбцовые базы данных — MariaDB ColumnStore и HBase.

---

[1] https://en.wikipedia.org/wiki/Columnar_database.

[2] https://www.predictiveanalyticstoday.com/top-wide-columnar-store-databases/.

16.3.4. Графовые базы данных NoSQL

Для знакомства с графовой БД NoSQL важно следующее. Граф моделирует отношения (связи) между объектами[1]. Объекты называются *узлами* (или *вершинами*), отношения — *ребрами*. Ребра обладают *направленностью*: так, ребро, представляющее маршрут полета, направлено от места вылета к месту посадки, но не наоборот. Таким образом, *графовая БД*[2] содержит узлы, ребра и их атрибуты.

Если вы пользуетесь социальными сетями — Instagram, Snapchat, Twitter или Facebook, — то представьте свой *социальный граф*, состоящий из людей, с которыми вы знакомы (узлы), и отношений между ними (ребра). Каждый человек обладает собственным социальным графом, и эти графы взаимосвязаны. Знаменитая задача «шести рукопожатий» гласит, что любые два человека в мире соединены друг с другом не более чем шестью ребрами во всемирном социальном графе[3]. Алгоритмы Facebook используют социальные графы своих миллиардов активных подписчиков[4] для определения того, какие истории должны включаться в ежедневную сводку новостей каждого пользователя. Анализируя ваши интересы, ваших друзей, их интересы и т. д., Facebook определяет истории, которые должны вас заинтересовать[5].

Многие компании используют аналогичные методы для создания рекомендательных систем. Когда вы просматриваете описание товара на Amazon, то Amazon использует граф пользователей и товаров для отображения похожих товаров, которые просматривались другими пользователями перед покупкой. При просмотре фильмов в Netflix сервис на основании графа пользователей и фильмов, которые им понравились, рекомендует фильмы, которые могут понравиться вам.

Одна из самых популярных графовых БД — Neo4j (рекомендуем прочитать бесплатную книгу Neo4j в формате PDF «Graph Databases»[6]). Многие реальные примеры применения графовых баз данных доступны по адресу:

https://neo4j.com/graphgists/

---

1 https://en.wikipedia.org/wiki/Graph_theory.
2 https://en.wikipedia.org/wiki/Graph_database.
3 https://en.wikipedia.org/wiki/Six_degrees_of_separation.
4 https://zephoria.com/top-15-valuable-facebook-statistics/.
5 https://newsroom.fb.com/news/2018/05/inside-feed-news-feed-ranking/.
6 https://neo4j.com/graph-databases-book-sx2.

Для большинства практических примеров приводятся диаграммы графов, построенные с использованием Neo4j. На них наглядно представлены отношения между узлами графа.

16.3.5. Базы данных NewSQL

Среди основных преимуществ реляционных баз данных обычно указываются безопасность и поддержка транзакций. В частности, реляционные БД обычно используют транзакции, характеристики которых обозначаются сокращением *ACID* (*Atomicity, Consistency, Isolation, Durability*)[1]:

- *Атомарность* (Atomicity) гарантирует, что БД изменяется только в том случае, если *все* составляющие транзакции прошли успешно. Если вы намереваетесь снять в банкомате 100 долларов, то эти деньги будут выданы только в случае, если у вас на счете хватает средств, а в банкомате — денег для выполнения транзакции.
- *Целостность* (Consistency) гарантирует, что БД всегда находится в корректном состоянии. В примере со снятием средств с банкомата новый баланс вашего счета будет в точности соответствовать сумме, снятой со счета (и, возможно, комиссионных платежей).
- *Изолированность* (Isolation) гарантирует, что параллельные транзакции выполняются так, как если бы они выполнялись последовательно. Например, если два человека совместно используют один банковский счет и пытаются одновременно снять с него деньги, то одна транзакция должна дождаться завершения другой.
- *Долговечность* (Durability) гарантирует, что БД сохраняет функциональность и при сбоях оборудования.

Если вы проанализируете достоинства и недостатки баз данных NoSQL, то увидите, что они обычно не предоставляют поддержку ACID. Для приложений, использующих базы данных NoSQL, обычно не нужны гарантии, предоставляемые ACID-совместимыми базами данных. Большинство баз данных NoSQL обычно придерживается модели *BASE* (*Basic Availability, Softstate, Eventual consistency* — «*базовая доступность, гибкое состояние, согласованность в конечном счете*»), которая в большей степени ориентируется на доступность базы данных. Если базы данных ACID гарантируют целостность состояния

[1] https://en.wikipedia.org/wiki/ACID_(computer_science).

при записи в базу данных, то базы данных BASE обеспечивают целостность в какой-то момент в будущем.

Базы данных NewSQL объединяют преимущества как реляционных баз данных, так и баз данных NoSQL для задач обработки больших данных. Популярные базы данных NewSQL — VoltDB, MemSQL, Apache Ignite и Google Spanner.

16.4. Практический пример: документная база данных MongoDB

MongoDB — документная БД, предоставляющая возможность хранения и загрузки документов в формате JSON. Twitter API возвращает твиты в виде объектов JSON, которые могут записываться напрямую в БД MongoDB. В этом разделе мы:

- используем Tweepy для потоковой выдачи твитов о 100 сенаторах США и сохранения их в базе данных MongoDB;
- используем Pandas для обобщения информации о 10 самых популярных сенаторах по их активности в Twitter;
- используем интерактивную карту Folium Соединенных Штатов с одной меткой на каждый штат. На метке выводится название штата и имена сенаторов, их политические партии и счетчики твитов.

Мы используем бесплатный облачный кластер MongoDB Atlas, не требующий установки и в настоящее время позволяющий хранить до 512 Мб данных. Чтобы хранить больший объем данных, загрузите MongoDB Community Server по адресу:

```
https://www.mongodb.com/download-center/community
```

и запустите его локально либо оформите подписку на платный сервис MongoDB Atlas.

Установка библиотек Python, необходимых для взаимодействия с MongoDB

Для взаимодействия с базами данных MongoDB из кода Python воспользуемся *библиотекой* `pymongo`. Также нам понадобится библиотека `dnspython` для подключения к кластеру MongoDB Atlas. Для установки введите команды:

```
conda install -c conda-forge pymongo
conda install -c conda-forge dnspython
```

keys.py

В подкаталоге TwitterMongoDB каталога ch16 находится код примера и файл keys.py. Отредактируйте файл, включите в него свои регистрационные данные Twitter и ключ OpenMapQuest из главы 12. После того как мы обсудим создание кластера MongoDB Atlas, вам также нужно будет добавить в файл строку подключения MongoDB.

16.4.1. Создание кластера MongoDB Atlas

Чтобы создать бесплатную учетную запись, откройте страницу:

https://mongodb.com,

введите свой адрес электронной почты и щелкните на ссылке Get started free. На следующей странице введите свое имя, создайте пароль и прочитайте условия обслуживания. Если вы с ними согласны, то щелкните на ссылке Get started free на этой же странице, и вы попадете на экран настройки своего кластера. Щелкните на ссылке Build my first cluster, чтобы перейти к созданию кластера.

Сервис проведет вас по серии «первых шагов» со всплывающими подсказками, которые описывают и указывают на каждую задачу, которую необходимо завершить. Для бесплатного кластера Atlas (обозначен как M0) предоставляются настройки по умолчанию; просто введите имя кластера в разделе Cluster Name и щелкните на ссылке Create Cluster. Открывается страница Clusters и начинается создание нового кластера, что может занять несколько минут.

Затем открывается учебное руководство Atlas со списком дополнительных действий, которые необходимо выполнить до начала работы:

- *Создание первого пользователя базы данных* — позволяет вам подключиться к кластеру.
- *Включение IP-адреса в «белый список»* — мера безопасности, гарантирующая, что с кластером смогут работать только проверенные вами IP-адреса. Для подключения к этому кластеру из разных мест (дом, работа, учебное учреждение и т. д.) необходимо включить в «белый список» все IP-адреса, с которых вы собираетесь создавать подключение.

- *Подключение к кластеру* — на этом шаге задается строка подключения к вашему кластеру, чтобы ваш код Python смог подключиться к серверу.

Создание первого пользователя базы данных

Во временном окне обучающего руководства щелкните на ссылке Create your first database user, чтобы продолжить работу. Выполняйте подсказки, перейдите на вкладку Security и щелкните на кнопке +ADD NEW USER. В диалоговом окне Add New User создайте имя пользователя и пароль. Запишите свои регистрационные данные — вскоре они вам понадобятся. Щелкните на кнопке Add User, чтобы вернуться к руководству Connect to Atlas.

Включение IP-адреса в «белый список»

Во временном окне обучающего руководства щелкните на ссылке Whitelist your IP address для продолжения работы. Выполняйте подсказки, перейдите на вкладку IP Whitelist и щелкните на кнопке +ADD IP ADDRESS. В диалоговом окне Add Whitelist Entry вы можете либо добавить текущий IP-адрес своего компьютера, либо разрешить доступ с любого адреса; второй вариант не рекомендован для реальной эксплуатации баз данных, но для учебных целей этого достаточно. Щелкните на кнопке ALLOW ACCESS FROM ANYWHERE, а затем на кнопке Confirm, чтобы вернуться к обучающему руководству Connect to Atlas.

Подключение к кластеру

Во временном окне обучающего руководства щелкните на ссылке Connect to your cluster для продолжения работы. Выполняйте подсказки, чтобы открыть диалоговое окно Connect to *ИмяВашегоКластера*. Для подключения к БД MongoDB Atlas из Python необходима строка подключения. Чтобы получить строку подключения, щелкните на ссылке Connect Your Application, а затем на ссылке Short SRV connection string. Ваша строка подключения появится внизу под полем Copy the SRV address. Щелкните на кнопке COPY, чтобы скопировать строку. Вставьте строку в файл keys.py как значение переменной `mongo_connection_string`. Замените `"<PASSWORD>"` в строке подключения вашим паролем, после чего замените имя базы данных `"test"` на `"senators"` (имя базы данных в данном примере). Щелкните на кнопке Close в нижней части окна Connect to *ИмяВашегоКластера*. Теперь все готово для взаимодействия с кластером Atlas.

16.4.2. Потоковая передача твитов в MongoDB

Сначала будет представлен интерактивный сеанс IPython, который подключается к БД MongoDB, загружает текущие твиты через систему потоковой передачи Twitter и определяет 10 самых популярных сенаторов по количеству твитов. Затем будет представлен класс `TweetListener`, который обрабатывает входящие твиты и сохраняет их разметку JSON в MongoDB. Наконец, мы продолжим сеанс IPython, создав интерактивную карту Folium для вывода информации из сохраненных твитов.

Использование Tweepy для аутентификации Twitter

Сначала используем Tweepy для выполнения аутентификации Twitter:

```
In [1]: import tweepy, keys

In [2]: auth = tweepy.OAuthHandler(
   ...:     keys.consumer_key, keys.consumer_secret)
   ...: auth.set_access_token(keys.access_token,
   ...:     keys.access_token_secret)
   ...:
```

Затем настроим объект Tweepy `API`, чтобы он переходил в режим ожидания при достижении ограничений частоты использования Twitter.

```
In [3]: api = tweepy.API(auth, wait_on_rate_limit=True,
    ...:                 wait_on_rate_limit_notify=True)
    ...:
```

Загрузка данных сенаторов

Воспользуемся информацией из файла senators.csv (находящегося в подкаталоге TwitterMongoDB каталога ch16) для отслеживания твитов, адресованных, отправленных и посвященных каждому сенатору США. Файл содержит двухбуквенный код штата, имя, партию, имя пользователя (handle) Twitter и идентификатор Twitter.

Twitter позволяет отслеживать конкретных пользователей по их числовым идентификаторам Twitter, но они должны передаваться в виде строковых представлений этих числовых значений. Итак, загрузим файл senators.csv в Pandas, преобразуем значения идентификаторов Twitter в строки (при помощи метода `astype` коллекции `Series`) и выведем несколько строк данных.

В данном случае мы указываем, что максимальное количество отображаемых столбцов равно 6. Позднее мы добавим в `DataFrame` еще один столбец, и этот параметр гарантирует, что будут выведены все столбцы вместо их подмножества, разделенного знаком ... :

```
In [4]: import pandas as pd

In [5]: senators_df = pd.read_csv('senators.csv')

In [6]: senators_df['TwitterID'] = senators_df['TwitterID'].astype(str)

In [7]: pd.options.display.max_columns = 6

In [8]: senators_df.head()
Out[8]:
  State            Name Party   TwitterHandle           TwitterID
0    AL  Richard Shelby     R       SenShelby            21111098
1    AL      Doug Jones     D    SenDougJones  941080085121175552
2    AK  Lisa Murkowski     R   lisamurkowski            18061669
3    AK    Dan Sullivan     R  SenDanSullivan          2891210047
4    AZ         Jon Kyl     R       SenJonKyl            24905240
```

Настройка MongoClient

Чтобы сохранить разметку JSON твитов в базе данных MongoDB, необходимо сначала подключиться к кластеру MongoDB Atlas при помощи функции `MongoClient` библиотеки `pymongo`, в аргументе которой передается строка подключения к вашему кластеру:

```
In [9]: from pymongo import MongoClient

In [10]: atlas_client = MongoClient(keys.mongo_connection_string)
```

Теперь получим объект `Database` библиотеки `pymongo`, представляющий БД `senators`. Следующая команда создает базу данных, если она не существует:

```
In [11]: db = atlas_client.senators
```

Создание потока твитов

Укажем количество твитов для загрузки и создадим объект `TweetListener`. При создании `TweetListener` передается объект `db`, представляющий БД MongoDB, чтобы твиты были записаны в базу данных. В зависимости от

скорости, с которой люди пишут твиты о сенаторах, сбор 10 000 твитов может занять от нескольких минут до нескольких часов. Для тестовых целей можно ограничиться меньшим числом:

```
In [12]: from tweetlistener import TweetListener

In [13]: tweet_limit = 10000

In [14]: twitter_stream = tweepy.Stream(api.auth,
    ...:     TweetListener(api, db, tweet_limit))
    ...:
```

Запуск потока твитов

Механизм потоковой передачи Twitter позволяет отслеживать до 400 ключевых слов и до 5000 идентификаторов Twitter одновременно. В данном примере мы будем отслеживать имена пользователей Twitter и идентификаторы сенаторов. В результате мы должны получить твиты отправленные, адресованные и относящиеся к каждому сенатору. Для демонстрации прогресса будем выводить экранное имя и временную метку для каждого полученного твита, а также общее количество обработанных твитов. Для экономии места приводим только один из результатов, заменив экранное имя пользователя XXXXXXX:

```
In [15]: twitter_stream.filter(track=senators_df.TwitterHandle.tolist(),
    ...:     follow=senators_df.TwitterID.tolist())
    ...:
    Screen name: XXXXXXX
     Created at: Sun Dec 16 17:19:19 +0000 2018
Tweets received: 1
...
```

Класс TweetListener

Для этого примера мы слегка изменили класс `TweetListener` из главы 12. Большая часть нижеследующего кода Twitter и Tweepy идентична приводившемуся ранее, поэтому мы сосредоточимся только на новых концепциях:

```
 1 # tweetlistener.py
 2 """TweetListener скачивает твиты и сохраняет их в MongoDB."""
 3 import json
 4 import tweepy
 5
 6 class TweetListener(tweepy.StreamListener):
```

```
 7     """Обрабатывает входной поток твитов."""
 8
 9     def __init__(self, api, database, limit=10000):
10         """Создает переменные экземпляров для отслеживания количества
              твитов."""
11         self.db = database
12         self.tweet_count = 0
13         self.TWEET_LIMIT = limit  # По умолчанию 10 000
14         super().__init__(api)  # Вызвать метод init суперкласса
15
16     def on_connect(self):
17         """Вызывается в том случае, если попытка подключения была успешной,
18         чтобы вы могли выполнить нужные операции в этот момент."""
19         print('Successfully connected to Twitter\n')
20
21     def on_data(self, data):
22         """Вызывается, когда Twitter отправляет вам новый твит."""
23         self.tweet_count += 1  # Количество обработанных твитов.
24         json_data = json.loads(data)  # Преобразование строки в JSON
25         self.db.tweets.insert_one(json_data)  # Сохранение в коллекции твитов
26         print(f'    Screen name: {json_data["user"]["name"]}')
27         print(f'     Created at: {json_data["created_at"]}')
28         print(f'Tweets received: {self.tweet_count}')
29
30         # При достижении TWEET_LIMIT вернуть False для завершения передачи
31         return self.tweet_count != self.TWEET_LIMIT
32
33     def on_error(self, status):
34         print(status)
35         return True
```

Ранее класс `TweetListener` переопределил метод `on_status` для получения объектов Tweepy `Status`, представляющих твиты. На этот раз переопределяется метод `on_data` (строки 21–31). Вместо объектов `Status` метод `on_data` получает необработанную *разметку JSON* для объекта твита. В строке 24 строка JSON, полученная методом `on_data`, преобразуется в объект Python JSON. Каждая БД MongoDB содержит одну или несколько коллекций `Collection` документов. В строке 25 выражение

```
self.db.tweets
```

обращается к коллекции `tweets` объекта `Database`, создавая ее в том случае, если она не существует. В строке 25 *метод* `insert_one` коллекции используется для сохранения объекта JSON в коллекции `tweets`.

Подсчет твитов для каждого сенатора

Затем выполним полнотекстовый поиск по коллекции твитов и подсчитаем количество твитов, содержащих имя пользователя Twitter каждого сенатора. Для выполнения полнотекстового поиска в БД MongoDB необходимо создать *текстовый индекс*[1]. Он указывает, в каком поле(-ях) документа следует вести поиск. Каждый текстовый индекс определяется как кортеж из имени поля для поиска и типа индекса (`'text'`). Универсальный спецификатор MongoDB (`$**`) означает, что каждое текстовое поле в документе (объекте твита JSON) должно индексироваться для полнотекстового поиска:

```
In [16]: db.tweets.create_index([('$**', 'text')])
Out[16]: '$**_text'
```

После того как индекс будет определен, вы сможете воспользоваться *методом* `count_documents` объекта `Collection` для подсчета общего количества документов в коллекции, содержащих указанный текст. Выполним поиск по коллекции `tweets` БД для каждого имени пользователя Twitter из столбца `TwitterHandle` коллекции `senators_df` (коллекция `DataFrame`):

```
In [17]: tweet_counts = []

In [18]: for senator in senators_df.TwitterHandle:
    ...:     tweet_counts.append(db.tweets.count_documents(
    ...:         {"$text": {"$search": senator}}))
    ...:
```

Объект JSON, передаваемый `count_documents`, в данном случае сообщает, что индекс с именем `text` будет использоваться для поиска значения `senator`.

Вывод счетчиков твитов для каждого сенатора

Создадим копию коллекции `senators_df` с добавленным столбцом `tweet_counts`, после чего выведем список из 10 сенаторов с наибольшими значениями счетчика твитов:

---

[1] За дополнительной информацией о разновидностях индексов в MongoDB, текстовых индексах и операторах обращайтесь по адресам https://docs.mongodb.com/manual/indexes, https://docs.mongodb.com/manual/core/index-text и https://docs.mongodb.com/manual/reference/operator.

```
In [19]: tweet_counts_df = senators_df.assign(Tweets=tweet_counts)

In [20]: tweet_counts_df.sort_values(by='Tweets',
    ...:     ascending=False).head(10)
    ...:
```

```
Out[20]:
   State                Name Party    TwitterHandle   TwitterID  Tweets
78    SC      Lindsey Graham     R  LindseyGrahamSC   432895323    1405
41    MA    Elizabeth Warren     D        SenWarren   970207298    1249
8     CA    Dianne Feinstein     D     SenFeinstein   476256944    1079
20    HI        Brian Schatz     D      brianschatz    47747074     934
62    NY       Chuck Schumer     D       SenSchumer    17494010     811
24    IL     Tammy Duckworth     D     SenDuckworth  1058520120     656
13    CT  Richard Blumenthal     D    SenBlumenthal   278124059     646
21    HI        Mazie Hirono     D      maziehirono    92186819     628
86    UT         Orrin Hatch     R    SenOrrinHatch   262756641     506
77    RI  Sheldon Whitehouse     D    SenWhitehouse   242555999     350
```

Получение координат штатов для вывода маркеров

Затем мы воспользуемся методами, описанными в главе 12, для получения широты и долготы каждого штата (в том числе, несколько ниже, — для вывода на карте Folium маркеров с названиями и количеством твитов, в которых упоминаются сенаторы каждого штата).

Файл state_codes.py содержит словарь `state_codes`, связывающий двухбуквенные обозначения штатов с их полными названиями. Полные названия штатов используются в сочетании с функцией `geocode` объекта `OpenMapQuest` библиотеки `geopy` для определения местонахождения каждого штата[1]. Начнем с импортирования необходимых библиотек и словаря `state_codes`:

```
In [21]: from geopy import OpenMapQuest

In [22]: import time

In [23]: from state_codes import state_codes
```

Затем получим объект `geocoder` для преобразования названий штатов в объекты `Location`:

```
In [24]: geo = OpenMapQuest(api_key=keys.mapquest_key)
```

---

[1] Мы будем использовать полные названия штатов, потому что в ходе тестирования по двухбуквенным обозначениям штатов не всегда возвращались правильные кооординаты.

Каждый штат представлен двумя сенаторами, поэтому мы можем определить местонахождение для каждого штата однократно, а затем использовать объект `Location` для обоих сенаторов от этого штата. Мы будем использовать уникальные названия штатов, отсортированные по возрастанию:

```
In [25]: states = tweet_counts_df.State.unique()

In [26]: states.sort()
```

Следующие два фрагмента используют код из главы 12 для определения местоположения каждого штата. Во фрагменте [28] при вызове функции `geocode` передается название штата с суффиксом `', USA'`, который гарантирует, что будут получены данные для США[1], потому что за пределами США существуют места, названия которых совпадают с названиями штатов. Чтобы отображать информацию о ходе выполнения, выведем строку каждого нового объекта `Location`:

```
In [27]: locations = []

In [28]: for state in states:
    ...:     processed = False
    ...:     delay = .1
    ...:     while not processed:
    ...:         try:
    ...:             locations.append(
    ...:                 geo.geocode(state_codes[state] + ', USA'))
    ...:             print(locations[-1])
    ...:             processed = True
    ...:         except:  # Тайм-аут, ожидаем перед повторной попыткой
    ...:             print('OpenMapQuest service timed out. Waiting.')
    ...:             time.sleep(delay)
    ...:             delay += .1
    ...:
Alaska, United States of America
Alabama, United States of America
Arkansas, United States of America
...
```

[1] Когда мы в первый раз проводили геокодирование для штата Вашингтон, объект OpenMapQuest вернул данные для города Вашингтон (округ Колумбия). Из-за этого мы внесли изменения в файл state_codes.py, чтобы в нем использовалась строка «Washington State».

Группировка счетчиков твитов по штатам

Используем общее количество твитов двух сенаторов штата для назначения цвета этого штата на карте. Более темными цветами обозначаются штаты с бóльшим количеством твитов. Для подготовки данных к нанесению на карту воспользуемся методом `groupby` коллекции `DataFrame` библиотеки Pandas для группировки сенаторов по штатам и вычисления суммарного количества твитов по штатам:

```
In [29]: tweets_counts_by_state = tweet_counts_df.groupby(
    ...:     'State', as_index=False).sum()
    ...:

In [30]: tweets_counts_by_state.head()
Out[30]:
  State  Tweets
0    AK      27
1    AL       2
2    AR      47
3    AZ      47
4    CA    1135
```

Ключевой аргумент `as_index=False` в фрагменте [29] показывает, что обозначения штатов должны быть значениями столбца полученного объекта `GroupBy` (вместо индексов строк). Метод `sum` объекта `GroupBy` суммирует числовые данные (количество твитов по штату). Фрагмент [30] выводит несколько строк объекта `GroupBy`, чтобы вы могли просмотреть часть результатов.

Создание карты

Перейдем к созданию карты. Возможно, вы захотите отрегулировать масштаб. В нашей системе приведенный ниже фрагмент создает карту, на которой изначально видна только континентальная часть США. Помните, карты Folium интерактивны, так что после отображения карты вы сможете отрегулировать масштаб или перетащить карту для просмотра других областей, включая Аляску или Гавайи:

```
In [31]: import folium

In [32]: usmap = folium.Map(location=[39.8283, -98.5795],
    ...:                    zoom_start=4, detect_retina=True,
    ...:                    tiles='Stamen Toner')
    ...:
```

Создание картограммы для определения цветов карты

На *картограмме* области карты раскрашиваются в разные цвета в соответствии со значениями, которые вы задали для определения карты. Создадим картограмму, выбирающую окраску штатов по количеству твитов, в которых встречаются имена сенаторов. Сохраним файл Folium us-states.json по адресу:

https://raw.githubusercontent.com/python-visualization/folium/master/examples/data/us-states.json

в каталоге этого примера. Файл содержит разметку в диалекте JSON, который называется *GeoJSON* (*Geographic JSON*) и предназначается для описания границ фигур — в данном случае границ всех штатов США. Картограмма использует эту информацию для определения окраски каждого штата. За более подробной информацией о GeoJSON обращайтесь по адресу http://geojson.org/.[1] Следующие фрагменты создают картограмму, а затем добавляют ее к карте:

```
In [33]: choropleth = folium.Choropleth(
    ...:     geo_data='us-states.json',
    ...:     name='choropleth',
    ...:     data=tweets_counts_by_state,
    ...:     columns=['State', 'Tweets'],
    ...:     key_on='feature.id',
    ...:     fill_color='YlOrRd',
    ...:     fill_opacity=0.7,
    ...:     line_opacity=0.2,
    ...:     legend_name='Tweets by State'
    ...: ).add_to(usmap)
    ...:

In [34]: layer = folium.LayerControl().add_to(usmap)
```

В данном случае были использованы следующие аргументы:

- `geo_data='us-states.json'` — файл с разметкой GeoJSON, определяющей границы окрашиваемых областей.
- `name='choropleth'` — Folium отображает картограмму `Choropleth` как слой на карте. Это имя будет отображаться в инструментарии управления слоями, который позволяет отображать и скрывать слои. Инструменты

[1] Folium предоставляет ряд других файлов GeoJSON в своем каталоге примеров по адресу https://github.com/python-visualization/folium/tree/master/examples/data. Вы также можете создать собственный файл по адресу http://geojson.io.

управления вызываются щелчком на значке с изображением слоев () на карте;

- `data=tweets_counts_by_state` — коллекция pandas `DataFrame` (или `Series`) со значениями, определяющими цвета картограммы;
- `columns=['State', 'Tweets']` — если данные содержатся в коллекции `DataFrame`, то аргумент содержит список двух столбцов, представляющих ключи и соответствующие значения цветов для окраски;
- `key_on='feature.id'` — переменная из файла GeoJSON, с которой картограмма связывает значения в аргументе `columns`;
- `fill_color='YlOrRd'` — цветовая карта, определяющая цвета для заполнения штатов. Folium предоставляет 12 цветовых карт: `'BuGn'`, `'BuPu'`, `'GnBu'`, `'OrRd'`, `'PuBu'`, `'PuBuGn'`, `'PuRd'`, `'RdPu'`, `'YlGn'`, `'YlGnBu'`, `'YlOrBr'` и `'YlOrRd'`. Поэкспериментируйте с разными картами, чтобы найти самую эффективную и эстетическую подборку цветов для своего приложения;
- `fill_opacity=0.7` — значение в диапазоне от `0.0` (полная прозрачность) до `1.0` (непрозрачность), определяющее степень прозрачности цветов заливки, выводимой в границах штатов;
- `line_opacity=0.2` — значение в диапазоне от `0.0` (полная прозрачность) до `1.0` (непрозрачность), определяющее прозрачность линий, которые обозначают границы штатов;
- `legend_name='Tweets by State'` — в верхней части карты выводится цветовая шкала, обозначающая диапазон значений, представленных цветами. Текст `legend_name` выводится под цветовой шкалой и сообщает, что представляют разные цвета.

Полный список ключевых аргументов `Choropleth` документирован по адресу:

http://python-visualization.github.io/folium/modules.html#folium.features.Choropleth

Создание маркеров для каждого штата

Затем создадим маркеры для каждого штата. Чтобы сенаторы отображались в каждом маркере в порядке убывания количества твитов, отсортируем `tweet_counts_df` по убыванию столбца `'Tweets'`:

```
In [35]: sorted_df = tweet_counts_df.sort_values(
    ...:     by='Tweets', ascending=False)
    ...:
```

Цикл в приведенном ниже фрагменте создает объекты `Marker`. Сначала вызов

```
sorted_df.groupby('State')
```

группирует `sorted_df` по значению `'State'`. Метод `groupby` коллекции `DataFrame` поддерживает *исходный порядок строк* в каждой группе. В заданной группе сенатор с наибольшим количеством твитов будет стоять на первом месте, так как сортировка выполнена по убыванию количества твитов во фрагменте [35]:

```
In [36]: for index, (name, group) in enumerate(sorted_df.groupby('State')):
    ...:     strings = [state_codes[name]]  # используется для сборки всплывающего
                                            # текста
    ...:
    ...:     for s in group.itertuples():
    ...:         strings.append(
    ...:             f'{s.Name} ({s.Party}); Tweets: {s.Tweets}')
    ...:
    ...:     text = '<br>'.join(strings)
    ...:     marker = folium.Marker(
    ...:         (locations[index].latitude, locations[index].longitude),
    ...:         popup=text)
    ...:     marker.add_to(usmap)
    ...:
    ...:
```

Передадим сгруппированную коллекцию `DataFrame` для `перебора`, чтобы получить индекс для каждой группы, по которой будет выбран объект `Location` для каждого штата из списка `locations`. Каждая группа состоит из названия (обозначение штата, по которому выполнялась группировка) и коллекции элементов этой группы (два сенатора от этого штата). Цикл работает следующим образом:

- Полное название штата ищется в словаре `state_codes` и сохраняется в списке `strings` — этот список будет использоваться для формирования текста подсказки `Marker`.
- Вложенный цикл перебирает элементы коллекции `group`, возвращая каждый элемент в виде именованного кортежа с данными конкретного сенатора. Для каждого сенатора строится отформатированная строка с именем, партией и количеством твитов, которая затем присоединяется к списку `strings`.
- В тексте `Marker` может использоваться разметка HTML для форматирования. Мы объединяем элементы списка `strings`, разделяя их элементом HTML `<br>`, создающим новую строку в HTML.

- Создается объект `Marker`. Первый аргумент определяет местоположение объекта `Marker` в виде кортежа, содержащего широту и долготу. Ключевой аргумент `popup` задает текст, выводимый по щелчку на маркере.
- Объект `Marker` добавляется на карту.

Вывод карты

Наконец, карта сохраняется в файле HTML:

```
In [17]: usmap.save('SenatorsTweets.html')
```

Чтобы просмотреть карту и взаимодействовать с ней, откройте файл HTML в своем браузере. Напомним, карту можно перетаскивать, чтобы вывести Аляску и Гавайи. Текст маркера для Южной Каролины:

Вы можете доработать этот пример и воспользоваться средствами анализа эмоциональной окраски (см. ранее), чтобы оценить сообщения с упоминанием каждого сенатора как положительные, нейтральные или отрицательные.

16.5. Hadoop

В нескольких ближайших разделах мы покажем, как технологии Apache Hadoop и Apache Spark решают проблемы хранения и обработки больших данных при помощи огромных компьютерных кластеров, массово-параллельной обработки, MapReduce-программирования Hadoop и средств Spark для обработки данных в памяти. В этом разделе рассматривается Apache Hadoop — ключевая технология больших данных, которая также лежит в основе многих последних достижений в области обработки больших данных и всей экосистемы программных инструментов, постоянно развивающихся для современных потребностей больших данных.

16.5.1. Обзор Hadoop

На момент запуска сервиса Google в 1998 году объем сетевых данных на 2,4 миллиона веб-сайтов[1] уже был огромным. В наши дни количество сайтов увеличилось почти до 2 миллиардов[2] (почти тысячекратное увеличение), а компания Google обрабатывает свыше 2 триллионов поисковых запросов в год[3]! К слову, авторы этой книги пользовались поисковой системой Google с момента ее появления, и, на их взгляд, скорость отклика в наши дни значительно выше.

Когда компания Google разрабатывала свою поисковую систему, она знала, что система должна быстро возвращать результаты поиска. Существовал лишь один реальный способ решения этой задачи — хранение и индексирование всего интернета, что можно было сделать только с применением умного сочетания механизмов использования внешней и оперативной памяти. Компьютеры того времени не могли хранить такие объемы данных или анализировать их достаточно быстро для того, чтобы гарантировать быструю выдачу ответов. Так компания Google разработала систему *кластеризации*, которая объединяла огромные количества компьютеров в так называемые *узлы* (nodes). Поскольку увеличение количества компьютеров и связей между ними означало более высокую вероятность аппаратных сбоев, в систему также были встроены высокие уровни *избыточности*, которые гарантировали, что система продолжит функ-

[1] http://www.internetlivestats.com/total-number-of-websites/.
[2] http://www.internetlivestats.com/total-number-of-websites/.
[3] http://www.internetlivestats.com/google-search-statistics/.

ционировать даже при сбое узлов внутри кластера. Данные распределялись между множеством недорогих «серийных компьютеров». Для удовлетворения поискового запроса все компьютеры кластера параллельно проводили поиск в той части системы, которая была им доступна локально. Затем результаты поиска собирались и возвращались пользователю.

Чтобы добиться этой цели, компания Google должна была разработать оборудование и программное обеспечение кластеризации, включая распределенное хранение. Компания Google опубликовала описание своей архитектуры, но не стала публиковать свой код. Затем программисты из Yahoo!, работавшие с описанием архитектуры Google из статьи «Google File System»[1], построили собственную систему. Они опубликовали свою работу на условиях открытого кода, а организация Apache реализовала систему в форме Hadoop (система получила свое название в честь плюшевого слона, который принадлежал ребенку одного из ее создателей).

Технологическому развитию Hadoop также способствовали две публикации Google — «MapReduce: Simplified Data Processing on Large Clusters»[2] и «Bigtable: A Distributed Storage System for Structured Data»[3] — эта технология была заложена в основу Apache HBase (база данных «ключ-значение» и столбцовая база данных NoSQL).[4]

HDFS, MapReduce и YARN

Ключевые компоненты Hadoop:

- *HDFS* (*Hadoop Distributed File System*) — файловая система для хранения огромных объемов данных в кластере;
- технология *MapReduce* для реализации задач обработки данных.

Ранее в книге мы представили основы программирования в функциональном стиле и парадигму «фильтрация/отображение/свертка». Hadoop MapReduce использует похожую концепцию, но в массово-параллельном масштабе. Задача MapReduce выполняет две операции — *отображение* и *свертку*. Шаг отобра-

---

[1] http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf.

[2] http://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf.

[3] http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf.

[4] Много других авторитетных публикаций, относящихся к большим данным (включая упомянутые), можно найти по адресу: https://bigdata-madesimple.com/research-papers-that-changed-the-world-of-big-data/.

жения, который также может включать *фильтрацию*, обрабатывает исходные данные по всему кластеру и отображает их на кортежи пар «ключ-значение». Затем шаг свертки объединяет эти кортежи для получения результатов задачи MapReduce. Здесь принципиальное значение имеет то, как выполняются операции MapReduce. Hadoop делит данные на *пакеты*, распределяемые по узлам кластера — от нескольких узлов до кластеров Yahoo! с 40 000 узлов и свыше 100 000 ядер[1]. Hadoop также распределяет код задачи MapReduce по узлам кластера и выполняет этот код параллельно на всех узлах. Каждый узел обрабатывает только пакет данных, хранящийся на этом узле. Шаг свертки объединяет результаты всех узлов для получения итогового результата. Для координации происходящего Hadoop использует механизм *YARN* (*«Yet Another Resource Negotiator»*), управляющий всеми ресурсами кластера и планирующий выполнение задач.

Экосистема Hadoop

Хотя существование Hadoop начиналось с HDFS и MapReduce, за которыми последовала технология YARN, в настоящее время на основе Hadoop сформировалась крупная экосистема, включающая Spark (см. разделы 16.6–16.7) и многие другие проекты Apache[2,3,4]:

- *Ambari* (https://ambari.apache.org) — инструменты для управления кластерами Hadoop.
- *Drill* (https://drill.apache.org) — SQL-запросы к нереляционным данным в базах данных Hadoop и NoSQL.
- *Flume* (https://flume.apache.org) — сервис сбора и хранения (в HDFS и других системах хранения) потоковых событийных данных (серверные журналы большого объема, сообщения IoT и т. д.).
- *HBase* (https://hbase.apache.org) — БД NoSQL для больших данных с «миллиардами строк и до 31 миллиона столбцов — на кластерах, построенных из серийного оборудования».
- *Hive* (https://hive.apache.org) — использование SQL для взаимодействия с данными в хранилищах данных. *Хранилище данных* (data warehouse) объединяет данные разных типов из разных источников. Основные опе-

[1] https://wiki.apache.org/hadoop/PoweredBy.
[2] https://hortonworks.com/ecosystems/.
[3] https://readwrite.com/2018/06/26/complete-guide-of-hadoop-ecosystem-components/.
[4] https://www.janbasktraining.com/blog/introduction-architecture-components-hadoop-ecosystem/.

рации — извлечение данных, их преобразование и загрузка (эти операции обозначаются сокращением *ETL*) в другую БД, обычно для анализа и построения отчетов.

- *Impala* (https://impala.apache.org) — БД для SQL-запросов в реальном времени к распределенным данным, хранимым в Hadoop HDFS или HBase.
- *Kafka* (https://kafka.apache.org) — передача сообщений в реальном времени, обработка и хранение потоковых данных (обычно с целью преобразования и обработки потоковых данных большого объема — скажем, активности на веб-сайте или потоковой передачи данных IoT).
- *Pig* (https://pig.apache.org) — сценарная платформа, преобразующая задачи анализа данных с языка сценариев Pig Latin в задачи MapReduce.
- *Sqoop* (https://sqoop.apache.org) — инструмент для перемещения структурированных, полуструктурированных и неструктурированных данных между базами данных.
- *Storm* (https://storm.apache.org) — система обработки потоковых данных в реальном времени для таких задач, как аналитика данных, машинное обучение, ETL и т. д.
- *ZooKeeper* (https://zookeeper.apache.org) — сервис для управления конфигурацией кластера и координацией между кластерами.

Провайдеры Hadoop

Многие провайдеры облачных сервисов предоставляют Hadoop как сервис — Amazon EMR, Google Cloud DataProc, IBM Watson Analytics Engine, Microsoft Azure HDInsight и др. Кроме того, такие компании, как Cloudera и Hortonworks (которые на момент написания книги проходили слияние), предоставляют интегрированные компоненты и инструменты экосистемы Hadoop через крупных провайдеров облачных сервисов. Они также предоставляют *бесплатные* загружаемые среды, которые можно запустить на настольном компьютере[1] для обучения, разработки и тестирования до перехода на облачное размещение, которое может быть сопряжено со значительными затратами. Программирование MapReduce будет представлено в примерах следующих разделов с использованием кластера на базе облачного сервиса Microsoft Azure HDInsight, предоставляющего Hadoop как сервис.

---

[1] Сначала проверьте их серьезные системные требования и убедитесь в том, что вы располагаете дисковым пространством и памятью, необходимыми для запуска.

Hadoop 3

Apache продолжает развивать Hadoop. В декабре 2017 года была выпущена версия Hadoop 3[1] с многочисленными усовершенствованиями, включая повышенное быстродействие и значительно улучшенную эффективность хранения данных[2].

16.5.2. Получение статистики по длине слов в «Ромео и Джульетте» с использованием MapReduce

В нескольких ближайших подразделах мы создадим облачный многоузловой кластер в Microsoft Azure HDInsight. Затем воспользуемся функциональностью сервиса для демонстрации выполнения задач Hadoop MapReduce в этом кластере. Задача MapReduce будет определять длину каждого слова в файле RomeoAndJuliet.txt (из главы 11), а затем выводить статистику по количеству слов каждой длины. После определения шагов отображения и свертки мы отправим задачу в кластер HDInsight, а Hadoop решит, как лучше использовать компьютерный кластер для ее выполнения.

16.5.3. Создание кластера Apache Hadoop в Microsoft Azure HDInsight

Большинство крупных провайдеров облачных сервисов поддерживают кластеры Spark и Hadoop, которые можно настраивать под потребности вашего приложения. Многоузловые облачные кластеры обычно являются *платными* сервисами, хотя многие провайдеры предоставляют бесплатные ознакомительные версии или кредиты, позволяющие опробовать эти сервисы.

Поэкспериментируем с процессом настройки кластеров и воспользуемся ими для выполнения задач. В этом примере Hadoop воспользуемся сервисом Microsoft Azure's HDInsight для создания облачных кластеров компьютеров для тестирования наших примеров. Откройте страницу

```
https://azure.microsoft.com/en-us/free
```

для создания учетной записи. Microsoft требует ввести данные кредитной карты для проверки личности. Некоторые сервисы бесплатны бессрочно, и вы

---

[1] За списком возможностей Hadoop 3 обращайтесь по адресу https://hadoop.apache.org/docs/r3.0.0/.

[2] https://www.datanami.com/2018/10/18/is-hadoop-officially-dead/.

можете продолжать пользоваться ими хоть круглый год. За информацией об этих сервисах обращайтесь по адресу:

https://azure.microsoft.com/en-us/free/free-account-faq/

Microsoft также предоставляет кредит для экспериментов с *платными* сервисами, такими как сервис Spark и HDInsight Hadoop. По истечении кредита или через 30 дней (в зависимости от того, что произойдет ранее) вы не сможете продолжить пользоваться платными сервисами, пока не разрешите Microsoft снимать средства с вашей карты.

Поскольку в наших примерах используется кредит новой учетной записи Azure[1], обсудим настройку низкозатратного кластера, использующего меньше вычислительных ресурсов по сравнению с тем, что Microsoft выделяет по умолчанию[2]. **Внимание: после выделения кластера плата будет взиматься независимо от того, пользуетесь вы им или нет. Следовательно, после завершения этого примера обязательно удалите свой кластер(-ы) и другие ресурсы во избежание лишних трат.** За дополнительной информацией обращайтесь по адресу:

https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-portal

Документацию и видеоролики, относящиеся к Azure, можно найти здесь:

- https://docs.microsoft.com/en-us/azure/ — документация Azure.
- https://channel9.msdn.com/ — видеосеть Microsoft Channel 9.
- https://www.youtube.com/user/windowsazure — канал Microsoft Azure на YouTube.

Создание кластера Hadoop в HDInsight

По следующей ссылке доступно описание процесса создания кластера для Hadoop с использованием сервиса Azure HDInsight:

---

[1] За последней информацией о возможностях бесплатных учетных записей Microsoft обращайтесь по адресу https://azure.microsoft.com/en-us/free/.

[2] Информация о рекомендуемых Microsoft конфигурациях кластеров доступна по адресу https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-component-versioning#default-node-configuration-and-virtual-machine-sizes-for-clusters. Если вы настроите кластер, размеры которого недостаточны для конкретного сценария, то при попытке развертывания кластера будет получено сообщение об ошибке.

https://docs.microsoft.com/en-us/azure/hdinsight/hadoop/apache-hadoop-linux-create-cluster-get-started-portal

Выполняя серию шагов Create a Hadoop cluster, обратите внимание на следующее:

- На *шаге 1* для обращения к порталу Azure вам следует ввести регистрационные данные своей учетной записи по адресу:

  https://portal.azure.com

- На *шаге 2* операция Data + Analytics теперь называется Analytics, а внешний вид значка HDInsight и его текст отличаются от того, что показано в учебнике.

- На *шаге 3* необходимо выбрать свободное имя кластера. При вводе имени Microsoft проверяет имя и выводит соответствующее сообщение, если оно занято. Также вы должны создать пароль. В группе Resource также необходимо щелкнуть на кнопке Create new и ввести имя группы. Оставьте все остальные настройки этого шага без изменений.

- На *шаге 5*: в разделе Select a Storage account щелкните на кнопке Create new и введите имя учетной записи хранения данных, содержащее только буквы нижнего регистра и цифры. Имя учетной записи хранения данных, как и имя кластера, должно быть уникальным.

Когда вы доберетесь до раздела Cluster summary, то увидите, что компания Microsoft изначально определила конфигурацию кластера `Head (2 x D12 v2), Worker (4 x D4 v2)`. На момент написания книги оцениваемая почасовая стоимость такой конфигурации составляла 3,11 доллара. Кластер использует шесть вычислительных узлов с 40 ядрами — много более, чем необходимо для демонстрационных целей. Вы можете отредактировать эту конфигурацию и сократить количество процессоров и ядер, что обеспечит экономию средств. Изменим конфигурацию и переключимся на кластер из четырех процессоров и 16 ядер, использующий менее мощные компьютеры. В разделе Cluster summary:

1. Щелкните на кнопке Edit справа от поля Cluster size.
2. Измените количество рабочих узлов Number of Worker на 2.
3. Щелкните на кнопке Worker node size, затем на кнопке View all, выберите вариант D3 v2 (минимальный размер для узлов Hadoop) и щелкните на кнопке Select.

4. Щелкните на кнопке Head node size, затем на кнопке View all, выберите вариант D3 v2 и щелкните на кнопке Select.
5. Щелкните на кнопке Next, затем снова на кнопке Next для возврата к разделу Cluster summary. Microsoft проверяет новую конфигурацию.
6. Когда кнопка Create станет доступной, щелкните на ней, чтобы развернуть кластер.

«Развертывание» кластера занимает 20–30 минут. В это время Microsoft выделяет все ресурсы и программное обеспечение, необходимое кластеру.

После внесения изменений оценочная стоимость кластера составляла 1,18 доллара в час (при *среднем* уровне использования кластеров с аналогичной конфигурацией). Наши фактические затраты были ниже оцениваемых. Если вы столкнетесь с какими-либо проблемами в ходе настройки кластера, то Microsoft предоставит техническую поддержку HDInsight в форме чата по адресу:

https://azure.microsoft.com/en-us/resources/knowledge-center/technical-chat/

16.5.4. Hadoop Streaming

В языках, не имеющих встроенной поддержки в Hadoop, таких как Python, для реализации задач приходится использовать технологию *Hadoop Streaming*. В Hadoop Streaming сценарии Python, реализующие шаги отображения и свертки, используют *стандартные потоки ввода и вывода* для взаимодействия с Hadoop. Обычно стандартный поток ввода читает данные с клавиатуры, а стандартный поток вывода записывает данные в командную строку. Тем не менее эти потоки могут быть *перенаправлены* (как это делает Hadoop) для чтения из других источников и записи в другие приемники. Hadoop использует потоки следующим образом:

- Hadoop поставляет ввод *сценарию отображения*. Этот сценарий читает свои данные из стандартного потока ввода.
- Сценарий отображения записывает свои результаты в стандартный поток вывода.
- Hadoop поставляет вывод сценария отображения на вход сценария свертки, который читает данные из стандартного потока ввода.
- Сценарий свертки записывает свои результаты в стандартный поток вывода.

- Hadoop записывает вывод сценария свертки в файловую систему Hadoop (HDFS).

Термины «сценарий отображения» и «сценарий свертки» должны быть вам знакомы по предшествующему обсуждению программирования в функциональном стиле, а также фильтрации, отображения и свертки в главе 5.

16.5.5. Реализация сценария отображения

В этом разделе мы создадим сценарий отображения, который получает строки текста в виде входных данных от Hadoop и отображает их на пары *«ключ-значение»*, в которых ключом является длина слова, а соответствующее значение равно 1. Сценарий отображения «видит» каждое слово по отдельности, так что с его точки зрения каждое слово существует только в одном экземпляре. В следующем разделе сценарий свертки обобщает эти пары «ключ-значение» по ключу, сводя их к одному значению счетчика для каждого ключа. По умолчанию Hadoop ожидает, что вывод сценария отображения, а также ввод и вывод сценария свертки существуют в форме пар «ключ-значение», разделенных символом *табуляции*.

В сценарии отображения (length_mapper.py) синтаксис `#!` в строке 1 сообщает Hadoop, что код Python должен выполняться `python3` вместо установки Python 2 по умолчанию. Эта строка должна предшествовать всем остальным комментариям и коду в файле. На момент написания книги использовались Python 2.7.12 и Python 3.5.2. Если в кластере не установлена версия Python 3.6 и выше, то вы не сможете использовать форматные строки в коде.

```
 1 #!/usr/bin/env python3
 2 # length_mapper.py
 3 """Отображает строки текста на пары "ключ-значение" из длины слова и 1."""
 4 import sys
 5
 6 def tokenize_input():
 7     """Каждая строка стандартного ввода разбивается на список строк."""
 8     for line in sys.stdin:
 9         yield line.split()
10
11 # Прочитать каждую строку в стандартном вводе и для каждого слова
12 # построить пару "ключ-значение" из длины слова, табуляции и 1
13 for line in tokenize_input():
14     for word in line:
15         print(str(len(word)) + '\t1')
```

Функция-генератор `tokenize_input` (строки 6–9) читает строки текста из стандартного потока ввода, возвращая каждой список строк. В данном примере мы не удаляем знаки препинания или игнорируемые слова, как это делалось в главе 11.

Когда Hadoop выполняет сценарий, в строках 13–15 перебирают списки строк, полученные от `tokenize_input`. Для каждого списка (строки входных данных) и для каждого элемента (слова) этого списка строка 15 выводит пару «ключ-значение», состоящую из длины слова (ключ), символа табуляции (`\t`) и значения 1. Это означает, что (пока) существует только одно слово этой длины. Конечно, наверняка существует много других слов с такой же длиной. Алгоритм MapReduce на шаге свертки обобщает эти пары «ключ-значение» и сводит все пары с одинаковым ключом к одной паре «ключ-значение» со значением-счетчиком.

16.5.6. Реализация сценария свертки

В сценарии свертки (length_reducer.py) функция `tokenize_input` (строки 8–11) представляет собой функцию-генератор, которая читает и разделяет пары «ключ-значение», произведенные сценарием отображения. И снова алгоритм MapReduce предоставляет стандартный ввод. Для каждой строки `tokenize_input` отделяет все начальные и завершающие пропуски (в частности, завершающие символы новой строки) и строит список, содержащий ключ и значение.

```
 1 #!/usr/bin/env python3
 2 # length_reducer.py
 3 """Подсчитывает количество слов каждой длины."""
 4 import sys
 5 from itertools import groupby
 6 from operator import itemgetter
 7
 8 def tokenize_input():
 9     """Разбивает каждую строку стандартного ввода на ключ и значение."""
10     for line in sys.stdin:
11         yield line.strip().split('\t')
12
13 # Построить пары "ключ-значение" из длины слова и счетчика, разделенные
   # табуляцией
14 for word_length, group in groupby(tokenize_input(), itemgetter(0)):
15     try:
16         total = sum(int(count) for word_length, count in group)
17         print(word_length + '\t' + str(total))
18     except ValueError:
19         pass  # Если счетчик не является целым числом, то слово игнорируется
```

Когда алгоритм MapReduce выполняет этот сценарий свертки, в строках 14–19 функция groupby из *модуля* itertools используется для группировки всех длин слов с одинаковыми значениями. Первый аргумент вызывает tokenize_input для получения списков, представляющих пары «ключ-значение». Второй аргумент означает, что пары «ключ-значение» должны группироваться на основании элемента с индексом 0 в каждом списке (то есть ключа). Строка 16 суммирует все счетчики для заданного ключа. Строка 17 выводит новую пару «ключ-значение», которая состоит из слова и его счетчика. Алгоритм MapReduce берет все итоговые счетчики и записывает их в файл в HDFS — файловой системе Hadoop.

16.5.7. Подготовка к запуску примера MapReduce

Файлы необходимо загрузить в кластер для выполнения примера. В приглашении командной строки, терминале или командной оболочке перейдите в каталог, содержащий сценарии отображения и свертки, а также файл RomeoAndJuliet.txt. Программа предполагает, что все три файла находятся в каталоге ch16, поэтому не забудьте скопировать файл RomeoAndJuliet.txt в каталог.

Копирование файлов сценариев в Hadoop-кластер HDInsight

Введите приведенную ниже команду для отправки файлов. Не забудьте заменить *ИмяКластера* тем именем, которое было задано при создании кластера Hadoop, и нажмите клавишу Enter только после того, как будет введена вся команда. Двоеточие в этой команде обязательно; оно означает, что пароль кластера будет введен по запросу. Введите пароль, заданный при создании кластера, и нажмите клавишу Enter:

```
scp length_mapper.py length_reducer.py RomeoAndJuliet.txt
    sshuser@ИмяКластера-ssh.azurehdinsight.net:
```

При выполнении команды впервые в целях безопасности вам будет предложено подтвердить, что вы доверяете хосту (то есть Microsoft Azure).

Копирование файла RomeoAndJuliet.txt в файловую систему Hadoop

Чтобы прочитать содержимое RomeoAndJuliet.txt и передать строки текста сценарию отображения, необходимо сначала скопировать файл в файловую

систему Hadoop. Используйте ssh[1], чтобы войти в кластер и получить доступ к командной строке. В приглашении командной строки, терминале или командной оболочке введите приведенную ниже команду. Не забудьте заменить *ИмяКластера* именем своего кластера. Вам снова будет предложено ввести пароль кластера:

```
ssh sshuser@ИмяКластера-ssh.azurehdinsight.net
```

В данном примере мы воспользуемся следующей командой Hadoop для копирования текстового файла в уже существующий каталог /examples/data, который предоставляется кластером для учебных руководств Microsoft Azure Hadoop. И снова клавиша Enter должна быть нажата только после того, как вы введете всю команду:

```
hadoop fs -copyFromLocal RomeoAndJuliet.txt
    /example/data/RomeoAndJuliet.txt
```

16.5.8. Выполнение задания MapReduce

Теперь вы можете запустить задание MapReduce для файла RomeoAndJuliet.txt в вашем кластере, выполнив приведенную ниже команду. Для вашего удобства мы включили текст этой команды в файл yarn.txt, так что вы можете скопировать ее из файла. Мы, кроме того, переформатировали команду для удобочитаемости:

```
yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar
   -D mapred.output.key.comparator.class=
      org.apache.hadoop.mapred.lib.KeyFieldBasedComparator
   -D mapred.text.key.comparator.options=-n
   -files length_mapper.py,length_reducer.py
   -mapper length_mapper.py
   -reducer length_reducer.py
   -input /example/data/RomeoAndJuliet.txt
   -output /example/wordlengthsoutput
```

Команда `yarn` запускает программу Hadoop YARN, предназначенную для управления и координации доступа к ресурсам Hadoop, используемым

[1] Пользователям Windows: если ssh у вас не работает, установите и активируйте ssh так, как описано по адресу https://blogs.msdn.microsoft.com/powershell/2017/12/15/using-the-openssh-beta-in-windows-10-fall-creators-update-and-windows-server-1709/. После завершения установки выйдите из системы и войдите снова или перезапустите систему, чтобы активировать ssh.

задачей MapReduce. Файл hadoop-streaming.jar содержит утилиту Hadoop Streaming, которая позволяет использовать Python для реализации сценариев отображения и свертки. Два параметра `-D` задают свойства Hadoop, которые позволяют отсортировать итоговые пары «ключ-значение» по ключу (`KeyFieldBasedComparator`) по убыванию в числовом порядке (`-n`; минус означает сортировку по убыванию) вместо алфавитного. Другие аргументы командной строки:

- `-files` — список имен файлов, разделенных запятыми. Hadoop копирует эти файлы в каждый узел в кластере, чтобы они могли выполняться локально на каждом узле;
- `-mapper` — имя файла сценария отображения;
- `-reducer` — имя файла сценария свертки;
- `-input` — файл или каталог с файлами, передаваемыми сценарию отображения в качестве входных данных;
- `-output` — каталог HDFS, в который будет записываться весь вывод. Если каталог уже существует, происходит ошибка.

В следующем листинге приведена часть результатов, которые выдает Hadoop при выполнении задания MapReduce. Мы заменили части вывода многоточиями (`...`) для экономии места, а также выделили жирным шрифтом некоторые строки:

- Общее количество обрабатываемых входных путей (`Total input paths to process`) — единственным источником ввода в данном примере является файл RomeoAndJuliet.txt.
- Количество разбиений (`number of splits`) — два в данном примере; определяется количеством рабочих узлов в кластере.
- Проценты завершения.
- Счетчики файловой системы (`File System Counters`), включающие количество прочитанных и записанных байтов.
- Счетчики задания (`Job Counters`) с количеством использованных задач отображения и свертки, а также различной хронометражной информацией.
- Структура Map-Reduce (`Map-Reduce Framework`) с различной информацией об этапах выполнения.

```
packageJobJar: [] [/usr/hdp/2.6.5.3004-13/hadoop-mapreduce/hadoop-
streaming-2.7.3.2.6.5.3004-13.jar] /tmp/streamjob2764990629848702405.jar
tmpDir=null
...
18/12/05 16:46:25 INFO mapred.FileInputFormat: Total input paths to
process : 1
18/12/05 16:46:26 INFO mapreduce.JobSubmitter: number of splits:2
...
18/12/05 16:46:26 INFO mapreduce.Job: The url to track the job: http://
hn0-paulte.y3nghy5db2kehav5m0opqrjxcb.cx.internal.cloudapp.net:8088/
proxy/application_1543953844228_0025/
...
18/12/05 16:46:35 INFO mapreduce.Job:  map 0% reduce 0%
18/12/05 16:46:43 INFO mapreduce.Job:  map 50% reduce 0%
18/12/05 16:46:44 INFO mapreduce.Job:  map 100% reduce 0%
18/12/05 16:46:48 INFO mapreduce.Job:  map 100% reduce 100%
18/12/05 16:46:50 INFO mapreduce.Job: Job job_1543953844228_0025
completed successfully
18/12/05 16:46:50 INFO mapreduce.Job: Counters: 49
        File System Counters
             FILE: Number of bytes read=156411
             FILE: Number of bytes written=813764
...
        Job Counters
             Launched map tasks=2
             Launched reduce tasks=1
...
        Map-Reduce Framework
             Map input records=5260
             Map output records=25956
             Map output bytes=104493
             Map output materialized bytes=156417
             Input split bytes=346
             Combine input records=0
             Combine output records=0
             Reduce input groups=19
             Reduce shuffle bytes=156417
             Reduce input records=25956
             Reduce output records=19
             Spilled Records=51912
             Shuffled Maps =2
             Failed Shuffles=0
             Merged Map outputs=2
             GC time elapsed (ms)=193
             CPU time spent (ms)=4440
             Physical memory (bytes) snapshot=1942798336
             Virtual memory (bytes) snapshot=8463282176
```

```
                Total committed heap usage (bytes)=3177185280
...
18/12/05 16:46:50 INFO streaming.StreamJob: Output directory: /example/
wordlengthsoutput
```

Просмотр счетчиков

Hadoop MapReduce сохраняет вывод в HDFS, поэтому для просмотра счетчиков длин слов необходимо просмотреть файл в HDFS в кластере следующей командой:

```
hdfs dfs -text /example/wordlengthsoutput/part-00000
```

Результаты выполнения предшествующей команды:

```
18/12/05 16:47:19 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
18/12/05 16:47:19 INFO lzo.LzoCodec: Successfully loaded & initialized
native-lzo library [hadoop-lzo rev
b5efb3e531bc1558201462b8ab15bb412ffa6b89]
1       1140
2       3869
3       4699
4       5651
5       3668
6       2719
7       1624
8       1062
9       855
10      317
11      189
12      95
13      35
14      13
15      9
16      6
17      3
18      1
23      1
```

Удаление кластера для предотвращения затрат

Внимание: обязательно удалите свой кластер(-ы) и связанные с ним ресурсы (такие, как пространство хранения данных) во избежание лишних расходов. На портале Azure щелкните на кнопке All resources для просмотра списка ресурсов, в котором будет присутствовать созданный кластер и учетная

запись хранилища. И то и другое может привести к списанию средств, если не удалить эти ресурсы. Выберите каждый ресурс и удалите его кнопкой Delete. Вам будет предложено подтвердить свое решение (введите `yes`). За дополнительной информацией обращайтесь по адресу:

https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-portal

16.6. Spark

В этом разделе приведен обзор Apache Spark. Мы воспользуемся *библиотекой PySpark* для Python и средствами Spark в стиле функционального программирования (фильтрация/отображение/свертка) для реализации простого примера с подсчетом слов, который генерирует статистику по длине слов в пьесе «Ромео и Джульетта».

16.6.1. Краткий обзор Spark

При обработке действительно больших данных эффективность становится критически значимым фактором. Технология Hadoop адаптирована для пакетной обработки на базе дисков — данные читаются с диска, обрабатываются, а результаты записываются обратно на диск. Во многих случаях практическое применение больших данных требует эффективности более высокой, чем та, которую можно достичь при интенсивной работе с диском. В частности, быстрые потоковые приложения, требующие обработки в реальном времени или почти в реальном времени, не будут работать в дисковой архитектуре.

История

Технология Spark была разработана в 2009 году в Университете Беркли (Калифорния, США) и финансировалась DARPA (Управление перспективных исследований Министерства обороны США). Изначально она создавалась как ядро распределенного выполнения для высокопроизводительного машинного обучения[1]. Spark использует *архитектуру обработки в памяти*, которая «была применена для сортировки 100 Тбайт данных в 3 раза быстрее, чем Hadoop MapReduce, на 1/10 машин»[2], а по скорости выполнения некоторых задач

[1] https://gigaom.com/2014/06/28/4-reasons-why-spark-could-jolt-hadoop-into-hyperdrive/.

[2] https://spark.apache.org/faq.html.

превосходила Hadoop до 100 раз[1]. Существенно более высокая эффективность Spark в задачах пакетной обработки заставила многие компании перейти с Hadoop MapReduce на Spark[2,3,4].

Архитектура и компоненты

Хотя изначально технология Spark разрабатывалась для выполнения на базе Hadoop и использовала такие компоненты Hadoop, как HDFS и YARN, Spark может работать автономно: на одном компьютере (обычно для обучения и тестирования), а также в кластере или с использованием разных менеджеров кластеров и распределенных систем хранения данных. Для управления ресурсами Spark работает на базе Hadoop YARN, Apache Mesos, Amazon EC2 и Kubernetes, поддерживая массу распределенных систем хранения, включая HDFS, Apache Cassandra, Apache HBase и Apache Hive[5].

Центральное место в Spark занимают *отказоустойчивые распределенные наборы данных* (*RDD*, *Resilient Distributed Datasets*), используемые для обработки распределенных данных с помощью программирования в функциональном стиле. Кроме чтения данных с диска и записи данных на диск, Hadoop применяет репликацию для обеспечения отказоустойчивости, что создает дополнительные потери ресурсов из-за дисковых операций. RDD устраняют эти потери за счет работы с памятью (диск используется только в том случае, если данные не помещаются в памяти), а не за счет репликации. В Spark отказоустойчивость обеспечивается запоминанием действий, использованных для создания каждого RDD, что позволяет заново построить RDD в случае сбоя кластера[6].

Spark распределяет операции, заданные в Python, по узлам кластера для параллельного выполнения. Механизм Spark Streaming позволяет обрабатывать данные по мере получения. Коллекции Spark `DataFrame`, сходные с коллекциями `DataFrame` библиотеки Pandas, позволяют просматривать RDD как коллекцию именованных столбцов. Коллекции Spark `DataFrame` могут использоваться в сочетании с Spark SQL для выполнения запросов к распределенным данным. Spark также включает библиотеку *Spark MLlib* (Spark

---

1 https://spark.apache.org/.

2 https://bigdata-madesimple.com/is-spark-better-than-hadoop-map-reduce/.

3 https://www.datanami.com/2018/10/18/is-hadoop-officially-dead/.

4 https://blog.thecodeteam.com/2018/01/09/changing-face-data-analytics-fast-data-displaces-big-data/.

5 http://spark.apache.org/.

6 https://spark.apache.org/research.html.

Machine Learning Library) для выполнения различных алгоритмов машинного обучения, сходных с теми, которые были описаны в главах 14 и 15. RDD, Spark Streaming, коллекции `DataFrame` и Spark SQL будут описаны в нескольких ближайших примерах.

Провайдеры

Провайдеры Hadoop обычно предоставляют поддержку Spark. Кроме провайдеров, перечисленных в разделе 16.5, также существуют провайдеры, специализирующиеся на Spark, например Databricks. Они предоставляют «облачную платформу, построенную на базе Spark, с нулевыми потребностями в управлении»[1]. Кроме того, превосходным ресурсом для изучения Spark является веб-сайт Databricks. Платная платформа Databricks работает на базе Amazon AWS или Microsoft Azure. Databricks также предоставляет бесплатный уровень Databricks Community Edition, идеально подходящий для изучения Spark и среды Databricks.

16.6.2. Docker и стеки Jupyter Docker

В этом разделе мы покажем, как загрузить и выполнить стек Docker, содержащий Spark и модуль PySpark для работы со Spark из Python. Код примера Spark будет написан в Jupyter Notebook. Начнем с обзора Docker.

Docker

Docker — инструмент для упаковки программного обеспечения в *контейнеры* (также называемые *образами*), соединяющие воедино *все* необходимое для выполнения этого программного обеспечения на разных платформах. Некоторые программные пакеты, используемые в этой главе, требуют сложной настройки. Во многих случаях существуют готовые контейнеры Docker, которые можно загрузить бесплатно и выполнять локально на настольных или портативных компьютерах. Все это делает Docker отличным вариантом для быстрого и удобного освоения новых технологий.

Docker также способствует обеспечению *воспроизводимости* результатов в исследованиях и аналитике. Вы можете создавать специализированные контейнеры Docker, настроенные с версиями всех программных продуктов

[1] https://databricks.com/product/faq.

и всех библиотек, используемых в исследованиях. Это позволит другим специалистам воссоздать среду, которая использовалась вами, и поможет им воспроизвести ваши результаты в последующий момент. В этом разделе мы используем Docker для загрузки и выполнения контейнера Docker, настроенного для запуска приложений Spark.

Установка Docker

Docker для Windows 10 Pro или macOS можно установить по адресу:

https://www.docker.com/products/docker-desktop

В Windows 10 Pro вы должны разрешить установочной программе `"Docker for Windows.exe"` вносить изменения в вашу систему для завершения процесса установки. Когда Windows спросит, хотите ли вы разрешить программе установки вносить изменения в вашу систему, щелкните на кнопке Yes [1]. Пользователям Windows 10 Home придется использовать Virtual Box так, как описано по адресу:

https://docs.docker.com/machine/drivers/virtualbox/

Пользователи Linux должны установить Docker Community Edition согласно следующему описанию:

https://docs.docker.com/install/overview/

Для получения общего представления о Docker прочитайте руководство «Getting started» по адресу:

https://docs.docker.com/get-started/

Стеки Jupyter Docker

Команда Jupyter Notebook создала несколько готовых «стеков Docker» для Jupyter, содержащих стандартные сценарии развертывания Python. Каждая ситуация позволяет использовать документы Jupyter Notebook для экспериментов с функциональностью, не отвлекаясь на сложные аспекты настройки программного обеспечения. В каждом случае вы можете открыть JupyterLab

[1] Возможно, некоторым пользователям Windows придется выполнить инструкции из раздела «Allow specific apps to make changes to controlled folders» по адресу https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-exploit-guard/customize-controlled-folders-exploit-guard.

в браузере, открыть документ Notebook в JupyterLab и начать программирование. JupyterLab также предоставляет *окно терминала*, которое можно использовать в браузере по аналогии с окном терминала, приглашением Anaconda или командной оболочкой. Все, что мы приводили для IPython до настоящего момента, можно также выполнить с использованием IPython в окне терминала JupyterLab.

Мы будем использовать стек Docker `jupyter/pyspark-notebook`, заранее настроенный для создания и тестирования приложений Apache Spark на вашем компьютере. При установке других библиотек Python, использованных в книге, вы сможете реализовать большую часть примеров книги в этом контейнере. За дополнительной информацией о доступных стеках Docker обращайтесь по адресу:

https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html

Запуск стека Jupyter Docker

Прежде чем выполнять следующий шаг, убедитесь в том, что JupyterLab в настоящее время не выполняется на вашем компьютере. Загрузим и запустим стек Docker `jupyter/pyspark-notebook`. Чтобы ваша работа не была потеряна при закрытии контейнера Docker, присоединим к контейнеру каталог локальной файловой системы и используем его для сохранения вашего документа Notebook — пользователи Windows должны заменить `\` на `^.` :

```
docker run -p 8888:8888 -p 4040:4040 -it --user root \
    -v полныйПутьКИспользуемомуКаталогу:/home/jovyan/work \
    jupyter/pyspark-notebook:14fdfbf9cfc1 start.sh jupyter lab
```

При первом выполнении этой команды Docker загрузит контейнер Docker с именем:

```
jupyter/pyspark-notebook:14fdfbf9cfc1
```

Запись `":14fdfbf9cfc1"` обозначает конкретный контейнер `jupyter/pyspark-notebook` для загрузки. На момент написания книги новейшая версия контейнера была равна `14fdfbf9cfc1`. Указание конкретной версии, как это сделали мы, помогает обеспечить *воспроизводимость* результатов. Без включения `":14fdfbf9cfc1"` в команду Docker загрузит *новейшую* версию контейнера, которая может содержать другие версии программных продуктов и может оказаться несовместимой с выполняемым кодом. Размер контейнера Docker

составляет почти 6 Гбайт, так что исходное время загрузки будет зависеть от скорости подключения к интернету.

Открытие JupyterLab в браузере

После того как контейнер будет загружен и заработает, в окне приглашения командной строки, терминала или командной оболочки появится команда:

```
Copy/paste this URL into your browser when you connect for the first
time, to login with a token:

        http://(bb00eb337630 or 127.0.0.1):8888/?token=
            9570295e90ee94ecef75568b95545b7910a8f5502e6f5680
```

Скопируйте длинную шестнадцатеричную строку (в вашей системе она будет выглядеть иначе)

```
9570295e90ee94ecef75568b95545b7910a8f5502e6f5680
```

откройте адрес http://localhost:8888/lab в своем браузере (localhost соответствует 127.0.0.1 в предшествующем выводе) и *вставьте* скопированный маркер в поле Password or token. Щелкните на кнопке Log in, чтобы перейти в интерфейс JupyterLab. Если вы случайно закроете окно браузера, то откройте адрес http://localhost:8888/lab, чтобы продолжить сеанс.

При выполнении в контейнере Docker рабочий каталог на вкладке Files в левой части JupyterLab представляет каталог, присоединенный к контейнеру при помощи параметра `-v` команды `docker run`. С этого момента вы можете открывать файлы документов Notebook, предоставленные нами. Любые новые документы Notebook или другие файлы, которые вы будете создавать, будут сохраняться в этом каталоге по умолчанию. Так как рабочий каталог контейнера Docker связан с каталогом на вашем компьютере, все файлы, созданные в JupyterLab, останутся на вашем компьютере даже в том случае, если вы решите удалить контейнер Docker.

Обращение к командной строке контейнера Docker

Каждый контейнер Docker имеет интерфейс командной строки, сходный с тем, который использовался для запуска IPython в этой книге. Через этот интерфейс можно устанавливать пакеты Python в контейнере Docker и даже использовать IPython так, как это делалось ранее. Откройте отдельное приглашение

Anaconda, терминал или командную оболочку и выведите список контейнеров Docker, работающих в настоящий момент, при помощи следующей команды:

```
docker ps
```

Вывод команды получается довольно длинным, поэтому текст с большой вероятностью будет переноситься:

```
CONTAINER ID        IMAGE                                       COMMAND
           CREATED              STATUS             PORTS
  NAMES
f54f62b7e6d5        jupyter/pyspark-notebook:14fdfbf9cfc1   "tini -g --
/bin/bash"  2 minutes ago      Up 2 minutes      0.0.0.0:8888->8888/tcp
  friendly_pascal
```

В последней строке вывода в нашей системе под заголовком столбца `NAMES` из третьей строки выводится имя, случайным образом присвоенное Docker работающему контейнеру — `friendly_pascal`; в вашей системе имя будет другим. Чтобы обратиться к командной строке контейнера, выполните следующую команду, заменив *имя_контейнера* именем работающего контейнера:

```
docker exec -it имя_контейнера /bin/bash
```

Контейнер Docker использует Linux во внутренней реализации, поэтому вы увидите приглашение Linux, в котором сможете вводить команды.

Приложение из этого раздела будет использовать ту же функциональность библиотек NLTK и TextBlob, что и в главе 11. Ни одна из этих библиотек не устанавливается в стеках Jupyter Docker заранее. Чтобы установить NLTK и TextBlob, введите команду:

```
conda install -c conda-forge nltk textblob
```

Остановка и перезапуск контейнера Docker

Каждый раз, когда вы запускаете контейнер командой `docker run`, Docker предоставляет новый экземпляр, не содержащий ранее установленных библиотек. По этой причине вам следует отслеживать имя своего контейнера, чтобы вы могли использовать его из другого окна командной оболочки, приглашения Anaconda или терминала для остановки и перезапуска контейнера. Команда

```
docker stop container_name
```

завершает работу контейнера. Команда

```
docker restart container_name
```

перезапускает контейнер. Docker также предоставляет GUI-приложение с именем Kitematic, которое может использоваться для управления контейнерами, включая их остановку и перезапуск. Приложение можно загрузить на сайте https://kitematic.com/ и работать с ним из меню Docker. В следующем руководстве пользователя приведена краткая инструкция по управлению контейнерами из приложения:

https://docs.docker.com/kitematic/userguide/

16.6.3. Подсчет слов с использованием Spark

В этом разделе мы воспользуемся средствами фильтрации, отображения и свертки Spark для реализации простого примера, который строит сводку использования слов в «Ромео и Джульетте». Вы можете работать с существующим документом Notebook RomeoAndJulietCounter.ipynb из каталога SparkWordCount (в который вам следует скопировать файл RomeoAndJuliet.txt из главы 11) или же создать новый документ, а затем ввести и выполнить приведенные ниже фрагменты.

Загрузка игнорируемых слов NLTK

Воспользуемся методами, представленными в главе 11, для исключения игнорируемых слов из текста перед подсчетом частот слов. Сначала загрузим список игнорируемых слов NLTK:

```
[1]: import nltk
     nltk.download('stopwords')
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[1]: True
```

Затем загрузим игнорируемые слова в программе:

```
[2]: from nltk.corpus import stopwords
     stop_words = stopwords.words('english')
```

Настройка SparkContext

Объект `SparkContext` (из модуля `pyspark`) предоставляет доступ к функциональности Spark из Python. Многие среды Spark создают `SparkContext` за вас, но в стеке Docker Jupyter `pyspark-notebook` объект придется создать вам.

Сначала задайте параметры конфигурации, создав объект `SparkConf` (из модуля `pyspark`). Следующий фрагмент вызывает метод `setAppName` объекта для назначения имени приложения Spark и вызывает метод `setMaster` объекта для определения URL-адреса кластера Spark. URL-адрес `'local[*]'` означает, что Spark выполняется на вашем локальном компьютере (вместо кластера на базе облака), а звездочка сообщает Spark, что код должен выполняться с количеством *программных потоков*, равным количеству ядер на компьютере:

```
[3]: from pyspark import SparkConf
     configuration = SparkConf().setAppName('RomeoAndJulietCounter')\
                                .setMaster('local[*]')
```

Потоки позволяют одноузловому кластеру *совместно* (concurrent) выполнять части задач Spark, чтобы моделировать параллелизм, обеспечиваемый кластерами Spark. Когда мы говорим, что две задачи выполняются совместно, имеется в виду, что они продвигаются к завершению в одно время — как правило, одна задача выполняется в течение короткого промежутка времени, а затем дает возможность выполняться другой задаче. Под *параллельным* (parallel) выполнением понимается, что задачи выполняются одновременно; это одно из ключевых преимуществ выполнения Hadoop и Spark в компьютерных кластерах на базе облака.

Затем создадим объект `SparkContext`, передавая объект `SparkConf` в аргументе:

```
[4]: from pyspark import SparkContext
     sc = SparkContext(conf=configuration)
```

Чтение текстового файла и отображение его на слова

Для работы со `SparkContext` используются средства программирования в функциональном стиле (фильтрация, отображение и свертка), применяемые к *отказоустойчивым распределенным наборам данных* (*RDD*). RDD берет данные, хранящиеся в кластере в файловой системе Hadoop, и позволяет задать серию шагов обработки для преобразования данных в RDD. Действия по обработке выполняются *в отложенном режиме* (глава 5) — задание не выполняется до тех пор, пока вы не прикажете Spark приступить к его обработке. Следующий фрагмент задает три шага:

- Метод `textFile` объекта `SparkContext` загружает строки текста из файла RomeoAndJuliet.txt и возвращает их в виде `RDD` (из модуля `pyspark`) со строками, представляющими каждую строку.

- Метод `map` объекта `RDD` использует свой аргумент `lambda` для удаления всех знаков препинания функцией `strip_punc` объекта TextBlob и для преобразования каждой строки текста к нижнему регистру. Этот метод возвращает новый объект `RDD`, с которым можно задать дополнительные выполняемые операции.
- Метод `flatMap` объекта `RDD` использует свой аргумент `lambda` для отображения каждой строки текста на слова, и строит единый список слов вместо отдельных строк текста. Результатом выполнения `flatMap` является новый объект `RDD`, представляющий все слова «Ромео и Джульетты».

```
[5]: from textblob.utils import strip_punc
     tokenized = sc.textFile('RomeoAndJuliet.txt')\
                   .map(lambda line: strip_punc(line, all=True).lower())\
                   .flatMap(lambda line: line.split())
```

Удаление игнорируемых слов

Теперь используем метод `filter` объекта `RDD` для создания нового объекта `RDD`, из которого были исключены игнорируемые слова:

```
[6]: filtered = tokenized.filter(lambda word: word not in stop_words)
```

Подсчет всех оставшихся слов

Теперь в наборе остались только значимые слова, и мы можем подсчитать количество вхождений каждого слова. Для этого каждое слово сначала отображается на кортеж, содержащий слово и значение счетчика `1`. Здесь происходит примерно то же, что мы делали с Hadoop MapReduce. Spark распределяет задачу свертки по узлам кластера. Для полученного объекта `RDD` вызывается метод `reduceByKey`, которому в аргументе передается функция `add` модуля `operator`. Тем самым вы приказываете методу `reduceByKey` *просуммировать* счетчики для кортежей, содержащих одно значение `word` (ключ):

```
[7]: from operator import add
     word_counts = filtered.map(lambda word: (word, 1)).reduceByKey(add)
```

Поиск слов со счетчиками, большими или равными 60

Поскольку в тексте «Ромео и Джульетты» встречаются сотни слов, отфильтруем набор RDD, чтобы в нем остались только слова с 60 и более вхождениями:

```
[8]: filtered_counts = word_counts.filter(lambda item: item[1] >= 60)
```

Сортировка и вывод результатов

На данный момент заданы все операции для подсчета слов. При вызове метода `collect` объекта `RDD` Spark инициирует все действия обработки, заданные выше, возвращая список с окончательными результатами — в данном случае кортежи из слов и счетчиков. С вашей точки зрения все выглядит так, словно все вычисления выполнялись на одном компьютере. Но если объект `SparkContext` настроен для использования кластера, то Spark разделит задачи среди рабочих узлов кластера за вас. В следующем фрагменте список кортежей упорядочивается по убыванию (`reverse=True`) значений счетчиков (`itemgetter(1)`).

Вызовем метод `collect` для получения результатов и их сортировки по убыванию счетчика слов:

```
[9]: from operator import itemgetter
     sorted_items = sorted(filtered_counts.collect(),
                           key=itemgetter(1), reverse=True)
```

Наконец, выведем результаты. Сначала определим слово с наибольшим количеством букв, чтобы выровнять все слова по полю этой длины, а затем выведем каждое слово и его счетчик:

```
[10]: max_len = max([len(word) for word, count in sorted_items])
      for word, count in sorted_items:
          print(f'{word:>{max_len}}: {count}')
[10]:    romeo: 298
          thou: 277
        juliet: 178
           thy: 170S
         nurse: 146
       capulet: 141
          love: 136
          thee: 135
         shall: 110
          lady: 109
         friar: 104
          come: 94
      mercutio: 83
          good: 80
      benvolio: 79
         enter: 75
            go: 75
          i'll: 71
        tybalt: 69
         death: 69
```

```
   night: 68
lawrence: 67
     man: 65
    hath: 64
     one: 60
```

16.6.4. Подсчет слов средствами Spark в Microsoft Azure

В книге представлены как инструменты, которые могут использоваться бесплатно, так и инструменты для коммерческой разработки. Рассмотрим пример с подсчетом слов Spark средствами в кластере Microsoft Azure HDInsight.

Создание кластера Apache Spark в HDInsight с использованием Azure Portal

О настройке кластера Spark с использованием сервиса HDInsight читайте здесь:

> https://docs.microsoft.com/en-us/azure/hdinsight/spark/apache-spark-jupyter-spark-sql-use-portal

На этапах Create an HDInsight Spark cluster следует помнить о проблемах, перечисленных при описании создания кластера Hadoop ранее; в поле Cluster type выберите вариант Spark.

Как и прежде, конфигурация кластера по умолчанию предоставляет больше ресурсов, чем требуется для наших примеров. В разделе Cluster summary выполните действия, описанные ранее для создания кластера Hadoop, чтобы изменить количество рабочих узлов до двух и настроить рабочие и ведущие узлы для использования компьютеров `D3 v2`. Когда вы щелкнете на кнопке Create, стартует процесс настройки и развертывания кластера, который займет от 20 до 30 минут.

Установка библиотек в кластере

Если для вашего кода Spark необходимы библиотеки, не установленные в кластере HDInsight, их нужно будет установить. Чтобы увидеть, какие библиотеки установлены по умолчанию, воспользуйтесь ssh для входа в кластер (как было показано ранее в этой главе) и выполните команду:

```
/usr/bin/anaconda/envs/py35/bin/conda list
```

Так как ваш код будет выполняться на нескольких узлах кластера, библиотеки должны быть установлены на *каждом* узле. Azure требует, чтобы вы создали сценарий командной оболочки Linux, который содержит команды установки библиотек. Когда вы отправляете этот сценарий, Azure проверяет сценарий, а затем выполняет его на каждом узле. Сценарии командной оболочки Linux выходят за рамки книги, и этот сценарий должен быть размещен на веб-сервере, с которого Azure сможет загрузить файл. Мы создали за вас сценарий установки, который устанавливает библиотеки, используемые в примерах Spark. Выполните следующие действия, чтобы установить библиотеки:

1. На портале Azure выберите свой кластер.
2. В списке под полем поиска кластера щелкните на варианте Script Actions.
3. Щелкните на кнопке Submit new, чтобы настроить параметры сценария установки библиотек. В поле Script type выберите значение Custom, в поле Name введите libraries, а в поле Bash script URI введите адрес:

   http://deitel.com/bookresources/IntroToPython/install_libraries.sh

4. Включите варианты Head и Worker, чтобы сценарий установил библиотеки на всех узлах.
5. Щелкните на кнопке Create.

Когда кластер завершит выполнение сценария, в случае успешного выполнения появится зеленая метка рядом с именем сценария в списке действий. В противном случае Azure сообщит об ошибках.

Копирование файла RomeoAndJuliet.txt в кластер HDInsight

Как и с демонстрационным приложением Hadoop, воспользуемся командой `scp` для отправки в кластер файла RomeoAndJuliet.txt, который использовался в главе 11. В приглашении командной строки, терминале или командной оболочке перейдите в каталог с файлом (предполагается, что это каталог ch16) и введите приведенную ниже команду. Замените *ИмяКластера* именем, заданным при создании кластера, и нажмите клавишу Enter только после того, как будет введена вся команда. Двоеточие в этой команде обязательно; оно означает, что пароль кластера будет введен по запросу. Введите пароль, заданный при создании кластера, и нажмите клавишу Enter:

```
scp RomeoAndJuliet.txt sshuser@YourClus terName-ssh.azurehdinsight.net:
```

Затем используйте ssh, чтобы войти в кластер и получить доступ к командной строке. В приглашении командной строки, терминале или командной оболочке введите приведенную ниже команду. Не забудьте заменить *ИмяКластера* именем своего кластера. Вам снова будет предложено ввести пароль кластера:

```
ssh sshuser@YourClusterName-ssh.azurehdinsight.net
```

Для работы с файлом RomeoAndJuliet.txt в Spark в сеансе ssh скопируйте его в файловую систему Hadoop кластера, выполнив нижеследующую команду. Вновь возьмем существующий каталог /examples/data, включенный Microsoft для использования с учебными руководствами HDInsight. Не забудьте, что клавишу Enter следует нажимать только после того, как будет введена вся команда:

```
hadoop fs -copyFromLocal RomeoAndJuliet.txt
    /example/data/RomeoAndJuliet.txt
```

Обращение к документам Jupyter Notebook в HDInsight

На момент написания книги в HDInsight использовался *старый* интерфейс Jupyter Notebook вместо более нового интерфейса, представленного ранее. Краткий обзор старого интерфейса доступен по адресу:

https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Notebook%20Basics.html

Чтобы обратиться к документам Jupyter Notebook в HDInsight, на портале Azure выберите вариант All resources, а затем свой кластер. На вкладке Overview выберите вариант Jupyter notebook в разделе Cluster dashboards. При этом открывается окно веб-браузера, где предлагается ввести свои регистрационные данные. Используйте имя пользователя и пароль, заданный при создании кластера. Если вы не указали имя пользователя, то по умолчанию используется имя admin. После того как данные будут приняты, Jupyter отображает каталог с подкаталогами PySpark и Scala. В них находятся учебники Python и Scala Spark.

Отправка файла RomeoAndJulietCounter.ipynb

Чтобы создать новый документ Notebook, щелкните на кнопке New и выберите вариант PySpark3 или же отправьте существующие документы Notebook со своего компьютера. В нашем примере отправим файл RomeoAndJulietCounter.ipynb из предыдущего раздела и модифицируем его для работы с Azure. Для этого щелкните на кнопке Upload, перейдите в подкаталог SparkWordCount каталога

ch16, выберите файл RomeoAndJulietCounter.ipynb и щелкните на кнопке Open. На экране отображается файл в каталоге с кнопкой Upload справа. Щелкните на кнопке, чтобы поместить документ Notebook в текущий каталог. Затем щелкните на имени файла, чтобы открыть его в новой вкладке браузера. Jupyter открывает диалоговое окно Kernel not found. Выберите вариант PySpark3 и щелкните на кнопке OK, после чего следуйте инструкциям следующего параграфа.

Модификация блокнота для работы с Azure

Выполните следующие действия (каждая ячейка выполняется при завершении шага):

1. Кластер HDInsight не позволит NLTK сохранить загруженные игнорируемые слова в каталог NLTK по умолчанию, потому что он входит в число защищенных каталогов системы. В первой ячейке измените вызов `nltk.download('stopwords')` так, чтобы игнорируемые слова сохранялись в текущем каталоге (`'.'`):

   ```
   nltk.download('stopwords', download_dir='.')
   ```

2. При выполнении первой ячейки под ячейкой появится сообщение `Starting Spark application`, пока HDInsight создает объект `SparkContext` с именем `sc` за вас. Когда это будет сделано, код ячейки загрузит игнорируемые слова.

3. Во второй ячейке перед загрузкой игнорируемых слов необходимо сообщить NLTK, что они находятся в текущем каталоге. Включите следующую команду после команды `import`, запустив NLTK на поиск данных в текущем каталоге:

   ```
   nltk.data.path.append('.')
   ```

4. Поскольку HDInsight создает объект `SparkContext` за вас, третья и четвертая ячейки исходного документа не нужны, поэтому их можно удалить. Для этого либо щелкните внутри ячейки и выберите команду Delete Cells в меню Jupyter Edit, либо щелкните на белом поле слева от ячейки и введите dd.

5. В следующей ячейке укажите местонахождение файла RomeoAndJuliet.txt в файловой системе Hadoop. Замените строку `'RomeoAndJuliet.txt'` строкой

   ```
   'wasb:///example/data/RomeoAndJuliet.txt'
   ```

6. Синтаксис `wasb:///` означает, что файл RomeoAndJuliet.txt хранится в Windows Azure Storage Blob (WASB) — интерфейсе Azure к файловой системе HDFS.
7. Так как Azure в настоящее время использует Python 3.5.x, форматные строки не поддерживаются, и в последней ячейке следует заменить форматную строку следующей конструкцией строкового форматирования с вызовом метода `format`:

```
print('{:>{width}}: {}'.format(word, count, width=max_len))
```

Приложение выводит тот же результат, что и приложение из предыдущего раздела.

Внимание: не забудьте удалить свой кластер и другие ресурсы после завершения работы с ними, чтобы избежать лишних затрат. За дополнительной информацией обращайтесь по адресу:

https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-portal

Учтите, что при удалении ресурсов Azure *также будут удалены ваши документы Notebook*. Чтобы загрузить только что выполненный файл, выберите команду File ▸ Download as ▸ Notebook (.ipynb) в Jupyter.

16.7. Spark Streaming: подсчет хештегов Twitter с использованием стека Docker pyspark-notebook

В этом разделе мы создадим и запустим приложение Spark Streaming, которое будет получать поток твитов по заданной теме(-ам) и выводить сводку 20 самых популярных хештегов на гистограмме, обновляемой каждые 10 секунд. Для решения задачи используется контейнер Docker Jupyter из первого примера Spark. Пример состоит из двух частей. Сначала с использованием методов из главы 12 создадим сценарий, получающий поток твитов от Twitter. Затем используем механизм Spark Streaming в Jupyter Notebook для чтения твитов и обработки хештегов.

Эти две части будут общаться друг с другом через сетевые *сокеты* — низкоуровневое представление *сетевых взаимодействий «клиент/сервер»*, в котором *клиентское* приложение взаимодействует с серверным приложением по сети с использованием операций, сходных с операциями файлового ввода/вывода.

Программа может читать данные из сокета или записывать их в сокет почти так же, как при чтении из файла или записи в файл. Сокет представляет одну конечную точку сетевого соединения. В данном случае *клиентом* является приложение Spark, а *сервером* будет сценарий, который получает потоковые твиты и отправляет их приложению Spark.

Запуск контейнера Docker и установка Tweepy

Установим библиотеку Tweepy в контейнере Jupyter Docker. Выполните инструкции по запуску контейнера и установке в нем библиотеку Python из раздела 16.6.2. Установите Tweepy следующей командой:

```
pip install tweepy
```

16.7.1. Потоковая передача твитов в сокет

Сценарий starttweetstream.py содержит измененную версию класса `TweetListener` (глава 12). Она получает заданное количество твитов и отправляет их в сокет на локальном компьютере. По достижении лимита твитов сценарий закрывает сокет. Вы уже использовали потоковую передачу Twitter, поэтому сосредоточимся только на новых возможностях. Убедимся, что файл keys.py (подкаталог SparkHashtagSummarizer каталога ch16) содержит ваши регистрационные данные Twitter.

Выполнение сценария в контейнере Docker

Используем окно терминала JupyterLab для выполнения starttweetstream.py в одной вкладке, а затем Notebook — для выполнения задачи Spark в другой вкладке. При работающем контейнере Docker `pyspark-notebook` откройте адрес

http://localhost:8888/lab

в своем браузере. В JupyterLab выберите команду File ▸ New ▸ Terminal, чтобы открыть новую вкладку с терминалом (командной строкой Linux). Введите команду `ls` и нажмите Enter для вывода содержимого текущего каталога. По умолчанию выводится рабочий каталог контейнера.

Чтобы выполнить starttweetstream.py, необходимо перейти в каталог SparkHashtagSummarizer следующей командой[1]:

```
cd work/SparkHashtagSummarizer
```

[1] Пользователям Windows: в Linux для разделения имен каталогов используется символ / вместо \, а в именах каталогов учитывается регистр символов.

Теперь сценарий можно выполнить командой вида

```
ipython starttweetstream.py количество_твитов условия_поиска
```

где *количество_твитов* задает общее количество твитов для обработки, а *условия_поиска* — одна или несколько разделенных пробелами строк, используемых для фильтрации твитов. Так, следующая команда передает 1000 твитов о футболе:

```
ipython starttweetstream.py 1000 football
```

В этот момент сценарий выводит сообщение «Waiting for connection» и ожидает подключения к Spark, чтобы начать потоковую передачу твитов.

Команды импортирования starttweetstream.py

Для целей нашего обсуждения мы разделили starttweetstream.py на части. Сначала импортируются модули, используемые в сценарии. *Модуль* `socket` стандартной библиотеки Python предоставляет функциональность, которая позволяет приложениям Python взаимодействовать через сокеты.

```
1 # starttweetstream.py
2 """Сценарий для получения твитов по теме(-ам), заданной в аргументе(-ах),
3    и отправки текста твитов в сокет для обработки средствами Spark."""
4 import keys
5 import socket
6 import sys
7 import tweepy
8
```

Класс TweetListener

Большая часть кода класса `TweetListener` уже приводилась ранее, поэтому мы снова сосредоточимся исключительно на новых аспектах:

- Метод `__init__` (строки 12–17) теперь получает параметр `connection`, представляющий сокет, и сохраняет его в атрибуте `self.connection`. Сокет используется для отправки хештегов приложению Spark.
- В методе `on_status` (строки 24–44) строки 27–32 извлекают хештеги из объекта Tweepy `Status`, преобразуя их к нижнему регистру и создавая разделенную пробелами строку хештегов для отправки Spark. Самое важное происходит в строке 39:

```
self.connection.send(hashtags_string.encode('utf-8'))
```

Метод `send` объекта `connection` используется для отправки текста твита приложению, читающему данные из сокета. Метод `send` ожидает, что в первом аргументе передается последовательность байтов. Вызов метода `encode('utf-8')` преобразует строку в байты; Spark автоматически читает байты и воссоздает строки.

```
 9 class TweetListener(tweepy.StreamListener):
10     """Обрабатывает входной поток твитов."""
11
12     def __init__(self, api, connection, limit=10000):
13         """Создает переменные экземпляров для отслеживания количества
               твитов."""
14         self.connection = connection
15         self.tweet_count = 0
16         self.TWEET_LIMIT = limit  # 10,000 by default
17         super().__init__(api)  # call superclass's init
18
19     def on_connect(self):
20         """Вызывается в том случае, если попытка подключения была успешной,
21         чтобы вы могли выполнить нужные операции в этот момент."""
22         print('Successfully connected to Twitter\n')
23
24     def on_status(self, status):
25         """Вызывается, когда Twitter отправляет вам новый твит."""
26         # Получить хештеги
27         hashtags = []
28
29         for hashtag_dict in status.entities['hashtags']:
30             hashtags.append(hashtag_dict['text'].lower())
31
32         hashtags_string = ' '.join(hashtags) + '\n'
33         print(f'Screen name: {status.user.screen_name}:')
34         print(f'   Hashtags: {hashtags_string}')
35         self.tweet_count += 1  # Количество обработанных твитов
36
37         try:
38             # send необходимы байты, поэтому строка кодируется в формате utf-8
39             self.connection.send(hashtags_string.encode('utf-8'))
40         except Exception as e:
41             print(f'Error: {e}')
42
43         # При достижении TWEET_LIMIT вернуть False для завершения передачи
44         return self.tweet_count != self.TWEET_LIMIT
45
46     def on_error(self, status):
47         print(status)
48         return True
49
```

Главное приложение

Строки 50–80 выполняются при запуске сценария. Ранее мы уже устанавливали связь с Twitter, и здесь будут рассматриваться только новые аспекты.

Строка 51 получает количество твитов для обработки — аргумент командной строки `sys.argv[1]` преобразуется в целое число. Напомним, что элемент 0 представляет имя сценария.

```
50 if __name__ == '__main__':
51     tweet_limit = int(sys.argv[1])  # Получить максимальное количество твитов
```

В строке 52 вызывается *функция* `socket` модуля `socket`; она возвращает объект `socket`, используемый для ожидания подключения от приложения Spark.

```
52     client_socket = socket.socket()  # Создать сокет
53
```

В строке 55 *метод* `bind` объекта `socket` вызывается с передачей кортежа, содержащего имя хоста или IP-адрес компьютера и номер порта на этом компьютере. Комбинация этих значений определяет параметры ожидания сценарием исходного подключения от другого приложения:

```
54     # Приложение будет использовать порт 9876 локального хоста
55     client_socket.bind(('localhost', 9876))
56
```

Строка 58 вызывает *метод* `listen` сокета, который заставляет сценарий ожидать запрос на подключение. Именно эта команда не позволяет потоку Twitter стартовать до подключения приложения Spark.

```
57     print('Waiting for connection')
58     client_socket.listen()  # Ожидать подключения клиента
59
```

После того как приложение Spark подключится, строка 61 вызывает метод `accept` сокета, принимающий подключение. Этот метод возвращает кортеж с новым объектом `socket`, который будет использоваться сценарием для взаимодействия с приложением Spark, и IP-адресом компьютера с приложением Spark.

```
60     # При получении запроса на подключение получить подключение и адрес клиента
61     connection, address = client_socket.accept()
62     print(f'Connection received from {address}')
63
```

Затем выполняется аутентификация Twitter и запускается поток. Строки 73–74 настраивают поток, передавая объекту `TweetListener` объект `connection`, чтобы он мог использовать сокет для отправки хештегов приложению Spark.

```
64     # Настройка доступа к Twitter
65     auth = tweepy.OAuthHandler(keys.consumer_key, keys.consumer_secret)
66     auth.set_access_token(keys.access_token, keys.access_token_secret)
67
68     # Настройка Tweepy для перехода в ожидание при превышении ограничения
69     api = tweepy.API(auth, wait_on_rate_limit=True,
70                      wait_on_rate_limit_notify=True)
71
72     # Создать объект Stream
73     twitter_stream = tweepy.Stream(api.auth,
74         TweetListener(api, connection, tweet_limit))
75
76     # sys.argv[2] - первый критерий поиска
77     twitter_stream.filter(track=sys.argv[2:])
78
```

Наконец, в строках 79–80 вызывается метод `close` для объектов сокета, чтобы они освободили свои ресурсы.

```
79     connection.close()
80     client_socket.close()
```

16.7.2. Получение сводки хештегов и Spark SQL

В этом разделе воспользуемся Spark Streaming для чтения хештегов, отправленных через сокет сценарием starttweetstream.py, и обобщим результаты. Создайте новый документ Notebook и введите код, приведенный в тексте, либо загрузите файл hashtagsummarizer.ipynb из подкаталога SparkHashtagSummarizer каталога ch16.

Импортирование библиотек

Начнем с импортирования библиотек, используемых в документе Notebook. Будем описывать классы pyspark по мере их использования. Мы импортировали из IPython модуль `display`, который содержит классы и вспомогательные функции, которые могут использоваться в Jupyter. В частности, используем функцию `clear_output` для удаления существующей диаграммы перед выводом новой:

```
[1]: from pyspark import SparkContext
     from pyspark.streaming import StreamingContext
     from pyspark.sql import Row, SparkSession
     from IPython import display
     import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline
```

Это приложение Spark обрабатывает хештеги 10-секундными пакетами. После обработки каждого пакета оно выводит гистограмму Seaborn. Магическая команда IPython

```
%matplotlib inline
```

означает, что графика Matplotlib должна отображаться в документе Notebook, а не в отдельном окне. Напомним, что Seaborn использует Matplotlib.

В этой книге мы использовали всего несколько магических команд IPython. Между тем существует много магических команд, предназначенных для использования в документах Jupyter Notebook. Их полный список доступен по адресу:

https://ipython.readthedocs.io/en/stable/interactive/magics.html

Вспомогательная функция для получения объекта SparkSession

Как вы вскоре увидите, для запроса данных из наборов RDD может использоваться *Spark SQL*. Spark SQL использует коллекцию Spark `DataFrame` для получения табличного представления базовых RDD. Объект `SparkSession` (модуль pyspark.sql) используется для создания коллекции `DataFrame` из RDD.

В каждом приложении Spark может быть только один объект `SparkSession`. Следующая функция, позаимствованная нами из руководства «*Spark Streaming Programming Guide*»[1], показывает, как правильно получить экземпляр `SparkSession`, если он уже существует, или создать его, если он еще не был создан[2]:

---

[1] https://spark.apache.org/docs/latest/streaming-programming-guide.html#dataframe-and-sql-operations.

[2] Так как функция была позаимствована из документации (https://spark.apache.org/docs/latest/streaming-programming-guide.html#dataframe-and-sql-operations), мы не стали переименовывать

```
[2]: def getSparkSessionInstance(sparkConf):
         """Рекомендованный способ получения существующего или создания
            нового объекта SparkSession из Spark Streaming Programming Guide."""
         if ("sparkSessionSingletonInstance" not in globals()):
             globals()["sparkSessionSingletonInstance"] = SparkSession \
                 .builder \
                 .config(conf=sparkConf) \
                 .getOrCreate()
         return globals()["sparkSessionSingletonInstance"]
```

Вспомогательная функция для вывода гистограммы по данным коллекции Spark DataFrame

После обработки каждого пакета хештегов вызывается функция `display_barplot`. Каждый вызов стирает предшествующую гистограмму Seaborn, а затем строит новую гистограмму на основании полученной коллекции Spark `DataFrame`. Сначала мы вызываем метод `toPandas` коллекции Spark `DataFrame`, чтобы преобразовать коллекцию в Pandas `DataFrame` для использования с Seaborn. Затем вызывается *функция* `clear_output` из модуля `IPython.display`. Ключевой аргумент `wait=True` означает, что функция должна удалить предыдущую диаграмму (если она есть), но только по готовности новой диаграммы к выводу. В остальном коде функции используются стандартные средства Seaborn, продемонстрированные ранее. Вызов функции `sns.color_palette('cool', 20)` выбирает 20 столбцов из цветовой палитры `'cool'` библиотеки Matplotlib:

```
[3]: def display_barplot(spark_df, x, y, time, scale=2.0, size=(16, 9)):
         """Возвращает содержимое Spark DataFrame в виде гистограммы."""
         df = spark_df.toPandas()

         # Удалить предыдущую диаграмму, когда новая будет готова к выводу
         display.clear_output(wait=True)
         print(f'TIME: {time}')

         # Создать и настроить объект Figure с гистограммой Seaborn
         plt.figure(figsize=size)
         sns.set(font_scale=scale)
         barplot = sns.barplot(data=df, x=x, y=y
                               palette=sns.color_palette('cool', 20))

         # Повернуть метки оси x на 90 градусов для удобства чтения
         for item in barplot.get_xticklabels():
             item.set_rotation(90)
```

ее в соответствии со стандартным форматом имен функций Python или использовать одинарные кавычки как ограничители строк.

```
        plt.tight_layout()
        plt.show()
```

Вспомогательная функция для обобщения 20 самых популярных хештегов

В Spark Streaming объект `DStream` — последовательность RDD, каждый из которых представляет мини-пакет данных, предназначенных для обработки. Вы можете задать функцию, вызываемую для выполнения операции с каждым RDD в потоке. В этом пакете функция `count_tags` обобщает хештеги в заданном наборе RDD, добавляет их к текущим счетчикам (которые поддерживает `SparkSession`), после чего выводит обновленную гистограмму 20 хештегов, чтобы вы видели, как состав самых популярных хештегов меняется со временем[1]. Для целей нашего обсуждения функция разбита на несколько меньших частей. Получим объект `SparkSession` вызовом вспомогательной функции `getSparkSessionInstance` с данными конфигурации `SparkContext`. Каждый набор RDD предоставляет доступ к `SparkContext` в атрибуте `context`:

```
[4]: def count_tags(time, rdd):
        """Подсчет хештегов и вывод начальных 20 в порядке по убыванию."""
        try:
            # Получить объект SparkSession
            spark = getSparkSessionInstance(rdd.context.getConf())
```

Затем вызывем метод `map` объекта RDD для отображения данных в RDD на объекты `Row` (из пакета `pyspark.sql`). RDD в данном примере содержат кортежи из хештегов и счетчиков. Конструктор `Row` использует имена ключевых аргументов для определения имен столбцов каждого значения в этой строке. В данном случае `tag[0]` — хештег из кортежа, а `tag[1]` — счетчик для данного хештега:

```
            # Отобразить кортежи с хештегом и счетчиком на Row
            rows = rdd.map(
                lambda tag: Row(hashtag=tag[0], total=tag[1]))
```

Следующая команда создает коллекцию Spark `DataFrame` с объектами `Row`. Она используется в сочетании со Spark SQL для выборки 20 самых популярных с их счетчиками:

---

[1] Если к моменту первого вызова этой функции еще не было получено ни одного твита с хештегом, может появиться сообщение об ошибке. Это объясняется тем, что мы просто выводим сообщение об ошибке в стандартный поток вывода. Сообщение исчезнет сразу же после появления твитов с хештегами.

```
# Создать DataFrame для объектов Row
hashtags_df = spark.createDataFrame(rows)
```

Чтобы сформировать запрос к коллекции Spark `DataFrame`, создайте *табличное представление*, позволяющее Spark SQL выдавать запросы к `DataFrame` как к таблице реляционной БД. Метод `createOrReplaceTempView` коллекции Spark `DataFrame` создает временное табличное представление для `DataFrame` и присваивает представлению имя для использования в условии `from` запроса:

```
# Создать временное табличное представление для Spark SQL
hashtags_df.createOrReplaceTempView('hashtags')
```

Когда у вас появится табличное представление, можно запросить данные с использованием Spark SQL[1]. Следующая команда использует метод `sql` экземпляра `SparkSession` для выполнения запроса Spark SQL, который выбирает столбцы `hashtag` и `total` из табличного представления `hashtags`, упорядочивает выбранные строки по убыванию (`desc`) `total`, после чего возвращает первые 20 строк результата (`limit 20`). Spark SQL возвращает новую коллекцию Spark `DataFrame` с результатами:

```
# Использовать Spark SQL для получения 20 начальных хештегов
# при сортировке по убыванию
top20_df = spark.sql(
    """выбрать хештег, итого
       из общего количества итого
       хештег limit 20""")
```

Наконец, передадим коллекцию Spark `DataFrame` вспомогательной функции `display_barplot`. Хештеги и счетчики будут выводиться на осях *x* и *y* соответственно. Также приложение выводит время вызова `count_tags`:

```
        display_barplot(top20_df, x='hashtag', y='total', time=time)
    except Exception as e:
        print(f'Exception: {e}')
```

Получение объекта SparkContext

Остальной код документа Notebook настраивает Spark Streaming для чтения текста от сценария и указывает, как должны обрабатываться твиты. Сначала создается объект `SparkContext` для подключения к кластеру Spark:

```
[5]: sc = SparkContext()
```

[1] За подробной информацией о синтаксисе Spark SQL обращайтесь по адресу https://spark.apache.org/sql/.

Получение объекта StreamingContext

Для Spark Streaming необходимо создать объект `StreamingContext` (модуль `pyspark.streaming`), передав в аргументах объект `SparkContext` и частоту обработки пакетов потоковых данных в секундах (пакетный интервал). В нашем приложении пакеты будут обрабатываться каждые 10 секунд:

```
[6]: ssc = StreamingContext(sc, 10)
```

В зависимости от скорости поступления данных пакетный интервал можно увеличить или уменьшить. За обсуждением этого и других аспектов, связанных с эффективностью, обращайтесь к разделу «Performance Tuning» руководства *Spark Streaming Programming Guide*:

https://spark.apache.org/docs/latest/streaming-programming-guide.html#performance-tuning

Создание контрольной точки для хранения состояния

По умолчанию Spark Streaming не поддерживает информацию состояния в процессе обработки RDD. Тем не менее можно воспользоваться механизмом *контрольных точек* RDD для отслеживания состояния. Назовем важнейшие возможности контрольных точек:

- отказоустойчивость для перезапуска потока в случае сбоя узлов кластера или приложений Spark;
- преобразования с состоянием — например, обобщение данных, полученных к настоящему моменту, как это делается в нашем примере.

Метод `checkpoint` объекта `StreamingContext` создает каталог для контрольных точек:

```
[7]: ssc.checkpoint('hashtagsummarizer_checkpoint')
```

Для приложения Spark Streaming в облачном кластере задается путь HDFS для хранения каталога контрольных точек. Наш пример выполняется в локальном образе Jupyter Docker, поэтому мы просто задаем имя каталога, который Spark создаст в текущем каталоге (в нашем случае SparkHashtagSummarizer в каталоге ch16). За дополнительной информацией о контрольных точках обращайтесь по адресу:

https://spark.apache.org/docs/latest/streaming-programming-guide.html#checkpointing

Подключение к потоку через сокет

Метод `socketTextStream` объекта `StreamingContext` подключается к сокету, от которого будет поступать поток данных, и возвращает объект `DStream` для получения данных. В аргументах метода передается имя хоста и номер порта, по которому должен создавать подключение объект `StreamingContext`, — они должны соответствовать параметрам, по которым сценарий starttweetstream.py ожидает подключения:

```
[8]: stream = ssc.socketTextStream('localhost', 9876)
```

Разбиение хештегов на лексемы

Используем вызовы программирования в функциональном стиле для `DStream`, чтобы определить этапы обработки потоковых данных. Следующий вызов метода `flatMap` объекта `DStream` осуществляет разбиение строки хештегов, разделенных пробелами, и возвращает новый объект `DStream`, представляющий отдельные теги:

```
[9]: tokenized = stream.flatMap(lambda line: line.split())
```

Отображение хештегов на кортежи пар «хештег-счетчик»

Затем по аналогии со сценарием отображения Hadoop, приведенным ранее в этой главе, используем метод `map` объекта `DStream` для получения нового объекта `DStream`, в котором каждый хештег отображается на пару «хештег-счетчик» (в данном случае кортеж), в которой счетчик изначально равен 1:

```
[10]: mapped = tokenized.map(lambda hashtag: (hashtag, 1))
```

Суммирование счетчиков хештегов

Метод `updateStateByKey` объекта `DStreamп` получает лямбда-выражение с двумя аргументами, которое суммирует счетчики для заданного ключа и прибавляет их к предыдущей сумме для этого ключа:

```
[11]: hashtag_counts = tokenized.updateStateByKey(
          lambda counts, prior_total: sum(counts) + (prior_total or 0))
```

Определение метода, вызываемого для каждого RDD

Наконец, используем метод `foreachRDD` объекта `DSteam` для указания того, что каждый обработанный набор RDD должен быть передан функции `count_tags`, которая выделяет 20 самых популярных хештегов и строит гистограмму:

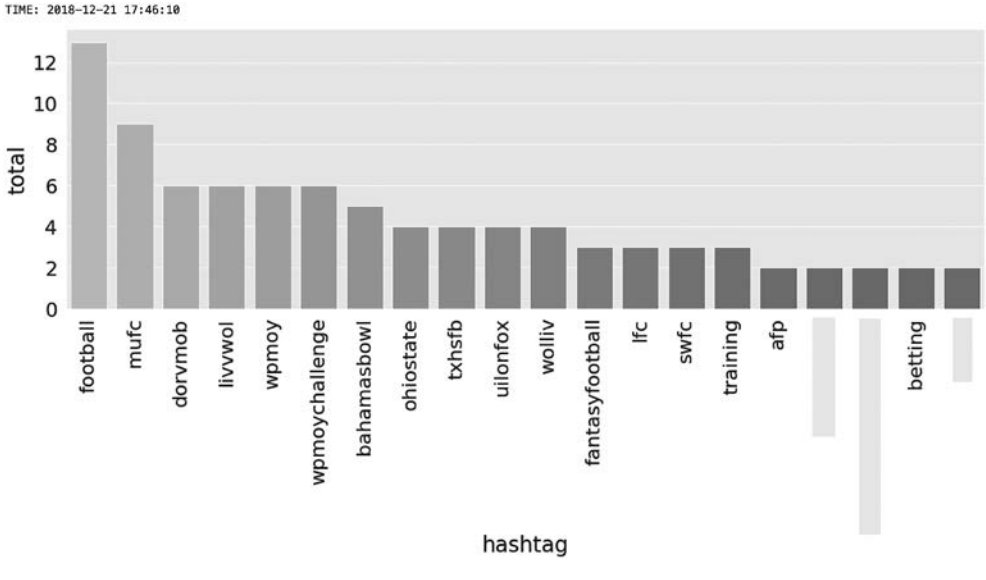
```
[12]: hashtag_counts.foreachRDD(count_tags)
```

Запуск потоковой передачи Spark

Определив процедуру обработки, вызовем метод `start` объекта `StreamingContext` для подключения к сокету и запуска процесса потоковой передачи.

```
[13]: ssc.start()  # Запустить Spark Streaming
```

На следующей иллюстрации изображена гистограмма, полученная при обработке потока твитов о футболе. Так как в США и во всем остальном мире словом «футбол» называются две разные игры, хештеги относятся как к американскому футболу, так и к игре, для которой в США используется название «соккер», — закрасим серым фоном три хештега, не подходящих для публикации:



16.8. «Интернет вещей»

В конце 1960-х годов интернет начинался в виде сети ARPANET, которая изначально объединяла четыре университета. Через 10 лет ее размер увеличился до 10 узлов[1]. Теперь же «глобальная паутина» разрослась до миллиардов компьютеров, смартфонов, планшетов и множества других типов устройств, подключенных к интернету по всему миру. *Любое* устройство, подключенное к интернету, рассматривается как «вещь» в концепции «интернета вещей» (IoT).

Каждое устройство обладает уникальным адресом протокола интернета (IP-адресом), который его идентифицирует. Стремительный рост количества подключенных устройств исчерпал приблизительно 4,3 миллиарда доступных адресов IPv4 (протокол IP версии 4)[2], что привело к необходимости разработки протокола IPv6, поддерживающего приблизительно $3{,}4 \times 10^{38}$ адресов[3].

«Ведущие исследовательские компании, такие как Gartner и McKinsey, прогнозируют скачок от 6 миллиардов подключенных устройств, существу-

Таблица 16.2. Основные типы устройств IoT

Устройства IoT	
Amazon Echo (Alexa), Apple HomePod (Siri), Google Home (Google Assistant) Автономные автомобили Датчики — химические, газовые, GPS, влажности, давления, температуры, ... Датчики цунами Устройства отслеживания активности — Apple Watch, FitBit, ... Домашние устройства — печи, кофеварки, холодильники, ... Здравоохранение — глюкометры для диабетиков, датчики давления, электрокардиограммы (ЭКГ), электроэнцефалограммы (ЭЭГ), кардиомониторы, системы внутреннего контроля, кардиостимуляторы, датчики сна, ...	Кнопки заказов Amazon Dash Оборудование беспроводных сетей Сейсмические датчики Системы охлаждения винных погребов Умный дом — свет, системы открывания гаража, видеокамеры, звонки, системы управления поливом, устройства безопасности, умные замки, датчики задымления, термостаты, вентиляция Устройства наблюдения

[1] https://en.wikipedia.org/wiki/ARPANET#History.
[2] https://en.wikipedia.org/wiki/IPv4_address_exhaustion.
[3] https://en.wikipedia.org/wiki/IPv6.

ющих в наши дни, до 20–30 миллиардов к 2020 году»[1]. Различные прогнозы утверждают, что это число может достигнуть 50 миллиардов. Подключенные к интернету устройства продолжают развиваться. В табл. 16.2 перечислены лишь немногие типы устройств и варианты применения IoT.

Проблемы IoT

Хотя мир IoT открывает много интересных возможностей, не все они положительные. Существует множество проблем в области безопасности, конфиденциальности и этики. Так, незащищенные устройства IoT использовались для проведения распределенных атак отказа в обслуживании (DDOS) на компьютерные системы[2]. Если видеокамеры безопасности, которые должны защищать ваш дом, будут взломаны, то другие пользователи получат доступ к вашему видеопотоку. Голосовые «шпионские» устройства постоянно ведут прослушивание, чтобы распознать управляющие слова. Дети случайно заказывали товары на Amazon, разговаривая с устройствами Alexa, а компании создавали телевизионную рекламу, которая активизировала устройства Google Home управляющими словами и заставляла Google Assistant зачитывать страницы «Википедии» с описанием товаров[3]. Люди беспокоятся, что эти устройства могут быть использованы и для подслушивания. Наконец, сравнительно недавно стало известно о том, что один судья запросил у Amazon записи Alexa для использования в криминальном судебном процессе[4].

Примеры этого раздела

В этом разделе рассматривается *модель публикации/подписки*, которая используется IoT и другими типами приложений для организации взаимодействий. Сначала без написания какого-либо кода мы построим информационную панель на базе Freeboard.io и подпишемся на поток от сервиса PubNub. Затем будет построено приложение, моделирующее термостат с подключением к интернету; моделируемое устройство публикует сообщения в бесплатном сервисе Dweet.io при помощи Python-модуля Dweepy. После этого будет создана визуализация данных на базе Freeboard.io. В завершение построим клиента Python,

---

[1] https://www.pubnub.com/developers/tech/how-pubnub-works/.

[2] https://threatpost.com/iot-security-concerns-peaking-with-no-end-in-sight/131308/.

[3] https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-security-voice-activated-smart-speakers-en.pdf.

[4] https://techcrunch.com/2018/11/14/amazon-echo-recordings-judge-murder-case/.

который подписывается на поток от сервиса PubNub и строит динамическую визуализацию потока средствами Seaborn и Matplotlib `FuncAnimation`.

16.8.1. Публикация и подписка

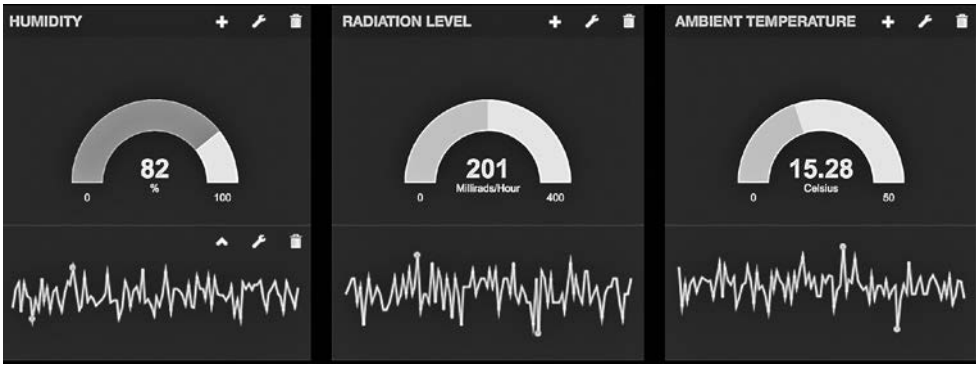
Устройства IoT (и многие другие разновидности устройств и приложений) обычно взаимодействуют друг с другом и с другими приложениями через *системы публикации/подписки*. *Публикатором* является любое устройство или приложение, отправляющее сообщение сервису на базе облака, который, в свою очередь, отправляет это сообщение всем *подписчикам*. Как правило, каждый публикатор указывает тему или канал, а каждый подписчик выбирает одну или несколько *тем* или *каналов*, для которых они хотели бы получать сообщения. В наше время существует много систем публикации/подписки. В оставшейся части этого раздела будут использоваться PubNub и Dweet.io. Мы также рекомендуем познакомиться с Apache Kafka — компонентом экосистемы Hadoop, предоставляющим высокопроизводительный сервис публикации/подписки, средства обработки потоков в реальном времени и хранения потоковых данных.

16.8.2. Визуализация живого потока PubNub средствами Freeboard

PubNub — сервис публикации/подписки, адаптированный для приложений реального времени, в которых произвольные программные компоненты и устройства, подключенные к интернету, общаются посредством малых сообщений. Некоторые стандартные применения таких систем — IoT, чаты, онлайновые многопользовательские игры, социальные сети и приложения для совместной работы. PubNub предоставляет несколько живых потоков для учебных целей, включая поток, моделирующий датчики IoT (другие перечислены в разделе 16.8.5).

Одно из распространенных применений живых потоков данных — визуализация для целей отслеживания. В этом разделе мы подключим смоделированный поток живых данных от датчика PubNub к информационной панели Freeboard.io. Приборная панель автомобиля используется для наглядного представления данных от датчиков машины: температуры окружающий среды, скорости, температуры двигателя, времени, объема оставшегося бензина и т. д. Информационная панель решает ту же задачу для данных из разных источников, включая устройства IoT.

Freeboard.io — средство динамической визуализации данных на информационных панелях для облачных сред. Даже без написания какого-либо кода Freeboard.io можно легко соединить с различными потоками данных и визуализировать данные по мере поступления. На следующей информационной панели визуализируются данные от трех из четырех датчиков IoT-потока, моделируемого средствами PubNub:



С каждым датчиком для визуализации данных используются представления `Gauge` (полукруг) и `Sparkline` (спарклайны). После выполнения инструкций из этого раздела вы увидите, что показания на полукруглых шкалах и линии быстро двигаются, так как новые данные поступают несколько раз в секунду.

Кроме платного сервиса, Freeboard.io предоставляет версию с открытым кодом (с меньшим набором возможностей) на GitHub. Также имеются учебники, показывающие, как пользоваться *плагинами расширения*; все это позволит вам самостоятельно разрабатывать визуализации для своих информационных панелей.

Подписка на Freeboard.io

Зарегистрируйтесь на сайте Freeboard.io и получите 30-дневный пробный период:

https://freeboard.io/signup

После этого откроется страница My Freeboards. При желании щелкните на кнопке Try a Tutorial и создайте визуализацию данных с вашего смартфона.

Создание новой информационной панели

Введите в поле поиска Enter a name в правом верхнем углу страницы My Freeboards строку `Sensor Dashboard`, после чего щелкните на кнопке Create New, открывающей конструктор информационной панели.

Добавление источника данных

Если вы добавили свой источник(-и) данных перед построением информационной панели, то сможете настроить каждую визуализацию при ее создании:

1. В разделе DATASOURCES щелкните на кнопке ADD, чтобы выбрать новый источник данных.
2. В раскрывающемся списке TYPE диалогового окна DATASOURCE выводится список источников данных, поддерживаемых в настоящий момент, хотя вы также можете разработать собственные плагины для новых источников данных[1]. Выберите вариант PubNub. Открывается страница с параметрами канала (Channel) и ключом подписки (Subscribe key) каждого живого потока PubNub. Скопируйте значения со страницы PubNub Sensor Network по адресу https://www.pubnub.com/developers/realtime-data-streams/sensor-network/, после чего вставьте их значения в соответствующих полях диалогового окна `DATASOURCE`. Введите имя своего источника данных (NAME) и щелкните на кнопке SAVE.

Добавление панели для датчика влажности

Информационная панель Freeboard.io разделена на субпанели, используемые для группировки визуализаций. Их можно перетаскивать мышью, располагая так, как вам нужно. Щелкните на кнопке +Add Pane, чтобы добавить новую панель. У каждой панели может быть свой заголовок. Для установки щелкните на значке с гаечным ключом, введите заголовок Humidity в поле TITLE, а затем щелкните на кнопке SAVE.

Добавление шкалы на панель Humidity

Чтобы добавить визуализации на панель, щелкните на кнопке +. На экране появляется диалоговое окно WIDGET. В раскрывающемся списке TYPE содержатся

---

[1] Некоторые из перечисленных источников данных доступны только через Freeboard.io, но не поддерживаются версией Freeboard с открытым кодом на GitHub.

некоторые встроенные виджеты. Выберите вариант Gauge. Справа от поля VALUE щелкните на кнопке +DATASOURCE, затем выберите имя своего источника данных. Открывается список доступных значений из этого источника данных. Щелкните на варианте humidity, чтобы выбрать значение от датчика влажности. В поле UNITS выберите вариант % и щелкните на кнопке SAVE. Появляется новая визуализация, которая немедленно начинает отображать данные из потока датчика.

Обратите внимание: значение отображается с четырьмя знаками в дробной части. PubNub поддерживает выражения JavaScript, которые могут использоваться для выполнения вычислений или форматирования данных. Например, вы можете воспользоваться функцией JavaScript `Math.round` для округления влажности до ближайшего целого числа. Для этого наведите указатель мыши на шкалу и щелкните на значке с гаечным ключом. Вставьте строку `"Math.round("` перед текстом в поле VALUE, вставьте `")"` после текста, затем щелкните на кнопке SAVE.

Добавление спарклайна на панель Humidity

Спарклайн представляет собой линейный график без осей, который дает представление об изменении значения данных со временем. Добавим спарклайн для датчика влажности; щелкните на кнопке + панели Humidity и выберите вариант Sparkline в раскрывающемся списке TYPE. В поле VALUE снова выберите источник данных и `humidity`, после чего щелкните на кнопке SAVE.

Завершение информационной панели

Повторив описанные выше действия, добавьте еще две панели и перетащите их справа от первой. Присвойте им названия `Radiation Level` и `Ambient Temperature` соответственно и разместите на каждой панели визуализации `Gauge` и `Sparkline` так, как показано выше. Для шкалы Radiation Level задайте в поле UNITS значение `Millirads/Hour`, а в поле MAXIMUM — значение 400. Для шкалы Ambient Temperature выберите в поле UNITS значение Celsius, а в поле MAXIMUM — значение 50.

16.8.3. Моделирование термостата, подключенного к интернету, в коде Python

Моделирование — одно из самых важных применений компьютеров. В одной из предыдущих глав моделировались результаты бросков кубиков. С IoT моделирование часто применяется для тестирования приложений, особенно

при отсутствии доступа к реальным устройствам и датчикам при разработке приложений. Сходные средства моделирования IoT предоставляются многими провайдерами облачных сервисов, например IBM Watson IoT Platform и IOTIFY.io.

В этом разделе будет создан сценарий, который моделирует подключенный к интернету термостат, публикующий периодические сообщения в формате JSON для `dweet.io`. Многие современные системы безопасности, подключенные к интернету, оснащаются температурными датчиками, которые могут выдавать предупреждения о слишком низкой или высокой температуре. Смоделированный нами датчик отправляет сообщения с данными местонахождения и температуры, а также оповещения о низкой или высокой температуре. Они срабатывают только в том случае, если температура достигает 3° или 35° по Цельсию соответственно. В следующем разделе мы воспользуемся `freeboard.io` для создания простой информационной панели, на которой отображаются изменения температуры при поступлении сообщений, а также индикаторы предупреждений о низкой или высокой температуре.

Установка Dweepy

Чтобы публиковать сообщения в `dweet.io` из кода Python, сначала необходимо установить библиотеку Dweepy:

```
pip install dweepy
```

Документацию библиотеки можно просмотреть по адресу:

https://github.com/paddycarey/dweepy

Запуск сценария simulator.py

Сценарий Python simulator.py, который моделирует термостат, находится в подкаталоге iot каталога ch16. Сценарий моделирования запускается с двумя аргументами командной строки, представляющими количество сообщений и задержку в секундах между их отправкой:

```
ipython simulator.py 1000 1
```

Отправка сообщений

Код simulator.py приведен ниже. Он использует генератор случайных чисел и средства Python, упоминавшиеся в книге, поэтому мы сосредоточимся

на нескольких строках кода, публикующих сообщения в сервисе `dweet.io` средствами Dweepy. Код сценария разбит на части для удобства рассмотрения.

По умолчанию `dweet.io` является общедоступным сервисом, так что любое приложение может публиковать сообщения или подписываться на них. При публикации сообщений *необходимо указать уникальное имя вашего устройства*. Мы использовали имя `'temperature-simulator-deitel-python'` (строка 17)[1]. В строках 18–21 определяется словарь Python, в котором будет храниться текущая информация от датчиков. Dweepy преобразует ее в формат JSON при отправке сообщения.

```
 1 # simulator.py
 2 """Модель подключенного к интернету термостата, который публикует
 3 сообщения JSON в dweet.io"""
 4 import dweepy
 5 import sys
 6 import time
 7 import random
 8
 9 MIN_CELSIUS_TEMP = -25
10 MAX_CELSIUS_TEMP = 45
11 MAX_TEMP_CHANGE = 2
12
13 # Получить количество моделируемых сообщений и задержку между ними
14 NUMBER_OF_MESSAGES = int(sys.argv[1])
15 MESSAGE_DELAY = int(sys.argv[2])
16
17 dweeter = 'temperature-simulator-deitel-python'  # Уникальное имя
18 thermostat = {'Location': 'Boston, MA, USA',
19               'Temperature': 20,
20               'LowTempWarning': False,
21               'HighTempWarning': False}
22
```

Строки 25–53 производят заданное число смоделированных сообщений. При каждой итерации цикла приложение:

- генерирует случайное изменение температуры в диапазоне от –2 до +2 и изменяет температуру;
- проверяет, что температура находится в разрешенном диапазоне;

---

[1] Чтобы имя было гарантированно уникальным, `dweet.io` может создать его за вас. В документации Dweepy объясняется, как это сделать.

- проверяет, сработал ли датчик низкой или высокой температуры, и обновляет словарь `thermostat` соответствующим образом;
- выводит количество сообщений, сгенерированных до настоящего момента;
- использует Dweepy для отправки сообщения dweet.io (строка 52);
- использует функцию `sleep` модуля `time` для ожидания заданного промежутка времени перед генерированием очередного сообщения.

```
23 print('Temperature simulator starting')
24
25 for message in range(NUMBER_OF_MESSAGES):
26     # Сгенерировать случайное число в диапазоне от -MAX_TEMP_CHANGE
27     # до MAX_TEMP_CHANGE и прибавить его к текущей температуре
28     thermostat['Temperature'] += random.randrange(
29         -MAX_TEMP_CHANGE, MAX_TEMP_CHANGE + 1)
30
31     # Убедиться в том, что температура остается в допустимом диапазоне
32     if thermostat['Temperature'] < MIN_CELSIUS_TEMP:
33         thermostat['Temperature'] = MIN_CELSIUS_TEMP
34
35     if thermostat['Temperature'] > MAX_CELSIUS_TEMP:
36         thermostat['Temperature'] = MAX_CELSIUS_TEMP
37
38     # Проверить предупреждение о низкой температуре
39     if thermostat['Temperature'] < 3:
40         thermostat['LowTempWarning'] = True
41     else:
42         thermostat['LowTempWarning'] = False
43
44     # Проверить предупреждение о высокой температуре
45     if thermostat['Temperature'] > 35:
46         thermostat['HighTempWarning'] = True
47     else:
48         thermostat['HighTempWarning'] = False
49
50     # Отправить сообщение dweet.io средствами dweepy
51     print(f'Messages sent: {message + 1}\r', end='')
52     dweepy.dweet_for(dweeter, thermostat)
53     time.sleep(MESSAGE_DELAY)
54
55 print('Temperature simulator finished')
```

Регистрация для использования сервиса необязательна. При первом вызове *функции* `dweet_for` библиотеки Dweepy для отправки сообщения (строка 52) `dweet.io` создает имя устройства. Функция получает в аргументе имя

устройства (`dweeter`) и словарь, представляющий отправляемое сообщение (`thermostat`). После выполнения сценария вы можете немедленно приступить к отслеживанию сообщений на сайте `dweet.io`, открыв следующий адрес в браузере:

```
https://dweet.io/follow/temperature-simulator-deitel-python
```

Если вы используете другое имя устройства, то замените `"temperature-simulator-deitel-python"` использованным именем. Веб-страница состоит из двух вкладок. На вкладке Visual показаны отдельные элементы данных со спарклайном для любых числовых значений. На вкладке Raw показаны фактические сообщения JSON, которые Dweepy отправляет dweet.io.

16.8.4. Создание информационной панели с Freeboard.io

Сайты `dweet.io` и `freeboard.io` принадлежат одной компании. На странице `dweet.io`, описанной в предыдущем разделе, кнопка Create a Custom Dashboard открывает новую вкладку браузера с уже реализованной информационной панелью по умолчанию для датчика температуры. По умолчанию `freeboard.io` настраивает источник данных с именем `Dweet` и автоматически генерирует информационную панель с одной панелью для каждого значения в JSON-разметке сообщения. Внутри каждой панели в текстовом виджете по мере поступления сообщений выводится соответствующее значение.

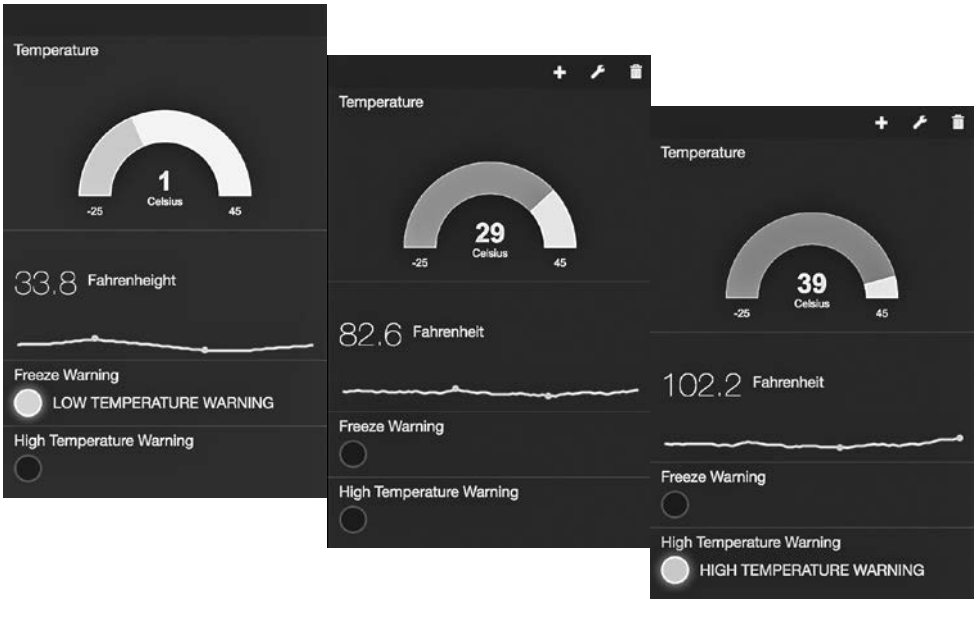
Если вы предпочитаете создать собственную информационную панель, то создайте источник данных так, как описано в разделе 16.8.2 (на этот раз выберите Dweepy): формируйте новые панели и виджеты или же внесите изменения в автоматически сгенерированную информационную панель. Ниже приведены три снимка экранов информационной панели, состоящей из четырех виджетов:

- Виджет Gauge с текущей температурой. Для параметра VALUE этого виджета мы выбрали поле `Temperature` источника данных. Параметру UNITS задано значение `Celsius`, а параметрам MINIMUM и MAXIMUM — –25 и 45 градусов соответственно. Виджет Text предназначен для вывода текущей температуры по шкале Фаренгейта. Для этого виджета мы присвоили параметрам INCLUDE SPARKLINE и ANIMATE VALUE CHANGES значение YES. Для параметра VALUE виджета выбрано поле `Temperature` источника данных, а в конец поля VALUE добавлен текст

  ```
  * 9 / 5 + 32
  ```

для выполнения вычислений, преобразующих значение по шкале Цельсия к шкале Фаренгейта. Также для параметра UNITS было выбрано значение `Fahrenheit`.

- Наконец, мы добавили виджеты Indicator Light. Для параметра VALUE первого виджета Indicator Light выбрано поле `LowTempWarning` источника данных, для параметра TITLE — значение `Freeze Warning`; для параметра ON TEXT выбрано значение `LOW TEMPERATURE WARNING`. ON TEXT показывает, какой текст должен отображаться при истинном значении. Для параметра VALUE второго виджета Indicator Light выбрано поле `HighTempWarning` источника данных, для параметра TITLE — значение `High Temperature Warning`; наконец, для параметра ON TEXT выбрано значение `HIGH TEMPERATURE WARNING`.



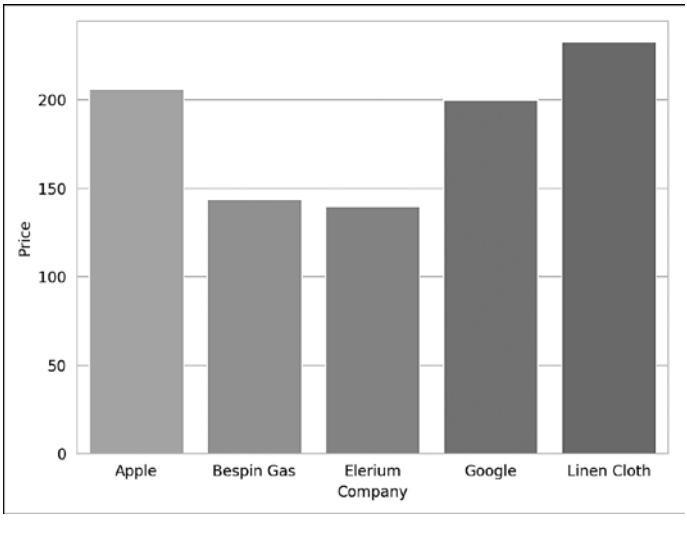
16.8.5. Создание подписчика PubNub в коде Python

PubNub предоставляет модуль Python `pubnub` для удобного выполнения операций публикации/подписки. Также предоставляются семь пробных потоков для экспериментов — четыре потока реального времени и три смоделированных потока[1]:

[1] https://www.pubnub.com/developers/realtime-data-streams/.

- Twitter Stream — предоставляет до 50 твитов в секунду из живого потока Twitter, не требуя регистрационных данных Twitter.
- Hacker News Articles — последние статьи с сайта.
- State Capital Weather — метеорологические данные для столиц штатов США.
- Wikipedia Changes — поток правок «Википедии».
- Game State Sync — смоделированные данные для многопользовательской игры.
- Sensor Network — смоделированные данные датчиков радиации, влажности, температуры и освещения.
- Market Orders — смоделированные заказы для пяти компаний.

В этом разделе мы используем *модуль* `pubnub` для подписки на смоделированный поток Market Orders, а затем построим визуализацию изменяющихся данных заказов на диаграмме Seaborn:



Вы можете публиковать сообщения и в потоке. Подробности — в документации модуля pubnub по адресу https://www.pubnub.com/docs/python/pubnub-python-sdk.

Чтобы подготовиться к использованию PubNub в Python, выполните следующую команду для установки новейшей версии модуля `pubnub` — часть

`'>=4.1.2'` гарантирует, что будет установлена как минимум версия 4.1.2 модуля pubnub:

```
pip install "pubnub>=4.1.2"
```

Сценарий stocklistener.py, который подписывается на поток и визуализирует данные заказов, находится в подкаталоге pubnub каталога ch16. Код сценария будет разбит на части для удобства обсуждения.

Формат сообщений

Смоделированный поток Market Orders возвращает объекты JSON, содержащие пять пар «ключ-значение» с ключами `'bid_price'`, `'order_quantity'`, `'symbol'`, `'timestamp'` и `'trade_type'`. В примере используются только ключи `'bid_price'` и `'symbol'`. Клиент PubNub возвращает данные JSON в формате словаря Python.

Импортирование библиотек

Строки 3–13 импортируют библиотеки, использованные в примере. Типы PubNub, импортированные в строках 10–13, будут рассматриваться по мере использования.

```
 1 # stocklistener.py
 2 """Визуализация живого потока PubNub."""
 3 from matplotlib import animation
 4 import matplotlib.pyplot as plt
 5 import pandas as pd
 6 import random
 7 import seaborn as sns
 8 import sys
 9
10 from pubnub.callbacks import SubscribeCallback
11 from pubnub.enums import PNStatusCategory
12 from pubnub.pnconfiguration import PNConfiguration
13 from pubnub.pubnub import PubNub
14
```

Список и коллекция DataFrame для хранения названий компаний и цен

Список `companies` содержит названия компаний, данные которых присутствуют в потоке Market Orders, а в коллекции `DataFrame` с именем `companies_df`

хранятся цены по каждой компании. Эта коллекция `DataFrame` будет использоваться для построения гистограммы средствами Seaborn.

```
15 companies = ['Apple', 'Bespin Gas', 'Elerium', 'Google', 'Linen Cloth']
16
17 # DataFrame для хранения последних цен
18 companies_df = pd.DataFrame(
19     {'company': companies, 'price' : [0, 0, 0, 0, 0]})
20
```

Класс SensorSubscriberCallback

Подписываясь на поток PubNub, вы должны добавить слушателя, который будет получать уведомления о статусе и сообщения от канала — по аналогии со слушателями Tweepy, которых вы определяли ранее. Для этого необходимо определить подкласс `SubscribeCallback` (модуль `pubnub.callbacks`), который будет рассмотрен ниже:

```
21 class SensorSubscriberCallback(SubscribeCallback):
22     """SensorSubscriberCallback получает сообщения от PubNub."""
23     def __init__(self, df, limit=1000):
24         """Создает переменные экземпляров для хранения количества твитов."""
25         self.df = df  # DataFrame для хранения последних цен
26         self.order_count = 0
27         self.MAX_ORDERS = limit  # 1000 по умолчанию
28         super().__init__()  # Вызов версии init суперкласса
29
30     def status(self, pubnub, status):
31         if status.category == PNStatusCategory.PNConnectedCategory:
32             print('Connected to PubNub')
33         elif status.category == PNStatusCategory.PNAcknowledgmentCategory:
34             print('Disconnected from PubNub')
35
36     def message(self, pubnub, message):
37         symbol = message.message['symbol']
38         bid_price = message.message['bid_price']
39         print(symbol, bid_price)
40         self.df.at[companies.index(symbol), 'price'] = bid_price
41         self.order_count += 1
42
43         # При достижении MAX_ORDERS отменить подписку на канал PubNub
44         if self.order_count == self.MAX_ORDERS:
45             pubnub.unsubscribe_all()
46
```

Метод `__init__` класса `SensorSubscriberCallback` сохраняет коллекцию `DataFrame`, в которую будут помещаться все новые цены. Клиент PubNub вы-

зывает переопределенный метод `status` каждый раз с поступлением нового сообщения статуса. В данном случае мы проверяем уведомления о подписке или отписке от канала.

Клиент PubNub вызывает переопределенный метод `message` (строки 36–45) при поступлении нового сообщения из канала. Строки 37 и 38 получают из сообщения название компании и цену, которые выводятся приложением, чтобы пользователь видел принимаемые сообщения. Строка 30 использует метод `at` коллекции `DataFrame` для нахождения строки соответствующей компании и ее столбца `'price'`, а затем присваивает этому элементу новую цену. После того как счетчик `order_count` достигнет `MAX_ORDERS`, строка 45 вызывает метод `unsubscribe_all` клиента PubNub, для того чтобы отписаться от канала.

Функция Update

В этом примере для визуализации цен применяются средства анимации, представленные в разделе «Введение в data science» главы 6. Функция `update` определяет прорисовку одного кадра анимации; она многократно вызывается объектом `FuncAnimation`, который будет определен ниже. Мы используем функцию `barplot` библиотеки Seaborn для визуализации данных от коллекции `DataFrame companies_df`, используя значения столбца `'company'` по оси x и значения столбца `'price'` по оси y.

```
47 def update(frame_number):
48     """Настраивает содержимое гистограммы для каждого кадра анимации."""
49     plt.cla()  # Стереть старую гистограмму
50     axes = sns.barplot(
51         data=companies_df, x='company', y='price', palette='cool')
52     axes.set(xlabel='Company', ylabel='Price')
53     plt.tight_layout()
54
```

Настройка объекта Figure

В основной части сценария сначала назначается стиль оформления диаграммы, после чего создается объект `Figure`, на котором будет выводиться гистограмма:

```
55 if __name__ == '__main__':
56     sns.set_style('whitegrid')  # Белый фон с серыми линиями сетки
57     figure = plt.figure('Stock Prices')  # Объект Figure для анимации
58
```

Настройка объекта FuncAnimation и отображение окна

Теперь настроим объект `FuncAnimation`, вызывающий функцию `update`, а затем и метод `show` библиотеки Matplotlib для отображения `Figure`. Обычно метод блокирует продолжение сценария до того, как вы закроете `Figure`. В данном случае передается ключевой аргумент `block=False`, чтобы сценарий продолжил выполнение, а мы могли настроить клиента PubNub и подписаться на канал.

```
59      # Настроить и запустить анимацию с вызовом функции update
60      stock_animation = animation.FuncAnimation(
61          figure, update, repeat=False, interval=33)
62      plt.show(block=False)  # display window
63
```

Настройка клиента PubNub

Затем мы настраиваем ключ подписки PubNub, используемый клиентом PubNub в сочетании с именем канала для подписки на канал. Ключ задается как атрибут объекта `PNConfiguration` (модуль `pubnub.pnconfiguration`), передаваемый в строке 69 новому объекту клиента `PubNub` (модуль `pubnub.pubnub`). Строки 70–72 создают объект `SensorSubscriberCallback` и передают его методу `add_listener` клиента `PubNub`, чтобы зарегистрировать его для получения сообщений от канала. Аргумент командной строки используется для определения количества обрабатываемых сообщений.

```
64      # Настроить ключ потока pubnub-market-orders
65      config = PNConfiguration()
66      config.subscribe_key = 'sub-c-4377ab04-f100-11e3-bffd-02ee2ddab7fe'
67
68      # Создать клиента PubNub и зарегистрировать SubscribeCallback
69      pubnub = PubNub(config)
70      pubnub.add_listener(
71          SensorSubscriberCallback(df=companies_df,
72              limit=int(sys.argv[1] if len(sys.argv) > 1 else 1000))
73
```

Подписка на канал

Следующая команда завершает процесс подписки, указывая, что мы хотим получать сообщения от канала с именем `'pubnub-market-orders'`. Метод execute запускает поток:

```
74      # Подписаться на канал pubnub-sensor-network и начать потоковую передачу
75      pubnub.subscribe().channels('pubnub-market-orders').execute()
76
```

Сохранение объекта Figure на экране

Второй вызов метода show библиотеки Matplotlib гарантирует, что объект `Figure` останется на экране, пока не будет закрыто окно.

```
77    plt.show()  # Диаграмма остается на экране, пока не будет закрыто окно
```

16.9. Итоги

В этой главе мы представили концепцию больших данных, рассказали, как развивается эта отрасль, и описали программную и аппаратную инфраструктуру для работы с большими данными. Были представлены традиционные реляционные базы данных и язык SQL, а модуль `sqlite3` использовался для создания и работы с базой данных `books` в SQLite. Также была продемонстрирована загрузка результатов запросов SQL в коллекции Pandas `DataFrame`.

Рассмотрены четыре основные разновидности баз данных NoSQL — базы данных «ключ-значение», документные, столбцовые и графовые базы данных, а также базы данных NewSQL. Объекты твитов JSON сохранялись в виде документов в кластере MongoDB Atlas на базе облака, а затем обобщались в интерактивной визуализации, отображаемой на карте Folium.

Далее была представлена технология Hadoop и ее применение в области больших данных. Мы настроили многоузловой кластер Hadoop с использованием сервиса Microsoft Azure HDInsight, после чего создали и выполнили задачу Hadoop MapReduce с применением потоковой передачи Hadoop.

Затем мы перешли к технологии Spark и ее применению для создания высокопроизводительных приложений больших данных, работающих в реальном времени. Мы использовали возможности программирования Spark в функциональном стиле «фильтрация/отображение/свертка» — сначала в стеке Jupyter Docker, работающем локально на вашем компьютере, а затем с многоузловым кластером Spark с использованием Microsoft Azure HDInsight. Затем был представлен механизм Spark Streaming для обработки данных мини-пакетами: в примере Spark SQL использовался для запросов на выборку данных, хранящихся в коллекции Spark `DataFrame`.

Глава завершается кратким введением в «интернет вещей» (IoT) и модель публикации/подписки. Средства Freeboard.io использовались для создания визуализации живого потока данных от сервиса. Мы смоделировали термо-

стат с подключением к интернету, который публикует сообщения на бесплатном сервисе `dweet.io` с использованием модуля Python Dweepy, а затем воспользовались Freeboard.io для визуализации данных смоделированного устройства. Наконец, мы подписались на поток PubNub при помощи модуля Python.

Спасибо всем читателям. Хочется верить, что книга вам понравилась, а материал показался интересным и содержательным. Надеемся, что благодаря прочитанному вы обретете необходимую уверенность для успешного применения технологий, рассмотренных в книге, и решения задач, с которыми столкнетесь в своей карьере.

Пол Дейтел, Харви Дейтел

Python: Искусственный интеллект, большие данные и облачные вычисления

Перевел с английского *Е. Матвеев*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*К. Тульцева*
Литературные редакторы	*М. Петруненко, М. Рогожин*
Художественный редактор	*В. Мостипан*
Корректоры	*Н. Викторова, М. Молчанова*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 03.2020. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 19.03.20. Формат 70×100/16. Бумага офсетная. Усл. п. л. 69,660. Тираж 1000. Заказ 0000.

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» предлагает профессиональную, популярную и детскую развивающую литературу

Заказать книги оптом можно в наших представительствах

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-83, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1, 6 этаж
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: hitsenko@piter.com

Екатеринбург: ул. Толедова, д. 43а; тел./факс: (343) 378-98-41, 378-98-42;
e-mail: office@ekat.piter.com; skype: ekat.manager2

Нижний Новгород: тел.: 8 930 712-75-13; e-mail: yashny@yandex.ru; skype: yashny1

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 277-89-66; e-mail: pitvolga@mail.ru,
pitvolga@samara-ttk.ru

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163; тел./факс: +37 517 208-80-01, 208-81-25;
e-mail: og@minsk.piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов:
тел./факс: (812) 703-73-72, (495) 234-38-15; e-mail: ivanovaa@piter.com
Подробная информация здесь: http://www.piter.com/page/avtoru

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок: *тел./факс: (812) 703-73-73; e-mail: sales@piter.com*

Заказ книг для вузов и библиотек:
тел./факс: (812) 703-73-73, доб. 6243; e-mail: uchebnik@piter.com

Заказ книг по почте: *на сайте www.piter.com; тел.: (812) 703-73-74, доб. 6216; e-mail: books@piter.com*

Вопросы по продаже электронных книг: *тел.: (812) 703-73-74, доб. 6217; e-mail: kuznetsov@piter.com*