1 Галузь людської інтелектуальної діяльності по застосуванню знань і навичок для проектування і створення процесів та технічних об'єктів, що реалізують такі процеси.

2 .<https://www.it-courses.by/basic-programming-principles/>

<https://techrocks.ru/2020/01/01/10-coding-principles-demystified/>

* **‌KISS Keep It Stupid Simple («Придерживайся простоты»**
* **‌DRY** Принцип Don’t Repeat Yourself (**«Не повторяйся»**, аббревиатура DRY в качестве отдельного слова означает «сухой»)
* **‌YAGNI** это определенно самая длинная аббревиатура в нашем списке. Принцип You Aren’t Gonna Need It (**«Тебе это не понадобится»**, YAGNI)
* **‌SLAP** Только представьте: код можно не только поцеловать (KISS) и высушить (DRY), но также и шлепнуть (SLAP)! **Single Level of Abstraction Principl**e (**«Принцип единого уровня абстракций»,** SLAP) диктует нам, как мы должны организовывать свой код (в частности, функции), чтобы он оставался поддерживаемым.Функции должны выполнять только одно действие, но выполнять его хорошо» (Роберт Мартин).
* **‌SRP** **Single Responsibility Principle**(**«Принцип единой ответственности»**, SRP) в чем-то похож на SLAP, но направлен на ofобъектно-ориентированное программирование. Этот принцип гласит, что объекты и классы (а также функции и методы) нужно организовывать так, чтобы каждый из них имел только одну зону ответственности. ***У КЛАССА ДОЛЖНА БЫТЬ ТОЛЬКО ОДНА ПРИЧИНА ДЛЯ ИЗМЕНЕНИЯ.***
* **‌OCP** Мы уже говорили о расширяемости кода, когда обсуждали YAGNI. **Open-Closed Principle** (**«Принцип открытости-закрытости»**, OCP) некоторым образом связан с предыдущим правилом, но представляет собой другой взгляд на вещи. OCP требует, чтобы код был открыт для новых, будущих дополнений, и чтобы при их добавлении не приходилось изменять уже написанный код. Этот принцип в большей степени затрагивает вопросы архитектуры, чем кода как такового.***ОБЪЕКТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (КЛАССЫ, МОДУЛИ, ФУНКЦИИ И Т.Д.) ДОЛЖНЫ БЫТЬ ОТКРЫТЫ ДЛЯ РАСШИРЕНИЯ, НО ЗАКРЫТЫ ДЛЯ МОДИФИКАЦИИ.***
* **‌LSP**. Liskov Substitution Principle (**«Принцип подстановки Барбары Лисков»**, LSP) назван в честь его автора, Барбары Лисков. Это принцип объектно-ориентированного программирования, касающийся классов, интерфейсов, типов и подтипов. Само по себе это правило довольно простое и логичное, но поначалу его может быть трудно применять на практике. Суть его в том, что каждый подтип должен дополнять, а не заменять базовый тип.***ПУСТЬ Φ(X) — СВОЙСТВО, ДОКАЗУЕМОЕ ДЛЯ ОБЪЕКТОВ X ТИПА T. ТОГДА Φ(Y) ДОЛЖНО БЫТЬ ИСТИННЫМ ДЛЯ ОБЪЕКТОВ Y ТИПА S, ГДЕ S — ПОДТИП T.***
* **‌ISP** **Interface Segregation Principle *КЛИЕНТЫ НЕ ДОЛЖНЫ ЗАВИСЕТЬ ОТ ИНТЕРФЕЙСОВ, КОТОРЫЕ ОНИ НЕ ИСПОЛЬЗУЮТ.***

 (**«Принцип разделения интерфейса»**, ISP) это еще один принцип, затрагивающий тему организации кода. У манипулирования объектами строго через интерфейс абстрактного класса есть два преимущества:

* клиенту не нужно иметь информации о конкретных типах объектов, которыми он пользуется, при условии, что все они имеют ожидаемый клиентом интерфейс;
* клиенту необязательно "знать" о классах, с помощью которых реализованы объекты. Клиенту известно только об абстрактном классе (или классах), определяющих интерфейс.
* Данные преимущества настолько существенно уменьшают число зависимостей между подсистемами, что можно даже сформулировать принцип объектно-ориентированного проектирования для повторного использования: программируйте в соответствии с интерфейсом, а не с реализацией.

* **‌DIP** Как и OCP, **Dependency Inversion Principle** (**«Принцип инверсии зависимостей»**, DIP. ***1. МОДУЛИ ВЫСОКОГО УРОВНЯ НЕ ДОЛЖНЫ ЗАВИСЕТЬ ОТ МОДУЛЕЙ НИЗКОГО УРОВНЯ. И ТЕ, И ДРУГИЕ ДОЛЖНЫ ЗАВИСЕТЬ ОТ АБСТРАКЦИЙ. 2. “АБСТРАКЦИИ НЕ ДОЛЖНЫ ЗАВИСЕТЬ ОТ ДЕТАЛЕЙ. ДЕТАЛИ ДОЛЖНЫ ЗАВИСЕТЬ ОТ АБСТРАКЦИЙ”.***
* **‌SOLID** Пять принципов, которые мы уже обсудили — SRP, OCP, LSP, ISP, DIP — вместе составляют набор принципов SOLID, описанный Робертом Мартином. Эти принципы способствуют созданию хорошего объектно-ориентированного (и не только) кода. <https://habr.com/ru/post/332578/>

**S.O.L.I.D принципы с примерами на Python** <https://gist.github.com/pavel-loginov-dev/8f3ef63e265c15763d169eff4627265d>

<https://nuancesprog.ru/p/14721/>

**3.Чим відрізняються процедурна та об’єкто-орієнтована парадигми програмування?**

Разница между ними тонкая, но существенная.

🎆В процедурной программе модули взаимодействуют путем чтения и записи состояния, которое хранится в общих структурах данных.

🎇В объектно-ориентированной программе модули в виде объектов взаимодействуют, отправляя сообщения другим объектам.

**Важность объектно-ориентированной парадигмы заключается не столько в языковом механизме, сколько в процессе мышления и проектирования .** В **процедурном программировании** речь идет **об операциях и разбиении этих операций на другие операции, группировании их в модули и** т. Д. Это означает, что данные или состояние имеют второстепенное значение. Это похоже на мышление о математических операциях.

С другой стороны, **объектно-ориентированная парадигма говорит, что вам нужно думать о состоянии и операциях вместе как о сущности, а затем проектировать свою программу как взаимодействие между сущностями, которые обмениваются состоянием и активируют операции.**

<https://coderoad.ru/530741/%D0%92-%D1%87%D0%B5%D0%BC-%D1%80%D0%B0%D0%B7%D0%BD%D0%B8%D1%86%D0%B0-%D0%BC%D0%B5%D0%B6%D0%B4%D1%83-%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B9-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BE%D0%B9-%D0%B8-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B9>

У **манипулирования объектами строго через интерфейс абстрактного класса есть два преимущества:**

* клиенту не нужно иметь информации о конкретных типах объектов, которыми он пользуется, при условии, что все они имеют ожидаемый клиентом интерфейс;

клиенту необязательно "знать" о классах, с помощью которых реализованы объекты. Клиенту известно только об абстрактном классе (или классах), определяющих интерфейс.

* Данные преимущества настолько существенно уменьшают число зависимостей между подсистемами, что можно даже сформулировать принцип объектно-ориентированного проектирования для повторного использования: программируйте в соостветствии с интерфейсом, а не с реализацией.

**Объектно-ориентированное программирование (ООП) и процедурное программирование** - это две парадигмы программирования.

**Парадигма программирования -** это **фундаментальный стиль компьютерного программирования,** который различается по способу представления различных элементов программы и определению шагов для решения проблем. Как следует из названия,

* **ООП** фокусируется на представлении проблем с **использованием реальных объектов и их поведения**,
* в то время как **процедурное программирование** занимается **представлением решений проблем с помощью процедур, которые представляют собой наборы кода, которые выполняются в определенном порядке**. Существуют языки программирования, которые поддерживают ключевые аспекты ООП (называемые языками ООП), процедурные (называемые процедурными языками) и то и другое. Но важно отметить, что ООП и процедурный - это два способа представления проблем, которые необходимо решить, и не имеет значения, какой язык используется. Другими словами, языки ООП могут использоваться для процедурного программирования, в то время как процедурные языки могут иногда использоваться для ООП, с некоторыми усилиями.

**4.Які основні принципи ООП (наслідування, інкапсуляція, поліморфізм)?**

<https://training.epam.ua/#!/News/275?lang=ua>

<http://pythoshka.ru/p1451.html>

[**Инкапсуляция**](https://younglinux.info/oopython/encapsulation)— ограничение доступа к составляющим объект компонентам (методам и переменным). Инкапсуляция делает некоторые из компонент доступными только внутри класса. Инкапсуляция в Python работает лишь на уровне соглашения между программистами о том, какие атрибуты являются общедоступными, а какие — внутренними .Под инкапсуляцией в объектно-ориентированном программировании понимается упаковка данных и методов для их обработки вместе, т. е. в классе. В Python инкапсуляция реализуется как на уровне классов, так и объектов. В ряде других языков, например в Java, под инкапсуляцией также понимают сокрытие свойств и методов, в результате чего они становятся приватными. Это значит, что доступ к ним ограничен либо пределами класса, либо модуля. В Python подобной инкапсуляции нет, хотя существует способ ее имитировать. Перед тем как выяснять, как это делается, надо понять, зачем вообще что-то скрывать.

[**Наследование**](https://younglinux.info/oopython/inheritance) подразумевает то, что дочерний класс содержит все атрибуты родительского класса, при этом некоторые из них могут быть переопределены или добавлены в дочернем.

[**Полиморфизм**](https://younglinux.info/oopython/polymorphism) в объектно-ориентированном программировании – это возможность обработки разных типов данных, т. е. принадлежащих к разным классам, с помощью "одной и той же" функции, или метода. На самом деле одинаковым является только имя метода, его исходный код зависит от класса. Кроме того, результаты работы одноименных методов могут существенно различаться. Поэтому в данном контексте под полиморфизмом понимается множество форм одного и того же слова – имени метода. **Полиморфизм** — разное поведение одного и того же метода в разных классах. Например, мы можем сложить два числа, и можем сложить две строки. При этом получим разный результат, так как числа и строки являются разными классами.

5. Що таке множинне наслідування?

Язык программирования **Python** являясь языком, поддерживающим парадигму [объектно-ориентированного программирования (ООП)](http://pythonicway.com/python-oop), также поддерживает и возможность **множественного наследования**. То есть, возможность у класса потомка наследовать функционал не от одного, а от нескольких родителей. Благодаря этому мы можем создавать сложные структуры, сохраняя простой и легко-поддерживаемый код.<http://pythonicway.com/education/python-oop-themes/35-python-multiple-inheritance>

**Множественное наследование в Python**

[**https://webdevblog.ru/mnozhestvennoe-nasledovanie-i-miksiny-v-python/**](https://webdevblog.ru/mnozhestvennoe-nasledovanie-i-miksiny-v-python/)

**6.Які є шість етапів розробки продукту в Software Development lifecycle і яка різниця між Agile і Kanban?**

* **Анализ требований** отвечает на вопрос «Какие проблемы требуют решений?»<https://habr.com/ru/company/dcmiran/blog/521718/>
* **Планирование** отвечает на вопрос «Что мы хотим сделать?»
* **Проектирование и дизайн** отвечает на вопрос «Как мы добьемся наших целей?»
* **Разработка ПО** регулирует процесс создания продукта.
* **Тестирование** регулирует обеспечение качественной работы продукта.
* **Развертывание** регулирует использование финального продукта.

**Жизненный цикл разработки ПО** (System/Software Development Life Cycle, SDLC) — процесс, состоящий из конкретных этапов, который начинается в момент принятия решения о необходимости создания программного продукта и заканчивается в момент прекращения поддержки ПО разработчиками. Переход между этапами определяется **моделью процессов разработки ПО**. Подробно цель, выходы, действия и задачи каждого из этих процессов описан в международном стандарте ISO/IEC 12207:2008 «System and software engineering — Software life cycle processes». 

<https://uk.myservername.com/weka-tutorial-how-download>

**Agile – это философия, scrum – структура, waterfall – метод, kanban – система управления. Scrum и kanban – варианты agile, но у них есть некоторые явные различия. Методика scrum требует фиксированных ролей, тогда как у kanban нет необходимых ролей.**

**Scrum** и **Kanban** — это гибкие методологии создания продукта. По ним можно работать в любой отрасли, но особенно хорошо они подходят для ИТ. В основе обеих методологий лежат принципы **Agile**.  Сам **Agile** (**agile software development**, от англ. agile – проворный) – это **семейство «гибких» подходов (методологий) к разработке программного обеспечения. Такие подходы также иногда называют фреймворками или agile-методологиями**   (Scrum, [Kanban](https://scrumtrek.ru/blog/kanban/5796/kanban-scrum-agile-otlichiya/), RUP, DSDM, FDD, XP, Crystal Clear).  Смысл Agile сформулирован в **Agile-манифесте** разработки ПО: «Люди и взаимодействие важнее процессов и инструментов. Работающий продукт важнее исчерпывающей документации. Сотрудничество с заказчиком важнее согласования условий контракта. Готовность к изменениям важнее следования первоначальному плану» и направлен  на быстрый выпуск продукта на рынок (сокращение TTM, Time-to-Market) за счет эффективного взаимодействия самоорганизованной команды. <https://babok-school.ru/blogs/sdlc-and-software-development-models-for-business-analyst/>

Говоря в терминах ООП, гибкие методологии являются классами, реализующими интерфейс Agile. А классы различных моделей разработки ПО реализуют интерфейс SDLC и связаны с методологиями через ассоциации. При этом отдельные методологии и модели разработки ПО можно считать объектами этих классов.



Как связаны модели и методологии разработки ПО: UML-диаграмма

К отдельным agile-подходам относятся **scrum** и **kanban**.      <https://web-academy.com.ua/stati/350-agile-scrum-kanban?fbclid=IwAR3E1FHMvNrMuSTMzvV8D88rcKUx9nA9MDCNWLSaNOQCYIKHkrOEyGLjP-o>

**Scrum – это «подход структуры».** Над каждым проектом работает универсальная команда специалистов, к которой присоединяется еще два человека: *владелец продукта* и *scrum-мастер*. Первый соединяет команду с заказчиком и следит за развитием проекта (это не формальный руководитель команды, а скорее куратор). Второй помогает первому организовать бизнес-процесс: проводит общие собрания, решает бытовые проблемы, мотивирует команду и следит за соблюдением scrum-подхода.

*Scrum-подход* делит рабочий процесс на равные **спринты** – обычно это периоды от недели до месяца, в зависимости от проекта и команды. Перед *спринтом* формулируются задачи на данный спринт, в конце – обсуждаются  результаты, а команда начинает новый спринт. Спринты очень удобно сравнивать между собой, что позволяет управлять эффективностью работы.

**Kanban – это «подход баланса».** Его задача – сбалансировать разных специалистов внутри команды и избежать ситуации, когда дизайнеры работают сутками, а разработчики жалуются на отсутствие новых задач.

Вся команда едина – в **kanban** нет ролей владельца продукта и scrum-мастера. Бизнес-процесс делится не на универсальные спринты, а на стадии выполнения конкретных задач: «Планируется», «Разрабатывается», «Тестируется», «Завершено» и др.

Главный показатель эффективности в kanban – это *среднее время прохождения задачи по доске*. Задача прошла быстро – команда работала продуктивно и слаженно. Задача затянулась – надо думать, на каком этапе и почему возникли задержки и чью работу надо оптимизировать.

**Когда-то давно Скрам и Канбан органично жили под одним брендом Agile. Однако, по мере развития Канбан-метода, его создатели поняли, что этот должен базироваться на других принципах, нежели Scrum**. Было отмечено, что Kanban-метод не вполне реализует 4 ценности Agile-манифеста, поэтому относить его к Agile-подходам неверно, и надо его из-под «зонтика Agile» вывести. Примерно в 2015-2017 годах создателями Канбан-метода было принято решение развивать концепцию Business Agility — в противовес [понятию Agile](https://scrumtrek.ru/blog/agile-scrum/4029/metodologiya-agile/). И с этого момента Канбан-метод начал позиционировать себя как альтернатива Agile-подходам, а не их часть. <https://scrumtrek.ru/blog/kanban/5796/kanban-scrum-agile-otlichiya/>

**7.Які є методи HTTP-запитів та яка між ними різниця?**

Разница между двумя основными методами HTTP-запроса GET и POST. Теги:  http  python [https://russianblogs.com/article/2414892906**/**](https://russianblogs.com/article/2414892906/)[**https://russianblogs.com/article/89871636032/**](https://russianblogs.com/article/89871636032/)

**Два метода HTTP-запроса: GET и POST**

В запросе-ответе между клиентом и сервером наиболее часто используются два метода: GET и POST.

* GET-запрос данных из указанного ресурса.
* POST - отправить данные для обработки на указанный ресурс

**GET метод**

**Обратите внимание, что строка запроса (пара имя / значение) отправляется в URL-адресе запроса GET:**

**/test/demo\_form.asp?name1=value1&name2=value2**

Некоторые другие примечания о запросах GET:

* GET-запросы можно кэшировать
* GET-запросы остаются в истории браузера
* GET запрос можно добавить в закладки
* Запросы GET не следует использовать при обработке конфиденциальных данных.
* Запрос GET имеет ограничение на длину
* Запросы GET следует использовать только для получения данных

**Метод POST**

**Обратите внимание, что строка запроса (пара имя / значение) отправляется в теле HTTP-сообщения запроса POST:**

* **POST /test/demo\_form.asp HTTP/1.1**
* **Host: w3schools.com**
* **name1=value1&name2=value2**

Некоторые другие примечания о запросах POST:

* Запросы POST не будут кэшироваться
* Запросы POST не останутся в истории браузера
* POST нельзя добавить в закладки
* Запрос POST не требует длины данных

**GET**

* Фильтры в интернет-магазинах
* Передача параметров через ссылку
* Другие безопасные запросы

**POST**

* Любые формы с паролями или банковскими картами
* Формы заявок с персональными данными
* Отправка файлов

[**https://russianblogs.com/article/89871636032**](https://russianblogs.com/article/89871636032)

**Вы легко дали «стандартный ответ»:**

* GET безвреден, когда браузер откатывается, в то время как POST отправляет запрос снова.
* URL-адрес, сгенерированный GET, можно добавить в закладки, но не POST.
* Запросы GET будут активно кэшироваться браузером, а запросы POST - нет, если они не установлены вручную.
* Запросы GET могут кодироваться только по URL-адресу, тогда как POST поддерживает несколько методов кодирования.
* Параметры запроса GET будут полностью сохранены в истории браузера, а параметры POST не сохранятся.
* Параметры, передаваемые в URL-адресе GET-запроса, ограничены по длине, но не для POST.
* Для типа данных параметра GET принимает только символы ASCII, тогда как POST не имеет ограничений.
* GET менее безопасен, чем POST, потому что параметры напрямую отображаются в URL-адресе, поэтому его нельзя использовать для передачи конфиденциальной информации.
* Параметры GET передаются через URL-адрес, а POST помещается в тело запроса.

**В чем разница между GET, POST и HEAD в протоколе HTTP?**

* ЗАГОЛОВОК: запрашивается только заголовок страницы.
* GET: запросить указанную информацию о странице и вернуть тело объекта.
* POST: запросить сервер принять указанный документ в качестве нового подчиненного объекта для указанного URL.
* (1) HTTP определяет различные методы взаимодействия с сервером, самые основные из них - GET и POST. Фактически, GET подходит для большинства запросов, в то время как зарезервированный POST используется только для обновления сайта.
* (2) При отправке ФОРМЫ, если метод не указан, по умолчанию используется запрос GET. Данные, представленные в форме, будут добавлены к URL-адресу и отделены от URL-адреса знаком?. Буквенно-цифровые символы отправляются как есть, но пробелы преобразуются в знаки «+», а другие символы преобразуются в% XX, где XX - это значение символа ASCII (или ISO Latin-1) в шестнадцатеричной системе счисления. Данные, отправленные запросом GET, помещаются в заголовок протокола HTTP-запроса, а данные, отправленные POST, помещаются в данные объекта;
* Данные, отправленные методом GET, могут иметь максимум 1024 байта, в то время как POST не имеет этого ограничения.
* (3) GET. Это наиболее часто используемый браузерами метод запроса сервера. Метод POST также используется для передачи данных, но, в отличие от GET, при использовании POST данные передаются не после URI, а в виде отдельной строки, которая должна быть передана, и в это время должен быть отправлен Content\_length. Заголовок указывает на длина данных, за которой следует пустая строка, а затем фактические переданные данные. Форма веб-страницы обычно отправляется POST.

**8.Як виглядають HTTP-request/response?**

**requests** — это библиотека, с помощью которой можно делать запросы в интернет. Само слово *“requests”* переводится как *запросы*.[Библиотека requests в Python | Devman](https://dvmn.org/encyclopedia/modules/requests/)

<https://developer.mozilla.org/ru/docs/Web/HTTP/Messages>

HTTP запросы и ответы имеют близкую структуру. Они состоят из:

1. Стартовой строки, описывающей запрос, или статус (успех или сбой). Это всегда одна строка.
2. Произвольного набора *HTTP заголовков,* определяющих запрос или описывающих тело сообщения.
3. Пустой строки, указывающей, что вся мета информация отправлена.
4. Произвольного тела, содержащего пересылаемые с запросом данные (например, содержимое HTML-формы ) или отправляемый в ответ документ. Наличие тела и его размер определяется стартовой строкой и заголовками HTTP.

Стартовую строку вместе с заголовками сообщения HTTP называют *головой* запроса, а его данные - *телом*.



**9.Що таке авторизація і як вона працює?**

HTTP предоставляет набор инструментов для разграничения доступа к ресурсам и авторизацией. Самой распространённой схемой HTTP авторизации является "Basic" (базовая) авторизация. Данное руководство описывает основные возможности HTTP авторизации и показывает способы ограничения доступа к вашему серверу с её использованием.<https://developer.mozilla.org/ru/docs/Web/HTTP/Authentication>

[**Общий механизм HTTP авторизации**](https://developer.mozilla.org/ru/docs/Web/HTTP/Authentication#%D0%BE%D0%B1%D1%89%D0%B8%D0%B9_%D0%BC%D0%B5%D1%85%D0%B0%D0%BD%D0%B8%D0%B7%D0%BC_http_%D0%B0%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8)

[RFC 7235](https://datatracker.ietf.org/doc/html/rfc7235) определяет средства HTTP авторизации, которые может использовать сервер для [запроса (en-US)](https://developer.mozilla.org/en-US/docs/Glossary/challenge) у клиента аутентификационной информации. Сценарий запрос-ответ подразумевает, что вначале сервер отвечает клиенту со статусом [401](https://developer.mozilla.org/ru/docs/Web/HTTP/Status/401) (Unauthorized) и предоставляет информацию о порядке авторизации через заголовок [WWW-Authenticate (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/WWW-Authenticate), содержащий хотя бы один метод авторизации. Клиент, который хочет авторизоваться, может сделать это, включив в следующий запрос заголовок [Authorization](https://developer.mozilla.org/ru/docs/Web/HTTP/Headers/Authorization) с требуемыми данными. Обычно, клиент отображает запрос пароля пользователю, и после получения ответа отправляет запрос с пользовательскими данными в заголовке Authorization.



В случае базовой авторизации как на иллюстрации выше, обмен **должен** вестись через HTTPS (TLS) соединение, чтобы обеспечить защищённость.

Вкратце, аутентификация проверяет, является ли пользователь тем, за кого себя выдает, а авторизация определяет, что разрешено делать аутентифицированному пользователю. [Аутентификация пользователей в Django](https://django.fun/docs/django/ru/3.2/topics/auth/)

**10.Що таке cookies?**

Файл **cookie** – это небольшой фрагмент текста, передаваемый в браузер с сайта, который вы посещаете. С его помощью сайт запоминает информацию о ваших посещениях. Cookie-файлы знают, где вы находитесь, на каком говорите языке, какой у вас компьютер и сколько примерно вам лет. Но нет, это не шпионы. Cookie-файл больше похож на библиотечную карточку или запись в журнале посетителей. Например, когда человек впервые приходит в фитнес-клуб, он оставляет о себе данные на ресепшен, и при каждом новом визите ему не нужно регистрироваться заново.

**Для чего нужны cookies**:

* хранят логин и пароль, ускоряют авторизацию
* запоминают, какие товары лежат в корзине
* сохраняют настройки интерфейса, языка и уведомлений
* позволяют предзагружать страницы для ускорения поиска
* помогают персонализировать рекламу и рекомендовать интересные материалы
* позволяют маркетологам анализировать информацию о пользователях и управлять расходами на рекламу более эффективно
* помогают определить ошибки в работе сайта

С их помощью специалисты анализируют количество посетителей сайта, их действия и источники трафика. Для этих целей куки очень удобны: сбор данных идет автоматически, их структура простая и не требует серьезных манипуляций.

Посетители всегда соглашаются на куки добровольно, поэтому имеют право в любое время отказаться от них. Но выключать эти файлы браузер не рекомендует — из-за этого некоторые веб-страницы могут работать неправильно, а ряд функций будет недоступен.

С момента изобретения cookie-файлов они обрели огромную популярность. Разработчики поняли, что cookie-файлы – это очень удобный инструмент и их начали массово использовать. Но через некоторое время стало понятно, что нельзя больше увеличивать размер этих файлов, ведь объём накопителей в персональных компьютерах тех лет был очень ограничен как и скорость интернета.

Решение появилось быстро. Вместо того чтобы хранить все данные на жестком диске пользователя, там хранился только универсальный идентификатор — номер пользователя, а все остальное переместилось на сервера сайтов. То есть сайты начали считывать ваш номер, находить ваши данные у себя на сервере по данному идентификатору и подстраиваться под вас.

**Устройство cookie-файлов**

* **Name**
* **Content or Value**
* **Domain**
* **Path (the “/” means the cookie is valid anywhere on that domain.)**
* **Send for (the connection security level)**
* **Created**
* **Expires**

**Что касается устройства cookie-файлы состоят из нескольких параметров:**

* Первый — это имя данного cookie-файла.
* Второе — само значение, тот самый уникальный идентификатор, по которому сайт определяет, что это именно вы посетили его.
* Третье — домен сайта, который создал этот cookie.
* Четвёртое — это путь, по которому внутри этого домена cookie действует. Во многих случаях здесь просто пишут слеш, чтобы cookie работал везде внутри данного домена.
* Пятое — уровень шифрования, которому подвергнется информация определенная этим cookie-файлом. А последние два параметра — это время когда файл был создан и когда он потеряет свое значение то есть удалится.

По значениям последних двух параметров определяются **два типа cookie: постоянные (persistent) и временные (session)**. **Временные** cookies **удаляются после прекращения сеанса**, то есть после того, как вы закройте браузер. Они раньше, например, использовались для хранения корзины в онлайн-магазинах. А **постоянные cookies** не удаляются, и **хранят долгосрочную информацию, например логин от учетной записи**.

**11.Що таке вебвразливість?**

**12.Які знаєте класичні бази даних?**

**13.Як читати специфікацію в конкретній мові (наприклад, PEP8 у Python)?**

**14.Як відбувається взаємодія клієнта і сервера?**

**15.Які є підходи до проєктування API?**

**16.Як використовувати патерни програмування?**

**17.Що таке Acceptance Testing і навіщо його використовують?**

**18.Що таке модульні та інтеграційні тести, API-тести?**

**19.Як писати unit-тести?**

**20.Які є best practices у написанні автотестів?**

**21.Які базові команди системи контролю версій?**

**22.Як використовувати Git?**

**23.У чому різниця між хешуванням і шифруванням?**

**Python:**

**24.Python — мова, що інтерпретується чи компілюється?**

**25.Які є змінні та незмінні типи даних?**

**26.Що таке область видимості змінних?**

**27.Що таке introspection?**

**28.Різниця між is і ==?**

**29.Різниця між \_\_init\_\_() і \_\_new\_\_()?**

**30.У чому різниця між потоками та процесами?**

**31.Які є види імпорту?**

**32.Що таке клас, ітератор, генератор?**

**33.Що таке метаклас, змінна циклу?**

**34.У чому різниця між ітераторами та генераторами?**

**35.У чому різниця між staticmethod та classmethod?**

**36.Як працюють декоратори, контекстні менеджери?**

**37.Як працюють dict comprehension, list comprehension і set comprehension?**

**38.Чи можна використовувати кілька декораторів для однієї функції?**

**39.Чи можна створити декоратор з класу?**

**40.Які є основні популярні пакети (requests, pytest, etc)?**

**41.Що таке lambda-функції?**

**42.Що означає \*args, \*\*kwargs та як вони використовуються?**

**43.Що таке exceptions, <try-except>?**

**44.Що таке PEP (Python Enhancement Proposal), які з них знаєте (PEP 8, PEP 484)?**

**45.Напишіть hello-world сервіс, використовуючи один із фреймворків.**

**46.Які є типи даних і яка різниця між list і tuple, навіщо вони?**

**47.Як використовувати вбудовані колекції (list, set, dictionary)?**

**48.У чому полягає складність доступу до елементів dict?**

**49.Як створюється об’єкт в Python, для чого \_\_new\_\_, навіщо \_\_init\_\_?**

**50.Що знаєте з модуля collections, якими ще built-in модулями користувались?**

**51.Що таке шаблонізатор та як у ньому виконувати базові операції (об’єднувати ділянки шаблона, виводити дату, виводити дані з серверного боку)?**

**52.Як Python працює з HTTP-сервером?**

**53.Що відбувається, коли створюється віртуальне середовище?**

**Бази даних:**

**54.Які є базові методи роботи з SQL- базою даних у Python?**

**55.Що таке SQL-транзакція?**

**56.Як зробити вибірку із SQL-бази з простою агрегацією?**

**57.Який вигляд має запит, який виконує JOIN між таблицями й до самих себе?**

**58.Як відправляти запити у SQL-базу даних без ORM?**

**Алгоритми:**

**59.Що таке алгоритми (наприклад, Big-O notation)?**

**60.Які є базові алгоритми сортування?**

**61.Що таке Bubble Sort і як це працює?**

**62.Що таке лінійна складність сортування?**

***ВОПРОСЫ  К  СОБЕСЕДОВАНИЯМ   НА  РАЗРАБОТЧИКА  PYTHON***

<https://gist.github.com/jigi-33/b9dbc60d7b62acd85a998c6086d3b058>

‌

‌