МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУ ІМ. ІГОРЯ СІКОРСЬКОГО»

ОСНОВИ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання комп'ютерних практикумів

на PYTHON

Київ – 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУ ІМ. ІГОРЯ СІКОРСЬКОГО»

ОСНОВИ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання комп'ютерних практикумів
на PYTHON

для студентів
спеціальності 122 «Комп'ютерні науки»
зі спеціалізації «Інформаційні технології в біології та медицині»

Рекомендовано вченою радою
факультету біомедичної інженерії
НТУУ «КПІ ім. Ігоря Сікорського»

Київ
НТУУ «КПІ ім. Ігоря Сікорського»
2017

ОСНОВИ ПРОГРАМУВАННЯ: методичні вказівки до виконання комп'ютерних практикумів на PYTHON з навчальної дисципліни «Основи програмування» для студентів спеціальності 122 «Комп'ютерні науки» зі спеціалізації «Інформаційні технології в біології та медицині» / Уклад. Л.М. Добровська. – К.: НТУУ «КПІ ім. Ігоря Сікорського», 2017. – 254 с.

Рекомендовано вченою радою ФБМІ НТУУ «КПІ ім.. І. Сікорського»
(Протокол № 6 від 27 лютого 2017 р.)

Навчально-методичне видання

ОСНОВИ ПРОГРАМУВАННЯ

методичні вказівки
до виконання комп'ютерних практикумів
на PYTHON

для студентів
спеціальності 122 «Комп'ютерні науки»
зі спеціалізації «Інформаційні технології в біології та медицині»

Укладачі: Л.М. Добровська, к. п. н., доцент кафедри БМК, ФБМІ

Відповідальний редактор: Яковенко Альона Вікторівна, к.т.н.

Рецензент: О. Г. Жданова, к.т.н., доцент кафедри АСОІУ, ФІОТ

Це навчально-методичне видання являє собою збірник комп'ютерних практикумів з основ програмування. Викладений матеріал дозволяє студентам без будь-яких базових знань з програмування за час навчання засвоїти основні конструкції мови Python (на основі стандартної бібліотеки) і оволодіти навичками алгоритмізації. Інструментальним засобом виконання практичних робіт є інтегроване середовище розробки програм на Python. Всі роботи супроводжуються прикладами розв'язання задач.

Сукупність придбаних навичок дозволяє студентам вже на першому курсі розробляти досить складні алгоритми.

З М І С Т

ВСТУП

Сучасні тенденції широкого використання інформаційних технологій і засобів комп'ютерного моделювання в усіх галузях економіки ставлять нові вимоги до рівня підготовленості студентів вищих технічних закладів в області програмування та інформатики. Створення у студентів достатньої теоретичної і практичної бази з цих предметів дозволяє їм швидко розв'язувати актуальні задачі, витрачаючи мінімальний час на перепідготовку. При цьому особлива увага повинна бути приділена підготовці майбутніх фахівців в області розробки сучасних систем.

Python – порівняно «молода» мова. Створюючи її у 1990-1991 роках, її автор Гвідо ван Россум (Guido van Rossum) врахував усі переваги та недоліки попередніх мов програмування. У наш час Python має широку область застосування. Це – мова, яка розвивається, її використовують в реальних проектах. Засоби для роботи з Python відносяться до категорії вільно поширюваного програмного забезпечення, що гарантує відсутність претензій щодо використання «інтелектуальної власності». Існує множина засобів, які полегшують процес створення програм на Python, серед них можна виділити спеціалізовані лексичні аналізатори і редактори для програмістів (наприклад, Kate і Bluefish), інтегровані середовища розробки. Багато засобів для роботи з Python є кросплатформними, в конструкціях мови підтримується багатобайтове кодування (Unicode), тому програми на Python легко переносити з одного середовища функціонування на інше.

Запропоноване видання дозволяє студентам без будь-яких базових знань з програмування за час навчання освоїти основні конструкції Python (на основі стандартної бібліотеки) і впевнено оволодіти навичками алгоритмізації. Це видання створено на основі комп'ютерних практикумів з дисципліни «Основи програмування», які виконуються в Національному технічному Університеті України «КПІ ім. Ігоря Сікорського» на факультеті біомедичної інженерії для студентів напряму підготовки 122 «Комп'ютерні науки». Виклад матеріалу дозволяє його використати не тільки викладачам, а й студентам під час самопідготовки. Всі роботи супроводжуються прикладами розв'язання задач. Програми виконувалися в середовищі Python 3.X. Сукупність придбаних навичок дозволяє студентам вже на першому курсі розробляти досить складні алгоритми.

Предмет навчальної дисципліни «Основи програмування» – базові поняття структурного та об'єктно-орієнтованого програмування на мові Python 3.X. *Метою навчальної дисципліни* є формування у студентів базових понять структурного та об'єктно-орієнтованого програмування на Python 3.X, вміння застосовувати їх на практиці.

Основні завдання навчальної дисципліни. Після засвоєння навчальної дисципліни студенти мають продемонструвати такі результати навчання:

знання: основних понять структурного та об'єктно-орієнтованого програмування:

- базові вбудовані типи об'єктів структурного програмування мови (числа, рядки, списки, словники, кортежі, файли, множини) і синтаксичні конструкції використання цих об'єктів у вигляді літералів (виразів, які генерують ці об'єкти); елементи структурного програмування (інструкції *if, while, for*);

- базові типи об'єктів об’єктно-орієнтованого програмування (клас, об’єкт-екземпляр класу, конструктор класу, метод класу тощо);

вміння: використовувати базові поняття структурного та об’єктно-орієнтованого програмування (створювати класи та об’єкти-екзампляри; використовувати наслідування класів тощо) для розв’язання різних задач на мові Python;

досвід в системі типових завдань діяльності: основи побудови сховищ даних.

Комп'ютерний практикум № 1. Подання інформації (цілих та дійсних чисел) у пам'яті комп'ютера

Мета роботи: ознайомитися з правилами перетворення цілих та дійсних чисел з однієї системи числення в іншу.

Об'єкт дослідження – процедури визначення внутрішнього зображення чисел в ПЕОМ.

ПЛАН

1. Подання цілих чисел у пам'яті ПЕОМ. Перетворення чисел з однієї системи числення (с/ч) в іншу
2. Подання інформації в ПЕОМ типу IBM PC/AT
3. Типи даних мови Python. Числа

Завдання

1. Вивчити теоретичні основи внутрішнього подання цілих та дійсних чисел в ПЕОМ.
2. Відповідно до свого варіанту отримати внутрішнє зображення:

№	цілого числа без знаку в одно- або дво-байтовій комірці пам'яті та у 8-ій с/ч	цілого числа в двобайтовій комірці, записати відповідь у 16-ій формі		дійсного числа в двійковій системі числення та в експоненційному нормалізованому вигляді (десятковий дріб $0,XXX_{10}$ перевести у двійкову систему числення з точністю до 2^{-3})
1	2	3	4	5
1	129	1687	-8542	125,1875
2	456	1597	-5687	159,275
3	423	4236	-1667	753,425
4	789	7524	-2546	851,953
5	159	7458	-9874	153,859
6	267	9532	-6523	753,159
7	357	8542	-9832	456,754
8	459	6578	-4596	159,753
9	852	2154	-7425	426,751
10	345	7596	-5623	953,751
11	259	7845	-8742	759,153
12	531	3256	-3657	698,412
13	214	9888	-8563	325,785
14	742	5236	-4253	254,145
15	853	4523	-7425	741,852
16	963	7845	-8632	963,258
17	751	6587	-7823	745,352
18	862	9832	-8523	752,154
19	953	1245	-9632	985,425
20	842	1597	-9874	965,742

1	2	3	4	5
21	751	3254	-5236	653,145
22	793	9863	-6497	751,425
23	315	7863	-9713	854,164
24	934	9632	-5612	856,235
25	468	7412	-4555	754,256
26	682	3298	-2385	154,752
27	921	8716	-4258	627,125
28	248	4512	-3296	597,531

3. Зобразити отримані результати на мові Python.
4. Скласти звіт і захистити його по роботі.

Контрольні запитання

1. Визначити поняття «система числення, основа системи числення». Що є доповненням числа «0» у двійковій с/ч? Що є основою двійкової с/ч?
2. Яку операцію необхідно виконати для перетворення цілої частини числа з однієї с/ч у іншу: а) ділення; б) віднімання; в) множення; г) додавання?
3. Які основні типи даних визначені в мові Python.

Аудиторна робота

Приклад 1.1. Приклади цілих чисел

```
# числа в 10-овій с/ч
dec1 = 8; dec2 = -3
dec3=258028365723567
# числа в 16-овій с/ч
hex1 = 0x9; hex2 = 0xA; hex3 = 0xFF; hex4 = 0x3de
# число у двійковій с/ч
bin1 = 0b11101101
# число у 8-овій с/ч
oct1 = 0o765
```

Приклад 1.2. Приклади дійсних чисел

```
a = 0.5; b = 3.2
# у експоненційній формі запису
c = 3.2e5  # 3.2 * 10**5
d = 1e-3   # 1 * 10**(-3)
# отримання дійсного значення
float("0.5")  # з рядка
float(3))     # з цілого числа
```

Приклад 1.3. Приклади комплексних чисел

```
c1 = 2 + 3j  # 2 + 3i, 2 - дійсна частина, 3 - уявна
c2 = 5 - 5j  # 5 + 5i
# побудова комплексного числа з дійсного
a = 2; b = 3
c3 = complex(a, b); c4 = complex(5, -5)
```

Теоретична частина

1. ПОДАННЯ ЦІЛИХ ЧИСЕЛ У ПАМ'ЯТІ ПЕОМ

Система числення – це сукупність прийомів та правил зображення чисел за допомогою символів (цифр, літер тощо), які мають визначені кількісні значення (числовий еквівалент).

Залежно від способів зображення чисел цифрами та способу визначення числового еквіваленту с/ч поділяють на непозиційні та позиційні. *Непозиційна* с/ч (наприклад, римська) – це така система, в якій значення символу (значення числового еквіваленту) не залежить від його позиції (розряду) у запису числа, а залежить лише від самого числа. *Позиційна* с/ч – це така система, в якій значення символу (числовий еквівалент) залежить від його місця у запису числа. Будь–яка позиційна с/ч характеризується *основою*. Основа (або базис p) позиційної с/ч – це ціле число, яке визначає кількість символів, що в ній використовують для зображення числа.

В позиційній с/ч з основою p будь-яке число R можна подати у вигляді:

$$R(p) = a_N p^N + a_{N-1} p^{N-1} + \ldots + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + \ldots + a_{-k} p^{-k}, \quad (1.1)$$

де коефіцієнти a_i – символи в зображенні числа R, які приймають значення від 0 до $p-1$. Зазвичай число $R(p)$ зображують у вигляді множини коефіцієнтів a_i: $R(p) = a_N a_{N-1} \ldots a_1 a_0 a_{-1} \ldots a_{-k}$. Наприклад:

$$265{,}23_{10} = 2 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0 + 2 \cdot 10^{-1} + 3 \cdot 10^{-2},$$

$$23{,}78_{16} = 2 \cdot 16^1 + 3 \cdot 16^0 + 7 \cdot 16^{-1} + 8 \cdot 16^{-2},$$

$$1001{,}01_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}.$$

У обчислювальній техніці широко використовують позиційні с/ч (2–ову, 8–ову, 10–ову, 16–ову).

Двійкова система числення – тут використовують дві цифри «0» і «1». Наприклад, $2_{10} = 2 \cdot 10^0 = 1 \cdot 2^1 + 0 \cdot 2^0 = 10_2$.

Двійкова таблиця додавання

+	0	1
0	0	1
1	1	10

Вісімкова с/ч – тут використовують цифри 0, 1, 2, 3, 4, 5, 6, 7. Наприклад, $8_{10} = 8 \cdot 10^0 = 1 \cdot 8^1 + 0 \cdot 8^0 = 10_8$.

Вісімкова таблиця додавання

+	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

Шістнадцятькова с/ч – тут використовують цифри 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 і літери A, B, C, D, E, F. Наприклад, $16_{10}=16\cdot10^0=1\cdot16^1+0\cdot16^0=10_{16}$.

Шістнадцяткова таблиця додавання

+	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Множина цілих чисел, розміщених в пам'яті ЕОМ, обмежена і залежить від розміру комірок пам'яті (машинного слова), які використовують для їхнього зберігання.

В *k*-розрядній комірці може зберігатися 2^k різних значень цілих чисел. Зазвичай розряди нумерують справа наліво, починаючи з «0». Нижче показана нумерація біт в двобайтовому машинному слові.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

В даній комірці можна зберігати 2^{16} цілих чисел. Діапазон допустимих значень залежить також від того, чи будуть в комірці зберігатися числа зі знаком чи без знаку. Для запису цілого числа без знаку досить перевести його в двійкову с/ч і при необхідності доповнити результат зліва незначущими нулями.

Приклад 1.4. Отримати внутрішнє машинне зображення цілого числа 129 без знаку в однобайтовій комірці пам'яті.

Розв'язання: $129_{10}=10000001_2$; 1000 0001

Приклад 1.5. Нехай задано внутрішнє машинне подання числа в однобайтовому машинному слові – $9C_{16}$. Визначити, що це за число.

Розв'язання: $9C_{16}=1001\ 1100_2=9\cdot16^1+12\cdot16^0=156_{10}$

$2^7+2^4+2^3+2^2=128+16+8+4=156_{10}$

Щоб отримати внутрішнє машинне подання цілого числа зі знаком, необхідно знайти його код доповнення.

Подання цілих додатніх чисел

Для їх отримання необхідно виконати такі кроки:

Крок 1. Перевести число *N* в двійкову систему числення.

Крок 2. Отриманий результат доповнити зліва незначущими нулями до k розрядів.

Крок 3. При необхідності перевести число в стислу 16-ову форму.

Приклад 1.6. Отримати внутрішнє машинне зображення цілого числа 1607 в двобайтовій комірці пам'яті. Записати відповідь в 16-овій формі.

Розв'язання: 160710 =110010001112. Внутрішнє машинне зображення цього числа: 0000 0110 0100 0111; 16-а форма: 0647.

Подання цілих від'ємних чисел

Для подання від'ємних чисел використовується код доповнення, який дозволяє замінити арифметичну операцію віднімання операцією додавання, що спрощує роботу процесора і збільшує його швидкодію. Алгоритм отримання внутрішнього подання цілого від'ємного числа *N*, що зберігається в k-розрядному машинному слові охоплює такі кроки:

Крок 1. Отримати внутрішнє зображення додатнього числа N.

Крок 2. Отримати зворотний код цього числа заміною «0» на «1» і «1» на «0», тобто значення всіх біт інвертувати.

Крок 3. До отриманого числа додати «1» (отримати код доповнення).

Крок 4. При необхідності записати стисле внутрішнє машинне подання.

Приклад 1.7. Отримати внутрішнє подання цілого числа –1607 в 2-х байтовій комірці пам'яті. Записати відповідь в 16–овій формі.

Розв'язання. Внутрішнє подання цього додатнього числа:

0000 0110 0100 0111

Зворотний код – 1111 1001 1011 1000

Код доповнення – 1111 1001 1011 1001

Стислий 16-ий код – $F9B9_{16}$

У разі подання величини зі знаком найлівіший (старший) розряд вказує на додатне число, якщо містить нуль, і на від'ємне, якщо – одиницю.

Приклад 1.8. Нехай задано стисле 16-е внутрішнє подання числа – $CF18_{16}$. Визначити, що це за число.

Розв'язання: CF18 = 1100 1111 0001 1000

Число від'ємне, оскільки старший розряд дорівнює «1», тому отримуємо зворотний код – 1100 1111 0001 0111 (відняти 1)

Прямий код – 0011 0000 1110 1000; $30E8_{16}= 12520_{10}$

ПЕРЕТВОРЕННЯ ЧИСЕЛ З ОДНІЄЇ СИСТЕМИ ЧИСЛЕННЯ В ІНШУ

Правило 1.Переведення змішаного числа (числа із дробовою частиною) з *p*–ої с/ч в *q*–у с/ч, коли має місце співвідношення $p=q^k$ (*k* – ціле додатне число), здійснюється поразрядно. Кожна *p*–а цифра замінюється рівним їй *k*–розрядним числом, записаним в *q*–ій с/ч (при цьому можна використовувати дані з табл. 1.1).

Приклад 1.4.

а) $A61,87C_{(16)} = \underbrace{(1010)}_{A_{(16)}}\underbrace{(0110)}_{6_{(16)}}\underbrace{(0001)}_{1_{(16)}},\underbrace{(1000)}_{8_{(16)}}\underbrace{(0111)}_{7_{(16)}}\underbrace{(1100)}_{C_{(16)}}{}_{(2)}.$

б) $31,471_{(8)} = \underbrace{(011)}_{3_{(8)}}\underbrace{(001)}_{1_{(8)}},\underbrace{(100)}_{4_{(8)}}\underbrace{(111)}_{7_{(8)}}\underbrace{(001)}_{1_{(8)}}{}_{(2)}$.

Таблиця 1.1

Запис чисел у різних с/ч

Десяткове число	Двійкове число	Вісімкове число	Шістнадцяткове число
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Зворотній переклад з *q*–ої с/ч в *p*–у с/ч здійснюють, розбиваючи *q*–ий запис числа на групи по *k* цифр, рухаючись від коми вправо і вліво і замінюючи кожну групу цифр її *p*–им зображенням. При цьому якщо крайні групи виявляться неповними, то їх доповнюють до *k* цифр незначущими нулями.

Приклад 1.5.

а) $1110010,01101111001_{(2)} = \underbrace{(0111)}_{7_{(16)}}\underbrace{(0010)}_{2_{(16)}},\underbrace{(0110)}_{6_{(16)}}\underbrace{(1111)}_{F_{(16)}}\underbrace{(0010)}_{2_{(16)}}{}_{(2)} = 72,6F2_{(16)}$;

б) $1001111,011100_{(2)} = \underbrace{(001)}_{1_{(8)}}\underbrace{(001)}_{1_{(8)}}\underbrace{(111)}_{7_{(8)}},\underbrace{(011)}_{3_{(8)}}\underbrace{(100)}_{4_{(8)}}{}_{(2)} = 117,34_{(8)}$.

Переведення змішаного числа виконується окремо для цілої і дробової частин числа.

Правило 2.

1. Перетворення цілої частини. Цілу частину числа, записану в *p*–ій с/ч, ділять на основу *q*–ої с/ч, записану в *p*–ій с/ч (всі операції виконуються за правилами *p*–ої с/ч). Отримане в залишку число є молодшою (останньої) цифрою в *q*–ій формі запису числа. Отриману частку знову ділять на основу *q*; залишок – це передостання цифра в шуканій формі запису числа; і т.д. Операцію ділення проводять до тих пір, поки часткою не стане число, менше за *q*; ця частка – це старша (перша) цифра в *q*–ій формі запису числа.

Приклад 1.6.

а) $345_{(10)} = 159_{(16)}$;

$$\begin{array}{r|l} 345 & 16 \\ \hline 32 & 21 \mid 16 \\ \hline 25 & 16 \quad 1 \\ \hline 16 & 5 \\ \hline 9 & \end{array}$$

б) $24_{(10)} = 11000_{(2)}$.

$$\begin{array}{r|l} 24 & 2 \\ \hline 24 & 12 \mid 2 \\ \hline 0 & 12 \quad 6 \mid 2 \\ \hline & 0 \quad 6 \quad 3 \mid 2 \\ \hline & \quad 0 \quad 2 \quad 1 \\ & \quad\quad 1 \end{array}$$

2. Перетворення дробової частини. Дробову частину числа, записану в *p*–ій с/ч, множать на основу *q*–ої с/ч, записану в *p*–ій с/ч. Ціла частина добутку буде старшою цифрою зображення дробу (першої після коми) в *q*–ій формі запису дробу. Дробову частину добутку знову множать на основу *q*; ціла частина – наступна цифра після коми в *q*–ій формі запису дробу. Процес продовжують, поки дробова частина не стане нулем або буде отримано необхідну кількість знаків після коми в дробовій частині запису числа. Цілі частини записують в *q*–ій с/ч.

Приклад 1.7.

а) $0,2_{(10)} = 0,(0011)_{(2)}$;

$$\begin{array}{r} 0,2 \\ \hline \times\ 2 \\ \hline 0_{(2)} = 0,4 \\ \hline \times\ 2 \\ \hline 0_{(2)} = 0,8 \\ \hline \times\ 2 \\ \hline 1_{(2)} = 1,6 \\ \hline \times\ 2 \\ \hline 1_{(2)} = 1,2 \\ \hline \times\ 2 \\ \hline 0_{(2)} = 0,4 \\ \hline \times\ 2 \\ \hline 0_{(2)} = 0,8 \end{array}$$

б) $0,12_{(10)} = 0,1EB_{(16)}$;

$$\begin{array}{r} 0,12 \\ \hline \times\ 16 \\ \hline 1_{(16)} = 1,92 \\ \hline 0,92 \\ \hline \times\ 16 \\ \hline E_{(16)} = 14,72 \\ \hline 0,72 \\ \hline \times\ 16 \\ \hline B_{(16)} = 11,52 \\ \hline \ldots\ldots \end{array}$$

в) $0,A_{(16)} = 0,625_{(10)}$.

$$\begin{array}{r} 0,A \\ \hline \times\ A \\ \hline 6_{(10)} = 6,4 \\ \hline 0,4 \\ \hline \times\ A \\ \hline 2_{(10)} = 2,8 \\ \hline 0,8 \\ \hline \times\ A \\ \hline 5_{(10)} = 5,0 \end{array}$$

При переведенні змішаного числа результати для цілої та дробової частин, отримані за правилами 2.1, 2.2, записують, відокремлюючи комою одну частину від іншої.

Приклад 1.8.

$$345{,}02_{(10)} = 101011001{,}(0011)_{(2)}.$$

$$345_{(10)} = 159_{(16)} = \underbrace{(0001)}_{1_{(16)}}\underbrace{(0101)}_{5_{(16)}}\underbrace{(1001)}_{9_{(16)}}{}_{(2)};$$

$$0{,}2_{(10)} = 0{,}(0011)_{(2)}.$$

$$\begin{array}{r|l} 345 & 16 \\ \underline{32} & 21 \\ 25 & \underline{16} \\ \underline{16} & 1 \\ 9 & \end{array} \qquad \begin{array}{r|l} 21 & 16 \\ \underline{16} & 1 \\ 5 & \end{array}$$

$$\begin{array}{r} 0{,}2 \\ \underline{\quad 2} \\ 0_{(2)} = 0{,}4 \\ \underline{\quad 2} \\ 0_{(2)} = 0{,}8 \\ \underline{\quad 2} \\ 1_{(2)} = 1{,}6 \\ \underline{\quad 2} \\ 1_{(2)} = 1{,}2 \\ \underline{\quad 2} \\ 0_{(2)} = 0{,}4 \\ \ldots\ldots \end{array}$$

Правило 3. Переведення чисел в 10−ову с/ч рекомендують виконувати додаванням з урахування «ваги» цифри у числі за формулою (1.1):

$$a_N a_{N-1} \ldots a_1 a_0 \,.\, a_{-1} \ldots a_{-K} = a_N p^N + a_{N-1} p^{N-1} + \ldots + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + \ldots + a_{-K} p^{-K}$$

Приклад 1.9.

а) $1101{,}111_{(2)} = 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} = 13{,}875_{(10)};$

б) $1A{,}02_{(16)} = 1\cdot 16^1 + 10\cdot 16^0 + 0\cdot 16^{-1} + 2\cdot 16^{-2} = 26{,}0078125_{(10)}.$

2. ПОДАННЯ ІНФОРМАЦІЇ В ПЕОМ ТИПУ ІВМ РС/АТ

Інформація в пам'яті ЕОМ зберігається і обробляється в двійковому вигляді. Форма запису даних в пам'яті машини називається внутрішнім поданням інформації в ЕОМ. Одиницею зберігання інформації є один біт, тобто двійковий розряд, який може приймати значення «0» або «1». Застосування двійкової с/ч дозволяє використовувати для зберігання інформації елементи, які мають лише два стійких стани. Один стан служить для зображення одиниці відповідного розряду числа, інший − для зображення нуля. Зазвичай біти пам'яті групуються в більш широкі структури. Група з восьми бітів називається байтом. Поля пам'яті ЕОМ мають спеціальні назви: 16 біт − слово, 32 біта − подвійне слово, 1024 байта (1 К байт) − лист. Всі байти пронумеровані, починаючи з нуля.

Адресою будь-якої інформації вважається адреса (номер) найпершого байта поля пам'яті, виділеного для її зберігання. Існують два основних способи подання чисел: з фіксованою і з плаваючою комою.

Двійкові числа з фіксованою комою. У вигляді з фіксованою комою можуть зберігатися тільки цілі числа (в пам'яті ЕОМ вони записуються в двійковому вигляді). Ціле двійкове число займає в пам'яті 16 або 32 (або 64) двійкових розряди. Це залежить від довжини числа і способу подання змінної.

Розглянемо, як записується число в подвійному слові (в слові воно записується аналогічно). Сьомий біт першого байта є службовим при зберіганні чисел зі знаком, і його вміст не впливає на абсолютну величину числа. Зазначимо, що для додатних чисел вмістом службового біта є нуль. Молодший двійковий розряд числа записується в 0-ий біт, тобто число заповнюють справа наліво. Якщо число додатнє, то біти, що залишилися, заповнюються нулями. Наприклад, число 12710 в 4 байтах буде подано таким чином:

$$127_{(10)} = 7F_{(16)} = 1111111_{(2)} = 1 \cdot 2^6 + 1 \cdot 2^5 + \ldots + 1 \cdot 2^1 + 1 \cdot 2^0$$

76543210	76543210	76543210	76543210	біти
0000 0000	0000 0000	0000 0000	01111111	байти
1 – ий байт	2-ий байт	3– ій байт	4– ий байт	

Із наведеного вище подання видно, що в молодшому розряді записаний коефіцієнт при 2^0, в наступному − при 2^1 і т.д. Очевидно, що якщо у всіх бітах з 0-го по 30-ий знаходяться 1, то максимальне ціле додатне число, яке можна записати в подвійному слові, має вигляд:

$$01111111\ 11111111\ 11111111\ 11111111 = 2^{31} - 1 = 2147483647_{10}$$

Форму запису додатних цілих двійкових чисел називають прямим кодом. У цьому випадку їх запис у відведеній пам'яті повністю відповідає символічному запису в двійковій системі числення.

Від'ємні числа записуються в додатковому коді. Використання цього коду дозволяє спростити апаратну реалізацію операції віднімання, яка замінюється операцією додавання зменшуваного, поданого в прямому коді, і від'ємника, поданого в додатковому коді. Додатковий код виходить з прямого шляхом інвертування кожного біта (зворотний код) і додавання «1» до молодшого біту числа. Подамо число -95_{10} як двійкове число з фіксованою точкою:

$$-95_{10} = -5F_{16} = -1011111_2$$

Крок 1. Запишемо прямий код числа в чотири байти:

00000000 00000000 00000000 01011111

Крок 2. Запишемо зворотний код числа:

11111111 11111111 11111111 10100000

Крок 3. Додавши до молодшого біту «1», отримаємо додатковий код числа:

	11111111	11111111	11111111	10100000
+				
				1
	11111111	11111111	11111111	10100001
	1-ий байт	2-ий байт	3-ій байт	4-ий байт

При зберіганні цілих додатних чисел існує можливість майже удвічі збільшити числовий діапазон. Це досягається за рахунок використання

службового сьомого біта першого байта для зберігання старшого коефіцієнта в двійковому поданні числа. Наприклад, число

$$65535_{10} = FFFF_{16} = 11111111\ 11111111_2,$$

оголошене як ціле без знака, в двох байтах запишеться у вигляді

11111111 11111111.

Двійкові числа з плаваючою крапкою. Числа, в яких положення крапки не зафіксовано після деякого розряду, а вказується спеціальним числом, називаються числами з плаваючою комою. У загальному вигляді будь−яке число *A* в системі числення з основою *p* можна подати у вигляді $A = m \cdot p^k$, де m − мантиса числа *A*, *k* − порядок числа. При цьому якщо мантиса числа задовольняє нерівності $1 \le |m| < p$, то число нормалізоване.

Розмір областей, які виділяються під числа з плаваючою комою, істотно залежать від апаратної реалізації обчислювальної техніки. Для IBM PC/AT, наприклад, числа з плаваючою комою можуть, в залежності від оголошених атрибутів *float* розташовуватися відповідно в 32 бітах пам'яті у вигляді:

float	±	характеристика		нормалізована мантиса	
біти	31	30	23	22	0

Мантиса числа записується в нормалізованому вигляді (в двійковому зображенні числа перед комою зберігається один значущий двійковий біт). Щоб збільшити діапазон подання чисел в форматі *float*, одиниця перед комою в двійковій нормалізованій формі запису числа в пам'ять не записується. Знак мантиси записується в останньому біті першого байта.

Для спрощення апаратної реалізації арифметичних операцій в поданні числа з плаваючою комою знак порядку явно не зберігається. Замість порядку в пам'ять записується величина, яку називають характеристикою: її отримують шляхом додавання коефіцієнта, який для чисел типу *float* дорівнює $127_{10} = 7F_{16}$.

Подамо число $-18.2_{10} = -10010.(0011)_2$, нормалізований запис якого має вигляд $-1.0010(0011) \cdot 2^{100}$, де нормалізована мантиса $m = -1.0010\ (0011)$, а порядок $k = 4_{10} = 100_2$. Перейдемо від порядку до його характеристики: $4 + 127 = 131_{10} = 83_{16} = 1000\ 0011_2$. Отримаємо зображення числа -18.2_{10} в форматі *float*:

1	10000011	00100011001100110011010	
31	30	23	22 0

Якщо кількість цифр в числі, яке записують в ПЕОМ або отримано в результаті обчислень, більше виділеного поля пам'яті, то надлишкові цифри відкидаються, а число, що записується в пам'ять, округляється з надлишком.

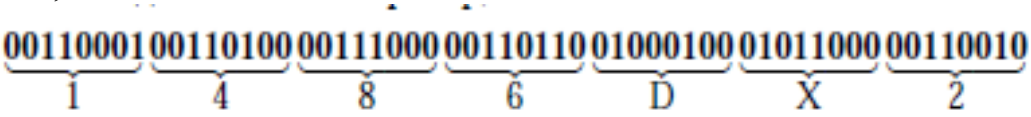
Розмір виділених для зберігання даних областей визначає інтервал допустимих для цього атрибута значень − діапазон (табл. 1.5). Будь-яке число, менше за абсолютною величиною додатного мінімального числа, поданого у відповідному форматі, буде записано в пам'ять у вигляді нуля. Для заданого

формату це так званий машинний нуль. Крім того, числа заданого формату не повинні перевищувати по абсолютній величині максимальне число, подане в цьому форматі. В іншому випадку старші біти числа будуть втрачені, а результат не правильним. Описана ситуація називається переповненням розрядної сітки, а самі числа − машинною нескінченністю.

Таблиця 1.5

Тип даних	Кіл-ть біт	Діапазон	
		Abs(min)	Abs(max)
float	32	$3.4 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$

Символьна інформація. В ПЕОМ для внутрішнього подання символьних даних використовується ASCII код, згідно з яким кожному символу відповідає 8-розрядний код, тобто в один байт записується один символ. Наприклад, текст «1486DX2» в пам'яті має вигляд:



3. ТИПИ ДАНИХ (ОБ'ЄКТІВ) МОВИ PYTHON

Програми на Python виконують дії, які приймають форму операцій над об'єктами (наприклад, складання або конкатенація), над якими виконуються операції. Дані в Python подані у формі вбудованих об'єктів або об'єктів, які створюють із застосуванням конструкцій мови Python або інших інструментів (наприклад, бібліотеки розширень, написані на мові С).

Об'єктами є всі дані, які доводиться обробляти в програмах. Об'єкти (структури даних, призначені для подання складових предметної області) − це область пам'яті зі значеннями і асоційованими з ними наборами операцій.

В Python використовують динамічну типізацію (типи даних визначаються автоматично, їх не потрібно оголошувати в програмному коді), але при цьому він є мовою із чіткою типізацією (ви зможете виконувати над об'єктом тільки ті операції, які застосовані до його типу). Програми на Python можна розкласти на такі складові: модулі, інструкції, вирази і об'єкти. При цьому програми діляться на модулі, модулі містять інструкції, інструкції складаються з виразів, вирази створюють і обробляють об'єкти.

Можна створювати власні типи об'єктів: вбудовані компоненти є стандартними складовими Python, тому вони завжди залишаються незмінними; власні структури мають властивість змінюватися від випадку до випадку. Навіть якщо ви створюєте власні типи об'єктів, вбудовані об'єкти будуть ядром будь−якої програми на Python. У табл. 1.3 наведено деякі вбудовані типи об'єктів і синтаксичні конструкції використання цих об'єктів у вигляді літералів – виразів, які генерують (створюють) ці об'єкти. Типи об'єктів, перераховані в табл. 1.3, називають базовими, тому що вони вбудовані безпосередньо в мову (для створення більшості з них використовується певний синтаксис). Наприклад, коли виконується програмний код: 'spam', то виконується вираз − літерал, який генерує і повертає новий об'єкт−рядок.

Таблиця 1.3

Деякі вбудовані об'єкти

Тип об'єкта	Приклад літерала
Числа	1234, 3.1415, 3+4j, Decimal, Fraction
Рядки	‘spam’, “guido’s” , b’a\x01c’
Списки	[1, [2, ‘three’], 4]
Словники	{‘food’: ‘spam’, ‘taste’: ‘yum’}
Кортежі	(1,’spam’, 4, ‘U’)
Файли	myfile = open(‘eggs’, ‘r’)
Множини	set(‘abc’), {‘a’, ‘b’, ‘c’}

Схожим чином вираз, розміщений в квадратних дужках, створює список, в фігурних дужках – словник тощо. Хоча в Python відсутня конструкція оголошення типу, сам синтаксис виконуваних виразів задає типи створюваних і використовуваних об'єктів. Розглянемо базовий тип об'єктів Python – числа.

ЧИСЛА

Python підтримує звичайні числові типи (цілі і дійсні), а також літерали – для їх створення і вирази – для їх обробки. Нижче наведений повний перелік числових типів та інструментів, які підтримуються в Python:

1. Цілі і дійсні числа
2. Комплексні числа
3. Числа фіксованої точності
4. Раціональні числа
5. Множини
6. Логічні значення
7. Цілі числа необмеженої точності
8. Вбудовані функції, модулі для роботи з числами

Крім базових типів даних Python використовує звичайні числові типи: цілі числа (додатні та від’ємні) і дійсні числа (з дробовою частиною), які іноді називають числами з плаваючою точкою. Python дозволяє записувати цілі числа у вигляді шістнадцяткових, вісімкових і двійкових літералів. Він підтримує комплексні числа і забезпечує необмежену точність подання цілих чисел (кількість цифр в цілих числах обмежується лише обсягом наявної пам'яті). У табл. 1.4 показано, як виглядають числа різних типів в Python в тексті програми (тобто у вигляді літералів).

Таблиця 1.4

Числові літерали

Літерал	Інтерпретація
1234, -24, 0, 9999999999999999999	Звичайні цілі числа (із необмеженою точністю зображення)
1.23, 1., 3.14e-10, 4E210, 4.0e+210	Дійсні числа
0o177, 0x9ff, 0b101010	Вісімкові, шістнадцяткові та двійкові літерали цілих чисел
3+4j, 3.0+4.0j, 3J	Літерали комплексних чисел

Комп'ютерний практикум № 2. Розробка блок-схеми алгоритму розв'язання задачі

Мета роботи: ознайомитися з правилами побудови та визначення блок–схем алгоритмів для написання програм, сформувати вміння і навички роботи з графічними елементами та базовими структурами блок-схем. *Об'єкт дослідження* – блок–схема алгоритму, її графічні елементи та базові структури.

ПЛАН

1. Алгоритм та його властивості**.**
2. Створення схем алгоритмів засобом Microsoft Visio

Завдання

1. Вивчити теоретичні основи опису алгоритмів за допомогою блок-схем. Опрацювати приклади.

2. Відповідно до свого варіанту завдання виконати такі кроки

1) побудувати блок-схему алгоритму обчислення значень за даними варіантів завдань в Microsoft Visio,

2) сформувати блок-схему у середовищі Microsoft Visio, зберегти її в форматі Microsoft Visio та у текстовому документі.

3. Скласти звіт і захистити його по роботі.

Варіанти

№	РОЗРАХУНКОВІ ФОРМУЛИ (вихідні дані)	Значення вхідних даних
1	2	3
1	$Y=\sqrt{x^3+ax^2+bx+c}$, де a=x+0,52b; b=$e^{(cx2+1)}$	x=0,35; c=0,8
2	$Y=5\sin^2\left\|\ln(cx^3+1)\right\|$, де x=$\cos^2(a+b)$; b=$\sin^2(a)$	a=0,52; c=1,5
3	$Y=\ln(e^x+bx^2)$, де x=$\cos^2(a+b)$; b=$\sin^2(a)$	a=0,52
4	$Z=\frac{\sqrt{x^4+ax+b}}{\sqrt{x^4+ax+b}}$, де a=x^2+lg(0,08)+e^{-x}; b=x+4a	x=0,75
5	$C=\frac{D+\sqrt{x}}{\ln(\sqrt{R+E})}$, де D=12,5+ln(E); R=8D–x^4+1	x=1,08; E=0,7
6	$Y=(1+z)\frac{x+\frac{y}{x}}{a-\frac{1}{1+x}}$, де $y=e^{\sqrt{x^3+1,75z}}$; z = $5\cdot10^{-1,8}$	x = $0,8\cdot10^{-2}$
7	$Y=\frac{(a+b)^n}{1+\frac{x}{x^m+b^{m-n}}}$, де m=n–1; n=$e^{a/b}$; a=lg(0,083); b=e^a	x=0,81
8	$A=\frac{\alpha_1\beta_1-\alpha_2\beta_2}{\alpha_1-\alpha_2^2}$, де α_1=sin(0,18); $\beta_1=e^{\alpha_1}$; $\alpha_2=\ln(0,05)+e^{\alpha_1}$	β_2=1,7

1	2	3
9	$S = K_1a_1 + K_2a_2 - \beta,$ де $a_1 = a_0 + \Delta a$; $a_2 = T\sin(30^\circ) + \omega$; $\omega = e^{K_1+K_2}$	K_1=0,05; K_2=0,03; T=1; a_0=1; $\Delta a = 0{,}1$
10	$Z = (a\sqrt{b} - c\sqrt{d})^2 \dfrac{5{,}6}{a+b+c}$, де $a = \sqrt{(b-c)^3}$; b=a^2–1; d=e$^{(b\wedge 2-a)}$	c=1,0
11	$Y = \sin\dfrac{1}{x+0.2} + \lg(0{,}08e^x)$, де $x = -2{,}5a + b\sqrt{c}$; a=0,8c+bx	c=0,5; b=1
12	$Y = \dfrac{\cos^2 ax+b}{\sin^2 x}$, де $a = 0{,}5c + e^x$; b=x^2–ac; c=0,8ln(0,072)	x=0,82
13	$Y = \dfrac{x+3a-K_1x}{K_2x+K_3x}$, де x=5a+K_1K_2	K_1=0,8; K_2=K_3=0,5; a=0,56
14	$Y = \dfrac{x^4-x^2-b}{x-b} + \dfrac{x^3-x-a}{x-a}$, де b=2^x–1; x=lg(0,005)+1	a=1,0
15	$Y = 5\sin^2\lvert cx^3+1\rvert$, де x=cos^2(a+b); b=sin^2(a)	a=0,52; c=1
16	$Z = \dfrac{\sqrt{x^4+ax+b}}{\sqrt{x^4+ax+b}}$, де b=x+4a; a=-x^2+lg(0,08)+e^{-x}	x=0,75
17	$Y = (1+z)\dfrac{x+\frac{y}{x}}{a-\frac{1}{1+x}}$, де $y = e^{\sqrt{x^3+1{,}75z}}$; z=5·10$^{-1,8}$	x=0,8·10^{-2}; a=1,0
18	$A = \dfrac{\alpha_1\beta_1 - \alpha_2\beta_2}{m\alpha_1 - \alpha_2^2}$, де $\beta_1 = e^{\alpha_1}$; $\alpha_2 = \ln(0{,}05) + e^{\alpha_1}$; α_1=sin(π/2)	β_2=1,7; m=1
19	$Z = (a\sqrt{b} - c\sqrt{d})^2 \dfrac{5{,}6}{a+b+c}$, де $a = \sqrt{(b-c)^3}$; b=a^2–1; $d = e^{(b^2-a^2)}$	c=1
20	$A = \dfrac{\alpha_1\beta_1 - \alpha_2\beta_2}{\alpha_1 - \alpha_2^2}$, де $\beta_1 = e^{\alpha_1}$; $\alpha_2 = \ln(0{,}05) + e^{\alpha_1}$; α_1=sin(π/3);	β_2=1,7
21	$y = \sqrt{x^3+ax^2+bx+c}$, де a=x+0,52b; b=$e^{(cx2+1)}$	x=0,35; c=0,8
22	$y = \ln(e^x + bx^2)$, де x=cos^2(a+b); b=sin^2a	a=0,52
23	$C = \dfrac{D+\sqrt{x}}{\ln(\sqrt{R+E})}$, де R=8D – x^4+1; D=12,5+ln(E)	x=1,08; E=0,7

1	2	3
24	$Y=\dfrac{(a+b)^n}{1+\dfrac{x}{x^m+b^{m-n}}}$, де a=lg(0,083); b=e^a; $n=e^{a/b}$; $m=n-1$	x=0,81
25	$S=K_1a_1+K_2a_2-\beta$; $a_1=a_2+\Delta a$; $a_2=T\sin 60^\circ+\omega$; $\omega=e^{K_1+K_2}$;	K_1=0,1; K_2=0,3; $\Delta a=0{,}1$; T=π/2; ; β=3
26	$y=\dfrac{2tg(b)}{\ln\lvert 2a\rvert}-\sqrt{\dfrac{\lvert d\rvert}{c}}$	a=-1,35; b=1,45; c=5; d=-2
27	$y=\sqrt{\dfrac{\ln\lvert d\rvert}{tg(c)}}+\dfrac{sin(b)}{2a}$	a=-1,35; b=1,45; c=5; d=-2
28	$y=4\sqrt{e^{c-a}}+\dfrac{\cos(b)}{d}$	a=-1,35; b=π; c=5; d=2

Контрольні запитання

1. Що таке алгоритм? Які існують способи опису алгоритмів?
2. Що таке блок-схема? Основні графічні елементи блок-схем, їх призначення.
3. Які існують правила оформлення блок-схем?
4. Який нормативний документ містить правила оформлення схем алгоритмів?
5. Які існують основні групи графічних форматів в Microsoft Visio? Які шаблони елементів застосовані для формування схеми алгоритму? Які засоби контролю за розмірами елементів передбачені Microsoft Visio? Які засоби вирівнювання і розподілу елементів застосовуються в Microsoft Visio? Як додати текст в діаграмі Microsoft Visio? Як здійснюється експорт фрагментів діаграм Microsoft Visio в текстовий редактор?

Теоретична частина

1. АЛГОРИТМ ТА ЙОГО ВЛАСТИВОСТІ

Розв’язання будь-якої задачі на ЕОМ відбувається в кілька етапів: формулювання постановки задачі; конструювання алгоритму розв'язання задачі; складання програми за розробленим алгоритмом; введення в ЕОМ програми і вихідних даних; налагодження і тестування програми; отримання розв’язку та аналіз результатів.

Алгоритм – це кінцева послідовність чітко визначених дій, які призводять до однозначного вирішення поставленого завдання. Головна особливість будь-якого алгоритму – формальне виконання, що дозволяє виконувати задані дії (команди) не тільки людині, але і різним технічним пристроям (виконавцям). Процес складання алгоритму називається алгоритмізацією.

Алгоритми характеризуються обчислювальною і ємнісною складністю. За видом використовуваної обчислювальної моделі алгоритми діляться на послідовні (або детерміновані), паралельні (або недетерміновані), розподілені та ін. Алгоритм можна реалізувати в ЕОМ, якщо він містить тільки елементарні команди. Елементарними, тобто не потребуючими деталізації, можна вважати такі команди (або операції): 1) початок, кінець; 2) список даних; 3) введення, виведення; 4) обчислювальні операції, реалізовані оператором присвоювання.

Для алгоритму характерні такі *властивості*: 1) детермінованість (або визначеність), тобто однозначність його розуміння для будь-якого виконавця, що приводить до точного виконання однієї і тієї ж послідовності дій; 2) результативність, чи спрямованість, тобто властивість досягнення за кінцеву кількість простих кроків шуканого результату задачі; 3) масовість, тобто придатність до розв'язання будь-якої задачі з деякого класу задач.

Розрізняють такі способи подання алгоритмів: текстовий, операторний і графічний. Найбільше поширення в наш час одержав графічний спосіб, при якому обчислювальний процес розчленовується на окремі операції, що відображаються у вигляді умовних графічних символів (блоків).

Послідовність блоків і сполучних ліній утворюють блок–схему. Блок–схеми розташовуються зверху вниз. Лінії з'єднання окремих блоків показують напрямок процесу обробки в схемі. Кожний такий напрямок називається гілкою. Незалежно від структури алгоритм завжди має по одному блоку «Початок» і «Кінець». Його гілки повинні в кінці зійтися, і за якою б гілки не було б розпочато рух, він завжди має привести до блоку «Кінець».

З 01.01.92 введено ДСТ 19.701-90 (ІСО 5807-85), який визначає правила виконання схем і перелік блоків, їхнього найменування, форму і розміри. Блоки з'єднуються між собою у визначеній послідовності стрілками. Усередині блоків у вигляді формул чи тексту вказують інформацію, яка пояснює та характеризує виконувані ними дії. Для визначення алгоритму за допомогою блок-схеми використовуються чітко визначені блоки, основні типи яких наведено в табл. 2.1. Розміри блоків розраховують відповідно до таких правил: 1) найменший геометричний розмір a={10, 15, 20, …} мм; 2) найбільший геометричний розмір $b = 1{,}5a$.

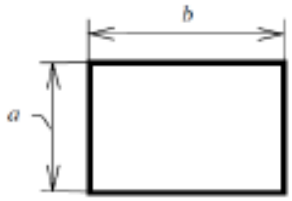
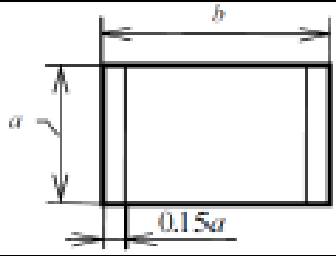
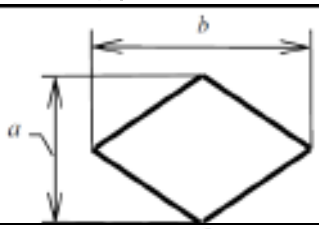
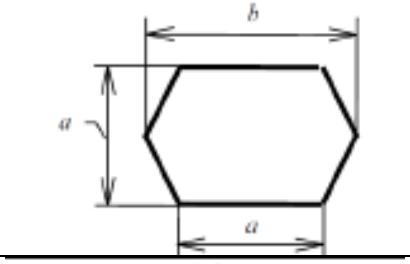
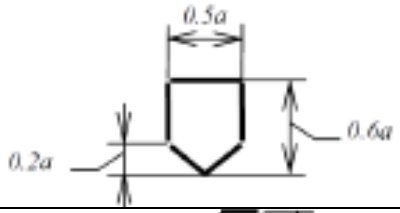
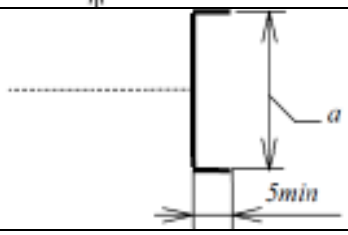
Теорія структурного програмування доводить, що алгоритм будь-якого ступеня складності можна побудувати за допомогою основного базового набору структур: послідовна (лінійна) структура; структура, що розгалужується; циклічна структура.

Найпростішими для розуміння і використання є лінійні структури.

Лінійним називають алгоритм (фрагмент алгоритму), в якому окремі команди виконуються послідовно одна за іншою, незалежно від значень вхідних даних і проміжних результатів. Розглянемо приклад алгоритму лінійної структури (рис. 2.1), який описує алгоритм обчислення площі трикутника зі сторонами *a, b, c* за формулою Герона (спочатку треба визначити значення на півпериметру трикутника, потім розрахувати його площину).

Таблиця 2.1

Графічні елементи, з яких будуються блок-схеми

Назва	*Графічний блок*	*Функціональне призначення*
Обчислювальний блок		Обчислення або послідовність обчислень
Визначений процес		Виконання підпрограми (функції)
Логічний блок (блок умови)		Перевірка умови
Цикл		Початок циклу
Перехід		З’єднання між двома сторінками
Коментарі		Пояснення

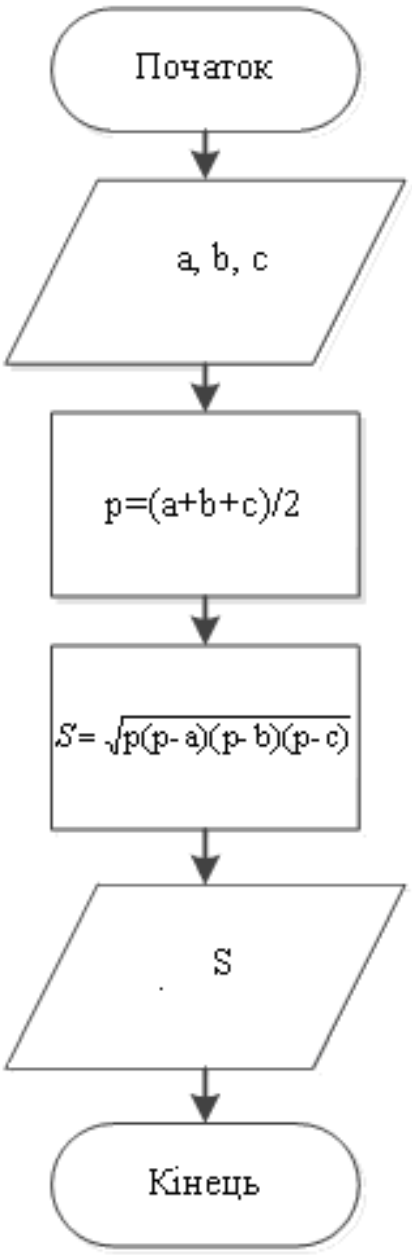


Рис. 2.1. Блок-схема алгоритму обчислення площини трикутника

2. СТВОРЕННЯ СХЕМ АЛГОРИТМІВ ЗАСОБАМИ MICROSOFT VISIO

MS Visio – засіб створення різних діаграм, у тому числі інформаційних моделей процесів, блок-схем, структурних схем, графіків робіт, призначений для фахівців; керівників проектів, він дозволяє описувати (моделювати) різні інформаційні технології і системи в керуванні й проектуванні.

Створення будь-якої діаграми розпочинається з команди Файл-створити - Вибір типу діаграми. Створення блок схеми розпочинається з того, що на діалоговій панелі вибираємо елемент меню – блок-схема, потім - Проста блок-схема. Після вибору типу діаграми відкривається порожнє вікно (blank drawing) - Visio готове для створення діаграми.

Ліворуч (на зеленому полі) розташовується перелік символів, специфічний для обраного типу *діаграми*.

Символи можуть розташовуватися на бланку діаграми простою операцією перетаскування (ліворуч/праворуч). Для кожного символу надається короткий опис, для цього досить встановити курсор миші на відповідному символі.

Процес побудови діаграм в Visio. Спочатку на робочий аркуш перетягуються необхідні символи-примітиви (SmartShapes), потім будуються зв'язки між ними і розташовуються підписи (текст). Технологія (Smart Connector) дозволяє переміщувати примітиви на робочому аркуші, при цьому зв'язки автоматично перебудовуються, а у випадку перетинання сполучних ліній автоматично створюється «перехрестя» у вигляді арки, яка складається з дуги або декількох (від двох до семи) відрізків.

Перш ніж розпочати створення діаграми, виберемо режим, в якому всі її елементи будуть з'єднуватися автоматично. Для цього використовуємо кнопку Автосоединение (вкл/выкл), розташовану на панелі інструментів

Комп'ютерний практикум № 3. Алгоритми послідовної (лінійної) структури. Числа та операції над ними. Введення даних з клавіатури

Мета роботи: ознайомитися з алгоритмами послідовної (лінійної) структури, з процедурами запуску программ, які реалізують ці алгоритми на мові Python; знайомство з інтегрованим середовищем розробки – integrated development environment (IDLE). *Об'єкт дослідження* – алгоритми послідовної (лінійної) структури, процедури запуску программ, середовище програмування IDLE.

ПЛАН

1. Python: запуск програми користувачем
2. Числа та операції над ними

Завдання

1. Вивчити теоретичні основи написання алгоритмів послідовної (лінійної) структури. Опрацювати приклади.

2. Відповідно до свого варіанту написати програму, яка, використовуючи складові модуля *math*, розраховує значення виразу (для тригонометричних функцій аргументи задано в радіанах, для введення даних з клавіатури та виведення даних на екран використовувати функції введення-виведення).

3. Засвоїти дві процедури запуску програми:

- в командному рядку;
- з інтерфейсу IDLE.

4. Скласти звіт і захистити його по роботі.

Варіанти

Відповідають лабораторній роботі № 2.

Завдання 1. *Запуск програми в командному рядку DOS* (більшість програмістів для розробки програм використовують вікно терминала з командною оболонкою і вікно текстового редактора)

1. За допомогою текстового редактора «Блокнот» («Notepad») створити файл *lab_rob*3.*py*, який містить в собі код програми (інструкції мови).

2. Запустити цей файл на виконання за допомогою інтерпретатора Python в командному рядку DOS:

меню «Все программы (Programs)», яке з'являється після кліку на кнопці Пуск (Start)),

меню Accessories (Стандартные) → Command Prompt (Командная строка) або Пуск (Start) → Выполнить… (Run…) → *cmd*.

3. Зафіксувати результат виконання коду програми (клавіша PtrScr).

```
>>> import sys
>>> a=sys.path
>>> a
['', 'C:\\WINDOWS\\system32\\python34.zip', 'C:\\Python34\\DLLs', 'C:\\Python34\
\lib', 'C:\\Python34', 'C:\\Python34\\lib\\site-packages']
>>>
```

Завдання 2. *Запуск програми у середовищі IDLE.*

1. За допомогою текстового редактора IDLE створити текстовий файл *l_r3.py*, який містить код програми (IDLE автоматично не додає розширення до імені файла, навіть «*.*py*»).

2. Запустити цей файл на виконання за допомогою інтерфейсу IDLE.

3. Зафіксувати результат виконання коду програми (клавіша PtrScr).

Програми мають бути записані в окремих файлах, які можуть виконуватися консольною командою python <ім'я файлу>.py.

Результат обчислень має виводитися на екран командою *print*.

Кожна програма має сприймати довільні значення, які задовольняють описаному в завданні формату, і передаються у вигляді аргументів командного рядка при виклику програми: *python <ім'я файлу>.py*

Формат введення даних та виведення результатів має чітко збігатися із вказаним.

Контрольні запитання

1. Що таке алгоритм послідовної (лінійної) структури, програма послідовної (лінійної) структури?

2. Які основні типи даних визначені в мові Python.

Аудиторна робота

Приклад 3.1. Вбудована функція *input* – це засіб отримання результату введення з клавіатури – вона виводить підказку, текст якої міститься в необов'язковому аргументі-рядку, і повертає введену користувачем відповідь у вигляді рядка. Функція *print* (засіб виведення) може приймати декілька параметрів, які розділяються комами

print ('first', 'second')

Результат – first second

Функція input призупиняє виконання програми, поки користувач не введе значення і натисне клавішу Enter. Вона повертає введене значення у вигляді рядка.

1)
```
text = input('Enter some text: '); print(text)
```

2)
```
string = input('Введите строку: ')  # или string = input()
print('Вы ввели "', string, '"', sep='')
# введём два числа
n = int(input('Введите первое число: '))
m = int(input('Введите второе число: '))
print('{} + {} = {}'.format(n, m, n + m))
```

3. Функції *bin, oct, hex* повертають рядки, які зображують задане число у відповідних с/ч

```
number = int(input('Введіть число: '))
print('Двійкова с/ч:      ', bin(number))
print('Вісімкова с/ч:     ', oct(number))
print('Шістнадцяткова с/ч:', hex(number))
```

Результат

```
Введіть число: 10
Двійкова с/ч:     0b1010
Вісімкова с/ч:      0o12
Шістнадцяткова с/ч: 0xa
```

4. Функції *min*, *max* повертають мінімальне/максимальне значення

```
a = 5;b = 7;c = 2
print(min(a, b, c)) # мінімальне значення
print(max(a, b, c)) # максимальне значення
```

5. Обчислити значення функції $y = (\sin(\pi/2)/b)\cdot\sqrt{m}$. Значення вхідних даних *m*=4, *b*=2.

```
import math as m1
 m=float(input("m=")); b=float(input("b="))
 a=m1.sin(m1.pi/2)
 y=(a/b)*m1.sqrt(m); print("y=", y)
```

Результат

```
m=4
b=2
y= 1.0
```

Приклад 3.2. Арифметичні операції

1.

```
# Функція float конвертує значення-рядок у дійсне,
# з яким можна виконувати арифметичні операції
x = float(input('Введите первое число: '))
y = float(input('Введите второе число: '))
# operation — рядок
operation = input('Введите знак арифметической операции: ')
# Присвоїмо змінній result значення None, яке вказує, що
# значення об’єкта не відомо
result = None
# if-elif-else (якщо - інакше якщо - інакше) — умовний оператор.
# дозволяє виконувати різні фрагменти кода в залежності від умов
# Операція «==» перевіряє два значення на рівність
if operation == '+':
    # до чисел можна застосувати  арифметичні операції
    result = x + y
elif operation == '-':
    result = x - y
elif operation == '*':
    result = x * y
elif operation == '/':
    result = x / y
elif operation == '^':
    result = x ** y  # ** — операція піднесення до степеня
else:
    print('Операція, яка не підтримується')
# отримаємо результат, якщо операція була припустимою
```

```
if result is not None:  print(result)
```
2.
```
x = 10; y = 8
print(x + y)  # додавання
print(x + 3)
print(x - y)  # віднімання
print(x * y)  # добуток
print(x / y)  # ділення
print(x // y) # цілочисельне ділення
print(x % y)  # залишок від ділення
print(x ** y) # піднесення до степеню
print(3.2 * 0.8 - 2 * 5 - 3**3)  # арифметичний вираз
print(4 ** 0.5)  # піднесення до степеню 0.5 – квадратний корінь

z = -2
print(abs(z))              # модуль числа
print(pow(z, 2), z ** 2)  # квадрат числа

m = 3.26
print(round(m), round(m, 1))
# Округлення числа до цілого і до одного знака після коми
```

Приклад 3.3.

1. Операції на основі модуля *math*

```
import math # імпортуємо модуль math
x = 3.265
# ціле число, найближче ціле знизу, найближче ціле зверху
print(math.trunc(x), math.floor(x), math.ceil(x))
print(math.pi)  # константа пі
print(math.e)   # число Эйлера

y = math.sin(math.pi / 4)  # math.sin – синус
print(round(y, 2))

y = 1 / math.sqrt(2)  # math.sqrt – квадратний корінь
print(round(y, 2))
```

2. Деякі математичні функції

```
>>> import math
>>> math.pi, math.e # константи pi, e=2.71
(3.1415926535897931, 2.7182818284590451)
>>> math.sin(2*math.pi/180) # Синус
0.034899496702500969
>>> math.sqrt(144), math.sqrt(2) # Квадратний корінь
(12.0, 1.4142135623730951)
>>> math.pow(2, 4), 2 ** 4 # Піднесення до степеня
(16, 16)
>>> abs(-42.0), sum((1, 2, 3, 4)) # абсолютне значення, сума
(42.0, 10)
>>> min(3, 1, 2, 4), max(3, 1, 2, 4) # Мінімум, максимум
(1, 4)
>>> math.log(10) #ln(10)
```

math.log(*x*[, *base*]) - with one argument, return the natural logarithm of *x* (to base *e*).

math.log1p(*x*) - return the natural logarithm of *1+x* (base *e*).

Теоретична частина

1. ЗАПУСК ПРОГРАМИ КОРИСТУВАЧЕМ

Розглянемо способи запуску програмного коду, який був збережений у файлі.

1. Запуск файлів з командного рядка. Відкрийте текстовий редактор (наприклад, Notepad або редактор IDLE) і збережіть такі інструкції в файлі з ім'ям *script1.py* (перший сценарій на мові Python):

```
import sys # Завантажує бібліотечний модуль
print(sys.platform)
print(2 ** 100) # Підносить число 2 до степеня 100
x = 'Spam!'
print(x * 8) # Дублює рядок
```

Цей файл:

1. Імпортує модуль *sys* Python (бібліотеку додаткових інструментів), щоб пізніше отримати назву платформи ОС.

2. Тричі викликає функцію *print*, щоб відобразити результати

3. Використовує змінну з ім'ям *x*, утворену в момент, коли їй присвоюють значення у вигляді об'єкта-рядка.

4. Виконує деякі операції над об'єктами.

Ім'я *sys.platform* – це змінна-рядок, вміст якої ідентифікує тип комп'ютера, на якому виконується сценарій. Ця змінна знаходиться в модулі з ім'ям *sys*, який необхідно завантажити з допомогою інструкції *import.*

Коментар – це текст, розташований за символом #. Коментарі можуть займати окремий рядок або додаватися в рядок з програмним кодом, праворуч від нього. Текст, розташований за символом «#», інтерпретатором ігнорується.

Файл модуля називається *script1.py*. Імена файлів з програмним кодом, які передбачається імпортувати з інших файлів, повинні закінчуватися розширенням *.py. Якщо зберегти цей текстовий файл, то можна запропонувати інтерпретатору Python виконати його, вказавши повне ім'я файлу в якості першого аргументу команди *python*, ввівши такий рядок в системній командному рядку:

```
% python script1.py
win32
1267650600228229401496703205376
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

І в цьому випадку також ви повинні використовувати командну оболонку, яка надається операційною системою - у вікні командного рядка (Command Prompt) в Windows. Не забувайте замінювати слово «python» на повний шлях до виконуваного файлу інтерпретатора, якщо змінна оточення PATH у вас не налаштована.

Якщо все було зроблено правильно, ця команда запустить інтерпретатор Python, який в свою чергу послідовно, рядок за рядком, виконає інструкції в файлі, і ви побачите на екрані результати виконання трьох інструкцій *print* – назва платформи, результат піднесення числа 2 до степеня 100 і результат багаторазового дублювання рядка. Якщо щось пішло не так, на екрані з'явиться повідомлення про помилку. Оскільки в даній ситуації для запуску програм використовують командну оболонку, можна застосовувати будь-які синтаксичні конструкції, припустимі у командній оболонці. Наприклад, можна направити висновок сценарію Python у файл, щоб детально досліджувати отримані результати пізніше, як показано нижче:

```
% python script1.py > saveit.txt
```

У цьому випадку три рядки, показані в попередньому прикладі запуску сценарію, не будуть виводитися на екран, а будуть записані в файл *saveit.txt*. Це – можливість перенаправлення потоків, її можна використати як для виведення тексту, так і для введення. В середовищі Windows цей приклад буде працювати так само, хоча командний рядок буде виглядати інакше:

```
C:\Python30> python script1.py
win32
1267650600228229401496703205376
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

Якщо у вас змінна оточення PATH не налаштована і не був виконаний перехід в каталог інтерпретатора, необхідно вводити повний шлях до виконуваного файлу інтерпретатора Python:

```
D:\temp> C:\python30\python script1.py
win32
1267650600228229401496703205376
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

У новітніх версіях Windows можна просто вводити ім'я файлу сценарію незалежно від того, в якому каталозі ви перебуваєте, бо новітні версії системи Windows відшукують програми, необхідні для запуску файлів, за допомогою реєстру Windows, і вам не потрібно явно вказувати її в командному рядку. Наприклад, в сучасних версіях Windows попередню команду можна спростити:

```
D:\temp> script1.py
```

Необхідно вказувати повний шлях до файлу сценарію, якщо він знаходиться в каталозі, відмінному від того, в якому ви працюєте. Наприклад, команда

```
D:\other> python c:\code\otherscript.py
```

буде працювати в каталозі *D:\other*, якщо шлях до команди *python* включений в змінну оточення PATH, при цьому вона повинна запустити сценарій, розташований в каталозі *D:\other*.

Якщо змінна оточення PATH не включає шлях до каталогу з виконуваним файлом інтерпретатора Python і при цьому файл сценарію не знаходиться в

поточному робочому каталозі, тоді необхідно вказати повний шлях як до виконуваного файлу інтерпретатора, так і до файлу сценарію:

```
D:\other> C:\Python30\python c:\code\otherscript.py
```

2. Інтерфейс користувача IDLE. Програма IDLE може запропонувати вам графічний інтерфейс користувача для розробки програм на мові Python (IDLE – інтегроване середовище розробки – integrated development environment). IDLE – це набір інструментальних засобів з графічним інтерфейсом, який здатний працювати на самих різних платформах, включаючи Microsoft Windows. IDLE - це програма на мові Python, яка створює графічний інтерфейс за допомогою бібліотеки *tkinter* GUI, що забезпечує її переносимість, але також означає, що для використання IDLE вам доведеться забезпечити підтримку *tkinter* в Python (версія Python для Windows має таку підтримкою за замовчуванням).

Запуск IDLE в Windows – для неї створюється окремий пункт в розділі Python меню кнопки Пуск (Start). У Windows IDLE є сценарієм Python, який за замовчуванням знаходиться в каталозі *C:\Python30\Lib\idlelib*.

В IDLE присутні звичні пункти меню, а для виконання найбільш поширених операцій можна використовувати короткі комбінації клавіш.

Щоб створити (або відредагувати) файл з вихідним програмним кодом в середовищі IDLE, відкрийте вікно текстового редактора: в головному вікні відкрийте меню File (Файл) → пункт New Window (Нове вікно) – відкрити вікно текстового редактора (або Open ... (Відкрити) – щоб відредагувати існуючий файл). Для запуску сценарію, відкритого у вікні редагування, використовують меню «Run» цього вікна (пункт «Run Module»).

IDLE забезпечує підсвічування синтаксису програмного коду, який вводиться як в головному вікні, так і у всіх вікнах текстового редактора – ключові слова виділяються одним кольором, літерали іншим кольором і т. д. Це дозволяє візуально виділяти елементи програмного коду і розрізняти синтаксичні елементи програмного коду.

Щоб запустити файл з програмним кодом в середовищі IDLE, виберіть вікно, де редагується текст, розкрийте меню Run (Запустити) і виберіть в ньому пункт Run Module (Запустити модуль) або скористайтеся комбінацією клавіш, яка відповідає цьому пункту меню. Якщо з моменту відкриття або останнього збереження файлу його вміст змінювалося, Python запропонує зберегти його.

Коли сценарій запускається таким способом, весь висновок, який він генерує, а також всі повідомлення про помилки з'являються в основному вікні інтерактивного сеансу роботи з інтерпретатором (командна оболонка Python).

Крім основних функцій редагування і запуску середовище IDLE надає цілий ряд додаткових можливостей, включаючи налагоджувач і інспектор об'єктів. Налагоджувач IDLE активується за допомогою меню Debug (Налагодження), а інспектор об'єктів – за допомогою меню File (Файл). Інспектор об'єктів дозволяє переходити, переміщаючись по шляху пошуку модулів, до файлів і об'єктів в файлах – клацання на файлі або об'єкті

призводить до відкриття відповідного вихідного тексту в вікні редагування.

Режим налагодження в IDLE ініціюється вибором пункту меню Debug (Налагодження) → Debugger (Отладчик) головного вікна, після цього можна запустити налагоджувач сценарію вибором пункту меню Run (Запустити) → Run Module (Запустити модуль). Як тільки налагоджувач буде активований, клацанням правої кнопки миші на вибраному рядку у вікні редагування ви зможете встановлювати точки зупинки в своєму програмному коді, щоб призупиняти виконання сценарію, переглядати значення змінних тощо Ви зможете стежити за ходом виконання програм – в цьому випадку поточний виконуваний рядок програмного коду виділяється кольором.

У разі появи помилок можна натиснути правою кнопкою миші на рядку з повідомленням про помилку і швидко перейти до рядка програмного коду, який викликав цю помилку. Це дозволяє швидко з'ясувати джерело помилки і ліквідувати її. Крім цього текстовий редактор IDLE володіє великим набором можливостей, які стануть в нагоді програмістам, включаючи автоматичне оформлення відступів, розширений пошук тексту і файлів тощо.

2. Числа та операції над ними

Крім числових літералів, наведених в табл. 1.6, Python надає набір операцій для роботи з числовими об'єктами:

1. Оператори виразів +, –, *, /, >>, **, &, інші.
2. Вбудовані математичні функції *pow, abs, round, int, hex, bin,* інші.
3. Допоміжні модулі *random, math,* інші.

Для роботи з числами в основному використовуються вирази, вбудовані функції і модулі (при цьому числа мають ряд власних, специфічних методів). Дійсні числа, наприклад, мають метод *as_integer_ratio,* який зручно використовувати для перетворення дійсного числа в раціональне, а також метод *is_integer_method*, який перевіряє – чи можна подати дійсне число як ціле значення. Цілі числа теж мають різні атрибутами, включаючи метод *bit_length,* він повертає кількість бітів, необхідних для подання значення числа. Крім того, множини підтримують власні методи і оператори виразів.

При роботі з числами часто використовують вирази: комбінації чисел (або інших об'єктів) і операторів, які повертають значення при виконанні інтерпретатором Python. Вирази в Python записуються з використанням звичайної математичної нотації і символів операторів. Наприклад, складання двох чисел X і Y записується у вигляді виразу X+Y, яке наказує інтерпретатору Python застосувати оператор «+» до значень з іменами X і Y. Результатом виразу X+Y буде інший числовий об'єкт. У табл. 3.1 наведено перелік всіх операторів, наявних в Python (математичні оператори: +, –, *, / і т. д.; оператор % обчислює залишок від ділення, оператор «<<» виконує побітовий зсув вліво, оператор «&» виконує побітову операцію «І» і т. д.). Деякі оператори більш характерні для Python. Наприклад, оператор «*is*» перевіряє ідентичність об'єктів, оператор *lambda* створює неіменовані функції. Нерівність значень можна перевірити як X! = Y. Операція ділення з округленням вниз (X//Y) усікає

дробову частину. Операція ділення X / Y виконує справжнє ділення (повертає результат з дробовою частиною).

Таблиця 3.1

Оператори виразів в Python і правила визначення старшинства

Оператори	Опис
yield x	Підтримка протоколу send у функціях-генераторах
lambda args: expression	Створює анонімну функцію
x if y else z	Тримісний оператор вибору (значення x обчислюється, тільки якщо значення y істинно)
x or y	Логічна операція «АБО» (значення y обчислюється, тільки якщо значення x = хибність)
x and y	Логічний оператор «І» (значення y обчислюється, тільки якщо значення x= істина)
not x	логічне заперечення
x in y, x not in y	Перевірка на входження (для ітерованих об'єктів і множин)
x is y, x is not y	Перевірка ідентичності об'єктів
x < y, x <= y, x> y, x>= y x == y, x != y	Оператори порівняння, перевірка на підмножину і над множину Оператори перевірки на рівність
x | y	Бітова операція «АБО», об'єднання множин
x ^ y	Бітова операція «виключне АБО» (XOR), симетрична різниця множин
x & y	Бітова операція «І», перетин множин
x << y, x >> y	Зсув значення x вліво або вправо на y бітів
x, + y x – y	Додавання, конкатенація Віднімання, різниця множин
x * y x % y x / y, x // y	Множення, повторення Залишок, формат Ділення: справжнє і з округленням вниз
-x, +x	Унарний знак «мінус», тотожність
~x	Бітова операція «НЕ» (інверсія)
x ** y	Піднесення до степеню
x[i]	Індексація (в послідовності, відображеннях тощо)
x[i:j:k]	витяг зрізу
x(...)	Виклик (функцій, класів і інших об'єктів)
x.attr	Звернення до атрибуту
(...)	Кортеж, підвираз, вираз-генератор
[...]	Список, генератор списків
{...}	Словник, множина, генератор словників і множин

Комп'ютерний практикум № 4. Алгоритми розгалуженої структури (інструкція if)

Мета роботи: ознайомитися з алгоритмами розгалуженої структури та їх реалізацією; можливою обробкою помилок (виключень). *Об'єкт дослідження* – умовний оператор (процедурна інструкція *if*), алгоритми розгалуженої структури, виключення у вигляді рядків.

ПЛАН

1. Динамічна типизація.
2. Умовна інструкція *if*.
3. Обробка виключень (помилок)

Завдання

1. Вивчити теоретичні основи написання алгоритмів розгалуженої структури. Опрацювати приклади.
2. Побудувати блок-схему алгоритму вирішення завдання.
3. Відповідно до свого варіанту
- визначити умови;
- за допомогою формул описати варіанти виконання необхідний дій;
- написати програму, яка розв'язує завдання.
- організувати введення даних з клавіатури, виведення у консоль.
4. Реалізувати обробку помилок (виключень у вигляді рядків)
   - за допомогою інструкції *try*;
   - перевіривши зміст рядку введення за допомогою методу *isdigit.*
5. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасировки виконання програми, переробку разгалужень, які входять одне до одного.

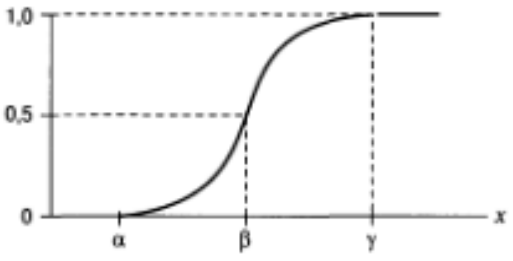
Вимоги: не можна використовувати масиви, цикли, власні функції.

Варіанти

1-13. Напишіть програму, яка обчислює значення визначеної функції (на вхід подають дійсні числа).

1. Задано функцію S з параметрами $\alpha=l$, $\beta=0{,}5(l+r)$, $\gamma=r$ ($l<r$) вигляду:

$$S(x;l,r)=\begin{cases}0, & x\le l;\\ 2\left(\dfrac{x-l}{r-l}\right)^2, & l<x<\dfrac{l+r}{2};\\ 1-2\left(\dfrac{r-x}{r-l}\right)^2, & \dfrac{l+r}{2}<x\le r;\\ 1, & r<x.\end{cases}$$



2. Задано функцію Z з параметрами l та r ($l<r$), яку визначають через функцію S:

$$Z(x;l,r)=1-S(x;l,r),\ \text{де}\ \ S(x;l,r)=\begin{cases}0, x\le l;\\ 2\left(\dfrac{x-l}{r-l}\right)^2, l<x<\dfrac{l+r}{2};\\ 1-2\left(\dfrac{r-x}{r-l}\right)^2, \dfrac{l+r}{2}<x\le r;\\ 1, r<x.\end{cases}$$

3.Задано функцію π з параметрами a,c, яку визначають через функції S, Z:

$$\pi(x;a,c)=\begin{cases}S(x;c-a,c), x\le c;\\ Z(x;c,c+a), x>c;\end{cases}$$

де $S(x;l,r)=\begin{cases}0, x\le l;\\ 2\left(\dfrac{x-l}{r-l}\right)^2, l<x<\dfrac{l+r}{2};\\ 1-2\left(\dfrac{r-x}{r-l}\right)^2, \dfrac{l+r}{2}<x\le r;\\ 1, r<x;\end{cases}$ $Z(x;l,r)=1-S(x;l,r)$

4. Задано функцію Гаусса з параметрами c_1, c_2, σ_1, σ_2 вигляду:

$$ts\_gaussian(x;c_1,\sigma_1,c_2,\sigma_2)=\begin{cases}exp\left[-\dfrac{1}{2}\left(\dfrac{x-c_1}{\sigma_1}\right)^2\right], & x\le c_1;\\ 1, & c_1<x<c_2;\\ exp\left[-\dfrac{1}{2}\left(\dfrac{x-c_2}{\sigma_2}\right)^2\right], & c_2\le x.\end{cases}$$

5. Задано функцію y з параметром b вигляду:

$$y(x,b)=\begin{cases}-x^2+b, \text{ якщо } x<0 \text{ та } b\ne 0;\\ \dfrac{x}{x-3}+5, \text{ якщо } x>0 \text{ та } b=0;\\ \dfrac{x}{-3} \text{ інакше.}\end{cases}$$

6. Задано функцію $y(x)$ вигляду: $y(x)=\begin{cases}1, \text{ якщо } x\le 0;\\ cos(x), \text{ якщо } 0<x\le pi;\\ -1 \text{ інакше.}\end{cases}$

7. Задано функцію ts_π з параметрами $\{a, b, c, d\}$, яку визначають через функції S і Z:

$$ts\_\pi(x;a,b,c,d)=\begin{cases}0, x\le a;\\ S(x;a,b), a<x<b;\\ 1, b\le x\le c;\\ Z(x;c,d), c<x<d;\\ 0, d\le x.\end{cases}$$ де $Z(x;l,r)=1-S(x;l,r)$, $S(x;l,r)=\begin{cases}0, x\le l;\\ 2\left(\frac{x-l}{r-l}\right)^2, l<x<\frac{l+r}{2};\\ 1-2\left(\frac{r-x}{r-l}\right)^2, \frac{l+r}{2}<x\le r;\\ 1, r<x.\end{cases}$

8. Задано функцію *triangle* з трьома параметрами $\{a, b, c\}$, де $a < b < c$, вигляду:

$$\mu(x)=triangle(x;a,b,c)=\begin{cases}\frac{x-a}{b-a}, \ a\le x\le b;\\ \frac{c-x}{c-b}, \ b\le x\le c;\\ 0, \ x\le a \text{ або } c\le x.\end{cases}$$

9. Задано функцію *trapezoid* з чотирма параметрами $\{a, b, c, d\}$, де $a < b \le c < d$, вигляду:

$$\mu(x)=trapezoid(x;a,b,c,d)=\begin{cases}\frac{x-a}{b-a}, \ a\le x\le b;\\ 1, \ b\le x\le c;\\ \frac{d-x}{d-c}, \ c\le x\le d;\\ 0, \ d\le x \text{ або } x\le a.\end{cases}$$

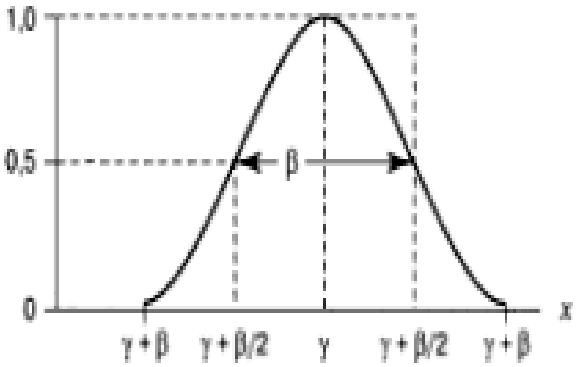
10. Задано функцію S з параметрами α, β, γ ($\alpha<\beta<\gamma$) вигляду:

$$S(x;\alpha,\beta,\gamma)=\begin{cases}0, \text{ для } x\le\alpha\\ 2\left(\frac{x-\alpha}{\gamma-\alpha}\right)^2, \text{для } \alpha\le x\le\beta\\ 1-2\left(\frac{x-\gamma}{\gamma-\alpha}\right)^2, \text{для } \beta\le x\le\gamma\\ 1, \text{ для } x\ge\gamma\end{cases}$$

11. Задано функцію π на основі функції S з параметрами $\{\beta, \gamma\}$ вигляду:

$$\Pi(x;\ \beta,\ \gamma)=\begin{cases}S(x;\gamma-\beta,\gamma-\beta/2,\gamma), \text{ якщо } x\le\gamma\\ 1-S(x;\gamma,\gamma+\beta/2,\gamma+\beta), \text{в інших випадках}\end{cases}$$

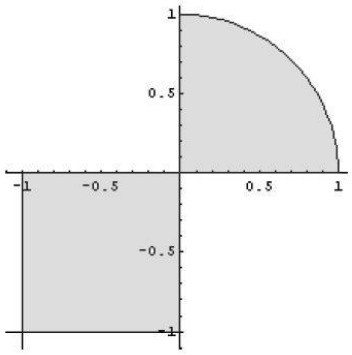
П-функція досягає нуля в точках $x = \beta \pm \gamma$ (параметр β визначає загальну ширину), а координати точок перетину функції з прямою $\pi = 0{,}5$ визначають як $x = \beta \pm \gamma / 2$.



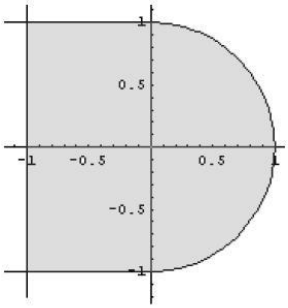
12. Задано функцію *y* з параметром *b* вигляду:

$$y(x,b)=\begin{cases} 2x^2+b, & \textit{якщо } x<1 \textit{ та } b\neq 0; \\ \dfrac{x}{4x-1}+6, & \textit{якщо } x>1 \textit{ та } b=0; \\ \dfrac{x}{-2} & \textit{інакше.} \end{cases}$$

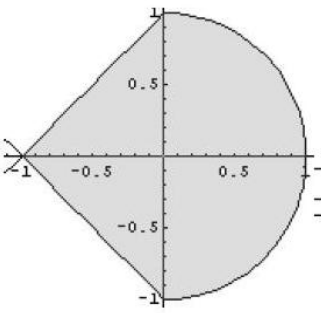
13-25. Напишіть програму, яка визначає, чи потрапляє задана точка (*x*, *y*) всередину вказаної області:



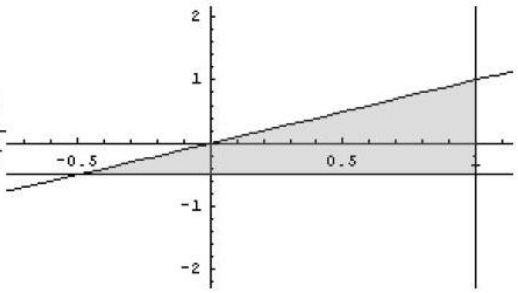
13



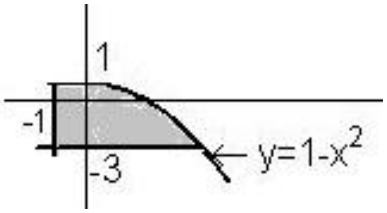
14



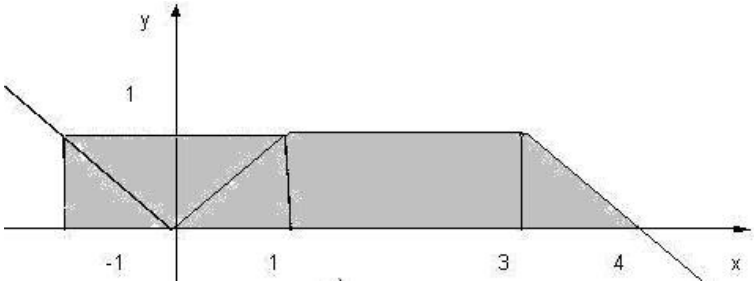
15



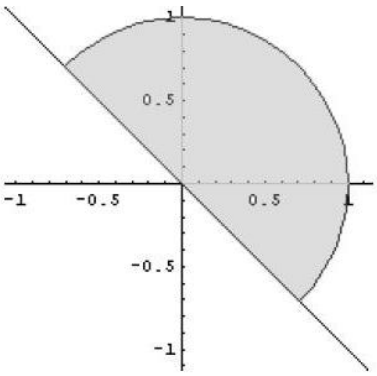
16



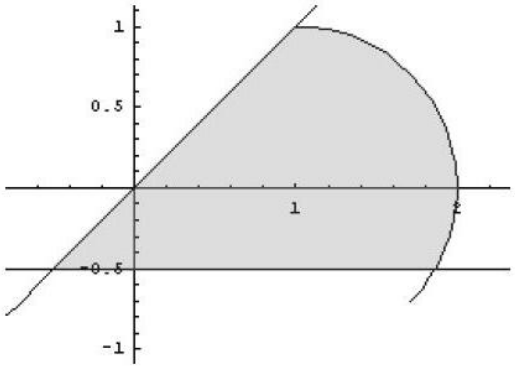
17



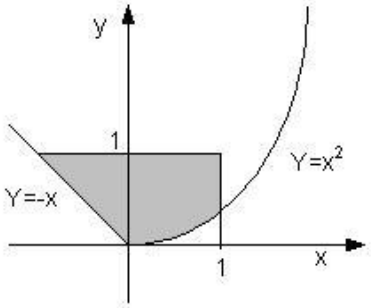
18



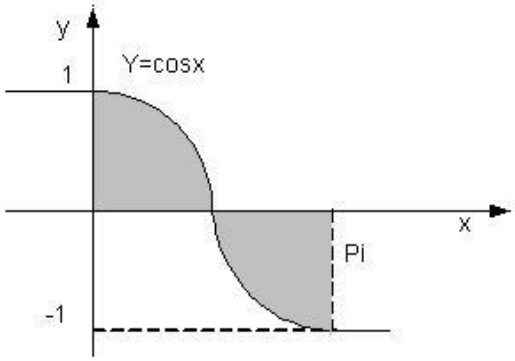
19



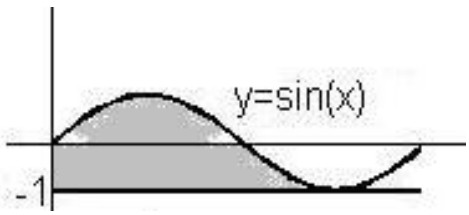
20



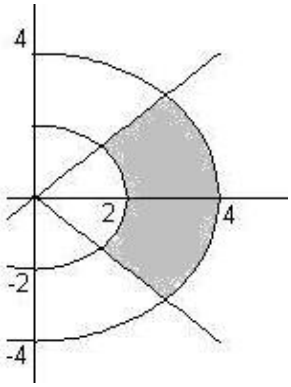
21



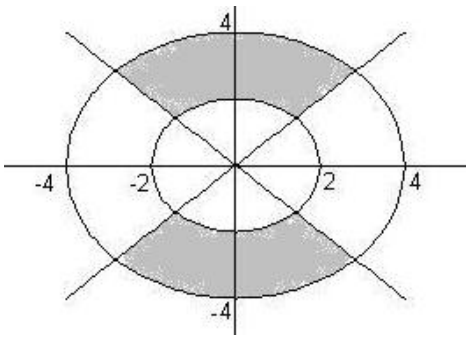
22



23



24



25

26. Нехай задано числа a_1, b_1, c_1, a_2, b_2, c_2. Вивести значення координат точки перетину прямих, які описують рівняння $a_1x+b_1y=c_1$ і $a_2x+b_2y=c_2$, або повідомити, що ці прямі співпадають, не перетинаються або не існують.

27. Нехай задано числа a, b, c - коефіцієнти рівняння $ax^2+bx+c=0$ ($a\neq0$). Вивести значення коренів $x1$, $x2$ (якшо вони існують) та змінної $t=true$, інакше $t=false$ (якщо коренів немає, повідомити про це).

28. Нехай задано числа *a*, *b* і *c* - довжина сторін трикутника. Якщо не можна побудувати трикутник із таким набором значень сторін, то надрукувати «0», інакше – «3», «2» або «1» в залежності від того, який це трикутник (рівносторонній, рівнобедрений або який-небудь інший).

Контрольні запитання

1. Що таке алгоритм розгалуженої структури?
2. В чому полягає різниця між умовними операторами з однією та двома гілками?
3. Що таке логічне висловлювання? Назвіть логічні операції, які використовують в логічних висловлюваннях при написанні програм на мові Python.
4. Який синтаксис має оператор розгалуження (множинного розгалуження)?

Аудиторна робота

Приклад 4.1. Приклади логічних операцій та виразів

```
print('and:')
print(False and False);print(False and True)
print(True and False);print(True and True); print()

print('or:')
print(False or False);print(False or True)
print(True or False);print(True or True); print()

print('not:')
print(not False); print(not True); print()

# Логічні вирази
a = True;b = False;c = True
f = a and not b or c or (a and (b or c))
print(f)
```

Приклад 4.2. Приклади порівнянь

```
a = 2; b = 5
print(a < b)        # менше
print(b > 3)        # білше
print(a <= 2)       # менше або дорівнює
print(b >= 7)       # більше або дорівнює
print(a < 3 < b)    # подвійне порівняння
print(a == b)       # рівність
print(a != b)       # нерівність
print(a is b)       # ідентичність об'єктів в пам'яті
print(a is not b)   # a и b – різні об'єкти
#(хоча їхні значення  можуть бути  однаковими - рівними)
```

Приклад 4.3.

```
1)x = int(input('x = '))
# якщо число x більше 5, то буде виведено повідомлення
```

```
if x > 5: print(“x більше п’яти”)

2)value = None
# Синтаксис мови вимагає наявність в блоці хоча б одного
# оператору, тому існує спеціальний оператор pass,
# який нічого не виконує,його можна використовувати як заглушка
if value is not None: pass # TODO: add logic

3)x = int(input('x = '))
if x < 0: x += 1
print(x)

4)x = int(input('x = '))
if x > 0: y = x ** 0.5
else: y = x ** 2
print(y)

5)x = int(input('x = '))
y = int(input(' y = '))
if 0 < x < 7:
    print('Значення x належить до заданого діапазону, продовжуємо')
    y = 2 * x - 5
    if y < 0:
        print(“Значення y - від'ємне”)
    else:
        if y > 0:
            print('Значення y - додатнє')
        else:
            print('y = 0')

6)# Замінимо другий вкладений if у прикладі 5
# на оператор розгалуження з декількома умовами
if 0 < x < 7:
    print('Значення x належить до заданого діапазону, продовжуємо')
    y = 2 * x - 5
    if y < 0:
        print(“Значення y - від'ємне”)
    elif y > 0:
        print(''Значення y - додатнє')
    else: print('y = 0')

7)print('''Меню:
        1. Файл
        2. Вигляд
        3. Вихід
''')
choice = int(input('Ваш вибір: '))
if choice == 1:
    print('Ви обрали пункт меню "Файл"')
```

```
    elif choice == 2:
        print('Ви відкрили меню "Вигляд"')
    elif choice == 3:
        print('Завершення.')
    else:
        print('Деякий вибір')

    8)is_ready = True
    # в залежності від умови присвоюємо значення змінній
    if is_ready:
        state_msg = 'Ready'
    else:
        state_msg = 'Not ready yet'
    print(state_msg)
    9)is_ready = True
# Теж саме, що і в прикладі 8, але замість умовного оператора
# використовуємо умовний вираз
    state_msg = 'Ready' if is_ready else 'Not ready yet'
    print(state_msg)

    10)string = input('Enter a string: ')
    # теж саме, що if string is not None and string != ''
    if string:
        print('The string is {}'.format(string))
    number = int(input('Enter a number: '))
    if number: print('Число не дорівнює нулю')
    else: print('Число дорівнює нулю')
```

Приклад 4.4. Визначити середнє арифметичне заданої непустої послідовності додатних цілих чисел, за якою слідує «0» (це ознака кінця послідовності).

```
from math import *
a=input ('Input first number: ')
if not a.isdigit():
 print("String value can not be entered")
 print("or number is negative, restart the program!")
 input ("reload the program!")
else: a=int(a)
if a==0:
 print("The number can not be zero"); input ()
count=0;ar=0
ar+=a; count+=1
if a!=0:
    try:
        a=int(input('Input next number or Enter 0 to finish: '))
    except:
        print("String value can not be entered")
        print("or number is negative, restart the program!")
        input ("reload the program!")
    else:
        ar+=a; count+=1
ar=ar/count; print("Average: ",ar)
```

```
========== RESTART: C:/Python34/1_r4.py ===
Input first number: 1
Input next number or Enter 0 to finish: 2
Average:  1.5
```

Теоретична частина

1. ДИНАМІЧНА ТИПИЗАЦІЯ

У сценаріях на Python не виконують оголошення об'єктів певних типів. Типи даних визначають під час виконання, а не в результаті оголошень у програмному коді.

Змінні утворюються при виконанні операції присвоєння, можуть посилатися на об'єкти будь-яких типів і перш ніж до них можна буде звернутися їм необхідно присвоїти деякі значення.

Наприклад, якщо ввести таку інструкцію: *a*=3, інтерпретатор Python виконає цю інструкцію в такі три етапи: 1) створює об'єкт, який подає число 3, 2) створює змінну *a*, якщо вона ще відсутня, 3) у змінну *a* записується посилання на новостворений об'єкт, який подає число «3». Результатом виконання цих етапів буде структура, зображена на рис. 4.1: змінні і об'єкти зберігаються в різних частинах пам'яті і пов'язані між собою посиланням (посилання на рисунку зображено у вигляді стрілки). Змінні завжди посилаються на об'єкти і ніколи – на всі інші змінні, але великі об'єкти можуть посилатися на інші об'єкти (наприклад, об'єкт списку містить посилання на об'єкти, які включені в список).

Рис. 4.1. Імена та об'єкти після виконання операції присвоювання *a* = 3. Змінна перетворюється на посилання на об'єкт «3», у внутрішньому поданні змінна є вказівником на простір пам'яті з об'єктом, створеним в результаті інтерпретації літерального виразу «3»

Якщо використовують змінну (тобто вказівник), інтерпретатор Python переходить за посиланням від змінної до об'єкта. У конкретних термінах: *змінні* – це записи в системній таблиці, де передбачено місце для зберігання посилань на об'єкти, *об'єкти* – це області пам'яті за об'ємом, достатнім для подання значень цих об'єктів, *посилання* – це вказівники на об'єкти.

Коли в сценарії в результаті виконання виразу створюється нове значення, інтерпретатор Python створює новий об'єкт (тобто виділяє область пам'яті), яка подає це значення. Внутрішня реалізація інтерпретатора для оптимізації кешує і повторно використовує деякі типи незмінних об'єктів, такі як малі цілі числа і рядки (кожен «0» насправді не є новою областю в пам'яті).

Об'єкти мають більш складну структуру, ніж просто простір в пам'яті, необхідний для зберігання значення. Кожен об'єкт має два стандартних поля:

описувач типу (який використовують для зберігання інформації про тип об'єкта) і лічильник посилань (який використовують для визначення моменту, коли пам'ять, яку займає об'єкт, може бути звільнена).

Інформація про тип зберігається в об'єкті, але не в змінній.

Щоб побачити, як використовується інформація про типи об'єктів, подивимося, що відбувається, якщо виконати кілька операцій присвоювання одній і тій же змінній:

```
>>> a = 3 # це ціле число
>>> a = 'spam' # тепер це - рядок
>>> a = 1.23 # тепер це – дійсне  число
```

Цей програмний код працює таким чином: спочатку створюється ціле число, потім рядок і, нарешті, дійсне число. Тип – це властивість об'єкта, а не імені. У коді змінюється посилання на об'єкт. Тому що змінні не мають типів, ми насправді не змінюємо тип змінної – ми записуємо в змінну посилання на об'єкти інших типів. Змінні посилаються на конкретні об'єкти в конкретні моменти часу. З іншого боку, об'єкти знають, до якого типу вони відносяться, кожен об'єкт містить поле, в якому зберігається інформація про його типі. Цілочисельний об'єкт «3», наприклад буде містити значення «3» плюс інформацію, яка повідомить інтерпретатор Python, що об'єкт є цілим числом (це вказівник на об'єкт з назвою *int*, який відіграє роль імені цілочисельного типу). Описувач типу для рядка '*spam*' вказує на тип рядок (з ім'ям *str*). Оскільки інформація про тип зберігається в об'єктах, її не потрібно зберігати в змінних.

Таким чином, тип в мові Python – це властивість об'єктів, а не змінних. У програмному коді змінна зазвичай посилається на об'єкти тільки одного типу.

2. УМОВНА ІНСТРУКЦІЯ IF

Логіка висловлювань. Логічний тип даних. Числа можна порівнювати. Для цього в Python є такі операції порівняння:

> більше	<= менше чи рівне
< менше	== дорівнює
>= більше чи рівне	!= не дорівнює

Наприклад:

```
6>5  –  True
7<1  –  False
7==7  – True
7 != 7  – False
```

Python повертає значення True (Істина == 1), коли порівняння істинне, False (Хибність == 0) – в іншому випадку. True і False відносяться до логічного (булевого) типу даних *bool*. В программах часто використовують більш складні вирази – висловлювання (це – означає деяке твердження, яке може бути істинним або хибним). Кажуть, що істинність (True) або хибність (False) – це логічні значення висловлювання.

Логіка висловлювань вивчає способи, за допомогою яких з одних висловлювань можна утворювати інші, причому в такий спосіб, щоб істинність або хибність нових висловлювань залежала лише від істинності або хибності

старих. Для цього використовуються так звані (логічні) сполучники. Python використовує три логічних оператора *and*, *or*, *not* які відповідають сполучникам І, АБО, НЕ в логіці висловлювань.

A B A∧B

T	T	T
T	F	F
F	T	F
F	F	F

A B A∨B

T	T	T
T	F	T
F	T	T
F	F	F

A ¬A

F	T
T	F

Наприклад:

```
x = 8
y = 13
x == 8 and y < 15 # x дорівнює 8 та y меньше 15
x > 8 and y < 15 # x більше 8 та y меньше 15
x != 0 or y >15 # x не дорівнює 0 або y меньше 15
x < 0 or y >15 # x меньше 0 або y меньше 15
```

Для Python істинним або хибним може бути не тільки логічне висловлювання, а й об'єкт. Будь-яке число, яке не дорівнює нулю (або непорожній об'єкт) інтерпретують як «істина». Числа, які дорівнюють нулю, порожні об'єкти та спеціальний об'єкт *None* інтерпретують як «хибність». Наприклад: 1) " and 2 # False and True маємо результат «"»; 2) " or 2 # False or True – «2»

```
>>> y = 6>8; y
False
>>> not y
True
>>> not None
True
>>> not 2
False
>>> 2>4 and 45>3 #  False and  True поверне значення False
False
```

При обчисленні оператора *and* Python обчислює операнди зліва направо і повертає перший об'єкт, який має помилкове значення.

```
>>> 0 and 3 # поверне перший помилковий об'єкт-операнд
```

```
0
>>> 5 and 4 # поверне крайній правий об'єкт-операнд
4
```

Якщо Python не знаходить помилковий об'єкт-операнд, він повертає крайній правий операнд.

Логічний оператор *or* діє схожим чином, але для об'єктів-операндів Python повертає перший об'єкт, який має значення «істина». Python припинить подальші обчислення, як тільки буде знайдений перший об'єкт, який має значення «істина».

```
>>> 2 or 3 # повертає перший об'єкт-операнд зі значення «істина»
2
>>> None or 5 # повертає другий операнд, тому що перший - хибний
5
>>> None or 0 # повертає об'єкт-операнд, що залишився
0
```

Логічні вирази можна комбінувати:

```
>>> 1+3 > 7 # пріоритет + вище, ніж >
False
>>> (1+3) > 7 # дужки сприяють наочності і позбавляють від помилок
False
>>> 1+(3>7)
1
```

В Python можна перевіряти належність до інтервалу:

```
>>> x=0
>>> -5<x<10 # еквівалентно: x > -5 and x<10
True
```

Рядки в Python можна порівнювати аналогічно числам. Символи, як і все інше, подають в комп'ютері у вигляді чисел. Існує таблиця, яка ставить у відповідність кожному символу деяке число. Визначити, яке число відповідає символу можна за допомогою функції *ord*():

```
>>> ord ('L')
76
>>> ord ('Ф')
1060
```

Тепер порівняння символів зводиться до порівняння чисел, які їм відповідають:

```
>>> 'A' > 'L' 6
False
```

При порівнянні рядків Python їх порівнює посимвольно:

```
>>> 'Aa' > 'Ll'
False
>>>
Оператор in перевіряє наявність підрядка в рядку:
>>> 'a' in 'abc'
True
```

```
>>> 'A' in 'abc' # великої літери А немає
False
>>> "" in 'abc' # порожній рядок є в будь-якому рядку
True
>>> '' in ''
True
```

Інструкція if. Розглянемо умовну інструкцію *if*, яку використовують для вибору серед альтернативних операцій на основі результатів перевірки. Інструкція *if* обирає, яку дію необхідно виконати. Вона може містити інші інструкції, в тому числі інші умовні інструкції *if*.

Спочатку записується частина *if* з умовним виразом, далі можуть слідувати одна або більше необов'язкових частин *elif* («*else if*») з умовними виразами і, нарешті, необов'язкова частина *else*. Умовні вирази і частина *else* мають асоційовані з ними блоки вкладених інструкцій, з відступом щодо основної інструкції. Під час виконання умовної інструкції *if* інтерпретатор виконує блок інструкцій, асоційований з першим умовним виразом, тільки якщо він повертає значення «істина», в іншому випадку виконується блок інструкцій *else*. Загальна форма запису умовної інструкції *if* виглядає таким чином:

```
if <test1>: # Инструкция if с условным выражением test1
     <statements1> # Ассоциированный блок
elif <test2>: # Необязательные части elif
     <statements2>
else: # Необязательный блок else
     <statements3>
```

Розглянемо кілька прикладів. Всі частини цієї інструкції, за винятком основної частини *if* з умовним виразом і пов'язаних з нею інструкцій, є необов'язковими. У найпростішому випадку інші частини інструкції опущено:

```
>>> if 1:
...  print 'true'
...
true
```

Запрошення до введення змінюється на «...» для рядків продовження в базовому інтерфейсі командного рядка, що використовується тут (в IDLE текстовий курсор переміщається на наступний рядок вже з відступом, а натискання на клавішу *Backspace* повертає на рядок вгору). Введення порожнього рядка (подвійним натисканням клавіші Enter) завершує інструкцію і призводить до її виконання. Число «1» – це логічна істина, тому дана перевірка завжди буде успішною. Щоб обробити помилковий результат, додайте частину *else:*

```
>>> if not 1:
     print 'true'
... else:
     print 'false'
...
```

```
     false
```

Множинне розгалуження. Розглянемо приклад умовної інструкції *if*, в якій присутні всі необов'язкові частини:

```
>>> x = 'killer rabbit'
>>> if x == 'roger':
     print "how's jessica?"
... elif x == 'bugs':
     print "what's up doc?"
... else:
     print 'Run away! Run away!'
...
Run away! Run away!
```

Ця багаторядкова інструкція простягається від рядка *if* до кінця блоку *else*. При виконанні цієї інструкції інтерпретатор виконає вкладені інструкції після тієї перевірки, яка дасть в результаті істину, або блок *else*, якщо всі перевірки дадуть результат «хибність». Обидві частини *elif* і *else* можуть бути опущені, і в кожній частині може бути більше однієї вкладеної інструкції. Зв'язок слів *if, elif* і *else* визначений тим, що вони знаходяться на одній вертикальній лінії, з одним і тим же відступом.

У Python множинне розгалуження оформляють у вигляді послідовності перевірок *if/elif*. Використання інструкції *if* є найбільш простим способом організації множинного розгалуження.

3. ОБРОБКА ПОМИЛОК

Існують три основних типи помилок: помилки етапу компіляції (інтерпретації), етапу виконання та логічні помилки.

Помилки етапу компіляції (або семантичні) відбуваються, коли код порушує правила синтаксису мови. Компілятор не може скомпілювати програму, поки вона не буде містити припустимі оператори і мати правильну структуру. Загальною причиною помилок етапу компіляції є помилки набору (друкарські помилки), посилання на невизначені змінні, пропущені крапка з комою, передача функції неправильних параметрів тощо.

Помилки етапу виконання (або семантичні) відбуваються, коли після компіляції програми під час її виконання робиться щось неприпустиме (наприклад, програма намагається виконати ділення на нуль або відкрити для запису неіснуючий файл).

Логічні помилки – це помилки проектування та реалізації програми. Ці помилки важко відстежити, оскільки інтерпретатор не може знайти їх автоматично, як синтаксичні та семантичні помилки. Зазвичай засоби налагодження дозволяють їх знайти. Логічні помилки призводять до некоректного або непередбаченого значення змінних, неправильного вигляду графічних зображень або невиконання коду, коли це очікується.

Іноді помилки важко знайти, оскільки вони можуть бути результатом взаємодії різних частин програми. У цьому випадку краще крок за кроком переглянути програму та стан змінних і виразів. Таке виконання – ключовий елемент від лагодження.

Нехай необхідно виконати математичні дії над введеними користувачем числами, наприклад, обчислити квадрати чисел. Для досягнення бажаного ефекту ми могли б спробувати використовувати такі інструкції:

```
while True:
     reply = input('Enter text:')
     if reply == 'stop': break
     print(int(reply) ** 2)
print('Bye')
```

У цьому сценарії використовується однорядкова інструкція *if*, яка виконує вихід із циклу після отримання від користувача рядку «*stop»*, а крім того, виконується перетворення введеного рядка для виконання необхідної математичної операції. У даний сценарій також додано повідомлення, яке виводиться в момент завершення роботи сценарію. Оскільки інструкція *print* в останньому рядку не має такого ж відступу, як інструкції вкладеного блоку, вона не вважається частиною тіла циклу і буде виконуватися тільки один раз – після виходу з циклу:

```
Enter text:2
4
Enter text:40
1600
Enter text:stop
Bye
```

Обробка помилок шляхом перевірки вводу. Якщо користувач введе неправильний рядок, маємо:

```
Enter text:xxx
...текст сообщения об ошибке опущен...
ValueError: invalid literal for int() with base 10: 'xxx'
```

Вбудована функція *int* активує виключення, коли стикається з помилкою. Якщо необхідно забезпечити стійкість сценарію, треба попередньо перевірити вміст рядка за допомогою методу рядків *isdigit*:

```
>>> S = '123'; T = 'xxx'; S.isdigit(), T.isdigit()
(True, False)
```

Для цього в наш приклад необхідно додати вкладені оператори. У наступній версії інтерактивного сценарію використовується версія умовної інструкції *if*, яка запобігає можливість появи винятків:

```
while True:
     reply = input('Enter text:')
     if reply == 'stop':
          break
     elif not reply.isdigit( ):
          print('Bad!' * 8)
     else:
          print(int(reply) ** 2)
print 'Bye'
```

У повній формі інструкція містить слово *if*, за яким слідують вираз перевірки умови і вкладений блок коду, один або більше необов'язкових перевірок *elif* («*else if*») і відповідних їм вкладених блоків коду і необов'язкова частина *else* із зв'язаним з нею блоком коду, який виконується при недотриманні умови. Інтерпретатор виконує перший блок коду (якщо перевірка дає в результаті значення «істина»), проходячи інструкцію зверху дониз, або частину *else* (якщо всі перевірки дали в результаті значення «хибність»).

Частини *if, elif* і *else* в попередньому прикладі належать одній і тій самій інструкції, тому що вертикально вони розташовані на одній лінії (тобто мають однакові відступи). Інструкція *if* простягається до початку інструкції *print* в останньому рядку. У свою чергу, весь блок інструкції *if* є частиною циклу *while,* тому що всю її зсунуто вправо щодо основної інструкції циклу. Тепер новий сценарій буде виявляти помилки перш, ніж вони будуть виявлені інтерпретатором, і виводити повідомлення:

```
Enter text:5
25
Enter text:xyz
Bad!Bad!Bad!Bad!Bad!Bad!Bad!Bad!
Enter text:10
100
Enter text:stop
```

Обробка помилок за допомогою інструкції try. В Python існує більш універсальний спосіб обробки помилок за допомогою інструкції *try*. Використавши цю інструкцію, можна спростити попередній код:

```
while True:
     reply = input('Enter text:')
     if reply == 'stop': break
     try:
          num = int(reply)
     except:
          print('Bad!' * 8)
     else:
          print(int(reply) ** 2)
print 'Bye'
```

Ця версія працює так само, як і попередня, тільки тут ми замінили явну перевірку наявності помилки програмним кодом, який передбачає, що перетворення буде виконано і виконує обробку виключення, якщо таке перетворення неможливо. Ця інструкція *try* складається зі слова *try*, слідом за яким розміщено основний блок коду (дії, які ми намагаємося виконати), з наступною частиною *except*, де розташовується програмний код обробки винятку. Далі розміщено частину *else*, програмний код якої виконується, якщо в частині *try* виключення не виникло. Інтерпретатор спочатку виконує частину *try*, потім виконує або частина *except* (якщо виник виняток), або частина *else* (якщо виняток не виник).

Тому що слова *try, except* і *else* мають однаковий відступ, всі вони вважаються частиною однієї і тієї ж інструкції *try*. В даному випадку частина *else* пов'язана з інструкцією *try*, а не з інструкцією *if*. Ключове слово *else* в мові Python може з'являтися не тільки в інструкції *if*, але і в інструкції *try* і в циклах – величина відступу наочно показує, частиною якої інструкції воно (слово *else*) є. В цьому випадку інструкція *try* розпочинається зі слова *try* і триває до кінця вкладеного блоку коду, розташованого за словом *else*, тому що *else* розміщено на одній відстані від лівого краю, що і *try*. Інструкція *if* в цьому прикладі займає всього один рядок і завершується відразу ж за словом *break*.

Комп'ютерний практикум № 5. Алгоритми циклічної структури (інструкція while)

Мета роботи: ознайомитися з алгоритмами циклічної структури, їх реалізацією на мові Python; вивчити формат оператору циклу *while*. *Об'єкт дослідження* – алгоритми циклічної структури, оператор циклу *while*, тип даних «рядки».

ПЛАН

1. Рядки.
2. Цикли *while.*

Завдання

1. Вивчити теоретичні основи написання алгоритмів циклічної структури. Опрацювати приклади.
2. Побудувати блок-схему алгоритму вирішення завдання 1.
3. Відповідно до свого варіанту
- визначити умови;
- за допомогою формул описати варіанти виконання необхідний дій;
- написати програму, яка розв'язує завдання, реалізуючи обробку помилок (виключень);
- організувати введення даних з клавіатури, виведення у консоль рядків або списків.
4. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасировки виконання програми.

Вимоги: не можна використовувати власні функції; при розв'язанні задачі треба використовувати вкладені цикли.

Варіанти

Завдання 1. Скласти програму обчислення значення функції на відрізку $[x_0, x_k]$ з кроком h за допомогою циклу *while*.

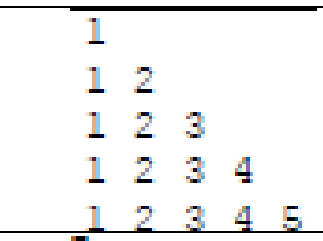
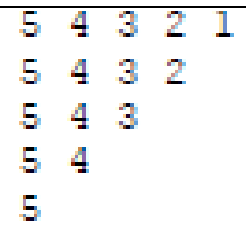
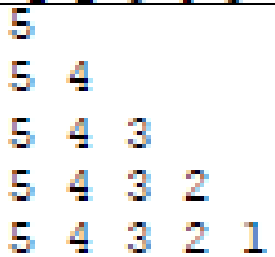
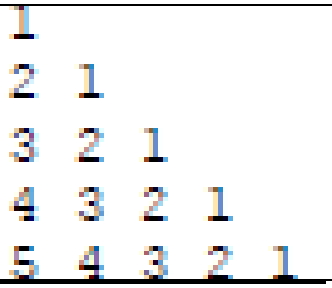
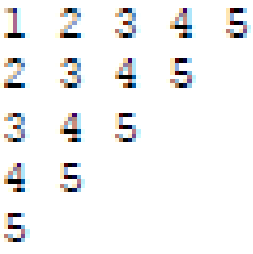
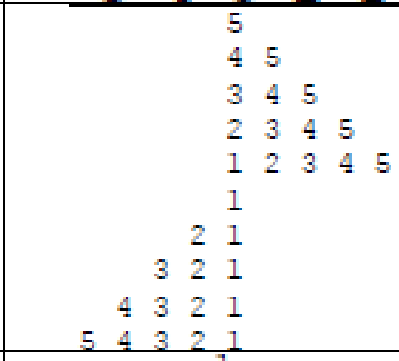
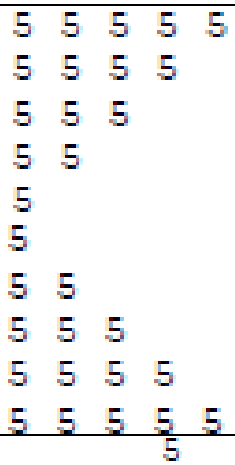
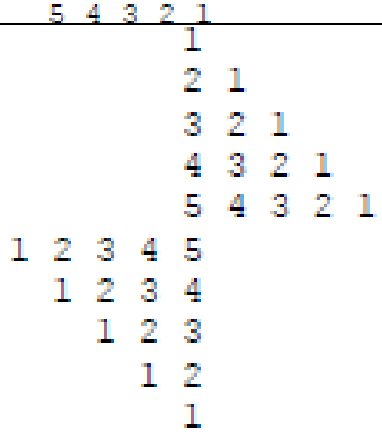
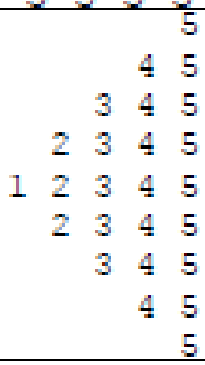
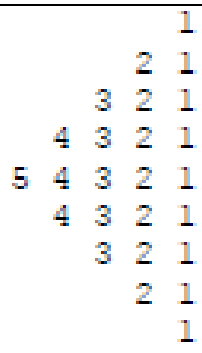
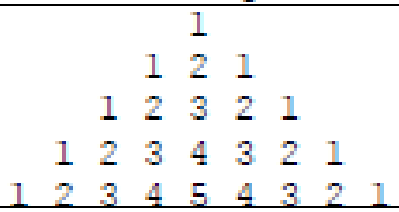
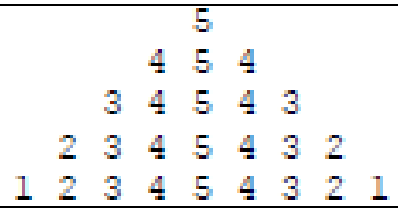
№	$f(x)$	h; [a; b]
1	2	3
1	$y=\ln(x)$	h=0.1; a=1; b=1.5
2	$y=1+\ln^2(x)$	h=0.1; a=0.4; b=1.0
3	$y=1+e^x$	h=0.01; a=0.5; b=0.6
4	$y=e^{x^2}/2$	h=0.2; a=2; b=3
5	$y=\cos(x)\cdot e^{-x}$	h=0.2; a=1; b=2
6	$y=1/(1+e^{-x})$	h=0.2; a=3; b=4
7	$y=\sin(x)\cdot\sinh(x)$	h=1; a=1; b=5
8	$y=0.5+\sinh^2(x)$	h=0.2; a=2; b=3
9	$y=\sqrt{x}\cdot\cosh(x)$	h=0.2; a=3; b=4

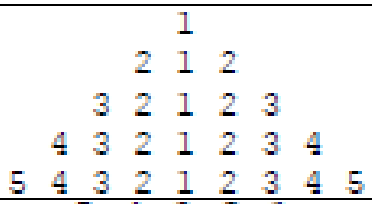
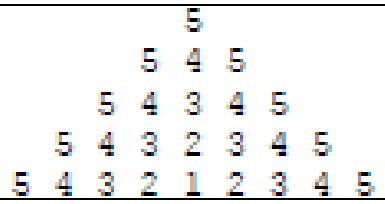
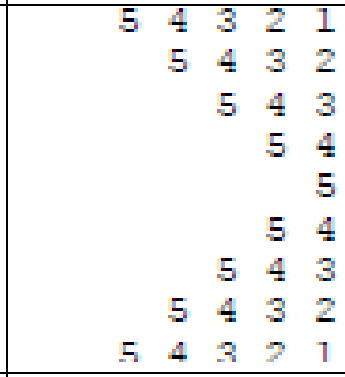
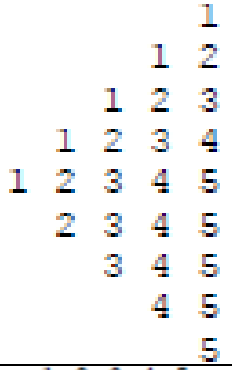
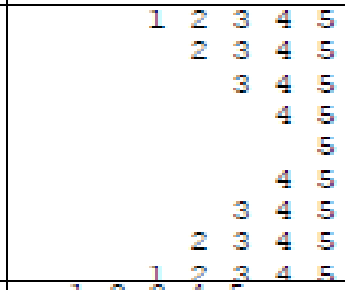
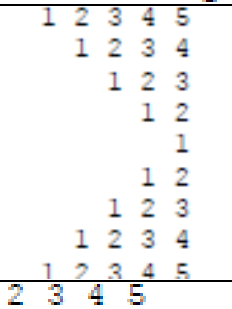
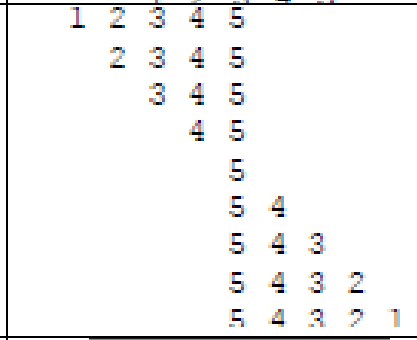
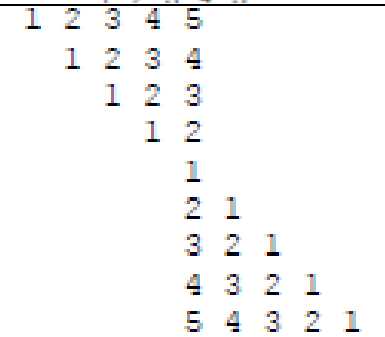
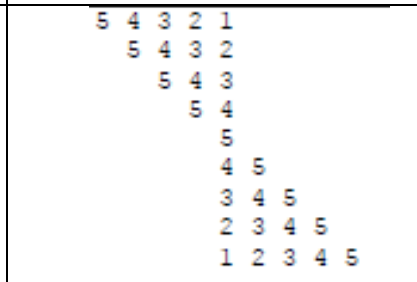
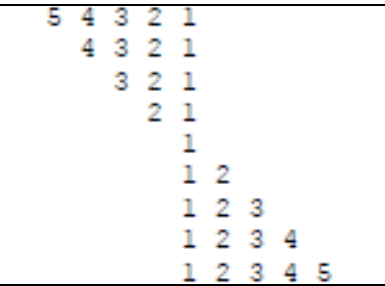
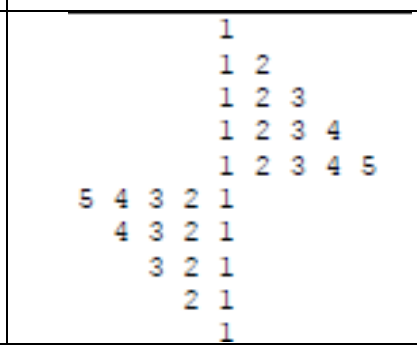
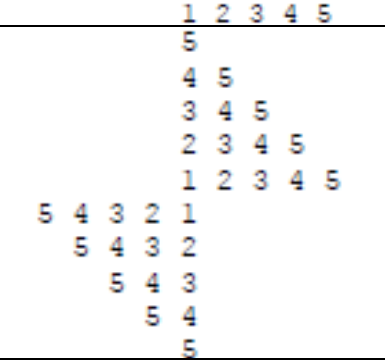
1	2	3
10	$y=1/(1+\cosh^2(x))$	h=0.5; a=2; b=4
11	$y=\sqrt{x}\cdot\sinh(x)$	h=1; a=1; b=5
12	$y=e^{-x}\cdot\cosh(x)$	h=1; a=1; b=4
13	$y=\ln(x^2)$	h=0.1; a=1; b=1.4
14	$y=x+\ln(x)$	h=1; a=1; b=5
15	$y=1/(1+\sin(x))$	h=π/10; a==π/6; b==π/3
16	$y=\sin(x)+\sqrt{x}$	h=π/10; a==π/6; b==π/4
17	$y=x\cdot(1-\cos(x))$	h=0.1; a=0.4; b=0.8
18	$y=e^{x+3}\ sin(x)$	h=0.5; a=0; b=2
19	$y=\cos(x)\cdot\cosh(x)$	h=1; a=1; b=5
20	$y=e^{x+1}\cdot\sinh(x)$	h=1; a=1; b=4
21	$y=10^{-2}(5+4x)-e^{x^3+4}$	h=0.1; a=-3.4; b=-1.4
22	$y=4x^3+2^{5/4}xe^{-x}$	h=1.01; a=2.4; b=10.4
23	$y=9(x^3+3.2)\cdot\mathrm{tg}(x)$	h=0.2; a=1; b=2.4
24	$y=1.2e^{x^2}+x$	h=-0.05; a=-0.75; b=-1.5
25	$y=x^{2^2}+\cos\left(2^{3/4}+x^{3/2}\right)$	h==π/3; a=14; b=19
26	$y=\left(x^{5/2}-0.8\right)\cdot\ln\left(x^2+12.7\right)$	h=0.3; a=0.25; b=5
27	$y=0.8\cdot10^{-5}(x^3+6.7)^{7/6}$	h=0.1; a=-0.5; b=0.4
28	$y=0.4+x^{2/3}\cos(x+e^x)$	h=π/10; a=5.6; b=15.4

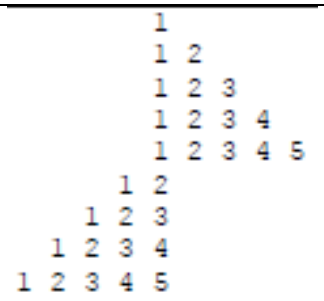
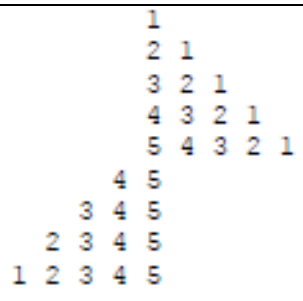
Завдання 2 (приклад 5.4). Вводиться ціле число N ($1\le N\le 9$), а виводяться *рядки з числами*, які утворюють визначений «рисунок» (останній задається варіантом). У варианті показано рисунок для N=5.

Таблиця 5.1

№ варіанту	*Рисунок*	*№ варіанту*	*Рисунок*
1	2	3	4
1	5 4 5 3 4 5 2 3 4 5 1 2 3 4 5	**2**	1 2 3 4 5 1 2 3 4 1 2 3 1 2 1

1	2	3	4
3		**4**	
5		**6**	
7		**8**	
9		**10**	
11		**12**	
13		**14**	

1	2	3	4
15		**16**	
17		**18**	
19		**20**	
21		**22**	
23		**24**	
25		**26**	

1	2	3	4
27		**28**	

Контрольні запитання

1. Що таке цикл, для чого його використовують?
2. Як описується та виконується циклічна інструкція while?
3. Як можна організувати нескінченні цикли? Наведіть декілька прикладів і поясніть їх. Як можна вийти з нескінченних циклів? Що відбувається при запуску нескінченного циклу?
4. Чи може оператор циклу не мати тіла? Чому?
5. Для чого служать оператори переривання *break* та *continue*?

Аудиторна робота

Приклад 5.1. Використання циклу *while*

```
1)
n = 1
while n<=10: # повторювати, поки n меньше або дорівнює десяти
    print('n =',n)  # виводимо поточне значення n
    n += 1  # і збільшуємо його на 1
2)x = 0
while x <= 0:  # повторювати, поки x не буде додатним
    x = int(input('Введіть додатне число: '))
print('Ви ввели число', x)

3)print('Усі натуральні числа:')
n = 1
while True:  # нескінчений цикл
    print(n); n += 1

4)name = None  # спочатку ми не знаємо імені користувача
# нескінчений цикл
while True:
    print('Меню:')
    print("1. Ввести ім'я")
    print('2. Вивести привітання')
    print('3. Вийти')
    response = input('Виберіть пункт: ')
    print()
    if response == '1':
        name = input("Введіть ваше ім'я: ")
    elif response=='2':
      if  name:  #вітаємося  з  користувачем,  якщо  ім'я  вже
```

```
введено
              print('Привіт, ', name, '!', sep='')
           else:
                 print('Я не знаю вашого імені.')
        elif response == '3':
            # оператор break завершує виконання циклу
            break #якщо користувач вибрав 3, то виходимо з циклу
        else:
            print('Неправильне введення.')
        print()

    5) while True:
        print('Введіть exit, щоб завершити цикл')
        response = input('> ')
        if response == 'exit': break

    6)x = 0
    while x < 10:
        x += 1
        if x == 5:  # пропускаємо число 5
            continue
        print(' Поточне число дорівнює ', x)
    7)x = 5
    while x:  # поки x не дорівнює нулю
        print(x); x -= 1
    else:
        print('Цикл виконано')
        print('Кінцеве значення x:', x)

    8)x = 3
    while x:
#цикл буде виконано 3 рази, якщо користувач не завершить його
раніше
        x-=1
        response=input('Введіь stop, щоб зупинити цикл (інакше що
завгодно): ')
        if response=='stop': break
    else:
        # ця гілка буде виконана, якщо цикл не був перерваний
        print('Цикл завершився сам')
    print('Кінець програми')
```

Приклад 5.2. Використання рядків.

```
    1)
    s1 = "Рядок 1";s2 = 'Рядок 2'
    print(s1, s2)
    # формування рядка з іншого значення
    s3 = str(8); print(s3)
    # рядки, які складаються з  багатьох рядків
    s4 = """ Lesson2. Variables and Data Types
    Some data types explained in this lesson:
     - int, - bool, - float, - complex, - str """
```

```
print(s4)
# символ \ використовують, щоб продовжити рядок
# або будь-який вираз в Python з наступного рядка кода
s5 = "started\
      continued"
print(s5)

2)
string = "a string" # Створення рядка
# Виведення окремих символів рядка
print(string[0])   # 'a'
print(string[2])   # 's'
print(string[-1])  # 'g'
```

Результат

```
a
s
g

3)
string = "a string" # Створення рядка
# Виведення зрізів (групи символів) рядка
print(string[2:5]) # str
print(string[:5])  # a str
print(string[2:])  # string
print(string[::2]) # asrn
# Отримання окремих елементів рядка та їх конкатенація
print(string[2] + string[-3:])  # sing
```

Результат

```
str
a str
string
asrn
sing

4)
string = input('Введіть рядок: ') # Введення рядка
# Перевірка, чи є в даному рядку символ «q»
if 'q' in string:
    print('В цьому рядку є символ "q"')
else:
    print('В цьому рядку немає символу "q"')
```

Результат

```
Введіть рядок: студент
В цьому рядку немає символа "q"

5)
string = input('Введіть рядок: ') # Введення рядка
# Виведення довжини рядка
print('Довжина цього рядка: ', len(string))
```

Приклад 5.3. Приклади операцій над рядками

```
str1 = 'hel'; str2 = 'lo'
```

```
result = str1 + str2  # конкатенація рядків
print(result)
# форматування рядків
a = 48;b = 73
message1 = '%d + %d = %d' %(a, b, a + b)
print(message1)
message2 = '{} - {} = {}'.format(a, b, a - b)
print(message2)
# індексація рядків
s = 'Hello, World!'
print(s[0])  # індексація розпочинається з нуля
print(s[4])  # четвертий (п'ятий реально) елемент (символ)
print(s[-1]) # від'ємні числа – індексація розпочинається з кінця
print(s[2:7])
# - символи з 2 (включно) по 7 (не включно)
print(s[2:7:2])  # теж саме, але з кроком два
```

Приклад 5.4. Визначити середнє арифметичне заданої непустої послідовності додатних цілих чисел, за якою слідує «0» (це ознака кінця послідовності).

```
1) from math import *
a=input ('Input first number: ')
if not a.isdigit():
 print("String value can not be entered")
 print("or number is negative, restart the program!")
 input ("reload the program!")
else:
 a=int(a)
if a==0:
 print("The number can not be zero"); input ()
2) count=0;ar=0
while True:
    ar+=a; count+=1
    try:
        a=int(input('Input next number or Enter 0 to finish: '))
    except:
        print("String value can not be entered")
        print("or number is negative, restart the program!")
        input ("reload the program!")
    else:
        if a==0: break
ar=ar/count; print("Average: ",ar)
```

Результат

```
Input first number: 1
Input next number or Enter 0 to finish: 2
Input next number or Enter 0 to finish: 0
Average:  1.5
```

Приклад 5.5. Вводиться ціле число N ($1 \le N \le 9$), а виводяться *рядки* з

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

числами, які утворюють визначений «рисунок»

```
try:
    N=int(input('Введите N = '))
except Exception:
    print('Введите число!!!')
else:
 M=N; pp=''
 while M!=0:
  i=M;  L=[]
  while i!=0:
   if i<=M:
      L.append(str(i)); i-=1
  a=list(L)
  pp=''.join(a)
  print(pp); M=M-1
```

Результат

```
Введите N = 9
987654321
87654321
7654321
654321
54321
4321
321
21
1
```

Приклад 5.6. Оператор *print*

```
print(1, end=' ')
print(2, end='\n\n')    # два переведення рядка
print('he', end='')     # пустий кінець
print('llo')
```

Приклад 5.7. Ввести числа *a*, *b*, *h*. Визначити на інтервалі [*a*, *b*] з кроком *h* (h приймає цілі значення) множину значень $y_1, \ldots, y_k$: $y = \sqrt{x}$.

```
import math
a=int(input('a=')); b=int(input('b='))
h=int(input('h=')) # h может быть только int
x=a
while x!=b:
 y=math.sqrt(x); print(x,y);x+=h
```

Теоретична частина

І. РЯДКИ

Рядки – це впорядковані послідовності символів, що використовують для зберігання і подання текстової інформації (символів і слів, наприклад, ваше ім'я, змісту текстових файлів, завантажених в пам'ять, адрес в Інтернеті, програми на Python тощо). Рядки володіють потужним набором засобів для їх обробки. У Python відсутній спеціальний тип для подання одного символу, тому в разі необхідності використовуються односимвольні рядки.

Рядки відносять до категорії незмінних послідовностей, в тому сенсі, що символи, які вони містять, мають певний порядок розміщення зліва направо і самі рядки неможливо змінити. Рядки – це представник великого класу об'єктів, які називають послідовностями. Зверніть увагу на операції над послідовностями, наведені тут, тому що вони схожим чином працюють і з іншими типами послідовностей, такими як списки і кортежі, які ми будемо розглядати пізніше. У табл. 5.1 наведені найбільш типові літерали рядків і операцій

Таблиця 5.1

Типові літерали рядків та операції над ними

Операція	Інтерпретація
S = ‘’	Пустий рядок
S = “spam’s”	Рядок у лапках
S = ‘s\np\ta\x00m’	Екрановані послідовності
block = “””...”””	Блоки в потрійних лапках
S = r’\temp\spam’	Неформатовані рядки
S = b’spam’	Рядки байтів
S1 + S2 S * 3	Конкатенація, повторення
S[i] S[i:j] len(S)	Звернення до символу за індексом Витяг підрядку (зрізу) Довжина
“a %s parrot” % kind	Вираз форматування рядка
“a {0} parrot”.format(kind)	Метод форматування рядка
S.find(‘pa’)	Виклик методу рядків: пошук
S.rstrip()	Видалення провідних й кінцевих символів пробілу
S.replace(‘pa’, ‘xx’)	Заміна
S.split(‘,’)	Разбиття за символом, який є роздільником
S.isdigit()	Перевірка вмісту
S.lower()	Перетворення регістра символів
S.endswith(‘spam’)	Перевірка закінчення рядка
‘spam’.join(strlist)	Формування рядка зі списку
S.encode(‘latin-1’)	Кодування рядків Юнікоду
for x in S: print(x) ‘spam’ in S [c * 2 for c in S] map(ord, S)	Обхід в циклі Перевірка на входження

Порожній рядок має вигляд пари лапок (або апострофів), між якими нічого немає. Для роботи з рядками підтримуються операції над виразами, такі як конкатенація (об'єднання рядків), виділення підрядка, вибірка символів за індексами (за зсувом від початку рядка) тощо. Python пропонує ряд методів, які реалізують різні завдання роботи з рядками.

II. ЦИКЛ WHILE

1. Інструкція while. Алгоритм, в якому передбачено неодноразове виконання певної послідовності дій, називається алгоритмом циклічної структури або циклом. Цикл дозволяє істотно скоротити розмір запису алгоритму, зобразити його компактно шляхом відповідної організації пропонуємих дій. Повторювати певні дії має сенс при різних значеннях параметрів, які змінюються. Такі параметри називаються *параметрами циклу*. Блок повторюваних операторів називають *тілом циклу* (це послідовність дій, які виконується багаторазово).

Циклічний процес називається ітераційним, якщо заздалегідь невідома кількість повторень циклу, а кінець обчислення визначається при досягненні деякою величиною заздалегідь заданої точності обчислення.

При програмуванні ітераційних процесів прийнято їх розділяти на цикли з «передумовою» і з «післяумовою». Їх відмінність полягає в тому, що перевірка досягнення деякою величиною заданої точності обчислення здійснюється або на початку циклу, або наприкінці циклу відповідно. Особливість циклу з «післяумовою» полягає в тому, що повторювана ділянка алгоритму виконається хоча б один раз, у той час як в циклі з «передумовою» ця ділянка може не виконатися жодного разу. Процес ініціалізації включає в себе визначення (введення) початкових значень змінних, які використовуються в тілі циклу.

Для запису ітераційних процесів в Python використовують лише один тип операторів циклу *while* – оператор з попередньою умовою (передумовою). Для всіх операторів циклу характерні такі особливості:

1. Повторювані обчислення записуються лише один раз.
2. Вхід в цикл можливий тільки через його початок.
3. Змінні оператора циклу повинні бути визначені до входу в цикл.
4. Потрібно передбачити вихід з циклу. Якщо цього не зробити, то обчислення будуть тривати нескінченно довго.

Нескінченний цикл – це циклічна ділянка в програмі, в якій не передбачені засоби виходу з циклу при досягненні деякого умови.

Інструкція *while* організує цикл з передумовою (перевірка виконується перед початком чергової ітерації), складається з рядка заголовка з умовним виразом, тіла циклу, що містить одну або більше інструкцій з відступами, і необов'язкової частини *else*, яка виконується, коли управління передається за межі циклу без використання інструкції *break*. Інтерпретатор продовжує обчислювати умовний вираз в рядку заголовка і виконувати вкладені інструкції в тілі циклу, поки умовний вираз не поверне значення «хибність»:

```
while <test>: # Умовний вираз test
     <statements1> # Тіло цикла
else: # Необов'язкова  частина else
     <statements2> # Виконується, якщо вихід із цикла
# виконується не інструкцією break
```

Оператор *while* дозволяє багаторазово виконувати певні дії в залежності від деякої <умови> (виразу логічного типу, який приймає тільки значення True або False). Цикл виконується поки <Умова>=«істина». Як тільки <Умова> порушується, виконання циклу завершується. Блок-схема циклу *while* наведена на рис. 5.1.

Інструкція *while* продовжує виконувати блок інструкцій (зазвичай з відступами), поки умовний вираз продовжує повертати значення «істина». Вона називається «циклом», тому що управління циклічно повертається до початку інструкції, поки умовний вираз не поверне значення «хибність». Як тільки в результаті перевірки буде отримано значення «хибність», управління буде передано першій інструкції, яка розташована відразу ж за вкладеним блоком тіла циклу *while*. Розглянемо приклади простих циклів *while.*

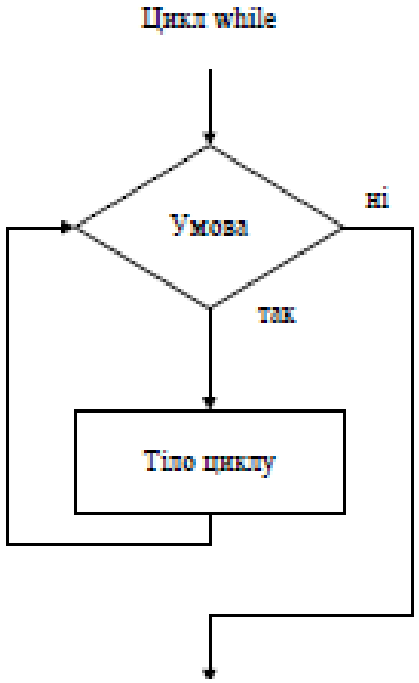


Рис.5.1. Функціональна схема циклу «Поки» (з передумовою)

Перший приклад (який містить інструкцію *print*, вкладену в цикл *while*), нескінченно виводить повідомлення (True – це особлива версія цілого числа «1», вона позначає значення «істина», тому результатом цього умовного виразу завжди буде «істина», і інтерпретатор нескінченно буде виконувати тіло циклу, поки ви не скасуєте його виконання). Такі цикли зазвичай називають нескінченними:

```
>>> while True:
          print('Type Ctrl-C to stop me!')
```

Наступний фрагмент продовжує видаляти з рядка перший символ, поки

він не стане порожнім, в результаті умова прийме значення «хибність». Перевірка об'єктів на значення «істина» здійснюється безпосередньо замість використання еквіваленту (*while* x != ‘’:).

```
>>> x = 'spam'
>>> while x: # Поки x – це не пустий рядок
...       print(x, end=' ')
...       x = x[1:] # Видалити перший символ з x
...
spam pam am m
```

Аргумент *end*= ‘’ забезпечує виведення значень в один рядок через пробіл. Наступний фрагмент перебирає значення від *a* до *b*, не включаючи значення *b*:

```
>>> a=0; b=10
>>> while a < b:
# Один зі способів організації циклів перебору
...       print(a, end=' ')
...       a += 1 # або a = a + 1
...
0 1 2 3 4 5 6 7 8 9
```

В Python відсутній цикл «*do until*», однак його можна імітувати, додавши в кінець тіла циклу умовну інструкцію та інструкцію *break*:

```
while True:
... <тіло цикла>...
     if exitTest(): break
```

2. Break, continue, pass і else. Розглянемо дві прості інструкції, які можна використати тільки усередині циклів – інструкції *break* і *continue*. У Python:

Break виконує перехід за межі циклу (всієї інструкції циклу).

Continue виконує перехід на початок циклу (в рядок заголовка).

Pass нічого не робить: це пуста інструкція.

Блок *else* виконується, тільки якщо цикл завершився звичайним чином (без використання інструкції *break*).

З урахуванням інструкцій *break* і *continue* цикл *while* має такий загальний вигляд:

```
while <test1>:
 <statements1> # тіло циклу
 if <test2>: break # Вийти з циклу, пропустивши частину else
 if <test3>:continue #Перейти на початок циклу, до вираз.
test1
else:
 <statements2>  # Виконується, якщо не була використана
                # інструкція 'break'
```

Інструкції *break* і *continue* можуть з'являтися в будь-якому місці всередині тіла циклу *while* (або for), але як правило, їх використовують в умовних інструкціях *if*, щоб виконати необхідну дію у відповідь на деяку умову.

Pass. Інструкція *pass* не виконує ніяких дій, її використовують у

випадках, коли синтаксис мови вимагає наявності інструкції, але ніяких корисних дій в цій точці програми виконати не можна. Вона часто використовується в якості порожнього тіла складної інструкції. Наприклад, створити нескінченний цикл, який нічого не робить, можна таким чином:

```
while 1: pass # Натисніть Ctrl-C, щоб припинити  цикл!
```

Цей приклад нескінченно робить «ніщо». Ця інструкція може використовуватися, наприклад, для того, щоб ігнорувати виключення в інструкції *try*. Іноді інструкцію *pass* використовують як заповнювач, замість того, «що буде написано пізніше», і в якості тимчасового фіктивного тіла функцій:

```
def func1():
     pass # Реалізація функції буде додана пізніше
def func2(): pass
```

Порожнє тіло функції викличе синтаксичну помилку, тому в подібних ситуаціях можна використати інструкцію *pass*. У Python 3.0 замість будь-якого виразу допускається використовувати три крапки «...», які самі по собі не виконують жодної дії, тому їх можна використовувати як альтернативу інструкції *pass*, зокрема замість програмного коду, який буде написано пізніше (примітка: To Be Done – слід реалізувати):

```
def func1():
     ... # Альтернатива інструкції pass
def func2():
     ...
func1() # Під час виклику не виконає жодних дій
```

Три крапки може бути присутнім в одному рядку із заголовком інструкції і використовуватися для ініціалізації змінних, коли не потрібно вказувати значення якогось певного типу:

```
def func1(): ... #Може бути присутнім в тому ж рядку
def func2(): ...
>>> X = ... # Альтернатива об'єкту  None
>>> X
Ellipsis
```

Continue. Інструкція *continue* виконує перехід на початок циклу. Вона іноді дозволяє уникнути використання вкладених інструкцій. У наступному прикладі цю інструкцію використовують для пропуску непарних чисел (цей фрагмент виводить парні числа менше «10» і більше або рівні «0»). Число «0» означає «хибність», а оператор % обчислює залишок від ділення, тому даний цикл виводить числа в зворотному порядку, пропускаючи значення, кратні 2 (він виводить 8 6 4 2 0):

```
x = 10
while x:
     x = x-1 # x -= 1
     if x%2!=0: continue # непарне, пропустить вивеедення
     print(x, end=’ ‘)
```

Останній приклад виглядав би зрозуміліше, якби інструкція *print* була складовою інструкції *if*:

```
x = 10
while x:
     x = x-1
     if x % 2 == 0: # парне? - вывести
          print(x, end=' ')
```

Break. Інструкція *break* виконує негайний вихід з циклу. Програмний код, розташований в циклі за цією інструкцією не виконується, якщо ця інструкція запущена. Нижче наведено інтерактивний цикл, який виконує введення даних за допомогою функції *input* і вихід з циклу, якщо у відповідь на запит імені буде введена рядок «*stop*»:

```
>>> while 1:
...  name = input('Enter name:')
...  if name == 'stop': break
...  age = input('Enter age: ')
... print('Hello', name, '=>', int(age) ** 2)
...
Enter name:mel
Enter age: 40
Hello mel => 1600
Enter name:bob
Enter age: 30
Hello bob => 900
Enter name:stop
```

Цей приклад виконує перетворення рядка *age* в ціле число за допомогою функції *int*, перед тим як звести його до другого степеня (це необхідно, тому що функція *input* повертає результат введення користувача у вигляді рядка).

Else. При об'єднанні з частиною *else* інструкція *break* дозволяє позбутися від необхідності зберігати прапор стану пошуку. Наступний фрагмент визначає, чи є додатне ціле число простим числом, виконуючи пошук дільників більше за значення «1»:

```
x = y // 2 # Для значень y>1
while x > 1:
     if y % x == 0: # залишок
          print(y, 'has factor', x)
          break # Переступити  блок else
     x -= 1
else: # Нормальне завершення цикла
     print(y, 'is prime')
```

Замість того щоб встановлювати прапор, який буде перевірений після закінчення циклу, досить вставити інструкцію *break* в місці, де буде знайдений дільник. При такій реалізації управління буде передано блоку *else*, тільки якщо інструкція *break* не була виконана, тобто коли з упевненістю можна сказати, що число є простим. Блок *else* циклу виконується також у разі, коли тіло циклу

жодного разу не виконувалося, оскільки в цій ситуації інструкція *break* також не виконується. У циклах *while* це відбувається, коли перша ж перевірка умови в заголовку дає значення «хибність». Внаслідок цього в попередньому прикладі буде отримано повідомлення «is prime» (просте число), якщо спочатку *x* менше або дорівнює «1» (тобто, коли y=2). Блок else в циклах дозволяє обробити «інший» спосіб виходу з циклу, без необхідності встановлювати і перевіряти прапори або умови.

Цей приклад визначає прості числа, але недостатньо точно. Числа, менші «2», не вважають простими у відповідності із суворим математичним визначенням. Якщо бути більш точним, цей код також буде терпіти невдачі при від'ємних значеннях і виконуватися успішно при використанні дійсних чисел без дробової частини. В Python замість оператора ділення / використовують оператор //, тому що тепер оператор / виконує операцію «істинного ділення» (початкове ділення необхідно, щоб відсікти залишок!).

Нехай ми створюємо цикл пошуку деякого значення в списку і після виходу з циклу необхідно дізнатися, чи було знайдено це значення. Це завдання можна вирішити таким чином:

```
found = False
# прапор, щоб визначити, чи закінчився пошук успіхом
while x and not found:
     if match(x[0]): # шукане значення є першим?
          print('Ni'); found = True
     else:
          x = x[1:] # Видалити перше значення й повторити
if not found:
     print('not found')
```

Нижче наводиться еквівалентний фрагмент, де використано блок *else* в циклі:

```
while x: # Вийти, коли x спорожніє
     if match(x[0]):
          print('Ni'); break # Вихід, в обхід блоку else
          x = x[1:]
else:
  print('Not found') #цей блок працює, якщо рядок x вичерпано
```

Комп'ютерний практикум № 6. Алгоритми циклічної структури (інструкція for)

Мета роботи: ознайомитися з алгоритмами циклічної структури, їх реалізацією на мові Python; вивчити формат оператору циклу *for*. *Об'єкт дослідження* – алгоритми циклічної структури на основі оператору циклу *for*, модуль *itertools* (функція *itertools.count*), функція *range*, ітераційні процеси.

ПЛАН

1. Цикл *for*.
2. Побудова блок-схеми обчислення ітераційних процесів

Завдання

1. Вивчити теоретичні основи написання алгоритмів циклічної структури. Опрацювати приклади.
2. Побудувати блок-схему першого підзавдання.
3. Відповідно до свого варіанту
   - визначити умови;
   - за допомогою формул описати варіанти виконання необхідний дій;
   - написати програму, яка розв'язує завдання «Обчислення суми» з використанням циклу *for* та функції *range*.
   - організувати введення даних з клавіатури, виведення у консоль.
4. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасировки виконання програми.

Вимоги: не можна використовувати власні функції; при розв'язанні задачі можна використовувати вкладені цикли, функцію *math.faktorial*(*i*).

Варіанти

Обчислити значення суми, при необхідності використовуючи функції модуля *math* (наприклад, *math.faktorial*(*i*)).

Завдання 1. **1-14.** Задано натуральне n та формула

1	$\frac{1!}{1}+\frac{2!}{(1+2)}+\ldots+\frac{n!}{(1+2+\ldots+n)}$	2	$\frac{1}{1\cdot 2}-\frac{1}{2\cdot 3}+\ldots+\frac{(-1)^{n+1}}{n(n+1)}$
3	$\left(1+\frac{1}{1^2}\right)+\left(1+\frac{1}{2^2}\right)+\ldots+\left(1+\frac{1}{n^2}\right)$	4	$\frac{1}{3^2}+\frac{1}{5^2}+\ldots+\frac{1}{(2n+1)^2};$
5	$\frac{1}{1^5}-\frac{1+2}{2^5}+\ldots+\frac{(-1)^{n+1}(1+2+\ldots+n)}{n^5}$	6	$\frac{1!}{1}+\frac{2!}{1+\frac{1}{2}}+\ldots+\frac{n!}{1+\frac{1}{2}+\ldots+\frac{1}{n}};$
7	$-\frac{1}{3}+\frac{1}{5}+\ldots+\frac{(-1)^n}{2n+1};$	8	$\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+\ldots+\frac{1}{\sin 1+\ldots+\sin n}$

9	$\underbrace{\sqrt{2+\sqrt{2+....\sqrt{2}}}}_{\text{n корней}};$	10	$\left(1-\frac{1}{1^3}\right)+\left(1-\frac{1}{2^3}\right)+...\left(1-\frac{1}{n^3}\right)$
11	$\frac{\sin 1}{\sqrt{1}}+\frac{\sin 1+\sin 2}{\sqrt{1}+\sqrt{2}}+...+\frac{\sin 1+...+\sin n}{\sqrt{1}+\sqrt{2}+...+\sqrt{n}}$	12	$\frac{1!}{\cos 1}+\frac{2!}{\cos 1+\cos 2}+...+\frac{n!}{\cos 1+...+\cos n}$
13	$\frac{1}{1}-\frac{1+2}{1\cdot 2}+...+\frac{(-1)^{n+1}(1+2+...+n)}{n!}$	14	$\frac{1}{\cos 1}+\frac{1+2}{\cos 1+\cos 2}+...+\frac{1+...+n}{\cos 1+...+\cos n}$

15-28. Задано натуральне n, дійсне $x\in\Re$ та формула

15	$1+\frac{x}{1!}+\frac{x^2}{2!}+...+\frac{x^n}{n!}$
16	$\left(\frac{1}{1!}+\sqrt{\lvert x\rvert}\right)+\left(\frac{1}{2!}+\sqrt{\lvert x\rvert}\right)+...+\left(\frac{1}{n!}+\sqrt{\lvert x\rvert}\right)$
17	$\left(1-\frac{\sin(x)}{1!}\right)+\left(1-\frac{\sin(2x)}{2!}\right)+...+\left(1-\frac{\sin(nx)}{n!}\right)$
18	$x-\frac{x^3}{3!}+\frac{x^5}{5!}-...+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$
19	$1-\frac{x^2}{2!}+\frac{x^4}{4!}-...+(-1)^m\frac{x^{2m}}{(2m)}$
20	$\frac{1}{\cos(x)}+\frac{1+2}{\cos(x)+\cos(2x)}+...+\frac{1+2+...+n}{\cos(x)+\cos(2x)+...+\cos(nx)}$
21	$1+\frac{x^2}{2!}+\frac{x^4}{4!}+...+\frac{x^{2n}}{(2n)!}$
22	$x-\frac{x^2}{2}+\frac{x^3}{3}-...+(-1)^{n-1}\frac{x^n}{n}$
23	$x-\frac{x^3}{3}+\frac{x^5}{5}-...+(-1)^{n-1}\frac{x^{2n-1}}{2n-1}$
24	$-x-\frac{x^2}{2}-\frac{x^3}{3}-...-\frac{x^n}{n}$
25	$\left(1+\frac{\sin(x)}{1^2}\right)+\left(1+\frac{\sin(2x)}{2^2}\right)+...+\left(1+\frac{\sin(nx)}{n^2}\right)$
26	$\frac{1}{\sin(x)}+\frac{2}{\sin(x)+\sin(2x)}+...+\frac{n}{\sin(x)+\sin(2x)+...+\sin(nx)}$
27	$x+\frac{x^3}{3!}+\frac{x^5}{5!}+...+\frac{x^{2n-1}}{(2n-1)!}$
28	$\frac{x}{\cos(x)}+\frac{x^{(1+2)}}{\cos(x)+\cos(2x)}+...+\frac{x^{(1+2+...+n)}}{\cos(x)+\cos(2x)+...+\cos(nx)}$

Завдання 2. Обчислити значення суми.

1-16. Задано дійсне число *x*, ціле значення *n* та формула

№	Формула	*x*, *n*	№	Формула	*x*, *n*
1	2	3	1	2	3
1	$\sum_{k=0}^{n}\frac{(-1)^k x^{2k+1}}{k!(2k+1)}$	n=10, $x\in(0.1;1)$	2	$\sum_{k=1}^{n}\frac{1}{x^2 k^3}$	
3	$\sum_{k=0}^{n}\frac{(-1)^k x^{4k+3}}{(2k+1)!(4k+3)}$		4	$\sum_{k=0}^{n}\frac{(-1)^k}{k!(k+1)!}\left(\frac{x}{5}\right)^{6k}$	
5	$\sum_{k=0}^{n}\frac{(-1)^k}{((k+1)!)^2}\left(\frac{x}{2}\right)^{2(k+1)}$		6	$\sum_{k=1}^{n}\frac{(-1)^{k+1}}{(2k+1)!}\left(\frac{x}{3}\right)^{4k+2}$	
7	$\sum_{k=0}^{n}\frac{(-1)^k}{k!(k+1)!}\left(\frac{x}{2}\right)^{2k+1}$		8	$\sum_{k=0}^{n}\frac{(-1)^k x^{4k+1}}{(2k)!(4k+1)}$	
9	$\sum_{k=0}^{n}\frac{(-1)^{k+1} x^{2k-1}}{(2k-1)(2k+1)!}$		10	$\sum_{k=0}^{n}\frac{(-1)^{k+1}}{2k!}\left(\frac{x}{3}\right)^{4k}$	
11	$\sum_{k=0}^{n}\frac{x^{2k}}{2^k k!}$		12	$\sum_{k=0}^{n}\frac{(-x)^{2k}}{2k!}$	
13	$\sum_{k=1}^{n}\frac{x^{2k}}{x^2+k^3}$		14	$\sum_{k=0}^{n}\frac{(-1)^{k+1} x^{k+2}}{(k+1)(k+2)!}$	
15	$\sum_{k=0}^{n}\frac{(-1)^k (k+1) x^k}{3^k}$		16	$\sum_{k=0}^{n}\frac{(-1)^k x^k}{(k+1)^2}$	

17-28. Нехай задано дійсні числа *x*, $\varepsilon=10^{-6}$ ($x \neq 0$, $\varepsilon>0$). Обчислити наближене значення нескінченної суми, обчислення виконати із заданою точністю ε (поки поточний член ряда не перевищує за абсолютною величиною заданого ε).

№	Формула	*x*, ε
1	2	3
17	$1+\frac{x}{1!}+\ldots+\frac{x^n}{n!}+\ldots;$ $term_0=1;\ term_1=term_0\left(\frac{x}{1}\right);\ term_n=term_{n-1}\left(\frac{x}{n}\right)$	$\varepsilon=10^{-6}$, $x\in(0.1;1)$
18	$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\ldots+(-1)^{n-1}\frac{x^{2n}}{(2n)!}+\ldots,\ term_0=1;$ $term_n=term_{n-1}\left(-\frac{x\cdot x}{(2\cdot n-1)\cdot 2\cdot n}\right)$	
19	$x+\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots+\frac{x^{2n-1}}{(2n-1)!}+\ldots,\ term_0=x;\ term_n=term_{n-1}\left(\frac{x\cdot x}{2\cdot n\cdot(2\cdot n-1)}\right)$	

1	2	3
20	$1+\frac{x^2}{2!}+\frac{x^4}{4!}+...+\frac{x^{2n}}{(2n)!}+...,$ $term_0=1;\ term_n= term_{n-1}\cdot\left(\frac{x\cdot x}{(2\cdot n-1)\cdot 2\cdot n}\right)$	$\varepsilon=10^{-6}$, $x\in(0.1;1)$
21	$x-\frac{x^2}{2}+\frac{x^3}{3}+...+(-1)^{n-1}\frac{x^n}{n}+...,$ $term_1=x;\ term_n=term_{n-1}\cdot\left(-\frac{x(n-1)}{n}\right)$	
22	$x-\frac{x^3}{3}+\frac{x^5}{5}-...+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)}+...,$ $term_1= x;\ term_n = term_{n-1}\left(-\frac{x*x*(2*n-3)}{2*n-1}\right.$	
23	$-x-\frac{x^2}{2}-\frac{x^3}{3}-...-\frac{x^n}{n}-...,$ $term_1= -x;\ term_n = -term_{n-1}\cdot\left(-\frac{x\cdot(n-1)}{n}\right.$	
24	$1-x+x^2-...+(-1)^n x^n+...,\ term_0=1;$ $term_n= term_{n-1}\cdot(-x)$	
25	$1+x+x^2+...+x^n+...,\ term_0=1;$ $term_n= term_{n-1}\cdot(x)$	
26	$1+\frac{\alpha x}{1!}+\frac{\alpha(\alpha-1)x^2}{2!}...+\frac{\alpha(\alpha-1)...(\alpha-n+1)x^n}{n!}+...;$ $term_0=1;\ term_n= term_{n-1}(\alpha-n+1)\cdot\frac{x}{n}$	
27	$2\cdot\left(x+\frac{x^3}{3}+\frac{x^5}{5}+...+\frac{x^{2n-1}}{(2n-1)}+...\right),$ $term_1=2\cdot x;\ term_n =(½)\cdot term_{n-1}\left(2\cdot\frac{x\cdot x\cdot(2\cdot n-3)}{2\cdot(n-1)}\right.$	
28	$1-\frac{1}{2}x+\frac{3}{8}x^2-\frac{5}{16}x^3+...+(-1)^{n-1}\frac{(2n-1)!!x^n}{(2n)!!}+...,$ $term_0=1;\ term_n= term_{n-1}\left(\frac{(2\cdot n-1)\cdot x}{2\cdot n}\right.$	

Контрольні запитання

1. Що таке цикл з параметром, формат його використання?
2. Що таке вкладені цикли?
3. Як працює інструкція *for*? Для організації яких циклів її застосовують ?

Аудиторна робота

Приклад 6.1. Функція *range*

```
1. Параметр  i приймає значення в діапазоні [0, 10)
     for i in range(10):
         print('i =', i)

2. Параметр i приймає значення в діапазоні [5, 10)
     for i in range(5, 10):
         print('i =', i)

3. Параметр i приймає значення в діапазоні [5, 10) з кроком 2
     for i in range(5, 10, 2):  print('i =', i)

4. Цикл буде повторюватися 3 рази, якщо користувач не завершить
його  раніше
     for i in range(3):
         response=input('Введіть stop, щоб зупинити цикл (інакше
що завгодно): ')
         if response=='stop':  break
     else:
     # цю гілку виконують тільки якщо цикл не був перерваний
         print('Цикл сам був завершений')
     print('Кінець програми')

5)for x in range(1, 11):
         if x == 5:  # пропускаємо число 5
             continue
         print('Поточне число дорівнює ', x)

6) for i in range(10):
          for j in range(30):
             print('*', end='')
          print()

7. Функція reversed дозволяє обходити послідовність в зворотному
напрямку
     for i in reversed(range(5)):
         print(i)
```

Результат

```
4
3
2
1
0
```

Приклад 6.2. Задано натуральне *n*. Обчислити значення суми $1+\frac{2}{1!}+\frac{2^2}{2!}+...+\frac{2^n}{n!}$.

```
1)
import math
n=int(input('n= '))
```

```
s=1
for i in range(1,n+1):
 term=2**i/math.faktorial(i);  s+=term
print(s)

2)n=int(input('n= '))
s=1
for i in range(1,n+1):
 p=1
 for j in range(1,i+1):
  p=p*j; term=2**i/p; s+=term
print(s)
```

Приклад 6.3. Ввести числа *a, b, h*. Визначити множину значень $y_1, \ldots, y_k$ на інтервалі [*a, b*] з кроком *h* (*h* приймає лише цілі значення): $y = \sqrt{x}$.

```
import math
a=int(input('a='))
b=int(input('b='))
h=int(input('h=')) # h может быть только int
for i in range(a,b+1,h):
 x=i;  y=math.sqrt(i); print(x,y)
```

Приклад 6.4. Нехай задано дійсні числа *x*, $\varepsilon=10^{-6}$ ($x \neq 0$, $\varepsilon > 0$). Обчислити наближене значення нескінченної суми. Обчислення виконати із заданою точністю ε (поки поточний член ряда не перевищує за абсолютною величиною заданого ε): $\frac{x}{0!}+\frac{x}{1!}+\ldots+\frac{x}{n!}+\ldots$.

```
import math
x=int(input('x='))
eps=float(input('eps='))
s=x; term=1; i=0;
while (abs(term) > eps):
  term=term*(1/(i+1))
  s = s + x*term; i+=1
print(s)
```

Теоретична частина

1. ЦИКЛ FOR

Цикл *for* − універсальний ітератор послідовностей: він може виконувати обхід елементів в будь−яких впорядкованих об'єктах−послідовностях. Інструкція *for* здатна обробляти рядки, списки, кортежі, інші вбудовані об'єкти, які підтримують можливість виконання ітерацій. Схематично цикл *for* зображено на рис. 6.3.

Цикли *for* в Python розпочинають з рядка заголовка, де вказують змінну циклу для присвоювання, а також об'єкт, обхід якого буде виконано.

За заголовком розташовано блок (зазвичай з відступами) інструкцій, які потрібно виконати:

```
for <target> in <object>:
# Пов'язує елементи об'єкта зі змінною циклу
  <Statements>
# тіло циклу: використовує змінну циклу
else:
  <Statements>
# Якщо не потрапили на інструкцію 'break'
```

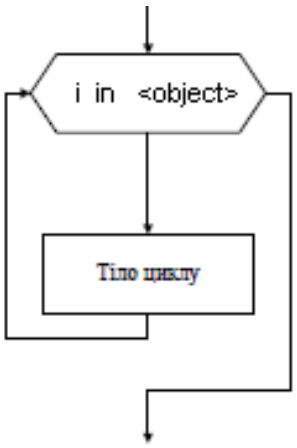


Рис. 6.1.

Коли інтерпретатор виконує цикл *for*, він по черзі присвоює змінній циклу елементи об'єкта–послідовності і виконує оператори з тіла циклу для кожного елементу. Змінну циклу можна змінити в тілі циклу, проте їй автоматично буде присвоєно наступний елемент послідовності, коли управління повернеться на початок циклу. Після виходу з циклу ця змінна зазвичай посилається на останній елемент послідовності (якщо цикл не був завершений інструкцією *break*).

Інструкція *for* підтримує необов'язкову частина *else*, яка працює так само, як і в циклах *while* − її виконують, якщо вихід з циклу реалізовано не інструкцією *break* (тобто, якщо в циклі не виконано обхід всіх елементів послідовності). Інструкції *break* і *continue* в циклах *for* працюють так само, як і у циклах *while*. Повна форма циклу *for* має такий вигляд:

```
for <target> in <object>:
# змінній циклу присвоюють  елементи об'єкта
     <Statements>
     if <test>: break # Вихід з циклу, минаючи блок else
     if <test>: continue # Перехід на початок циклу
else: <Statements> # Якщо не викликано інструкцію 'break'
```

Розглянемо приклад використання циклу for.

Приклад 6.5. *Перевірка гіпотези Сиракуз*.

Ввести число *N*. Гіпотеза стверджує: незалежно від вибору першого числа *N* рано чи пізно ми отримаємо значення «1», якщо виконаємо такі дії над цим числом: а) якщо число парне, то розділити його навпіл, б) якщо непарне − помножити на 3, додати 1 і результат розділити на 2. Над отриманим числом знову повторити дії а) або б) в залежності від його парності.

```
1)import itertools
try: N=int(input("Введіть додатне ціле число N= "))
except ValueError:
    print('Помилка:N - додатне ціле число ')
else:
 for i in itertools.count(start=1, step=1.0):
    if N<=0:
        print('Помилка: N - додатне число '); break
    if N%2==0: N=N//2
    else: N=(N*3+1)//2
```

```
    print('N = ', N); if N==1: break
2.n = int(input())
while n != 1:
     if n % 2 == 0: n = n // 2
     else:n = (3*n + 1) // 2
     print(n, end=' ')
```

2. ПОБУДОВА БЛОК-СХЕМИ ОБЧИСЛЕННЯ ІТЕРАЦІЙНИХ ПРОЦЕСІВ

Існують циклічні обчислювальні процеси, які називають ітераційними. Ітераційний процес продовжують, поки різниця між сусідніми значеннями, які уточнюються на кожному кроці циклу (ітерації), не стане менше або дорівнювати деякому заданому значенню (точності). Характерною особливістю ітераційного процесу є те, що кількість циклів (ітерацій) в ньому заздалегідь не відома. Вихід з циклу відбувається тоді, коли аналізована величина або величини на черговій ітерації відрізняються від еталонних (наприклад, заданої точності). У такому циклі, як правило, результати обчислення попереднього кроку циклу використовують як початкові дані для виконання наступного кроку циклу (приклад 6.5).

Приклад 6.6. Скласти блок−схему визначення з точністю ε=0.001 відрізку степеневого ряду для функції sin(*x*):

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ... + (-1)^{m-1}\frac{x^{2m-1}}{(2m-1)!} + ..., \; x \in \Re.$$

Точність отриманого значення вважати досягнутою, якщо останній член ряду не перевищує за абсолютною величиною значення ε (рис. 6.2).

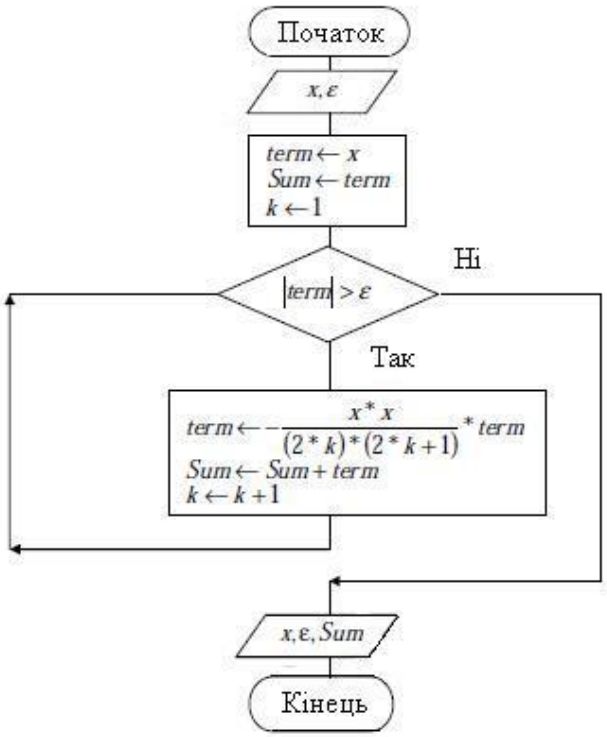


Рис. 6.2. Блок-схема прикладу 6.6

Приклад 6.7. Скласти блок−схему визначення наближеного значення квадратного кореня $y=\sqrt{x}$, $x>0$ як ліміт послідовності y_1, y_2, ..., y_k, ... де $y_{k+1}=\frac{1}{2}\left(y_k+\frac{x}{y_k}\right)$. Точність отриманого наближення оцінити за величиною відхилення двох сусідніх наближень y_k та y_{k+1}, $\Delta_{k+1}=|y_k-y_{k+1}|\le\varepsilon$, де ε=0.0001 − задана точність обчислень (рис. 6.3).

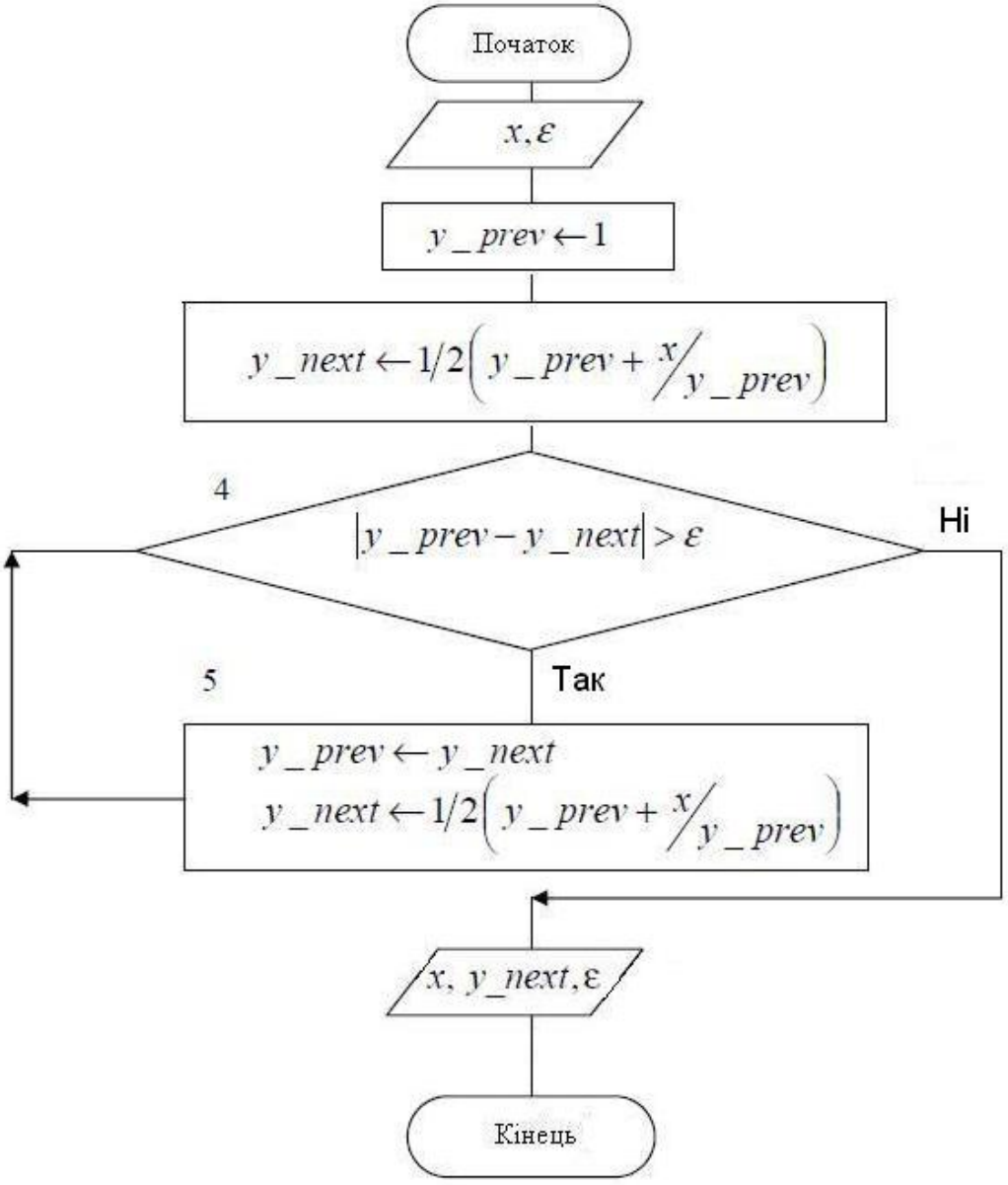


Рис. 6.3. Блок-схема прикладу 6.7

Заданий елемент арифметичної прогресії, наприклад, $-\frac{1}{3}+\frac{1}{5}-...+\frac{(-1)^n}{2n+1}$ можна визначити, вказавши номер елементу, перший і загальний елементи:

```
N = INT(INPUT('N = '))
Z=(-1/3)+(-1)**N/(2*N+1); PRINT (Z)
```

Комп'ютерні практикуми № 7-8. Списки: одновимірні та двовимірні масиви. Генерація випадкових чисел

Мета роботи: ознайомитися з особливостями визначення та використання одновимірних та двовимірних масивів, структурною організацією масивів та способів доступу до їх елементів. *Об'єкт дослідження* – списки, масиви даних, процедура генерації випадкових чисел (модуль *random*).

ПЛАН

1. Списки
2. Масиви
3. Модуль *random*

Завдання

1. Ознайомитися з теоретичним матеріалом. Опрацювати приклади.
2. Відповідно до свого варіанту
- визначити умови;
- за допомогою формул описати варіанти виконання необхідний дій;
- розробити програмний додаток, який розв'язує завдання
- організувати генерацію випадкових чисел (якщо треба, введення даних з клавіатури і виведення у консоль);
- вивести на екран початковий та отриманий (отримані) вектори/матриці (або отриманий результат).

3. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасировки виконання програми.

Варіанти

Завдання 1. Одновимірні масиви (вектори)

Використовуючи генератор випадкових чисел, заповнити список [a_1, …, a_n] елементами:

а) дійсними числами, які лежать в діапазоні від 0 до 1;

б) цілими додатними та від'ємними числами, які лежать в діапазоні від – 10 до 10 включно;

в) цілими додатними числами, які лежать в діапазоні від 0 до 50 включно.

Нехай задано список різних випадкових чисел [a_1, …, a_n], значення n визначає користувач програми.

1. Задано список (*б*). Написати програму формування іншого списку, в якому усі елементи, які передують найбільшому від'ємному елементу, замінити на значення їх квадратів.

2. Задано список (*б*). Написати програму формування іншого списку, в якому, якщо еленти заданого списку не утворюють послідовності, яка зменшується, то замінити його від'ємні елементи одиницями.

3. Задано список (*б*). Написати програму формування іншого списку, в

якому переставити елементи таким чином, щоб спочатку були розташовані всі невід'ємні елементи, а вкінці − від'ємні елементи.

4. Задано список (*б*). Написати програму формування іншого списку, в якому елементи сформовані таким чином $[a_1, a_{n+1}, a_2, a_{n+2}, ..., a_n, a_{2n}]$.

5. Задано список (*б*). Перевірити чи утворюють елементи заданого масиву послідовність, яка чітко зменшується або збільшується. Вивести відповідне повідомлення.

6. Задано список (*а*). За заданими дійсними числами $a_0, a_1, ..., a_n, t$ обчислити значення багаточлена $a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ і його похідної в точці t.

7. Задано список (*б*). Написати програму визначення суми всіх елементів, розміщених до останнього додатного елемента включно.

8. Задано список (*в*). Написати програму визначення суми лише тих елементів, які є непарними числами.

9. Задано список (*б*). Написати програму визначення добутку елементів, розміщених між максимальним за модулем та мінімальним за модулем елементами.

10. Задано список (*б*). Написати програму формування іншого списку, в якому елементи сформовані таким чином, що нульові елементи перенесено у хвіст списку.

11. Задано список (*б*). Написати програму визначення суми модулів елементів, розміщених після першого нульового.

12. Задано список (*б*). Написати програму визначення суми елементів, розміщених між першим та другим від'ємними елементами.

13. Задано список (*б*). Написати програму формування іншого списку, в якому елементи сформовані таким чином: якщо хоча б одне значення елементів належить проміжку $[x, y]$, то всі елементи, які не належать цьому проміжку, замінити на z.

14. Задано список (*б*). Написати програму визначення суми чисел цієї послідовності, розташованих між максимальним и мінімальним числами (до суми включити й обидва цих числа).

15. Задано різні два списки різних цілих випадкових чисел $[a_1, a_2, \ldots, a_{3n}]$ (0.б). Написати програму визначення найменшого серед тих чисел першого списку, які не входять до другого (вважаючи, що хоча б одне таке число існує).

16. Задано список (*б*). Написати програму формування іншого списку, в якому всі від'ємні елементи списку перенести в його початок, а всі інші – в кінець, сбурігаючи початкове взаємне розміщення як серед від'ємних елементів, так и серед інших элементі.

17-19. Задано список (*в*). Написати програму формування іншого списку, в якому елементи впорядковані таким чином, що для всіх k виконується умова $x_k \le x_{k+1}$, використовуючи такий алгоритм сортування (впорядкування):

а) *сортування вибором*: виконується пошук максимального елементу, який переноситься в кінець масиву; потім цей метод застосовують до всіх

елементів, крім останнього і т. д.;

б) *сортування обміном (метод бульбашки)*: послідовно порівнють пари сусідніх елементів $x_к$ і x_{k+I} (k=1, 2, 3,…, n-1) і, якщо $x_к > x_{к+1}$, то їх переставляють місцями; тим самим найбільший елемент буде розташований в кінці списку; потім цей метод застосовують до всіх елементів, окрім останнього і т. д.;

в) *сортування вставками*: нехай перші k елементів масиву вже впорядковані: $x_k \le x_{k+1}$; береться (k+1)-й елемент і розміщується серед перших k елементів таким чином, щоб впорядкованими стали вже k+1 перших елементів; цей метод застосовують для k від 1 до n–1.

20. Задано список (*в*). Написати програму формування іншого списку, в якому $[a_1, a_1+a_2, a_1+a_2+a_3, \ldots, a_1+a_2+\ldots+a_n]$.

21. Задано список (*в*). Написати програму формування іншого списку, в якому виконаний циклічний зсув усіх елементів на k позицій вліво, наприклад, для k=1 маємо $a_2, \ldots, a_n, a_1$.

22. Задано список (*в*). Написати програму формування двох списків $[x_1, \ldots, x_n]$ і $[y_1, \ldots, y_n]$, в яких елементи сформовані таким чином $[a_1, a_3, \ldots, a_{2n-1}]$ і $[a_2, a_4, \ldots, a_{2n}]$.

23. Задано різні два списки $[x_1, \ldots, x_n]$ і $[y_1, \ldots, y_n]$ (*в*). Написати програму формування списку, елементи якого дорівнюють відповідним значенням $[x_1, y_1, \ldots, x_n, y_n]$.

24. Задано список $[a_1, a_2, \ldots, a_{3n}]$ (*в*). Написати програму формування іншого списку, в якому елементи дорівнюють середньому арифметичному значенню трьох наступних елементів списку.

25. Задано список випадкових чисел з нулів та одиниць $[a_1, \ldots, a_{3n}]$. Написати програму пошуку найбільшої за довжиною ділянки заданої послідовності, яка заповнена одиницями. Вивести на екран індекси початку та кінця знайденої ділянки.

26. Задано список (*в*). Написати програму пошуку всіх локальних мінімумів та максимумів в списку (елемент називається локальним мінімумом/максимумом, якщо в нього немає сусідів, які менші/більші за нього).

27. Задано різні два списки цілих випадкових чисел $[x_1, \ldots, x_n]$ і $[y_1, \ldots, y_n]$ (*в*). Написати програму формування двох списків, елементи яких сформовані за правилами: $x_1=\max(x_1, y_1)$; $y_1=\min(x_1, y_1)$.

28. Задано список (*б*). Написати програму визначення кідькості елементів, що передують першому від'ємному елементу та їх значення належать проміжку $[x, y]$ (значення x, y введені з клавіатури).

Завдання 2. Двовимірні масиви (матриці)

Використовуючи датчик випадкових чисел, заповнити список $[[a_{11}, \ldots, a_{1n}], \ldots, [a_{m1}, \ldots, a_{mn}]]$ елементами:

а) дійсними числами, які лежать в діапазоні від 0 до 1;

б) цілими додатними та від'ємними числами, які лежать в діапазоні від –10 до 10 включно;

в) цілими додатними числами, які лежать в діапазоні від 0 до 20 включно;

г) дійсними числами, які лежать в діапазоні від –10 до 10;

Нехай задано список [[a_{11}, ..., a_{1n}], ..., [a_{m1}, ..., a_{mn}]] різних випадкових чисел, значення n,m визначає користувач програми.

Елементи головної діагоналі розташовані з лівого верхнього кута матриці до правого нижнього; друга, зворотна діагональ матриці є побічною.

1. Задано список (*б*). Написати програму, яка змінить місцями два стовпчики: стовпчик, який містить максимальний від'ємний елемент, і стовпчик, який містить мінімальний додатний елемент матриці.

2. Задано список (*в*). Написати програму, яка змінить місцями перший рядок з рядком, що містить максимальний елемент матриці.

3. Задано список (*б*). Написати програму, в якій визначити k – кількість «особливих» елементів матриці.

Примітка. Елемент є «особливим», якщо: а) він більше суми інших елементів свого стовпчика; б) в його рядку зліва від нього розташовані елементи, які меньші за нього, а справа – більші.

4. Задано список (*б*). Написати програму, яка змінить місцями останній рядок з рядком, який містить мінімальний додатний елемент матриці.

5. Задано список (*б*), $n=m$. Написати програму, яка визначить, чи є задана квадратна матриця симетричною відносно головної діагоналі.

6. Згенерувати список, $n=m$, який визначає квадратну матрицю – магічний квадрат (тобто така, в якій суми елементів у всіх рядках і стовпчиках є однаковими). Вивести на екран отриманий результат.

7. Задано список (*б*) (якщо треба згенерувати відповідний список). Написати програму, яка визначить добуток від'ємних елементів другого рядка та кількість елементів в другому стовпчику, які не кратні «5».

8. Задано список (*б*). Написати програму, яка визначить новий список, в якому кожен елемент матриці помножений на мінімальний за абсолютною величиною елемент у поточному стовпчику.

9. Задано список (*г*). Написати програму, яка змінить місцями останній стовпчик і стовпчик, який містить мінімальний додатний елемент матриці.

10. Задано список (*б*). Написати програму, яка визначити максимальний елемент третього стовпчика та суму непарних елементів першого рядка.

11. Задано список (*г*). Написати програму, яка визначить рядок, сума елементів якого мінімальна. Вивести на екран початкову матрицю та визначений рядок.

12. Задано список (*г*). Написати програму, яка визначить суму елементів кожного стовпчика (результат записати в інший список).

13. Задано список (*в*). Написати програму, яка визначить суму елементів тих рядків, у яких на побічної діагоналі стоять невід'ємні числа.

14. Задано список (*в*). Написати програму, яка сформує новий список, в якому всі елементи матриці вище побічної діагоналі є нульовими.

15. Задано список (*г*). Написати програму, яка змінить місцями перший стовпчик і стовпчик, який містить мінімальний за абсолютною величиною елемент матриці.

16. Задано список (*г*). Написати програму, яка визначить суму елементів парних рядків, записати результат у новий список.

17. Задано список (*г*). Написати програму, яка визначить суму елементів в кожному стовпчику, записати результат у новий список.

18. Задано список (*г*). Написати програму, яка змінить місцями перший рядок і рядок, який містить мінімальний елемент матриці.

19. Задано список (*г*). Написати програму, в якій кожен елемент матриці помножено на максимальний елемент у поточному рядку.

20. Задано список (*г*). Написати програму, яка змінить матрицю таким чином, щоб всі елементи нижче головної діагоналі були нульовими.

21. Задано список (*в*). Написати програму, яка сформує новий список в якому змінено місцями два рядки: рядок, що містить максимальний елемент, і рядок, що місмтить мінімальний елемент.

22. Задано список (*б*) (якщо треба згенерувати відповідний список). Написати програму, яка визначає добуток від'ємних парних чисел першого стовпчика.

23. Задано список (*б*). Написати програму, яка сформує новий список в якому змінено місцями останній рядок і рядок, що містить мінімальний за абсолютною величиною елемент.

24. Задано список (*б*). Написати програму, яка визначає скалярний добуток рядка, в якому знаходиться найбільший елемент матриці, на стовпчик із найменшим елементом.

25. Задано список (*б*)), $n=m$ (якщо треба згенерувати відповідний список). Написати програму, яка визначає чи є задана квадратна матриця ортонормованою.

Підказка: квадратна матриця є ортонормованою, якщо в ній скалярний добуток кожної пари різних рядків дорівнює «0», а скалярний добуток кожного рядка на себе – «1».

26. Задано список (*б*) (якщо треба згенерувати відповідний список). Написати програму, яка визначає суму чисел, які кратні трьом, та суму від'ємних чисел елементів третього стовпчика.

27. Задано список (*б*) (якщо треба згенерувати відповідний список). Написати програму, яка визначає добуток парних елементів другого стовпчика та добуток непарних елементів другого рядка матриці.

28. Задано список (*б*). Написати програму, яка перетворить масив, виконавши поворот елементів навколо його центра на 90° проти часової стрілки.

29. Задано список (*в*). Для заданої матриці написати програму, яка визначить індекси всіх її сідлових точок.

Примітка. Елемент матриці назвемо сідловою точкою, якщо він є найменшим у своєму рядку й одночасно найбільшим у своєму стовпчику або навпаки (є найбільшим у своєму рядку й найменшим у своєму стовпчику).

Контрольні запитання

1. Що таке одновимірний масив? Для чого використовують одномірні масиви? Як їх описують в Python?

2. Як в програмі використати значення конкретного елемента одновимірного масиву?

3. Для чого в програмах використовуються двовимірні масиви? Як їх описують в Python?

4. Скільки індексів характеризують конкретний елемент двовимірного масиву?

5. Як в програмі використати значення конкретного елемента двовимірного масиву?

6. Який індекс двовимірного масиву змінюється швидше при послідовному розміщенні елементів масиву в оперативній пам'яті?

Аудиторна робота

Приклад 7.1. Список

1. Список – впорядкована послідовність певних значень, які можуть повторюватися. Для створення списку необхідно записати його елементи через кому у квадратних дужках

```
int_list = [1,2,3,5] #  список із чотирьох цілих чисел
char_list = ['a', 'c', 'z', 'x'] #  список  із чотирьох символів
empty_list = []# empty_list – пустий список

print('Список чисел:', int_list)
print('Список символів:', char_list)
print('Пустий список:', empty_list)
```

Результат

```
Список чисел: [1, 2, 3, 5]
Список символів: ['a', 'c', 'z', 'x']
Пустий список: []
```

2. Значення елемента можна отримувати за його індексом (номером). Індексація починається з нуля.

```
# Створення списку чисел
my_list = [5, 7, 9, 1, 1, 2]
# Виведення  першого елемент
print (my_list [0]); print (my_list)
# Введення індексу
index = int (input ('Введіть номер елемента:'))
# Отримання відповідного елемента
element=my_list[index]
# Виведення  його значення на екран
print (element)
```

Результат

```
[5, 7, 9, 1, 1, 2]
Введіть номер елемента: 2
9
```

3. Якщо використати від'ємні індекси, то обхід елементів розпочинається з останнього. Індекс останнього елемента списку – «–1», передостаннього – «–2».

```
# Створення списку чисел
my_list = [5, 7, 9, 1, 1, 2]
# Отримання передостаннього значення
pre_last = my_list [-2] # pre_last == «1»
print (pre_last)
# Обчислення суми першого і останнього значень
result = my_list[0] + my_list[-1];print(result)
```

Результат

```
1
7
```

4. Методи обробки списків:

– метод *append* додає значення в список

```
my_list = []# Створення пустого списку
my_list.append(3); my_list.append(5)
my_list.append(my_list[0]+my_list[1])

print(my_list) # Виведення списку на екран
```

Результат

```
[3, 5, 8]
```

- видалення елемента списку

```
my_list = [5, 1, 5, 7, 8, 1, 0, -23] # Створення списку чисел
print(my_list) # Виведення списку
# Оператор del видаляє заданий елемент
del my_list[2]
print(my_list) # Виведення списку
```

Результат

```
[5, 1, 5, 7, 8, 1, 0, -23]
[5, 1, 7, 8, 1, 0, -23]
```

5. Заміна елементу списка

```
my_list = [5, 1, 5, 7, 8, 1, 0, -23] # Створення списку чисел
print(my_list) # Виведення списку
length = len(my_list) # Отримання довжини списка
# Введення индексу
index = length
while not -length <= index < length:
 index=int(input('Введіть індекс ел-та списку (від %d до
%d):'
                 % (-length, length - 1)))
# Введення нового значення
value = int(input('Введіть нове значення заданого ел-та: '))
my_list[index]=value # зміна елемента списку
print(my_list) # Виведення списку на екран
```

Результат

```
[5, 1, 5, 7, 8, 1, 0, -23]
```

```
Введіть індекс елемента списку (від -8 до 7): 7
Введіть нове значення заданого елемента: 5
[5, 1, 5, 7, 8, 1, 0, 5]
```

6. Виведення квадратів чисел зі списку

```
my_list = [5, 1, 5, 7, 8, 1, 0, -23] # Створення списку чисел
for x in my_list:
    print('{}^2={}'.format(x, x ** 2))
```

Результат

```
5 ^ 2 = 25
1 ^ 2 = 1
5 ^ 2 = 25
7 ^ 2 = 49
8 ^ 2 = 64
1 ^ 2 = 1
0 ^ 2 = 0
-23 ^ 2 = 529
```

7. Функція-конструктор, яка створює список із значення іншого типу

```
empty_list=list() # Створення пустого списку
print(empty_list)

# Створення списку з послідовності range
numbers = list(range(10))
print(numbers)

# Створення списку символів із рядка
chars = list("a string"); print(chars)
```

Результат

```
[]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['a', ' ', 's', 't', 'r', 'i', 'n', 'g']
```

8. Числа Фібоначчі

```
n=10 # Кількість чисел в послідовності
# Список чисел Фібоначчі (напочатку має дві одиниці)
fibs = [1, 1]
# Повторюємо (n-2) рази, тому що два числа вже є в списку
for i in range(n-2):
   # Додаємо суму двох останніх чисел
   fibs.append(fibs[i]+fibs[i+1])
print(fibs) # Виведення списку на екран
```

Результат

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Приклад 7.2. Зріз списка – отримання групи елементів за їхніми індексами

1.

```
my_list = [5, 7, 9, 1, 1, 2] # Створення списку чисел
# Отримання зрізу списка від нульового (першого) елемента
# (включаючи його) до третього (четвертого) (не включаючи)
```

```
sub_list = my_list[0:3]
print(sub_list) # Виведення отриманого списку
# Виведення елементів списка від другого до переостаннього
print(my_list[2:-2])
# Виведення ел-тів списка від 4-ого (5-ого) до 5-ого (6-ого)
print(my_list[4:5])
```

Результат

```
[5, 7, 9]
[9, 1]
[1]
```

2. Зріз з використанням кроку

```
my_list = [5, 7, 9, 1, 1, 2]
# Вибір кожного другого ел-та списку (починаючи з першого),
# не включаючи останній елемент
sub_list = my_list[0:-1:2]
print(sub_list) # виведення отриманого списку
# Виведення елементів від 2-ого (3-ього) до передостаннього
# з кроком 2
print(my_list[2:-2:2])
# Виведення ел-тів списка, крім першого, в зворотному
порядку
print(my_list[-1:0:-1])
```

Результат

```
[5, 9, 1]
[9]
[2, 1, 1, 9, 7]
```

3. Будь-який параметр зріза можна пропустити (при умові дотримання правильного розміщення двокрапок)

```
# За замовчуванням початок списка – 0, кінець – довжина списка,
крок – 1
my_list=[5, 7, 9, 1, 1, 2]
# Виведення елементів списка від 2-ого (3-ього) значення до кінця
print(my_list[2:])
# Виведення всіх ел-тів списка від початку до передостаннього ел-
та
print(my_list[:-2])
# Виведення всіх елементів списку в зворотному порядку
print(my_list[::-1])
[-1:0:-1])
```

Результат

```
[9, 1, 1, 2]
[5, 7, 9, 1]
[2, 1, 1, 9, 7, 5]
```

Приклад 7.3. Для перевірки входження елемента в список використовують операцію *in*

1. Перевірка, чи є задане число у списку

```
my_list = [5, 7, 9, 1, 1, 2]
# Введення значення
```

```
value=int(input('Введіть число: '))
if value in my_list:
    print('Число входить у список')
else:
    print('Число не входить у список')
```

Результат

```
Введіть число: 5
Число входить у список
```

2. Визначення кількості елеметів списка

```
my_list = [1, 5, 1, 3, 7, 8, 124]
print(len(my_list))
```

Результат

```
7
```

Приклад 7.4. Нехай задано два списки цілих випадкових чисел від 0 до 5: [a_1, ..., a_n] і [b_1, ..., b_n], n=10. Написати програму їх формування. Вивести списки на екран.

```
import random
a=[random.randint(0,5) for i in range(0,10)]
b=[random.randint(0,5) for j in range(0,10)]
print(a); print(b)
```

Результат

```
[4, 3, 5, 5, 1, 4, 5, 2, 1, 5]
[4, 0, 4, 5, 4, 2, 2, 2, 4, 5]
```

Приклад 7.5. Нехай згенеровано список із цілих випадкових чисел та нулів [a_1, ..., a_n]. Написати програму визначення елементів, розміщених після першого нульового. Вивести на екран початковий та отриманий списки.

```
import random
arr=random.sample(range(-6,6), 12)
print("our random list: " , arr)
flag=0
while flag==0:
 if 0 in arr: # check if we have at list one zero in list
  first_zero_index=arr.index(0)#find index of first zero in list
  flag=1;
 else:
   print("we have not at list one zero in list: ")
arr1=[]
if flag==1:
 for i in arr[first_zero_index:]:
   arr1.append(i)
print (arr1)
```

Результат

```
our random list:  [-2, 3, 5, -5, -6, -4, 4, 1, 0, -3, -1, 2]
[0, -3, -1, 2]
```

Приклад 7.6. Нехай задано список-матриця цілих випадкових чисел (додатних та від'ємних) [[a_{11}, ..., a_{1n}], ..., [a_{m1}, ..., a_{mn}]]. Написати програму, яка

визначить список мінімальних елементів кожного рядка (результат записати в інший список). Вивести на екран початкову матрицю та отриманий список.

```
import random
n=3 # кількість стовпчиків
m=4 # кількість рядків
x=[0]*m
for i in range(m):
    x[i]=[0]*n
y=[0]*m; i=0
while i<m :
j=0; k=0
while j < n :
     x[i][j] = random.randint(-5,5)
     j+=1
y[i] = min(x[i][0:m])
k+=1; i+= 1
print('x= ', x); print('min = ', y)
```

Теоретична частина

1. СПИСКИ

Список − колекція інших об'єктів, змінюваний об'єкт, вони можуть бути вкладеними, збільшуватися і зменшуватися, містити об'єкти будь-яких типів. Завдяки спискам можна створювати і обробляти в своїх сценаріях структури даних будь-якого ступеня складності. Нижче наводяться основні властивості списків, списки в мові Python − це:

1. *Впорядковані колекції об'єктів довільних типів.*

2. *Доступ до елементів за зсувом*. Можна використовувати операцію індексування для отримання окремих об'єктів зі списку за їхнім зсувом.

3. *Змінна довжина, гетерогенність і довільна кількість рівнів вкладеності*. Списки можуть збільшуватися і зменшуватися безпосередньо (їх довжина може змінюватися), вони можуть містити не тільки односимвольні рядки, а й будь-які інші об'єкти (списки гетерогенні). Списки можуть містити інші складні об'єкти, вони підтримують можливість створення довільної кількості рівнів вкладеності, тому є можливість створювати зі списків списки списків.

4. *Відносяться до категорії змінних об'єктів.*

У табл. 6.1 наведено найбільш типові операції, які застосовують до списків. Коли список визначається виразом (літерально), його записують як послідовність об'єктів в квадратних дужках, розділених комами, наприклад:

```
x='123'; a=list(x); x=''.join(a); print(x)
>>1
```

Вкладені списки описують як вкладені послідовності квадратних дужок (рядок 3 у табл. 6.1), а порожні списки визначають як порожню пару квадратних дужок (рядок 1 у табл. 6.1).

Таблиця 6.1

Літерали списків і операції

Операція	Інтерпретація
L = []	Пустий список
L = [0, 1, 2, 3]	Чотири елемента з індексами 0..3
L = [‘abc’, [‘def’, ghi’]]	Вкладені списки
L = list(‘spam’)	Створення списка із рядка
L = list(range(−4, 4))	Створення списка із безперервної послідовності цілих чисел
L[i]	Індекс
L[i][j]	індекс індекса
L[i:j]	зріз
len(L)	довжина
L1 + L2	Конкатенація
L * 3	дублювання
for x in L: print(x)	Обхід у циклі, x – змінна для присвоювання
3 in L	перевірка входження
	Методи:
L.append(4)	додавання елемента «4» у список
L.extend([5,6,7])	додавання списка у список
L.insert(1, X)	додавання списку елементів у вказану позицію
	Методи:
L.index(1)	Визначення зсуву елемента за заданим значенням
L.count()	Підрахунок кількості елементів
L.sort()	Сортування
L.reverse()	Зміна порядку слідування елементів на зворотний
	Зменшення списку
del L[k]	видалення елемента
del L[i:j]	видалення групи елементів
L.pop()	видалення останнього елемента й повернення його значення
L.remove(2)	видалення елементі з визначеними значеннями
L[i:j] = []	
L[i] = 1	Присвоювання за індексом
L[i:j] = [4,5,6]	Присвоювання зрізу значень
L=[x**2 for x in range(5)]	Генератори списків
list(map(ord, ‘spam’))	відображення

Оскільки списки є послідовностями, вони, як і рядки, підтримують оператори + і * (для списків вони так само виконують операції конкатенації і повторення), а в результаті виходить новий список:

```
>>> len([1, 2, 3]) # Довжина
3
>>> [1, 2, 3] + [4, 5, 6] # Конкатенація
[1, 2, 3, 4, 5, 6]
>>> ['Ni!']*4 # Повторення
['Ni!', 'Ni!', 'Ni!', 'Ni!']
```

Не можна виконати операцію конкатенації для списку і рядка, якщо попередньо не перетворити список в рядок (використовуючи, наприклад, функцію *str* або оператор форматування %) або рядок в список (за допомогою вбудованої функцією *list*):

```
>>> str([1, 2]) + "34" # То же, что и "[1, 2]" + "34"
'[1, 2]34'
>>> [1, 2] + list("34") # То же, что и [1, 2] + ["3", "4"]
[1, 2, '3', '4']
```

Методи списків. Об'єкти списків в Python підтримують специфічні методи, багато з яких змінюють сам список безпосередньо:

```
>>> L.append('please')
# Виклик метода додавання елемента у кінець списка
>>> L
['eat', 'more', 'SPAM!', 'please']
>>> L.sort() # Сортування елементів списка ('S'<'e')
>>> L
['SPAM!', 'eat', 'more', 'please']
```

Методи – це функції, пов'язані з певним типом об'єктів.

Метод *append* додає один елемент (посилання на об'єкт) в кінець списку. На відміну від операції конкатенації, метод *append* приймає один об'єкт-список. За своєю дією вираз L.append (X) схожий на вираз L+[X], але в першому випадку змінюється сам список, а в другому – створюється новий список. На відміну від операції конкатенації (+), метод *append* не створює новий об'єкт, тому зазвичай він виконується швидше. Існує можливість імітувати роботу методу *append* за допомогою операції присвоювання зрізу: вираз L[len (L):]=[X] відповідає виклику L.append (X), а вираз L [:0] = [X] відповідає операції додавання в початок списку. В обох випадках видаляється порожній сегмент списку і вставляється елемент X, при цьому змінюється сам список L, так само швидко, як при використанні методу *append*.

Метод *sort* виконує перевпорядкуванне елементів в списку. За замовчуванням він використовує стандартні оператори порівняння мови Python (в даному випадку виконується порівнювання рядків) і виконує сортування в порядку зростання значень. Існує можливість змінити порядок сортування за допомогою іменованих аргументів – спеціальних синтаксичних конструкцій типу «= name == value», які використовують під час виклику функцій для передачі параметрів налаштування за їхніми іменами. Іменований аргумент *key* у виклику методу *sort* дозволяє визначити власну функцію порівняння, яка приймає один аргумент і повертає значення, яке буде використано в операції порівняння, а іменований аргумент *reverse* дозволяє виконати сортування не в порядку зростання, а в порядку убування:

```
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort() # Сортування з урахуванням регістру символів
>>> L
['ABD', 'aBe', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
```

```
>>> L.sort(key=str.lower)
# Приведення символів до нижнього регістру
>>> L
['abc', 'ABD', 'aBe']
>>>
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower, reverse=True)
# Змінює напрямок сортування
>>> L
['aBe', 'ABD', 'abc']
```

Методи *append* і *sort* змінюють сам об'єкт списку і не повертають список у вигляді результату (точніше кажучи, обидва методи повертають значення None). Якщо ви написали інструкцію L = L.append (X), ви не отримаєте змінене значення L (насправді ви взагалі втратите посилання на список) – використання атрибутів *append* і *sort*, призводить до зміни самого об'єкта, тому немає ніяких причин виконувати повторне присвоювання. Вбудована функція *sorted* здатна сортувати списки і будь-які інші послідовності, вона повертає новий список з результатом сортування (оригінальний список при цьому не змінюється):

```
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted(L, key=str.lower, reverse=True)
# функція сортування
['aBe', 'ABD', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted([x.lower() for x in L], reverse=True)
# елементи попередньо
['abe', 'abd', 'abc'] # зменюються
```

В останньому прикладі перед сортуванням за допомогою генератора списків виконується приведення символів до нижнього регістра, і значення елементів в отриманому списку відрізняються від значень елементів в оригінальному списку. В останньому прикладі виконується сортування тимчасового списку, створеного в процесі сортування. Іноді вбудована функція *sorted* може виявитися більш зручною, ніж метод *sort*.

Метод *reverse* змінює порядок проходження елементів в списку на зворотний, а методи *extend* і *pop* вставляють кілька елементів в кінець списку і видаляють елементи з кінця списку відповідно. Крім того, існує вбудована функція *reversed*, яка нагадує вбудовану функцію *sorted*, але її необхідно обгорнути в виклик функції *list*, тому що вона повертає ітератор:

```
>>> L=[1,2]; L.extend([3,4,5]) # Додавання елементів у кінець
списку
>>> L
[1, 2, 3, 4, 5]
>>> L.pop() # видаляє і повертає останній елемент списка
5
>>> L
[1, 2, 3, 4]
>>> L.reverse() # Змінює порядок слідування елементів на зворотний
>>> L
```

```
[4, 3, 2, 1]
>>> list(reversed(L))
# Вбудована функція сортування в зворотному порядку
[1, 2, 3, 4]
```

Інші методи списків дозволяють видаляти елементи з певними значеннями (*remove*), вставляти елементи у визначену позицію (*insert*), визначяти зсув елемента за заданим значенням (index) тощо:

```
>>> L = ['spam', 'eggs', 'ham']
>>> L.index('eggs') # індекс об'єкта
1
>>> L.insert(1, 'toast') # Вставка у потрібну позицію
>>> L
['spam', 'toast', 'eggs', 'ham']
>>> L.remove('eggs') # видалення елемента із визначеним значенням
>>> L
['spam', 'toast', 'ham']
>>> L.pop(1) # видалення елемента у вказаній позиції
'toast'
>>> L
['spam', 'ham']
```

Можна використати інструкцію *del* для видалення елемента або зрізу безпосередньо зі списка:

```
>>> L
['SPAM!', 'eat', 'more', 'please']
>>> del L[0] # видалення одного елемента списка
>>> L
['eat', 'more', 'please']
>>> del L[1:] # видалення цілого сегмента списка
>>> L # То же, что и L[1:] = []
['eat']
```

Можна видаляти зрізи списку, привласнюючи їм порожній список (L[*i:j*]=[]) – інтерпретатор спочатку видалить зріз, який визначається зліва від оператора =, а потім вставить порожній список. Присвоювання порожнього списку за індексом елемента призведе до збереження посилання на порожній список в цьому елементі, а не до його видалення:

```
>>> L = ['Already', 'got', 'one']; L[1:] = []
>>> L
['Already']
>>> L[0] = []
>>> L
[[]]
```

2. МАСИВИ

Для зберігання і обробки в програмах складних видів інформації використовують структурні типи. Їх утворюють шляхом об'єднання простих елементів даних (компонентів). Компоненти можуть бути при цьому однорідними або різнорідними. Необхідність у масивах виникає щоразу, коли в

пам'яті потрібно зберігати велику, але скінченну кількість однотипних впорядкованих даних.

Масив – це структура даних, яку можна розглядати як набір змінних однакового типу, що мають загальне ім'я.

Доступ до будь-якого елементу масиву здійснюється за його номером. У масиви можна об'єднати результати експериментів, списки прізвищ співробітників, різні складні структури даних.

У масиві дані різняться своїм порядковим номером (індексом). Якщо кожний елемент масиву визначається за допомогою одного номера, то такий масив називається *одновимірним*, якщо за двома — то *двовимірним*.

Двовимірний масив — це таблиця з рядків і стовпчиків. У таблицях перший номер вказує на рядок, а другий — на положення елемента в рядку. Усі рядки таблиці мають однакову довжину.

Одновимірний масив (лінійна таблиця) може бути набором чисел, сукупністю символьних даних чи елементів іншої природи (навіть масив масивів). Так само, як і в послідовності, в одновимірному масиві можна вказати елемент з конкретним номером, наприклад a5, або записати загальний вигляд елемента, використовуючи як індекс змінну *i*, вказуючи діапазон її зміни: *a*[*i*], *i*=1, 2, ... , *n*. Для позначення окремої компоненти до імені масиву додається індекс, який і виділяє потрібну компоненту (наприклад, a_1, …, a_{50}).

Найменший індекс називається *нижньою межею*, найбільший – *верхньою межею*, а число елементів – *розміром масиву*. Розмір масиву фіксується при описі і в процесі виконання програми не змінюється. Індекси можна обчислювати. Найчастіше в якості типу індексу використовується обмежений цілий тип.

Базові типи мови Python підтримують можливість створення вкладених конструкцій довільної глибини і в будь-яких комбінаціях (наприклад, зображення матриць, або «багатовимірних масивів»). Це можна зробити за допомогою списку, що містить вкладені списки:

```
>>> M = [[1, 2, 3], # Матриця 3x3 у вигляді вкладених списків
[4, 5, 6], # Вираз в квадратних дужках може
[7, 8, 9]] # займати декілька рядків
>>> M
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Тут реалізовано список, який містить три інших списків. В результаті отримано матрицю чисел 3×3. Звернутися до такої структури можна різними способами:

```
>>> M[1] # отримати рядок 2
[4, 5, 6]
>>> M[1][2]
# отримати рядок 2, а потім елемент 3 в цьому рядку
6
```

Один з найпростіших способів подання матриць (багатовимірних масивів) полягає у використанні вкладених списків. Нижче наводиться приклад

двовимірного масиву розміром 3х3, побудованого на базі списків на Python:

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matrix[1]
[4, 5, 6]
>>> matrix[1][1]
5
>>> matrix[2][0]
7
>>> matrix = [[1, 2, 3],
... [4, 5, 6],
... [7, 8, 9]]
```

Нижче наведено приклад двовимірного масиву:

```
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> for i in a
…    for j in i
…         print ("i=", i, "j=", j)
>>>i=[1, 2, 3] j=1
>>>i=[1, 2, 3] j=2
>>>i=[1, 2, 3] j=3
>>>i=[4, 5, 6] j=4
>>>i=[4, 5, 6] j=5
>>>i=[4, 5, 6] j=6
```

3. МОДУЛЬ RANDOM

Модуль *random* із стандартної бібліотеки також необхідно імпортувати. Цей модуль дозволяє отримати випадкові дійсні числа в діапазоні від 0 до 1, випадкові цілі числа в заданому діапазоні, послідовність випадкових елементів, виконати випадковий вибір (в тому числі і із списка), тощо:

```
>>> import random
>>> random.random()
0.44694718823781876
>>> random.randint(1, 10)
5
>>>  random.choice(['Life  of  Brian','Holy  Grail','Meaning  of
Life'])
'Life of Brian'
>>>  random.choice(['Life  of  Brian','Holy  Grail','Meaning  of
Life'])
'Holy Grail'
>>> random.choice([1, 2, 3, 4])
1
```

До складу модуля *random* входять такі функції:

1) *random.randin (a,b)* – випадкове ціле число від *a* до *b*;
2) *random.random*() – випадкове число з інтервалу [0, 1);
3) *random.choice*(*x*) – обирає випаджковий елемент послідовності;
4) *random.shuffle(x)* – перемішує елементи послідовності;
5) *random.uniform(a,b)* – випадкове дійсне число від *a* до *b*.

Комп'ютерний практикум № 9. Рядки символів. Множини

Мета роботи: ознайомлення з об'єктом множина мови Python. *Об'єкт дослідження* – множина, рядки символів.

ПЛАН

1. Множина

Завдання

1. Вивчити теоретичний матеріал. Опрацювати приклади.
2. Відповідно до свого варіанту
- визначити умови;
- за допомогою формул описати варіанти виконання необхідний дій;
- написати програму, яка розв'язує завдання, реалізуючи обробку помилок (виключень);
- організувати введення даних з клавіатури, виведення у консоль.
3. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасировки виконання програми.

Варіанти

Завдання 1 (обробка рядків).

1. Задано речення. Скласти програму, яка визначає і виводить на екран найбільшу кількість прогалин, розташованих підряд.

2. Задано текст, в якому є дві і більше однакові літери. Скласти програму, яка визначає і виводить на екран найбільшу кількість однакових символів, розташованих підряд.

3. Задано слово. Скласти програму, яка визначає і виводить на екран кількість різних символів в ньому.

4. Задано слово, в якому є дві і більше однакові літери. Скласти програму, яка їх визначає і виводить на екран.

5. Задано три слова. Скласти програму, яка визначає і виводить на екран літери, які не повторюються в них.

6. Задано два слова. Скласти програму, яка визначає і виводить на екран ті літери слів, які є тільки в одному з них (в тому числі повторювані). Наприклад, якщо задано слова «процесор» та «інформація», то відповідь має вигляд: п е з с і ф м а я.

7. Задано два слова. Скласти програму, яка визначає і виводить ті літери слів, які зустрічаються в обох словах тільки один раз. Наприклад, якщо задано слова «процесор» та «інформація», то відповідь має вигляд: п е ф м а я.

8. Задано два слова. Скласти програму, яка визначає, чи можна з літер першого з них здобути друге. Розглянути такі варіанти: 1) повторювані літери другого слова можуть в першому слові не повторюватися; 2) кожна літера другого слова повинна входити у перше слово стільки раз, скільки вона входить у друге.

9. Задано три слова. Скласти програму, яка визначає і виводить на екран ті літери слів, які є лише в одному зі слів. Розглянути такі варіанти: 1) повторювані літери кожного слова розглядаються; 2) повторювані літери кожного слова не розглядаються.

10. Задано три слова. Скласти програму, яка визначає і виводить на екран їх загальні літери. Повторювані літери кожного слова не розглядати.

11–**34.** У цих завданнях врахувати, що в заданому реченні немає початкових і кінцевих прогалин і символів «–», кількість слів у реченні не перевищує десять.

11. Задано речення з десяти слів. Скласти програму, яка визначає і виводить на екран заповнений ними список з 10 елементів.

12. Задано речення. Скласти програму, яка визначає і виводить на екран речення, в якому слова розташовані в зворотному порядку (наприклад, речення «мама мила раму» буде змінено на «раму мила мама»).

13. Задано речення. Скласти програму, яка визначає і виводить на екран речення, в якому слова змінено місцями (наприклад, замість першого слова розташовано останнє, а замість останнього – перше).

14. Задано речення. Скласти програму, яка визначає і виводить на екран всі його слова, відмінні від слова «привіт».

15. Задано речення. Скласти програму, яка визначає і виводить на екран: а) кількість слів, які розпочинаються з літери «н»; б) кількість слів, які закінчуються на літеру «р».

16. Задано речення. Скласти програму, яка визначає і виводить на екран: слова а) які розпочинаються і закінчуються на одну і ту ж літеру; б) які містять три літери «е»; в) які містять хоча б одну літеру «о».

17. Задано речення. Скласти програму, яка визначає і виводить на екран будь-яке його слово, що розпочинається на літеру «к».

18. Задано речення. Скласти програму, яка визначає і виводить на екран довжину його самого короткого слова.

19. Задано речення. Скласти програму, яка визначає і виводить на екран його найдовше слово (прийняти, що таке слово є одним).

20. Задано речення. Скласти програму, яка визначає і виводить на екран, чи правдивим є твердження, що його найдовше слово має більше 10 символів.

21. Задано речення. Скласти програму, яка визначає і виводить на екран всі слова в порядку неспадання їх довжин.

22. Задано речення. Скласти програму, яка визначає і виводить на екран всі слова, які зустрічаються в реченні один раз.

23. Задано речення. Скласти програму, яка визначає і виводить на екран всі його різні слова.

24. Задано речення, в якому є тільки два однакових слова. Скласти програму, яка визначає їх і виводить на екран.

25. Задано речення. Скласти програму, яка визначає і виводить на екран всі його слова, попередньо перетворивши кожне слово за таким правилом: а) замінити першу зустрінуту літеру «а» на «о»; б) видалити зі слова всі

входження останньої літери (крім неї самої), в) залишити в слові тільки перші входження кожної літери (інші видалити); г) в самому довгому слові видалити середню (середні) літери (прийняти, що таке слово є одним0.

26. Задана послідовність слів. Скласти програму, яка визначає і виводить на екран ті слова послідовності, які відмінні від першого слова і задовольняють такій властивості: а) в слові немає повторюваних літер; б) слово є симетричним.

27. Задано два речення. Скласти програму, яка для кожного слова першого речення визначає, чи входить воно в друге речення: повторювані слова першого речення: а) не розглядати; б) розглядати.

28. Задано два речення. Скласти програму, яка визначає і виводить на екран слова, які є тільки в одному з них (в тому числі повторювані).

29. Задано два речення. Скласти програму, яка визначає і виводить на екран слова, які зустрічаються в двох реченнях тільки один раз.

30. Задано текст. Скласти програму, яка перевіряє, чи правильно в ньому розставлені круглі дужки (чи знаходиться праворуч від кожної відкриваючої дужки відповідна їй закриваюча дужка, а зліва від кожної закриваючої – відповідна їй відкриваюча; припускають, що всередині кожної пари дужок немає інших дужок): відповідь має вигляд: «так» або «ні».

Якщо дужки розставлено неправильно, то повідомлення включає таку інформацію: а) якщо є зайві праві (закриваючі) дужки, то вивести повідомлення із зазначенням позиції першої такої дужки; б) якщо є зайві ліві (відкриваючі) дужки, то вивести повідомлення із зазначенням кількості таких дужок.

32. Задано текст: рядок містить арифметичний вираз, в якому використовують круглі дужки, в тому числі вкладені. Скласти програму, яка перевіряє, чи правильно в ньому розставлено дужки: відповідь має вигляд «так» або «ні».

Якщо дужки розставлено неправильно, то повідомлення включає таку інформацію: а) якщо є зайві праві (закриваючі) дужки, то вивести повідомлення із зазначенням позиції першої такої дужки; б) якщо є зайві ліві (відкриваючі) дужки, то вивести повідомлення із зазначенням кількості таких дужок.

Завдання 2 (використання множин).

1. Задано множину цілих чисел від «1» до «50». Скласти програму, яка визначає, скільки з них є числами Фібоначчі і скільки чисел, в запису яких перша значуща цифра дорівнює «1» або «2».

2. Задано множину символів від '*a*' до 'z':

Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz

Скласти програму, яка визначає і виводить на екран елементи цієї множини в алфавітному порядку.

3. Задано текст з цифр і літер латинського алфавіту. Скласти програму, яка визначає, яких літер – голосних {*a, e, i, o, u, y*} або приголосних більше в цьому тексті.

4. Задано множини символів A і символ «x». Скласти програму, яка формує множину B з множини A за таким правилом: а) додавання елемента x, якщо він відсутній в A; б) видалення елементу x, якщо він присутній в A.

5. Задано текст з латинських літер. Скласти програму, яка визначає і виводить на екран такі множини: а) символи – перші входження в текст, зберігаючи при цьому їх вихідний ваємний порядок; б) всі літери, які входять в текст не менше двох разів; в) всі літери, які входять в текст по одному разу.

6. Задано текст з латинських літер. Скласти програму, яка визначає і виводить на екран в алфавітному порядку по одному разу всі голосні літери латинського алфавіту (множина {*a, e, i, o, u, y*}), які входять в цей текст. Текст та елементи множини задано в одному реєстрі (нижньому або верхньому).

7. Скласти програму, яка формує і виводить на екран множину цілих чисел в порядку зростання з діапазону 1..1000, які можна подати у вигляді n^2+m^2, де $n, m \geq 0$.

8. Скласти програму, яка формує і виводить на екран множину простих цілих чисел в порядку спадання з діапазону від 2 до 1000, використовуючи метод решета Ератосфена.

Суть цього методу. Виписують всі цілі числа, більші за одиницю; вибирається перше з них (це 2 – просте число) і викреслюють всі кратні йому числа, крім нього самого; потім береться наступне з невикреслених чисел (це 3 – також просте число) і викреслюють всі кратні йому, крім нього самого; дії повторюють для всіх невикреслених раніше чисел. Зрештою, залишаться тільки прості числа, починаючи з двох (від 2 до 1000).

9. Задано текст в декілька рядків. Видалити з кожного непарного рядку слово, яке містить найбільшу кількість голосних {*a, e, i, ї, o, u, y, я, є, ю*}.

10. Маємо множину імен всіх однокласників: {Вася, Володя, Іра, Ліда, Марина, Міша, Наташа, Олег, Оля, Свєта, Юля). В гості до кожного з них запрошують деяку підмножину хлопців. Сформуйте масив, який містить відомості про запрошених і, використовуючи його, визначте: а) чи є серед них хоча б одна людина, що побував в гостях у всіх однокласників; б) чи є люди, які не запрошені до жодного з однокласників.

11. Задано непорожню послідовність слів з малих українських літер, між сусідніми словами розташована кома. Скласти програму, яка формує в алфавітному порядку такі множини: а) всі голосні літери українського алфавіту {*a, e, i, ї, o, u, y, я, є, ю*}, які входять в кожне слово; б) всі приголосні літери, які не входять до жодного слова; в) всі приголосні літери, які входять тільки в одне слово; г) всі голосні літери, які не входять більше ніж в одне слово.

12. Задано множину міст деякого регіону: {*a, b, c, d, e, f, g, h*}. Для кожного з них відомо множину міст, в які можна потрапити із заданого міста за один автобусний рейс без пересадки. Скласти програму, яка визначає (використовуючи масив, що містить задану інформацію для всіх міст): а) множину міст, в які можна потрапити із заданого користувачем міста; б) найкоротший (в сенсі кількості пересадок) шлях між парою заданих міст.

13. Задано текст з латинських літер. Скласти програму, яка визначає і видаляє з цього тексту три слова, які містять найбільшу кількість різних приголосних букв.

14. Задано речення на українській мові. Видалити з усього тексту одне слово, яке містить найбільшу кількість різних приголосних букв.

15. Задано речення з латинських літер з малих літер. Скласти програму, яка визначає і виводить всі голосні літери, що містить слово найбільшої довжини, і кількість повторень кожної літери. Це слово в речені видалити

16. Задано речення з латинських літер. Скласти програму, яка визначає кількість символів в усіх словах. Слова, які містять більше чотирьох різних символів, видалити.

17. Задано речення на українській мові з малих літер. Скласти програму, яка визначає і робити великими (заголовними) всі літери в тих словах, в яких приголосних менше голосних.

18. Задано речення на українській мові з великих літер. Скласти програму, яка визначає і робити маленькими всі літери в тих словах, в яких приголосних літер більше голосних.

19. У місті є множина з n вищих навчальних закладів (ВНЗ), які здійснюють закупівлю комп'ютерної техніки. Маємо множину з m=6 комп'ютерних фірм {«Діалог», «Avicom», «Нета», «Сервер», «Декада», «Dega.ru»}. Скласти програму, яка визначає: 1) в яких фірмах закупівля проводилася кожним з ВНЗ; 2) в яких фірмах закупівля проводилася хоча б одним з ВНЗ; 3) в яких фірмах жоден з вищих навчальних закладів не закуповував комп'ютери.

Підказка. Вирішити задачу з використанням операцій над множинами. Для зручності подальших маніпуляцій треба пронумерувати комп'ютерні фірми, починаючи з одиниці; занести інформацію про місце закупівель комп'ютерів кожним з ВН в окрему множину. Відповідь на перше запитання можна отримати, виконавши перетин всіх таких множин; на друге запитання – це результат об'єднання множин; на останнє запитання – це результат різниці множини всіх фірм і множини фірм, де хоча б один ВНЗ робив покупки.

20. Маємо множину день_тижня={пон, вівт, сер, чет, п’ят, суб, нед}. Скласти програму, яка формує множину, що включає в себе: а) назви днів тижня визначеного місяця; б) назви робочих днів тижня визначеного місяця.

21. Маємо рядок символів s і множину символів {c, d}. Скласти програму, яка змінній s присвоює: а) множину голосних латинських літер {*a, e, i, o, u, y*}; б) множину усіх цифр; в) множину літер, які більше c, але менше d (c<d).

22. Задано текст з малих латинських літер. Скласти програму, яка формує список (список містить в алфавітному порядку по одній літері латинські голосні літери з множини {*a, e, i, o, u, y*}, що входять в цей текст) і виводить його.

23. Задано речення на українській мові з малих літер. Скласти програму, яка визначає і видаляє з нього три слова, що містять найбільшу кількість різних приголосних літер.

24. Задано речення. Скласти програму, яка визначає і видаляє з нього два слова, які містять найбільшу кількість різних голосних латинського алфавіту (множина {*a, e, i, o, u, y*}).

25. Задано набір цифр. Скласти програму, що визначає, яких цифр з множини {1,2,3,4,5,6,7,8,9,0} більше в цьому наборі.

26. Маємо множину імен Name={*Вася, Володя, Іра, Ліда, Марина, Міша, Наташа, Олег, Оля, Свєта, Юля*}, гості – це підмножина з Name, група ГР – це список з імен гостей. Описати логічну функцію Скрізь (ГР), яка визначає, чи є в групі ГР хоча б одна особа, що побувала в гостях у всіх інших з групи (ГР[*x*] – множина людей, що побували в гостях у людини з ім'ям *x*; $x \neq$ (ГР[*x*]).

27. Задано текст з латинських літер. Символи цього тексту формують множину. Скласти програму, яка визначає і виводить на екран: а) перші входження літер в текст, зберігаючи їх вихідний взаємний порядок; б) всі літери, які входять в текст не менше двох разів; в) всі літери, які входять в текст один раз.

28. Маємо множину натуральних чисел {1,2,3,4,5,6,7,8,9}. Описати функцію: а) *digies*(*n*), яка підраховує кількість різних (значущих) цифр в запису натурального числа *n* у десятковій с/ч; б) *print*(*n*), яка виводить в порядку зростання всі цифри, що не входять в запис натурального числа *n* у десятковій с/ч.

29. Задано текст з цифр і латинських літер. Скласти програму, що визначає, яких літер – голосних {*a, e, i, o, i, y*} або приголосних – більше в цьому тексті.

30. Маємо три множини *x, y, z* цифр {8,9,10,11, …, 21,22}. Змінній *x* присвоїли множину всіх цілих чисел від 8 до 22, змінній *y* – множину всіх простих чисел з цього діапазону, а змінній z – множину всіх складових чисел з цього ж діапазону.

31. Маємо деякий набір продуктових товарів *n*, асортимент магазину – це множина товарів з цього набору. Компанія володіє m магазинами. За інформацією щодо асортименту, наданому кожним з цих магазинів скласти програму, яка сформує такі множини: множину А продуктів, які є в усіх магазинах; В – множину продуктів, кожен з яких є хоча б в одному магазині; С – множину продуктів, яких немає в жодному магазині.

32. Скласти програму, яка реалізовує гру «бики та корови». *Опис гри*: комп'ютер загадує чотиризначне число, яке не мітить двох однакових цифр; користувач називає цифри, з яких складається це число; на кожному кроці повідомляють кількість «биків» (правильно відгаданих цифр) і «корів» (неправильно відгаданих цифр); гра триває, поки не буде набрано чотири «бика».

33. Згенерувати *n* множин цілих чисел, нумерацію (назва) множин розпочати з «1». Скласти програму, яка визначає елементи множини, які входять в усі множини з номерами, кратними трьом, але не входять у першу множину.

Контрольні запитання

1. Що таке множина?

2. Як зберігається множина в пам'яті ЕОМ? Який максимальний обсяг оперативної пам'яті можна відвести під зберігання однієї множини?

3. Які операції можна виконувати над множинами?

4. Як додати елемент в множину?

5. Як видалити елемент з множини?

6. Як вивести елементи множини? Як підрахувати кількість елементів у множині?

Аудиторна робота

Пример 9.1: перетин послідовностей у вигляді рядків.

Пошук перетину двох послідовностей у вигляді рядків.

1. Розглянемо цикл *for*, який вибирає елементи, загальні для двох рядків. Після того як цикл *for* виконано, змінна *res* буде посилатися на список, який містить всі однакові елементи, виявлені в *seq1* і *seq2*:

```
seq1 = "spam"; seq2 = "scam"
res = [] # спочатку список пустий
for x in seq1: # виконати обхід першої послідовності
... if x in seq2: # загальний елемент?
... res.append(x) # Додати в кінець результата
res
```

2. Генератор списку

```
s1 = "spam"; s2 = "scam"
[x for x in s1 if x in s2]
```

Результат:

```
['s', 'a', 'm']
```

3. Використаємо множини для визначення перетину послідовностей у вигляді рядків

```
x=set("spam"); y = set("scam")
z=x & y # Перетин множин
print(z)
```

Результат

```
{'m', 'a', 's'}
```

Пример 9.2. Множини.

1. Створення множин – екземплярів класу *set*

```
my_set={1, 5, 3, 9}  # множина цілих чисел
print(my_set)
basket={'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)
```

2. Створення множин на основі генераторів множин

```
number_set={i + j for i in range(10) for j in range(5)}
print(number_set)
char_set={x for x in 'abracadabra' if x not in 'abc'}
print(char_set)
```

3. Використання конструктору *frozenset*. Множини можуть включати об'єкти тільки незмінних типів. Якщо необхідно зберегти одну множину всередині іншої, створюють множину за допомогою вбудованої функції *frozenset*, яка діє так само, як функція *set*, але формує *незмінну множину*.

```
empty_set=set(); print(empty_set) # Пуста множина
empty_frozenset=frozenset() # Пуста незмінювана множина
print(empty_frozenset); my_frozenset=frozenset([4,1,3,8])
my_set=set(my_frozenset); print(my_set); print(my_frozenset)
```

4. Операції з множинами (*set, frozenset*)

```
my_set = {4, 5, 1, 2}
# Кількість елементів множини
print('len({}) = {}'.format(my_set, len(my_set)))
print()
# Перевірка входження елемента
print(4 in my_set); print(3 not in my_set)
print(9 in my_set);print()
# Чи перетинаються множини
print({3, 4, 5}.isdisjoint({8, 1, 0}))
print({3, 4, 5}.isdisjoint({1, 2, 3}));print()
# Перевірка включення однієї множини в іншу
print({1, 7, 9}.issubset({1, 2, 3, 7, 9}))
print({1, 7, 9} <= {1, 2, 3, 7, 9})
print({1, 7, 9, 2, 3} <= {1, 2, 3, 7, 9});print()
# Перевірка чіткого включення
print({1, 7, 9} < {1, 2, 3, 7, 9})
print({1, 7, 9, 2, 3} < {1, 2, 3, 7, 9});print()
# Перевірка включення однієї множини в іншу
print({1, 2, 3, 4}.issuperset({1, 2}))
print({1, 2, 4, 4} >= {1, 2})
print({1, 2, 3, 4} >= {1, 2, 3, 4});print()
# Перевірка чіткого включення
print({1, 2, 4, 4} > {1, 2})
print({1, 2, 3, 4} > {1, 2, 3, 4});print()
# Об'єднання множин
print({1, 3}.union({2, 3, 4}))
print({1, 3} | {2, 3, 4});print()
# Перетин множин
print({1, 3}.intersection({2, 3, 4}))
print({1, 3} & {2, 3, 4});print()
# Різниця множин
print({1, 2, 3, 4}.difference({3, 4, 5}))
print({1, 2, 3, 4} - {3, 4, 5});print()
# Симетрична різниця
print({1, 2, 3, 4}.symmetric_difference({3, 4, 5, 6}))
print({1, 2, 3, 4} ^ {3, 4, 5, 6});print()
# Копіювання множини
my_set = set('chars')
copy = my_set.copy();print(copy)
```

5. Операції над множинами, які є методами, мають в якості аргументів будь-які

ітерабельні об'єкти. Операції над множинами, записані у вигляді бінарних операцій, вимагають, щоб другий операнд операції теж був множиною, і повертають множину того типу, якою була перше множина

```
print(frozenset('abc').union(frozenset('cdef')))  # коректно
print(frozenset('abc') | frozenset('cdef'))  # коректно
print(frozenset('abc').union('cdef'))  # коректно
print(frozenset('abc') | 'cdef')  # помилка
```

6. Для обробки об'єктів-множин існують такі методи: *add* вставляє новий елемент в множину, *update* виконує об'єднання, *remove* видаляє елемент за його значенням (викличте функцію *dir*, передавши їй будь□який екземпляр типу *set*, щоб отримати повний перелік всіх доступних методів).

```
my_set = {1, 3, 5}
my_set.update({2, 3, 4})  # my_set |= {2,3,4}
print(my_set)
my_set.intersection_update({0,1,2,3,10})  # my_set &= {0,1,2,3,10}
print(my_set)
my_set.difference_update({1})  # my_set -= {1}
print(my_set)
my_set.symmetric_difference_update({3, 4})  # my_set ^= {3, 4}
print(my_set)
my_set.add(5)  # my_set |= {5}
print(my_set)
my_set.remove(2) #my_set-={2},виводить KeyError, якщо елемента
немає
print(my_set); my_set.discard(2)  # my_set -= {2}
print(my_set)
print(my_set.pop());print(my_set)
my_set.clear();print(my_set)
```

Результат

```
{1, 2, 3, 4, 5}
{1, 2, 3}
{2, 3}
{2, 4}
{2, 4, 5}
{4, 5}
{4, 5}
4
{5}
set()
```

7. Перевірка множин на рівність виконується поелементно, незалежно від типів множин

```
print({1, 2, 3} == frozenset([1, 2, 3]))
print(set('abc') == frozenset('abc'))
print(set('abc') in set([frozenset('abc')]))
```

Результат

```
True
True
True
```

8. Сформувати і вивести множину

1) символів в алфавітному порядку

```
a = set('qwertyuiopasdfghjklzxcvbnm') # формуємо множину
print (a) #виведення множини
b=list(a) # перетворюємо множину у список (її не можна сортувати)
b.sort () #сортуємо список
print (b)  #виведення
```

Результат

```
{'g', 'f', 'u', 'z', 'm', 'n', 'y', 'e', 'd', 'k', 'v', 'l', 's',
'p', 'x', 'h', 'r', 'o', 'i', 'j', 'c', 'w', 'b', 't', 'a', 'q'}

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

2) цілих чисел у зростаючому порядку з діапазону 1..100, які можна подати у вигляді n^2+m^2, де $n, m \geq 0$.

```
a={k**2+i**2 for i in range(0,8) for k in range(0,8)}
b=list(a); b.sort()
print(b)
```

Результат

```
[0, 1, 2, 4, 5, 8, 9, 10, 13, 16, 17, 18, 20, 25, 26, 29, 32, 34,
36, 37, 40, 41, 45, 49, 50, 52, 53, 58, 61, 65, 72, 74, 85, 98]
```

9. Застосування операцій над множинами, які використовують до списку людей – службовців гіпотетичної компанії:

```
>>> engineers = {‘bob’, ‘sue’, ‘ann’, ‘vic’}
>>> managers = {‘tom’, ‘sue’}
>>> ‘bob’ in engineers # bob – інженер?
True

>>> engineers & managers # хто одночасно є інженером і менеджером?
{‘sue’}
>>> engineers | managers # Всі співробітники з обох категорій
{‘vic’, ‘sue’, ‘tom’, ‘bob’, ‘ann’}
>>> engineers – managers # Інженери, які не є менеджерами
{‘vic’, ‘bob’, ‘ann’}
>>> managers – engineers # Менеджери, які не є інженерами
{‘tom’}
>>> engineers > managers # чи всі менеджериы є інженерами?
False # (надмножина)
>>> {‘bob’, ‘sue’} < engineers
# Обидва співробітника є інженерами? (підмножина)
True
>>> (managers | engineers) > managers
# Множина всіх співробітників є надмножиною менеджерів?
True >>> managers ^ engineers
# Співробітники, які належать до однієї  категорії
{‘vic’, ‘bob’, ‘ann’, ‘tom’}
>>> (managers | engineers) - (managers ^ engineers)
# Перетин!
{‘sue’}
```

Приклад 9.3. Задано два слова. Для кожної літери першого слова (в тому числі для літер, які повторюються в цьому слові) визначити, чи входить літера до складу другого слова. Наприклад, якщо задано слова «*информация*» і «*процессор*», то для літер першого із них відповідь повинна мати вигляд: *нет нет нет да да нет нет да нет нет*.

версія 1

```
import itertools
sl1=input('Введіть слово 1 = ')
sl2=input('Введіть слово 2 = ')
m=len(sl1); n=len(sl2);i=0; L=[]
while m!=i:
  fl=0;
  for j in itertools.count(start=0, step=1):
    if j>=n: break;
    if j<n:
      if sl1[i]==sl2[j]: fl=1
  if fl==1:
        L.append('да ')
  else:
        L.append('нет ')
  i+=1
a=list(L)
pp=''.join(a);print(pp)
```

версія 2

```
sl1=input('Введіть слово 1 = ')
sl2=input('Введіть слово 2 = ')
m=len(sl1)
i=0; L=[]
while m!=i:
  fl=0;
  if sl1[i] in sl2: fl=1
  if fl==1:
        L.append('да ')
  else:
        L.append('нет ')
  i+=1
a=list(L); pp=''.join(a); print(pp)
```

Результат

```
информация
процессор
нет нет нет да да нет нет да нет нет
```

Теоретическая часть

1. МНОЖИНА

Множина – невпорядкований набір унікальних і незмінних об'єктів, який підтримує операції, що відповідають математичній теорії множин. За визначенням, кожний елемент може бути присутнім в множині в одному екземплярі незалежно від того, скільки разів він буде доданий. Оскільки множина є набором інших об'єктів, їй притаманні деякі властивості списків і

словників (множини підтримують ітерації, при необхідності можуть змінюватися в розмірах і містити об'єкти різних типів).

Щоб створити об'єкт множина, потрібно передати послідовність або інший об'єкт, що підтримує можливість ітерацій за його вмістом, вбудованій функції *set*:

```
x=set('abcde'); y=set('bdxyz')
```

Функція повертає об'єкт–множину, який містить всі елементи об'єкта, переданого функції:

```
>>> x
set(['a', 'c', 'b', 'e', 'd'])
```

Множини, створені таким способом, підтримують звичайні математичні операції над множинами за допомогою операторів виразів (перетин, об'єднання, різниця, симетрична різниця множин, перевірка входження в множину, надмножина, підмножина):

```
>>> 'e' in x # Перевірка входження в множину
True
>>> x - y # Різниця множин
set(['a', 'c', 'e'])
>>> x | y # Об'єднання множин
set(['a', 'c', 'b', 'e', 'd', 'y', 'x', 'z'])
>>> x & y # Перетин множин
set(['b', 'd'])
>>> x ^ y # Симетрична різниця (XOR)
set(['a', 'c', 'e', 'y', 'x', 'z'])
>>> x > y, x < y # Надмножина, підмножина
(False, False)
>>> S1 = {1, 2, 3, 4}
>>> S1 & {1, 3} # Перетин
{1, 3}
>>> {1, 5, 3, 6} | S1 # Об'єднання
{1, 2, 3, 4, 5, 6}
>>> S1 - {1, 3, 4} # Різниця
{2}
>>> S1 > {1, 3} # Надмножина
True
```

Для множин визначені такі операції:

+ – об'єднання множин;

– – різниця множин;

* – перетин множин;

= – перевірка еквівалентності двох множин;

<> – перевірка нееквівалентності двох множин;

<= – перевірка, чи є ліва множина підмножиною правої множини;

> = – перевірка, чи є права множина підмножиною лівої множини;

in – перевірка, чи входить елемент, який зображено зліва, в множину, вказану справа.

Результатом операції об'єднання, різниці або перетину є відповідна множина, інші операції дають результат логічного типу. Нехай змінні *x* і *y* зберегли свої значення, присвоєні в попередньому прикладі:

```
>>> x=set('abcde'); y=set('bdxyz')
>>> z=x.intersection(y) # x & y
>>> z
set(['b', 'd'])
>>> z.add('SPAM') # додає один елемент
>>> z
set(['b', 'd', 'SPAM'])
>>> z.update(set(['X', 'Y'])) # об'єднання множин
>>> z
set(['Y', 'X', 'b', 'd', 'SPAM'])
>>> z.remove('b') # видалить один елемент
>>> z
set(['Y', 'X', 'd', 'SPAM'])
```

Будучи ітерованими контейнерами, множини можна передавати функції *len*, використовувати в циклах *for* і в генераторах списків. Однак множини є невпорядкованими колекціями, тому не підтримують операції над послідовностями, такі як індексування і витяг зрізу:

```
>>> for item in set('abc'): print(item * 3)
aaa
ccc
bbb
>>> S = set([1, 2, 3])
>>> S | set([3, 4]) # Оператори виразу вимагають щоб
set([1, 2, 3, 4]) # обидва операнди були множинамии
>>> S | [3, 4]
TypeError: unsupported operand type(s) for |: 'set' and
'list'
>>> S.union([3, 4])
set([1, 2, 3, 4])
>>> S.intersection((1, 3, 5))
set([1, 3])
>>> S.issubset(range(-5, 5))
True
```

Літерали множин. В Python існує можливість використовувати вбудовану функцію *set* для створення множин, при цьому є форма літералів множин, в яких використовують фігурні дужки, раніше зарезервовані для літералів словників. В Python 3.0 такі інструкції є еквівалентними:

```
>>> set([1, 2, 3, 4]) # Виклик функції set
{1, 2, 3, 4} # Літерал множини
>>> set('spam')
{'a', 'p', 's', 'm'}
>>> {1, 2, 3, 4} # Літерали множин
{1, 2, 3, 4}
>>> S = {'s', 'p', 'a', 'm'}
>>> S.add('alot')
```

```
>>> S
{'a', 'p', 's', 'm', 'alot'}
```

Конструкція {} створює пустий словник. Щоб створити пусту множину, треба викликати вбудовану функцію *set*; результат операції виведення пустої множини має інший вигляд:

```
>>> S1 - {1, 2, 3, 4} # Пуста множина
set()
>>> type({}) # Літерал {} позначає пустий словник
<class 'dict'>
>>> S = set() # Ініціалізація пустої множини
>>> S.add(1.23)
>>> S
{1.23}
>>> {1, 2, 3} | {3, 4}
{1, 2, 3, 4}
>>> {1, 2, 3} | [3, 4]
TypeError: unsupported operand type(s) for |: 'set' and 'list'
>>> {1, 2, 3}.union([3, 4])
{1, 2, 3, 4}
>>> {1, 2, 3}.union({3, 4})
{1, 2, 3, 4}
>>> {1, 2, 3}.union(set([3, 4]))
{1, 2, 3, 4}
>>> {1, 2, 3}.intersection((1, 3, 5))
{1, 3}
>>> {1, 2, 3}.issubset(range(-5, 5))
True
```

Множини можуть включати об'єкти тільки незмінних типів: списки і словники не можна додавати в множини, однак можна використати кортежі, якщо з'явиться необхідність зберігати складові значення. В операціях над множинами кортежі порівнюють за своїм повним значенням:

```
>>> S
{1.23}
>>> S.add([1, 2, 3])
# Додаватися можуть тільки незмінні об'єкти
TypeError: unhashable type: 'list'
>>> S.add({'a':1})
TypeError: unhashable type: 'dict'
>>> S.add((1, 2, 3))
>>> S # а ні  список, ні словник не можна додати
{1.23, (1, 2, 3)} # кортеж можна
>>> S | {(4, 5, 6), (1, 2, 3)}
# Об'єднання: теж, що й S.union(...)
{1.23, (4, 5, 6), (1, 2, 3)}
>>> (1, 2, 3) in S
True
>>> (1, 4, 3) in S
False
```

Генератори множин. Конструкцію генератора множин розміщують у фігурні дужки. Генератор множин виконує цикл і збирає результати виразу в кожній ітерації - доступ до значення в поточній ітерації забезпечує змінна циклу. Результатом роботи генератора є нова множина:

```
>>> {x**2 for x in [1, 2, 3, 4]} # Генератор множин
{16, 1, 4, 9}
```

Генератор множин повертає «нову множину, яка містить квадрати значень X, для кожного X зі списку». В генераторах можна також використовувати інші види ітерованих об'єктів, наприклад рядки:

```
>>> {x for x in 'spam'} # Те ж саме, що і: set('spam')
{'a', 'p', 's', 'm'}
>>> {c * 4 for c in 'spam'}
# Множина результатів виразу
{'ssss', 'aaaa', 'pppp', 'mmmm'}
>>> {c * 4 for c in 'spamham'}
{'ssss', 'aaaa', 'hhhh', 'pppp', 'mmmm'}
>>> S = {c * 4 for c in 'spam'}
>>> S | {'mmmm', 'xxxx'}
{'ssss', 'aaaa', 'pppp', 'mmmm', 'xxxx'}
>>> S & {'mmmm', 'xxxx'}
{'mmmm'}
```

Оскільки елементи множин є унікальними, їх можна використовувати для фільтрації повторюваних значень в інших наборах. Для цього досить перетворити набір значень у множину, а потім виконати зворотне перетворення (множина є ітерованим об'єктом, тому її можна передавати функції *list*):

```
>>> L = [1, 2, 1, 3, 2, 4, 5]
>>> set(L)
{1, 2, 3, 4, 5}
>>> L = list(set(L)) # видалення значень, які  повторюються
>>> L
[1, 2, 3, 4, 5]
```

Комп'ютерний практикум № 10. Функції користувача. Наближене обчислення функцій

Мета роботи: ознайомитися з принципами побудови функцій користувача на мові Python, з використанням локальних і глобальних змінних. *Об'єкт дослідження* – функції користувача (основні складові функцій, оголошення та опис функцій), оператор форматування %.

ПЛАН

1. Інструкції, які використовують при створенні функцій

Завдання

1. Вивчити теоретичні основи написання алгоритмів з використанням функцій користувача. Опрацювати приклади.
2. Побудувати блок-схему алгоритму вирішення завдання.
3. Відповідно до свого варіанту
- визначити умови;
- за допомогою формул описати варіанти виконання необхідний дій;
- написати програму, яка розв'язує завдання, реалізуючи обробку помилок (виключень);
- організувати введення даних з клавіатури, виведення у консоль;
- реалізувати різні режими співставлення аргументів: a) простий режим співставлення аргументів відповідно до позиції, b) передачу довільної кількості аргументів;
- під час виведення результату використати оператор форматування %.
4. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасировки виконання програми.

Постановка завдання. Описати функцію обчислення значень функції (визначити *m* значень заданої функції *f*(*x*) на відрізку [*a*, *b*]). Результат вивести на екран у вигляді табл. 10.1. Значення *a*, *b*, є ввести з клавіатури.

Таблиця 10.1

x_i	$\overline{f}(x_i)$	$f(x_i)$ - точне значення функції	Точність	Кількість ітерацій
$x_0 = a$				
$x_{m-1} = b$				

Стовпчики таблиці: 1 – значення x_i; 2 – значення функції $f(x_i)$, яке обчислено з використанням функцій інтерпретатора (модуль *math*); 3 – значення функції $\overline{f}(x_i)$, яке обчислено за допомогою розкладання в ряд із

точністю $\varepsilon=10^{-5}$; 4 – точність обчислень – значення $|\overline{f}(x_i)-f(x_i)|$; 5 – кількість ітерацій, необхідних для досягнення заданої точності.

Обмеження.

1. Масиви не використовувати.

2. Не застосовувати факторіали для членів ряду в явному вигляді, а використовувати тільки для обчислення точного значення функції в модулі *math*.

3. Кількість точок на відрізку [*a*, *b*] складає не менше 10 (розбиття може бути рівномірним або за визначеним правилом).

4. Для тригонометричних функцій в програмі привести значення аргумента до вигляду $0 \le x \le \pi/2$.

Коментар:

1. Для рівняння $y=\sqrt[n]{x}$ ($x>0$, $n>0$) необхідно використати рекурентну формулу Ньютона: $y_{k+1}=\frac{1}{k}\left[(k-1)y_k+\frac{x}{y_k^{k-1}}\right]$, $k=0, 1, \ldots$; $y_0=\frac{x+n-1}{2}$, яка має місце для $y_0>0$. Необхідну точність оцінюють співвідношенням $|y_{n+1}-y_n|\le\varepsilon$.

2. Використати табл. 10.2 для побудови розкладання елементарних функцій в ряд Тейлора.

Таблиця 10.2

Довідкова інформація

Розкладання елементарних функцій в ряд Тейлора	Функція
1	2
$x-\frac{x^3}{3!}+\frac{x^5}{5!}-\ldots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}+\ldots$	$\sin(x)$, $x\in\Re$
$1+\frac{x}{1!}+\frac{x^2}{2!}+\ldots+\frac{x^n}{n!}+\ldots$	e^x, $x\in\Re$
$x-\frac{x^2}{2!}+\frac{x^4}{4!}-\ldots+(-1)^n\frac{x^{2n}}{(2n)!}+\ldots$	$\cos(x)$, $x\in\Re$
$x+\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots+\frac{x^{2n-1}}{(2n-1)!}+\ldots$	$\mathrm{sh}(x)$, $x\in\Re$
$x+\frac{x^2}{2!}+\frac{x^4}{4!}+\ldots+\frac{x^{2n}}{(2n)!}+\ldots$	$\mathrm{ch}(x)$, $x\in\Re$
$x-\frac{x^2}{2}+\frac{x^3}{3}+\ldots+(-1)^{n-1}\frac{x^n}{n}+\ldots$	$\ln(1+x)$, $x\in(-1; 1]$
$x-\frac{x^3}{3}+\frac{x^5}{5}-\ldots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)}+\ldots$	$\mathrm{arctg}(x)$, $x\in[-1; 1]$
$-x-\frac{x^2}{2}-\frac{x^3}{3}-\ldots-\frac{x^n}{n}-\ldots$	$\ln(1-x)$, $x\in(-1; 1)$
$1-x+x^2-\ldots+(-1)^n x^n+\ldots$	$\frac{1}{1+x}$, $x\in(-1; 1)$

1	2
$1+x+x^2+\ldots+x^n+\ldots$	$\frac{1}{1-x}$, $x\in(-1;\,1)$
$1+\alpha x+\frac{\alpha(\alpha-1)}{2!}x^2+\ldots+\frac{\alpha(\alpha-1)\ldots(\alpha-n+1)}{n!}x^n+\ldots$	$(1+x)^{\alpha}$, $x\in(-1;\,1)$
$1-\frac{1}{2}x+\frac{3}{8}x^2-\frac{5}{16}x^3+\ldots+(-1)^{n-1}\frac{(2n-1)!!x^n}{(2n)!!}+\ldots,$	$\frac{1}{\sqrt{1+x}}$, $x\in(-1;\,1)$
$e^{2x}=1+2x+\frac{(2x)^2}{2!}+\frac{(2x)^3}{3!}+\ldots=\sum_{n=0}^{\infty}\frac{(2x)^n}{n!}$	e^{2x}
$e^{x^2}=1+x^2+\frac{x^4}{2!}+\frac{x^6}{3!}+\ldots=\sum_{n=0}^{\infty}\frac{x^{2n}}{n!}$	e^{x^2}
$1+(1+x)+\frac{(1+x)^2}{2!}+\frac{(1+x)^3}{3!}+\ldots=\sum_{n=0}^{\infty}\frac{(1+x)^n}{n!}$	$e^{(1+x)}$
$x\sin x=\frac{1}{1!}x^2-\frac{1}{3!}x^4+\frac{1}{5!}x^6-\frac{1}{7!}x^8+\ldots=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+2}$	x*sin(x)
$1+3x+3x^2+x^2$	$(1+x)^3$
$x^{1/2}+x^{5/2}+x^{9/2}+\ldots$	$\frac{\sqrt{x}}{1-x^2}$

Варіанти

№	$f(x)$	$[a,\ b]$
1	2	3
1	$e^{2x}\sqrt{x}$	1; 3
2	$\sqrt[3]{x}$	3; 4
3	e^{x^2}	4; 5
4	$1/\sqrt[3]{x}$	5; 7
5	$1/\sqrt{x}$	6; 8
6	$(2x+1)/\sqrt{x}$	8; 9
7	$\sqrt[4]{x}$	0.5; 1
8	$e^{(1+x)}$	2.5; 3
9	$\sqrt[5]{x}\cos(x)$	1; 1.5
10	$\ln(1+x)$	1; 1.5
11	$1+\ln^2(1-x)$	0.4; 1
12	$1+e^{-x}$	0.5; 1
13	$(e^x/2)\cdot 1/\sqrt{x}$	2; 3
14	$\cos(x)\cdot e^{2x}$	1; 2
15	$1/(1+e^{(1+x)})$	3; 4

1	2	3
16	$(2+x)/\sqrt[3]{x}$	5; 7
17	$x\sin(x)$	0; π
18	$(1+x)^3$	2; 3
19	$2+\sqrt[4]{x}$	2; 3
20	$\frac{1}{\sqrt{1+x}}$	-0.1; 1
21	$\ln\frac{1+x}{1-x}=\frac{\ln(1+x)}{\ln(1-x)}$	0.1; 0.9
22	$\frac{x}{1+x}$	-1; 1
23	$e^x\sin(x)$	0; π
24	$\frac{\sqrt[3]{x}}{1+x}$	0.1; 0.9
25	$\frac{\sqrt{x}}{1-x^2}$, $(1-x^2)=(1+x)\cdot(1-x)$	0.1; 2.9
26	$\sqrt[4]{x}\cdot\ln\frac{1+x}{1-x}$	0.4; 0.9
27	$\ln(1+x)\cdot\frac{1}{1+x}$	-1; 1
28	$\ln(1-x)\cdot\frac{1}{\sqrt{1+x}}$	1; 1.5

Контрольні запитання

1. Що таке процедура та функція? Навіщо їх використовують?
2. Охарактеризуйте локальні та глобальні змінні. В чому їх відмінність?
3. Навіщо використовувати параметри під час визначення функції? Які типи параметрів існують?

Аудиторна робота

Приклад 10.1.

1. Об’ява функції hello_world

```
def hello_world():
    print('Hello, World!')
# Виклик  функції
hello_world()
```

2. Формальний параметр функції *print_numbers – limit*

```
def print_numbers(limit):
    for i in range(limit):  print(i)
# Виклик функції  print_numbers,  її формальний
# параметр limit замінюють  фактичним параметром 10
print_numbers(10)
```

```
3)def print_numbers(limit):
    for i in range(limit):  print(i)
# Виклик  функції з фактичним параметром n
# print_numbers
n = int(input('Введіть n: '))
print_numbers(n)

4)def print_numbers(limit):
    for i in range(limit):  print(i)
def main():
    n=int(input('Введіть n: ')); print_numbers(n)
# Виклик функції
main()

5)def add_numbers(a, b):
    return a + b  # вихід функції - сума параметрів
# Виклик функції
x = add_numbers(2, 3);print(x)

6)def procedure():
    print('I return nothing... Or I do?')
# Виклик функції
value = procedure(); print('Результат функції:', value)
```

7. Функції можна передавати будь-яку кількість позиційних та іменованих аргументів. Для цього перед ім'ям заданого словника в списку формальних параметрів ставиться два символу **

```
def function(**kwargs):
    print(kwargs)

function (arg1='value1', arg2='value2')
# Можна розпакувати відображення
# в іменовані параметри при виклику функції
options = {
    'sep': ', ',
    'end': ';\n'}
print('value1', 'value2', **options)
```

Результат

```
{'arg2': 'value2', 'arg1': 'value1'}
value1, value2;
```

8. Ця функція повертає аргумент, помножений на два, якщо він – від'ємний, або аргумент, помножений на три, якщо він – більше або дорівнює нулю

```
def function(x):
    if x < 0: return x * 2
    else: return x * 3
def main():
    # Виведення значень функції з діапазону [-3, 3]
    for i in range(-3, 4):
        y = function(i)
```

```
        print('function(', i, ') = ', y, sep='')
# Виклик функції
if __name__ == '__main__':  main()

9)def hello(name):
    # Якщо ім’я - пусте, виходимо з функції
    if not name: return
    print('Hello, ', name, '!', sep='')
# Виклик функції
hello('Alex');hello('');hello('Python')

10)def add(a, b):
    return a + b
def sub(a, b):
    return a - b
# Виклик функції
# може бути частиною виразу
print(add(2, 3) + sub(2, 3))  # => print((2 + 3) + (2 - 3))
```

11. Функція, яка має три аргументи

```
def info(object, color, price):
   print('Объект:', object)
   print('Цвет:', color); print('Цена:', price)
   print()
# Виклик  функції, передача параметрів в прямому порядку
info('ручка', 'синий', 1)
# передача параметрів у довільному порядку
info(price=5, object='чашка', color='оранжевый')
# можна змішувати обидва способи, але спочатку
# розташовують параметри,
# які  передають в прямому порядку
info('кофе', price=10, color='чёрный')
```

12. Якщо параметр *name* не задано, то *name* = 'Alex'

```
def hello(name='Alex'):
    print('Hello, ', name, '!', sep='')
# Виклик функції
hello('Python'); hello()

13) def outer_function():
    # об’ява функції локально
    def inner_function():
        print('Внутрішня функція')
    print('Зовнішня функція')
    inner_function()    # виклик функції

# виклик зовнішньої функції
outer_function()
# inner_function()  # помилка, тут ця функція не має доступу
```

Результат

```
Зовнішня функція
Внутрішня функція
```

```
14. Отримання доступу до глобальної змінної
def function():
    print(var)

var = 'глобальна змінна'; function()
```
Результат
```
Глобальна змінна
```

15. Локальна змінна
```
def function():
    # визначення локальної змінної
    var = 'локальна змінна'
    # виведення значення локальної змінної на екран
    print(var)

# визначення глобальної змінної
var = 'глобальна змінна'
function()
# виведення на екран значення глобальної змінної
print(var)
```
Результат
```
локальна змінна
глобальна змінна
```

16. *global* вказує на необхідність отримати доступ до глобальної змінної *var*, а не створювати нову локальну під час спроби що-небудь їй присвоїти
```
def function():
        global var
    # виведення значення глобальної змінної на екран
    print(var)
    # зміна глобальної змінної
    var = 'нове значення'
    # виведення значення глобальної змінної на екран
    print(var)

var = 'глобальна змінна'
function(); print(var)
```
Результат
```
глобальна змінна
нове значення
нове значення
```

```
17)def function(c, d):
    # a, b – глобальні змінні; c, d -- локальні
    global a, b
    # зміна значення глобальної змінної
    a = 5; b = 7
    # зміна значення локальної змінної
    c = 10; d = 12
```

```
a, b, c, d = 1, 2, 3, 4  # множинне присвоєння
print(a, b, c, d)  # 1 2 3 4
function(c, d); print(a, b, c, d)  # 5 7 3 4
```

Результат

```
1 2 3 4
5 7 3 4
```

```
18)def outer_function():
    var = 8  # створення локальної змінної var
    def inner_function():
      # вказує на необхідність використання змінної із
зовнішньої функції
      nonlocal var
      print(var)  # 8
      var=10

    print(var)  # 8
    inner_function()  # виклик внутрішньої функції
    print(var)  # 10

var = 0 # створення глобальної змінної var
print(var)  # 0
outer_function()
print(var)  # 0
```

Результат

```
0
8
8
10
0
```

```
19)def outer_function():
    var = 8  # створення локальної змінної var
    def inner_function():
     # вказує на необхідність використання глобальної змінної
      global var
      print(var)  # 0
      var = 10

    print(var)  # 8
    inner_function()  # виклик внутрішньої функції
    print(var)  # 8

# створення глобальної змінної var
var = 0; print(var)  # 0
outer_function()
print(var)  # 10
```

Результат

```
0
8
0
8
```

```
10
```

20. Оператор форматування %.

```
# дійсні  числа
>>> '%e, %.3e, %g' % (3.14159, 3.14159, 3.14159)
'3.141590e+00, 3.142e+00, 3.14159'
>>> '%f, %.2f, %06.2f' % (3.14159, 3.14159, 3.14159)
'3.141590, 3.14, 003.14'
>>> '{0:d}'.format(999999999999)
'999999999999'
>>> '{0:,d}'.format(999999999999)
'999,999,999,999'
```

Приклад 10.2. Описати функцію, яка обчислює множину значень $y=\sqrt[n]{x}$ (x>0, n>0), для цього використати рекурентну формулу Ньютона: $y_{k+1}=\frac{1}{k}\left[(k-1)y_k+\frac{x}{y_k^{k-1}}\right]$, k=0, 1, …; $y_0=\frac{x+n-1}{2}$, яка має місце для y_0>0. Необхідну точність оцінюють співвідношенням $|y_{n+1}-y_n|\le\varepsilon$.

Розв'язання

```
def my_sqrt(a,n,Epsilon):
 term1=(a+n-1)/2;
 term2=(2/3)*term1 +(a/(3*term1**2));
 while (abs(term2- term1) > Epsilon):
  term1= term2; term2=((n-1)/n)* term1 + a/(n*(term1**(n-1)));
 return term2
# main ()
import itertools
import math
print("  x  ", ": ","  y   ", " : ","  yt  ", " : ","  error  "  )
print("-------------------------------------------------------")
Epsilon=0.0001
a=0.1;b=1;
for i in itertools.count(start=a, step=0.2):
    if i>b: break
    n=3
    y= my_sqrt(i,n,Epsilon)
    y1=i**(1/n) # точное значение функции
    error=abs(y-y1)
    print('%.2f' %i, " : ", '%.4f' %y, " : ", '%.4f' %y1, " : ",
'%.4f' %error)
```

Результат

```
  x   :    y     :    yt    :    error
-----------------------------------------
0.10  :  0.4642  :  0.4642  :  0.0000
0.30  :  0.6694  :  0.6694  :  0.0000
0.50  :  0.7937  :  0.7937  :  0.0000
0.70  :  0.8879  :  0.8879  :  0.0000
0.90  :  0.9655  :  0.9655  :  0.0000
```

Приклад 10.3. Описати функцію визначення множини значень $e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+..+\frac{x^n}{n!}+...$ з точністю ε на відрізку (0, 1).

Розв'язання. Функция ExpUser(*x, Epsilon*) повертає значення e^x при фіксованому значенні *x*, обчислене за допомогою розкладання в ряд Тейлора з точністю Epsilon.

Версія 1:

a) простий режим співставлення аргументів відповідно до позиції

```
def ExpUser(x, Epsilon):
 s=1; term=1; i=0;
 while (abs(term) > Epsilon):
  term=term*(x/(i+1))
  s = s + term; i+=1
 return s
# main ()
import itertools
import math
print("  x  ", ": ","  y   ", " : ","  yt  ", " : ","  error  "  )
print("-----------------------------------------------------------")
Epsilon=0.01;
a=0;b=1;
for i in itertools.count(start=a, step=0.3):
    if i>b: break
    y=ExpUser(i, Epsilon)
    y1=math.exp(i) # точное значение функции
    error=abs(y-y1)
    print('%.2f' %i, " : ", '%.4f' %y, " : ", '%.4f' %y1, " : ",
'%.4f' %error)
```

Версія 2:

b) передача довільної кількості аргументів;

```
def expUser(eps,*args):
 #можна використати def expUser(eps,args)
 xx=args[0]; n=len(xx)
 j=0;y=[]
 while (j<n):
  x=xx[j]; term=1.0; s=1.0; i=0
  while (abs(term)>eps):
    term=(term*(x/(i+1))); s=s+term; i+=1
  j+=1; y.append(s)
 return y
# main ()
import itertools
import math
print("  x  ", ": ","  y   ", " : ","  yt  ", " : ","  error  "  )
print("-----------------------------------------------------------")
eps=0.0001
x=[];y1=[]
a=0;b=1;
for i in itertools.count(start=a, step=0.3):
  if i>b: break
  x.append(i)
  y1.append(math.exp(i)) # точное значение функции
n=len(x); y=expUser(eps,x)
```

```
for i in range(n):
    print('%.4f' %x[i], " : ", '%.4f' %y[i], " : ", '%.4f' %y1[i],
" : ", '%.4f' %abs(y[i]-y1[i]))
```

Приклад 10.4. Описати функцію *adder*, яка обчислює суму довільної кількості аргументів. Організуйте її виклик таким чином, щоб функції було передано більше або менше двох аргументів. Який тип має отримане значення суми? Що відбудеться, якщо функції передати аргументи різних типів?

Підказка: зріз, наприклад S[:0], повертає пусту послідовність того ж типу, що й S, а за допомогою внутрішньої функції *type* можні дізнатися тип об'єкту.

Розв'язання. У файлі *adder.py* показано два варіанти реалізації функції *adder* (складність тут полягає в ініціалізації значення суми пустим значенням в залежності від типу аргументів, які передаються).

Перший розв'язок з'ясовує, чи є перший аргумент цілим числом, і дістає порожній зріз першого аргументу (передбачено, що він є послідовністю), якщо він не є цілим числом.

```
def adder1(*args):
 print("adder1", end=" ")
 if type(args[0]) == type(0): # Ціле число?
  sum = 0 # ініціалізація нулем
 else: # інакше - послідовність:
  sum = args[0][:0] # використати пустий зріз першого аргументу
 for arg in args:
  sum = sum + arg
 return sum
```

Другий розв'язок. В якості початкового значення використовують перший аргумент, а потім виконують сканування аргументів, починаючи з другого.

```
def adder2(*args):
 print("adder2", end=" ")
 sum = args[0] # ініціалізувати значенням першого аргументу
 for next in args[1:]:
  sum += next # додати аргументи 2..N
 return sum
# main ()
for func in (adder1, adder2):
 print(func(2, 3, 4))
 print(func("spam", "eggs", "toast"))
 print(func(["a", "b"], ["c", "d"], ["e", "f"]))
```

Результат

```
adder1 9
adder1 spameggstoast
adder1 ['a', 'b', 'c', 'd', 'e', 'f']
adder2 9
adder2 spameggstoast
adder2 ['a', 'b', 'c', 'd', 'e', 'f']
```

В обох варіантах передбачено, що всі аргументи належать до одного типу.

Теоретична частина

1. Інструкції, які використовують при створенні функцій

Функція – це засіб, який дозволяє групувати набори інструкцій таким чином, що в програмі вони можуть запускатися неодноразово. Функції – це 1) програмні структури, які забезпечують багаторазове використання програмного коду і зменшують його надлишковість; 2) засіб проектування, який дозволяє розбити складну систему на прості і легко керовані частини; 3) засіб структурування програми; 4) можуть обчислювати деякий результат і дозволяють вказувати вхідні параметри, які різняться за своїми значеннями від виклику до виклику; 5) забезпечують можливість розбити складну систему на частини, кожна з яких грає визначену роль.

У табл. 10.3 наводяться основні інструменти, які мають відношення до функцій. В процесі розробки функції дозволяють мінімізувати надлишковість програмного коду.

Таблиця 10.3

Інструкції та вирази, які мають відношення до функцій

Інструкція	Приклад
Виклик	myfunc('spam', 'eggs', meat=ham)
def, return	def adder(a, b=1, *c): return a+b+c[0]
global	def changer(): global x; x = 'new'
nonlocal	def changer(): nonlocal x; x = 'new'
yield	def squares(x): for i in range(x): yield i ** 2
lambda	funcs = [lambda x: x**2, lambda x: x*3]

Інструкція *def* створює об'єкт функції та зв'язує його з ім'ям. У загальному вигляді інструкція має такий формат:

```
def <name>(arg1, arg2,... , argN):
     <statements>
```

Інструкція *def* складається з рядка заголовка і слідуючого за ним блоку інструкцій, зазвичай з відступами (або простої інструкції слідом за двокрапкою). Блок інструкцій утворює тіло функції, тобто програмний код, який виконується інтерпретатором щоразу, коли здійснюється виклик функції. У рядку заголовка інструкції *def* визначаються ім'я функції, з яким буде пов'язаний об'єкт функції, і список з нуля або більше аргументів (іноді їх називають параметрами) в круглих дужках. Імена аргументів в рядку заголовка будуть пов'язані з об'єктами, переданими в функцію, в точці виклику.

Тіло функції часто містить інструкцію *return*:

```
def <name>(arg1, arg2,... argN):
 ...
 return <value>
```

Інструкцію *return* можна розташувати в будь-якому місці в тілі функції – вона завершує роботу функції і передає результат програмі, яка її викликає. Інструкція *return* містить об'єктний вираз, який надає результат функції. Інструкція *return* є необов'язковою – якщо вона відсутня, робота функції завершується, коли потік управління досягає кінця тіла функції. Функція без інструкції *return* повертає об'єкт None, проте це значення зазвичай ігнорується.

Визначення функції відбувається під час виконання, тому в іменах функцій немає нічого особливого. Важливим є тільки об’єкт, на який посилається ім’я:

```
othername = func # Зв’язування об’єкта функції з ім’ям
othername() # Виклик функції
```

В цьому фрагменті функція зв’язана з іншим ім’ям і викликана з використанням нового імені. Функції – це звичайні об’єкти; їх явно записують в пам’ять під час виконання програми. Крім підтримки можливості виклику, функції дозволяють приєднати будь-які *атрибути*, які можуть зберігати інформацію для наступного використання:

```
def func(): ... # Створюють об’єкт функції
func() # Виклик об’єкту
func.attr = value # Приєднання атрибуту до об’єкту
```

Приклад 10.5. Функції мають *визначення* (інструкція *def*, яка створює функцію) і *виклик* (вираз, який зобов'язує інтерпретатору виконати тіло функції).

Визначення. Нижче наведено фрагмент коду, в якому визначається функція з ім'ям *times*, ця функція повертає результат обробки двох аргументів:

```
>>> def times(x, y): # Створити функцію і зв’язати її з ім’ям
... return x*y # Тіло, яке виконують під час виклику функції
...
```

Виклик. Після виконання інструкції *def* можна виконати виклик відповідної функції, додавши круглі дужки після її імені. В круглих дужках можна вказати один або більше аргументів, значення яких будуть присвоєні іменам, зазначеним в заголовку функції:

```
>>> times(2, 4) # аргументи в круглих дужках
8
```

Цей вираз передає функції *times* два аргументи. Передача аргументів здійснюється за рахунок виконання операції присвоєння: імені *x* в заголовку функції присвоюють значення «2», а імені *y* – значення «4», після чого запускають тіло функції. В нашому випадку тіло функції становить одна інструкція *return*, яка відправляє назад результат виразу: об'єкт «8» ($2 \cdot 4 = 8$). Цей результат можна присвоїти змінній, наприклад:

```
>>> x = times(3.14, 4);
>>> x
12.56
```

Комп'ютерний практикум № 11. Рекурсивні функції. Документування коду.

Мета роботи: ознайомитися з організацією рекурсивних функцій користувача, освоїти методики визначення та практичного застосування рекурсивних функцій у програмах, написаних мовою Python. *Об'єкт дослідження* – рекурсивні функції (ефективний засіб реалізації циклічних алгоритмів), документування коду.

ПЛАН

1. Рекурсивні функції.
2. Коментар #. Функції *dir* і *help*
3. Рядки документації __doc__
4. Функції *lambda*

Завдання

1. Вивчити теоретичні основи написання алгоритмів з використанням функцій користувача. Опрацювати приклади.
2. Відповідно до свого варіанту

- визначити умови;
- за допомогою формул описати варіанти виконання необхідний дій;
- написати програму, яка розв'язує завдання, реалізуючи обробку помилок (виключень);
- організувати введення даних з клавіатури, виведення у консоль;
- реалізувати

а) рядки документації до функції, де розмістити таку інформацію:

Модуль. Для чого призначена програма.
Методи. Призначення та опис вхідних, вихідних даних.

б) такі режими (приклад 11.2.а, б, в): код з використанням

- циклу;
- функцій;
- lambda-виразу (по можливості)

3. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних.

Варіанти

1. Описати рекурсивну функцію *root (f, a, b, eps),* яка методом ділення відрізка навпіл знаходить з точністю *eps* корінь рівняння *f(x)=0* на відрізку [*a, b*] (вважати, що *eps>0, a<b, f(a)*f(b)<0* і *f*(*x*) – неперервна і монотонна функція на відрізку [*a,b*].)

2. Описать рекурсивну логічну функцію *симм(s,i,j),* яка перевіряє, чи є симетричною частина рядка *s,* що розпочинається *i*-м і закінчується *j*-м її елементами.

3. Описати функцію, яка обчислює рекурсивним методом добуток двох цілих додатних чисел $f(a,b)=ab$ за таким алгоритмом:

$f(a,b)=a+f(a,b-1)$, якщо b – непарне,

$f(a,b)=2f(a,b/2)$, якщо b – парне,

$f(a,b)=0$, якщо $b=0$.

4. Нехай задано цілі числа m, n: $0 \le m \le n$. Описати рекурсивну функцію $C(m,n)$ обчислення біноміального коефіцієнта C^m_n за такою формулою:

$$C_n^0 = C_n^n = 1;\ C_n^m = C_{n-1}^m + C_{n-1}^{m-1}.$$

5. Описати рекурсивну функцію визначення цифрового кореня натурального числа.

Підказка. Цифровой корень заданного числа получается таким образом: если сложить все цифры этого числа, затем все цифры найденной суммы и повторять этот процесс, то в результате будет получено однозначное число (цифра), которая и называется *цифровым корнем* данного числа.

6. Задано перший член і різницю арифметичної прогресії. Описати рекурсивну функцію визначення: а) n-го члена прогресії; б) суми n перших членів прогресії.

7. Задано перший член і знаменник геометричної прогресії. Описати рекурсивну функцію визначення: а) n-го члена прогресії; б) суми n перших членів прогресії.

8. Описати рекурсивну функцію, яка обчислює суму двох цілих невід'ємних чисел шляхом багаторазового додавання числа 1, наприклад:

6 + 10 = (6 + 1) + 9 = (7 + 1) + 8 = ...

9. Описати рекурсивну функцію, яка обчислює добуток двох чисел, використовуючи тільки операцію додавання: $x\times0=0$, $x\times y= x+ x(y-1)$, якщо $y>0$.

10. Описати рекурсивну функцію обчислення максимального елемента масива із n елементів.

11. Описати рекурсивну функцію визначення розбиття цілих чисел.

Підказка. Розбиттям цілого числа називають способи його зображення у вигляді суми цілих чисел. Наприклад, розбиттям числа 4 є:

4, 3+1, 2+2, 2+1+1, 1+1+1+1.

12. Описати рекурсивну функцію друку всіх перестановок з n символів.

13. Описати функцію, яка обчислює рекурсивним методом значення математичної залежності: $y(n) = \sqrt{n+\sqrt{n-1+...+\sqrt{1}}}$

14. Описати рекурсивну функцію обчислення значення *функції Аккермана* для невід'ємних цілих чисел n і m. Функція Аккермана має вигляд:

$$A(n,m)=\begin{cases} m+1, & \text{якщо } n=0, \\ A(n-1,1), & \text{якщо } n\neq 0, \ m \\ A(n-1,A(n,m-1)), & \text{якщо } n>0, \ m \end{cases}$$

Функцію Аккермана називають двічі рекурсивною: сама функція і один з її аргументів визначені через самих себе.

Визначити значення функції Аккермана для n=1, m =3.

15. Описати функцію обчислення рекурсивним методом значення математичної залежності:

$$y(n)=\cfrac{1}{n+\cfrac{1}{n-1+\cfrac{1}{n-2+\cfrac{1}{\ldots}}}}$$

$$\ldots+\cfrac{1}{1+\cfrac{1}{2}}$$

16. Описати рекурсивну функцію визначення числа, утвореного із заданого натурального числа шляхом запису його цифр у зворотному порядку. Наприклад, для числа 1234 необхідно отримати відповідь 4321.

17. Описати функцію обчислення рекурсивним методом значення математичної залежності, заданої рекурентною формулою для довільного числа параметрів. Значення полінома за схемою багаточлена Лагерра порядку k+1 має вигляд:

$L_0(x)=1$; $L_1(x)=1-x$; $L_2(x)=\frac{1}{2}\left(x^2-4x+2\right)$;

$$L_{k+1}(x)=\frac{1}{k+1}\left((2k+1-x)L_k(x)-kL_{k-1}(x)\right)$$

18. Описати функцію обчислення рекурсивним методом значення математичної залежності $y(n)=\sqrt{1+\sqrt{2+\ldots+\sqrt{n}}}$

19. Описати рекурсивну функцію переведення натурального числа із десяткової системи числення в двійкову.

20. Описати рекурсивну функцію, яка визначає, чи є задане натуральне число простим.

Підказка. Простим називають натуральне число, яке більше за «1» та не має інших дільників, окрім одиниці та самого себе.

21. Описати рекурсивну функцію обчислення індексу максимального элемента масиву з n елементів.

22. Описати рекурсивну функцію переведення натурального числа з десяткової системи числення в 8-ову.

23. Описати функцію обчислення рекурсивним методом значення математичної залежності, заданої рекурентною формулою для довільного значення параметрів. Значення полінома за схемою багаточлена Ерміта порядку n має вигляд:

$H_0(x)=1$; $H_1(x)=2x$; $H_2(x)=4x^2-2$; $H_2(x)=8x^3-12x$; …,

$H_n(x)=2x\ H_{n-1}(x)-2(n-1))\cdot H_{n-2}(x))$, $n=2,3,\ldots$;

24. Описати рекурсивну функцію підсумовування елементів масиву, який містить n дійсних чисел ($n \le 20$).

25. Описати рекурсивну функцію з одним аргументом, яка без використання операторів циклу визначає, чи є одновимірний цілочисловий масив симетричним

26. Описати функцію обчислення рекурсивним методом значення математичної залежності:

$$y(n)=\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{3+\cfrac{1}{\ldots}}}}$$

$$\ldots+\cfrac{1}{(n-1)+\cfrac{1}{n}}$$

27. Описати функцію обчислення рекурсивним методом кількості цифр в заданому числі.

28. Описати функцію обчислення рекурсивним методом значення математичної залежності, заданої рекурентною формулою для довільного значення параметрів. Значення полінома за схемою багаточлена Чебишева порядку *n* має вигляд:

$T_0(x)=1$; $T_1(x)=x$;

$T_n(x)=2x\,T_{n-1}(x) - T_{n-2}(x)$, $n=2, 3, \ldots$;

29. Задано квадратну матрицю А розміру $n\times n$, де n – натуральне число. Описати функцію обчислення рекурсивним методом значення її визначника розкладанням за першим рядком.

30. Описати рекурсивну функцію сортування за зростанням масиву з *n* цілих чисел. *Ідея методу*: помістити найменший елемент на першу позицію, після чого відсортувати частину масиву, що залишилася, за допомогою рекурсивного виклику.

30. Задано одновимірний масив. Описати рекурсивну функцію пошуку мінімального елемента масиву з *n* дійсних чисел, де *n* – натуральне число. Функція не повинна мати явно організований цикл.

Контрольні запитання

1. Що таке рекурсія?

2. Як описують рекурсивну функцію?

3. Чи може бути нескінченним рекурсивний процес? Як повинна бути оформлена рекурсивна функція, щоб кількість рекурсивних викликів було кінцевим?

4. Перелічіть переваги та недоліки звичайних і рекурсивних функцій.

Аудиторна робота

Приклад 11.1: рядки документації

```
1."""  В цьому прикладі розглянуто  документаційні рядки """
     def function():
```

```
        """ Рядок, який стоїть на самому початку функції (модуля,
класу або методу),
        відіграє роль особливого виду коментарів - рядку
документації (docstring). """
        print('function called\n\n')

    function()# Виклик функції
    # Виведення рядка документації на екран.
    # Після імені функції немає круглих дужок, тому вона не
викликається,
    # а сама розглядається як певний об'єкт
    print(function.__doc__)

    # Функцію help використовують для виведення довідки
    help(function)  # виклик довідки для визначеної користувачем
функції
    help (print) # виклик довідки для стандартної функції
```

2. Приклад документування коду

1) формування рядків документації у файлі *my_mod.py*

```
1. """ It is the program for definition of exponential function at point
x
2. Method ExpUser accepts twoo arguments (x, Epsilon).
4. The first  parameter - the point x,
5. the second parameter - the accuracy Epsilon
6. the output s – the definition of exponential function at point x """
7.def ExpUser(x, Epsilon):
8.  s=1; term=1; i=0;
9.  while (abs(term) > Epsilon):
10.   term=term*(x/(i+1))
11.   s = s + term; i+=1
12.return s
```

2) використовуючи метод __doc__, викличте рядки документації для різних об'єктів модуля; попередньо необхідно імпортувати модуль (щоб імпортувати його без проблем, помістіть файл *my_mod.py* в каталог установки *python*).

main () - l_r82.py

```
import my_mod
import itertools
import math
print("  x  ", ": ","  y   ", " : ","  yt  ", " : ","  error  "  )
print("-----------------------------------------------------------")
Epsilon=0.01
a=0;b=1.0
for i in itertools.count(start=a, step=0.2):
    if i>b+ step:  break
    y=ExpUser(i, Epsilon)
    y1=math.exp(i) # точное значение функции
    error=abs(y-y1)
    print('%.2f' %i, " : ", '%.4f' %y, " : ", '%.4f' %y1, " : ",
'%.4f' %error)
```

3) застосуйте функцію help для даного модуля (наприклад, якщо ім'я модуля my_mod, то виклик довідки про нього виглядає т. О .: help (my_mod)).

```
import my_mod
dir(my_mod)
print(help(my_mod))
print(my_mod.ExpUser.__doc__)
```

Приклад 11.2: *рекурсія.*

Описати функцію, яка виводить значення факторіала.

Факторіалом числа називають добуток всіх натуральних чисел до нього включно. Наприклад, факторіал числа 5 дорівнює добутку 1·2·3·4·5 = 120. Формулу визначення факторіала можна записати таким чином: $n! = 1 \cdot 2 \cdot ... \cdot n$, де n – це число, а $n!$ - факторіал цього числа. Можна записати трохи по–іншому: $n!=1 \cdot ... \cdot (n-2) \cdot (n-1) \cdot n$.

Визначення факторіала числа за першою формулою можна реалізувати за допомогою циклу *while*, а за другою формулою – за допомогою рекурсії.

```
import math   # math.factorial(n)
```

А. Код на Python з використанням циклу

```
# Факторіалом числа n (позначають n!) називають добуток
# всіх натуральних чисел від 1 до n включно.
n = input("Факторіал числа ")
n = int(n); fac = 1
i = 0
while i < n:
     i += 1; fac = fac * i
print ("равен",fac)
```

Б. Код на Python з використанням функцій

```
def fac(n):
     if n == 0:
          return 1
     return fac(n-1)*n

n=5; print(fac(n))
```

Крок 0. Виклик функції: *fac*(5)
Крок 1. *fac*(5) повертає *fac*(4) · 5
Крок 2. *fac*(4) => *fac*(3) · 4
Крок 3. *fac*(3) => *fac*(2) · 3
Крок 4. *fac*(2) => *fac*(1) · 2
Крок 5. fac(1) => 1
Крок 6. 1 · 2 – повернення до виклику *fac*(2)
Крок 7. 2 · 3 – *fac*(3)
Крок 8. 6 · 4 – *fac*(4)
Крок 9. 24 · 5 – *fac*(5)
Крок 10. Повернення у основну гілку програми значення «120».

В. Код на Python з використанням lambda -виразу

```
import sys
import functools

factorial=(lambda   z:functools.reduce(lambda   x,y:   x*y,   range
(1,z+1)))
if len(sys.argv)>1:
    n=int(sys.argv[1])
else:
 n = int(input("число n= "))
 print ('n= %d => n! = %d' % (n,factorial(n)))
```

Приклад 11.3: *рекурсія.*

Описати функцію, яка виводить ряд Фібоначчі до довільної межі *n*:

```
def fib(n):
...     """виводить ряд Фібоначчі до n"""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
>>> fib(2000) # Виклик функції
```

Результат

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

Пример 11.4: *рекурсия.*

Описати рекурсивну функцію обчислення суми чисел у списку:

Пряма рекурсія:

```
def mysum(L):
...  if not L:
...   return 0
...  else:
...   return L[0] + mysum(L[1:]) # Викликає саму себе
>>> mysum([1, 2, 3, 4, 5])
15
```

Рекурсивний цикл закінчується і повертається нуль, коли функція отримає порожній список. Коли використовують рекурсію, то на кожному рівні рекурсії формується змінна L (додайте у функцію оператор виведення L і запустіть приклад ще раз, щоб побачити вміст списку на кожному рівні рекурсії):

```
def mysum(L):
...  print(L) # допомогає відслідковувати рівні рекурсії
...  if not L: return 0
...  else:
...   return L[0] + mysum(L[1:])
>>> mysum([1, 2, 3, 4, 5])
[1, 2, 3, 4, 5]
[2, 3, 4, 5]
[3, 4, 5]
```

```
[4, 5]
[5]
[]
15
```

На кожному рівні рекурсії список стає все менше і менше, поки не спорожніє, що викличе кінець рекурсивного циклу. Сума обчислюється в процесі зворотного розкручування рекурсії.

Непряма рекурсія:

```
def mysum(L):
...  if not L: return 0
...  return nonempty(L) # Виклик функції, яка викличе цю функцію
def nonempty(L):
... return L[0] + mysum(L[1:]) # непряма рекурсія
>>> mysum([1.1, 2.2, 3.3, 4.4])
11.0
```

Теоретична частина

1. Рекурсивні функції

Кожна з функцій може викликати інші функції, у тому числі звертатися до самої себе. Функцію, яка звертається до самої себе, називають рекурсивною. Рекурсія – виклик функції з неї ж самої, безпосередньо (пряма) або через інші функції (складна або непряма): коли функція *first* викликає функцію *second*, а та у свою чергу викликає функцію *first*.

Кількість вкладених викликів функції називають глибиною рекурсії.

Обов'язковим елементом в описі будь-якого рекурсивного процесу є деяке твердження (оператор), який визначає умову завершення рекурсії (іноді його називають опорною умовою). Тут можна задати певне фіксоване значення, яке обов'язково буде досягнуто під час рекурсивного обчислення, дозволяючи організувати своєчасне призупинення процесу. Крім того, повинен існувати спосіб зображення одного кроку розв'язання за допомогою іншого, простішого. Кількість рівнів вкладеності не обмежена і може бути досить великою. Кожен виклик рекурсивної функції повинен наближати випадок, який зупиняє рекурсивні виклики, інакше виконання функції ніколи не припиниться через нескінченний ланцюжок рекурсивних викликів. Найпростіший способ гарантувати виконання опорної умови є зменшення деякої додатної величини до досягнення деякого «малого» значення.

Особливістю реалізації рекурсивних функцій є те, що у програмі існує тільки один екземпляр коду цієї функції. На кожному рівні рекурсії здійснюється нове звертання до цього коду. Рис. 1.1 ілюструє виконання рекурсивного процесу з трьома рівнями рекурсивного виклику, правда для простоти пояснення вважається, що на кожному рівні створюється новий екземпляр коду. Розглянемо послідовність дій, які виконують у цьому випадку.

Виклик функції забезпечує початок виконання її коду. Припускаючи, що на першому рівні рекурсії опорна умова не виконується, маємо виконання операторів, розташованих за опорною умовою, і новий виклик функції.

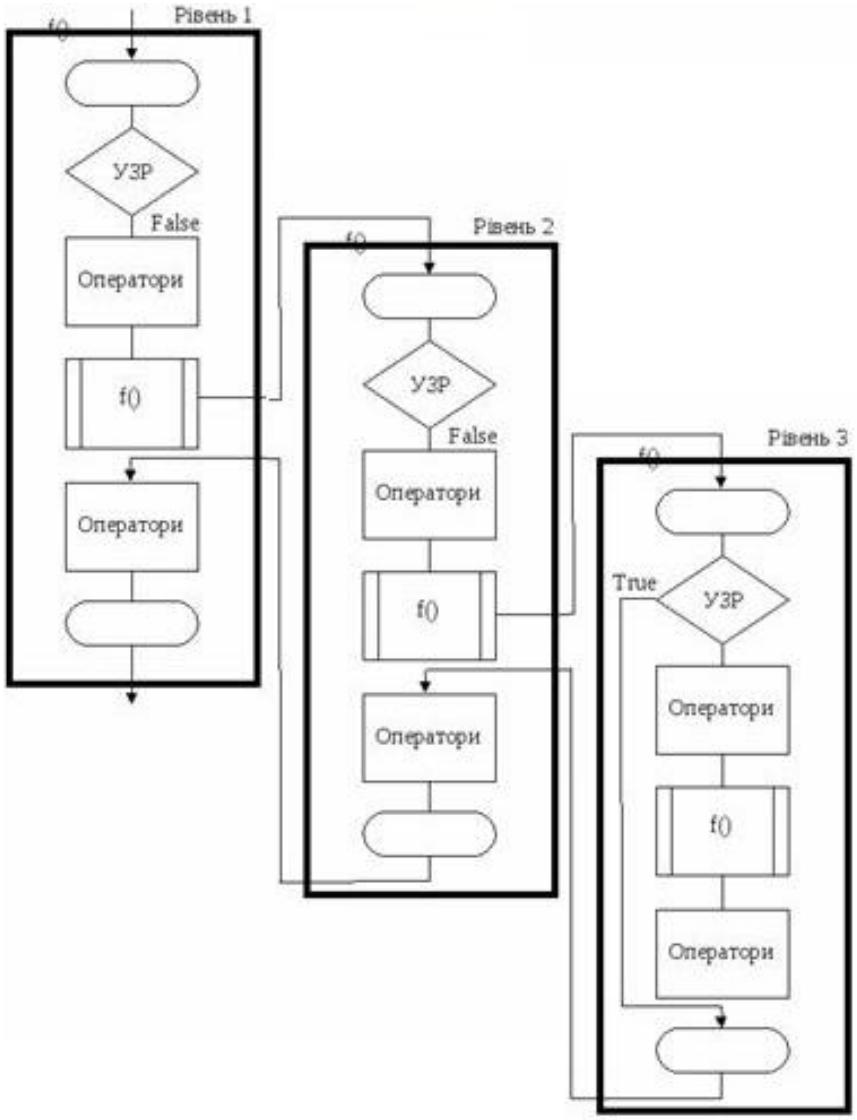


Рис. 9.1. Схема виконання рекурсивного процесу (УЗР– умова завершення рекурсії)

На цьому виконання функції на першому рівні тимчасово переривається, і розпочинається другий рівень рекурсії. Другий рівень рекурсії (рис.9.1) є аналогом першому. Він також тимчасово переривається з переходом до третього рівня рекурсії. Якщо припустити, що на третьому рівні рекурсії опорна умова виконана, маємо завершення третього рівня з поверненням на другий рівень у точку, де відбулося тимчасове переривання виконання процесу. Далі завершуються обчислення на другому рівні рекурсії з наступним поверненням на перший рівень. Після завершення обчислень на першому рівні, отримуємо остаточний результат.

Рекурсивні функції можуть вимагати великих обсягів пам’яті для свого виконання. Це пояснюється тим, що, як і у випадку звичайних функцій, локальні змінні створюються в програмному стеку на кожному рівні рекурсивного виклику, що може привести до ситуації, коли обсяг стека вичерпається, оскільки знищення локальних змінних здійснюється тільки при виході з функції.

Будь–яку рекурсивну функцію можна замінити циклом.

Функції, які виконують один рекурсивний виклик на кожній рекурсивній гілці. Структурно рекурсивна функція на верхньому рівні завжди має команду розгалуження у вигляді вибору однієї з двох або більше альтернатив в залежності від умови (умов), яку в цьому випадку доречно назвати «умовою припинення рекурсії». Умова має дві або більше альтернативні гілки, з яких хоча б одна є рекурсивною і хоча б одна – термінальною.

Рекурсивну гілку виконують, коли умова припинення рекурсії має значення «хибність», і містить хоча б один рекурсивний виклик – прямий або опосередкований виклик функцією самої себе.

Термінальну гілку виконують, коли умова припинення рекурсії має значення «істина» (вона повертає деяке значення, не виконуючи рекурсивного виклику). Правильно написана рекурсивна функція повинна гарантувати, що через скінчену кількість рекурсивних викликів буде досягнуто виконання умови припинення рекурсії, в результаті чого ланцюжок послідовних рекурсивних викликів буде перерваний.

Бувають випадки «паралельної рекурсії», коли на одній рекурсивній гілці виконують два або більше рекурсивних виклики. Паралельна рекурсія типова при обробці складних структур даних, таких як дерева. Найпростіший приклад паралельно-рекурсивної функції – обчислення ряду Фібоначчі, де для отримання значення *n*-го члена необхідно обчислити (*n*–1)-й і (*n*–2)-й.

Питання про бажаність використання рекурсивних функцій в програмуванні є неоднозначним. З одного боку, рекурсивна форма може бути структурно простішою і наочнішою, особливо, коли сам алгоритм, по суті є рекурсивним. З іншого боку, рекомендують уникати рекурсивних програм, які призводять (або в деяких умовах можуть призводити) до рекурсії великої глибини. Приклад рекурсивного обчислення факторіала є, скоріше, прикладом того, як не треба застосовувати рекурсію, тому що призводить до досить великої глибині рекурсії і має очевидну реалізацію у вигляді звичайного циклічного алгоритму.

2. Коментар #. Функції dir, help

Коментар #. Коментар, який розпочинають з символу решітки, є елементарним способом документування програмного коду. Інтерпретатор ігнорує весь текст, який слідує за символом # (за умови, що він знаходиться не всередині літерала-рядку). Вважають, що рядки документування краще підходять для створення функціонального опису (наприклад, «мій файл робить …»), а коментарі, які розпочинаються з символу #, краще підходять для опису деяких особливостей програмного коду (наприклад, «це дивний вираз робить …»).

Функція dir. Функція *dir* – це простий спосіб отримати список всіх атрибутів об'єкта (тобто методів і елементів даних). Цю функцію можна викликати для будь-якого об'єкта, який має атрибути. Наприклад, щоб дізнатися, що є в стандартному бібліотечному модулі *sys*, імпортуйте його і передайте ім'я модуля функції *dir*:

```
>>> import sys
>>> dir(sys)
```

```
    ['__displayhook__', '__doc__', '__excepthook__', '__name__',
'__package__',
    '__stderr__', '__stdin__', '__stdout__', '_clear_type_cache',
'_current_frames', '_getframe', 'api_version', 'argv',
'builtin_module_names', 'byteorder', 'call_ tracing', 'callstats',
'copyright', 'displayhook', 'dllhandle', 'dont_write_ bytecode',
'exc_info', 'excepthook', 'exec_prefix', 'executable', 'exit',
'flags', 'float_info', 'getcheckinterval', 'getdefaultencoding',
...остальные имена опущены...]
```

Тут показані тільки деякі з імен. Щоб дізнатися, які атрибути містять об'єкти вбудованих типів, передайте функції dir літерал (або існуючий об'єкт) необхідного типу. Наприклад, щоб побачити атрибути списків і рядків, можна передати функції порожній об'єкт:

```
>>> dir([])
['__add__', '__class__', ...остальные имена опущены...
'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
>>> dir('')
['__add__', '__class__', '__contains__', ...остальные имена
опущены...
'capitalize', 'center', 'count', 'encode', 'endswith',
'expandtabs',
'find', 'format', 'index', 'isalnum', 'isalpha', 'isdecimal',
'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex',
'rjust',...остальные имена опущены...]
```

Результат роботи функції *dir* для будь-якого вбудованого типу включає набір атрибутів, які мають відношення до реалізації цього типу (методи перевантажених операторів); всі вони розпочинаються і закінчуються двома символами підкреслення, щоб зробити їх відмінними від звичайних імен.

Функції *dir* і *help* в мові Python – це найперший засіб отримання доступу до документації. Функція *dir* повертає лише імена методів. Щоб дізнатися про призначення того чи іншого методу, можна передати його ім'я функції *help*:

```
>>> help(S.replace)
Help on built-in function replace:
(Справка о встроенной функции replace:)
replace(...)
S.replace (old, new[, count]) -> str
Return a copy of S with all occurrences of substring
old replaced by new. If the optional argument count is
given, only the first count occurrences are replaced.
```

3. Рядки документування: __doc__.

Python підтримує можливість створення документації, яка автоматично приєднується до об'єктів і доступна під час виконання. Синтаксично такі рядки розташовують на початку файлів модулів, функцій і класів, перед виконуваним програмним кодом (перед ними можна розташовувати коментарі #).

Інтерпретатор автоматично розміщує рядки документування в атрибут __doc__ відповідного об'єкта.

Рядки документування, визначені користувачем. Як приклад розглянемо файл *docstrings.py*. Рядки документування в ньому розташовано на початку файлу, а також на початку функції і класу, для створення описів використані рядки в потрійних лапках:

```
“”” Module documentation Words Go Here “””
spam = 40

def square(x):
 “”” function documentation can we have your liver then? “””
return x**2

class Employee:
 “”” class documentation “””
 pass

print(square(4))
print(square.__doc__)
```

Коментарі стають доступні для перегляду у вигляді атрибутів *__doc__* після того, як файл буде імпортовано. Щоб відобразити рядки документування, пов'язані з модулем і його об'єктами, досить імпортувати файл і вивести значення його атрибутів *__doc__*, де інтерпретатор зберігає текст:

```
>>> import docstrings
16
function documentation
can we have your liver then?
>>> print(docstrings.__doc__)
Module documentation
Words Go Here
>>> print(docstrings.square.__doc__)
function documentation
can we have your liver then?
>>> print(docstrings.employee.__doc__)
class documentation
```

Для виведення рядків документування необхідно використовувати функцію *print*, в іншому випадку буде виводитися один рядок з вбудованими символами нового рядка.

Вбудовані рядки документування. У вбудованих модулях і об'єктах Python використовують подібну методику приєднання документації – до і після списку атрибутів, які повертає функція *dir*. Наприклад, щоб побачити опис вбудованого модуля, його треба імпортувати і вивести рядок *__doc__*:

```
>>> import sys
>>> print(sys.__doc__)
This module provides access to some objects used or maintained by
the
interpreter and to functions that interact strongly with the
```

```
interpreter.
Dynamic objects:
argv -- command line arguments; argv[0] is the script pathname if
known
path -- module search path; path[0] is the script directory, else
''
modules -- dictionary of loaded modules
...остальной текст опущен...
```

Переглядаючи таким способом рядки документування вбудованих інструментів, можна отримати багато інформації, однак цю інформацію функція *help* надає автоматично.

Функцію *help* можна використати для отримання відомостей про вбудовані функції, методи і типи. Щоб отримати довідку про вбудований тип, необхідно передати функції ім'я типу (наприклад, *dict* – для словників, *str* – для рядків, *list* – для списків), буде надана інформація з описами всіх методів, доступних для цього типу:

```
>>> help(dict)
Help on class dict in module builtins:
class dict(object)
| dict() -> new empty dictionary.
| dict(mapping) -> new dictionary initialized from a mapping
object's
...остальной текст опущен...
>>> help(str.replace)
Help on method_descriptor:
replace(...)
S.replace (old, new[, count]) -> str
Return a copy of S with all occurrences of substring
...остальной текст опущен...
>>> help(ord)
Help on built-in function ord in module builtins:
ord(...)
ord(c) -> integer
Return the integer ordinal of a one-character string.
```

4.Функції lambda

Lambda-вирази. В Python існує можливість створювати об'єкти функцій у формі виразів. Через схожість з аналогічною можливістю в мові LISP вона отримала назву *lambda*. Ця назва походить з мови програмування LISP, в якій ця назва була запозичена з лямбда–числення. Однак в Python це – ключове слово, яке вводить вираз синтаксично. Подібно інструкції *def* цей вираз створює функцію, яку викличуть пізніше (але на відміну від інструкції *def*, вираз повертає функцію, а не зв'язує її з ім'ям). Саме тому *lambda*-вирази іноді називають анонімними (тобто безіменними) функціями. Їх часто використовують, як спосіб отримати вбудовану функцію або відкласти виконання фрагмента програмного коду. У загальному вигляді *lambda*-вираз складається з ключового слова *lambda*, за яким слідують один або більше аргументів (як список аргументів у круглих дужках у заголовку інструкції *def*) і

далі, слідом за двокрапкою, розташовано вираз:

lambda *argument1, argument2,... argumentN*:

вираз, який використовує аргументи

Результатом *lambda*–виразу є такі ж об'єкти функцій, які утворюють інструкції *def*, але тут є кілька відмінностей: 1) *lambda* – це вираз, а не інструкція; 2) тіло *lambda* – це не блок інструкцій, а один вираз. Наприклад, функцію:

```
def func(x, y, z): return x+y+z
>>> func(2, 3, 4)
9
```

можна описати за допомогою *lambda*–виразу, явно присвоївши результат імені, яке пізніше буде використано для виклику функції:

```
f=lambda x,y,z:x+y+z
>>> f(2,3,4)
9
```

Тут імені *f* присвоєно об'єкт функції, створений *lambda*-виразом, – інструкція *def* працює так само, але присвоювання виконують автоматично.

Коли можна використовувати *lambda*–вирази? – Їх використовують для створення маленьких функцій і дозволяють вбудовувати визначення функцій в код, який їх використовує. Вони не є предметом першої необхідності (можна замість них використовувати інструкції *def*), але вони дозволяють спростити сценарії, де потрібно впроваджувати невеликі фрагменти програмного коду. Вони корисні в якості скороченого варіанту інструкції *def*, коли необхідно вставити маленькі фрагменти виконуваного коду туди, де використання інструкцій неприпустимо. Слідуючий фрагмент коду створює список з трьох функцій, *lambda*–вирази вставлено в літерал списку. Інструкція *def* не може бути вставлена в літерал, тому що це – інструкція, а не вираз.

```
L = [lambda x: x**2, # Вбудовані визначення функцій
lambda x: x**3,
lambda x: x**4] # Список з трьох функцій
for f in L:
 print(f(2)) # Виведе 4, 8, 16
print(L[0](3)) # Виведе 9
```

Для реалізації еквівалентної таблиці переходів із застосуванням інструкцій *def* необхідно створити іменовані функції поза контекстом їх використання:

```
# визначення іменованих функцій
def f1(x): return x ** 2
def f2(x): return x ** 3
def f3(x): return x ** 4
L=[f1, f2, f3] # Посилання по імені
for f in L:
 print(f(2)) # Виведе 4, 8, 16
 print(L[0](3)) # Виведе 9
```

Комп'ютерний практикум № 12. Словник. Кортеж

Мета роботи: ознайомитися з такими об'єктами як словники, кортежі мови Python. *Об'єкт дослідження* – словник, кортеж.

ПЛАН

1. Словник
2. Кортеж

Завдання

1. Вивчити теоретичні основи написання алгоритмів з використанням словника і кортежу. Опрацювати приклади.
2. Відповідно до свого варіанту
- визначити умови;
- за допомогою формул описати варіанти виконання необхідний дій;
- написати програму, яка розв'язує завдання;
- організувати введення даних з клавіатури, виведення у консоль;
- для подання необхідної інформації реалізувати певну структуру даних, під час визначення якої використовувати будь-які комбінації вбудованих об'єктів (кортежі, словники, списки, рядки, числа); спробуйте звернутися до окремих елементів словника;
- реалізувати а) режим виведення на екран всіх значень словника; б) додавання (видалення) нового запису до (зі) словника; в) режим перегляду вмісту словника за відсортованими ключами (перетворити об'єкт подання ключів в список або скористатися функцією *sorted*, застосувавши її до словника, або об'єкту, який повертає метод *keys*);
3. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних.

Варіанти «Організація пошуку та вибору інформації»

1. Задано кількість очок, набраних кожною з n=9 команд–учасниць першості з футболу(перелік очок і команд задано у порядку зайнятих ними місць, тобто в порядку зменшення кількості набраних очок,жодна пара команд–учасниць не набрала однакову кількість очок). З'ясувалося, що в перелік забули включити ще одну, десяту, команду. Скласти програму, яка визначає: а) місце, яке зайняла зазначена команда (кількість набраних нею очок відомо, відомо також, що вона не стала чемпіоном і не зайняла останнє місце); б) назви команд, які набрали менше очок, ніж ця команда.

2. Задано дані про n=10 моменти часу однієї доби: години (значення від 0 до 23) і хвилини (від 0 до 59). Скласти програму, яка порівнює два будь-яких моменти часу за їх умовним порядковим номером і визначає, який з моментів був у цю добу раніше.

3. Задано дати кожної з n=10 подій, які відбулися після 1930 року: рік, номер місяця і число. Скласти програму, яка порівнює дві будь-яких події за

часом (визначає, яка з подій відбулася пізніше). Подію можна подати у вигляді рядку.

4. Задано дані про *n*=10 моменти часу однієї доби: години (значення від 0 до 23), хвилини (від 0 до 59) і секунди (від 0 до 59). Скласти програму, яка порівнює два будь-яких моменти часу (визначає, який з моментів відбувся у задану добу раніше).

5. Задано прізвища всіх *n*=10 співробітників фірми та їх адреси. Скласти програму, яка визначає, чи працюють у фірмі люди з прізвищем: Кузін, Куравльов, Кудін, Кульков або Кубиків. У разі позитивної відповіді надрукувати їх адреси.

6. Задано назви *n*=10 країн із загальною інформацією про них (площа, населення) і про частини світу, де вони розташовані. Скласти програму, яка визначає, чи є серед заданих країн ті, що знаходяться в Африці або в Азії. У разі позитивної відповіді надрукувати їх назви.

7. Задано дані про *n*=10 учнів класу: прізвище, ім’я, по батькові, дата народження (рік, номер місяця і число). Скласти програму, яка визначає, чи є в класі учні, у яких сьогодні день народження, і якщо так, то вивести їх ім'я та прізвище.

8. Задано прізвища та номери телефонів *n*=10 осіб у записнику. Скласти програму, яка визначає, чи є в записнику: а) телефон певної особи, і, якщо є, друкує номер її телефону; б) інформація про людину із заданим номером телефону, і, якщо є, вивести прізвище цієї людини.

9. Задано дані про *n*=10 учнів кількох шкіл, які займаються в районному Будинку творчості учнів (прізвище, ім'я, адреса, номер школи і клас). Скласти програму, яка визначає прізвище, ім'я та адресу учнів, що навчаються у визначеній школі в старших (10-11) класах, ці дані записати в окремий масив–список з елементами типу «ключ: кортеж».

10. Задано дані про кількість опадів, які випали за кожен день місяця, і про температуру повітря в ці дні. Скласти програму, що визначає, яка кількість опадів випала у вигляді снігу і яка – у вигляді дощу (вважати, що дощ іде, якщо температура повітря вище 0° С).

11. Задано дані про потужність двигуна (в кінських силах – к.с.) і вартість *n*=10 легкових автомобілів. Скласти програму, яка визначає загальну вартість автомобілів, у яких потужність двигуна перевищує 100 к. с.

12. Задано дані про вік і стать кожної з *n*=10 осіб. Скласти програму, яка визначає загальну кількість чоловіків.

13. Задано дані про кількість учнів у кожному з *n*=6 навчальних закладів і про тип закладів (школа, технікум або училище). Скласти програму, яка визначає загальну кількість учнів шкіл.

14. Задано дані про ціну і тираж кожного з *n*=5 журналів. Скласти програму, яка визначає середню вартість журналів, тираж яких менше 10 000 примірників.

15. Задано дані про вартість і «вік» кожної з n=10 моделей легкових автомобілів. Скласти програму, яка визначає середню вартість автомобілів, «вік» яких перевищує 6 років.

16. Задано дані про зріст і стать кожної з n=10 осіб. Скласти програму, яка визначає середній зріст чоловіків.

17. Задано дані про вартість кожної з n=5 моделей автомобілів і про їхній тип (легковий або вантажний). Скласти програму, яка визначає середню вартість легкових автобілів.

18. Задано дані про оцінки кожного з n=10 учнів класу з дванадцяти предметів. Скласти програму, яка визначає середню оцінку кожного учня і всього класу. Вивести прізвища учнів, у яких середня оцінка вище середньої в класі.

19. Задано дані у вигляді (маса; об'єм) n предметів, які виготовлені з різніх матеріалів. Скласти програму, яка визначає максимальну щільність матеріалу. *Підказка*: щільність = маса / об'єм.

20. Задано дані про чисельність населення (в мільйонах жителів) і площа (в тисячах квадратних км) n=10 держав. Скласти програму, яка визначає назву держави з максимальною щільністю населення.

21. Задано дані про оцінки кожного з n=10 учнів класу з десяти предметів. Скласти програму, яка визначає прізвище одного з учнів, який має: а) найбільшу суму оцінок; б) найменшу суму оцінок.

22. Задано дані про оцінки кожного з n=10 учнів класу з чотирьох предметів. Скласти програму, яка визначає прізвище одного з учнів, який має максимальну суму оцінок.

23. Задано дані про бали, набрані кожним з n=10 спортсменів-п'ятиборців в кожному з п'яти видів спорту. Скласти програму, яка визначає прізвище спортсмена-переможця змагань.

24. Задано дані про кількість очок, набраних кожною з n=10 команд-учасниць першості з футболу. Жодна пара команд не набрала однакову кількість очок. Скласти програму, яка визначає: а) назву команди, яка стала чемпіоном; б) назву команд, які посіли друге і третє місця; в) назву команд, які зайняли перше і друге місця, не використовуючи при цьому два оператори циклу (два проходи по масиву).

25. Задано дані про зріст кожного з n=15 учнів класу. Немає жодної пари учнів з однаковим зростом. Скласти програму, яка визначає: а) прізвища найвищого і найнижчого учнів класу; б) прізвища двох учнів, які є найвищими без урахування найвищого учня класу; в) прізвища двох учнів, які є найвищими в класі, не використовуючи при цьому два оператори циклу (два проходи по масиву).

26. Задано дані про n=10 співробітників фірми (прізвище, зарплата і стать). Скласти програму, яка визначає: а) прізвище особи, яка має найбільшу зарплату (вважати, що такий є лише один); б) прізвища чоловіка і жінки, які мають найменшу зарплату (вважати, що такі є і вони єдині в своїй групі співробітників).

27. Задано дані про n=10 співробітників фірми: прізвище, вік і відношення до військової служби (військовозобов'язаний чи ні). Скласти програму, яка визначає: а) прізвище наймолодшого за віком співробітника серед військовозобов'язаних (вважати, що такий є і він один); б) прізвища найстарших за віком людей серед військовозобов'язаних і серед невійськовозобов'язаних (вважати, що такі є і вони єдині в своїй групі).

28. Задано дані про розклад поїздів, які проходять через певну станцію: номер поїзда, призначення (звідки куди, наприклад, Москва – Омськ), час (години та хвилини) прибуття, час (години та хвилини) відправлення. Година та хвилина – цілі, додатні числа; значення годин не перевищує «23», значення хвилин – «59». Загальна кількість поїздів n=10. Поїзди приходять кожен день. Скласти програму, яка визначає, які поїзди (номер і призначення) стоять у визначений момент часу на станції.

29. Задано дані про про багаж (кількість речей і загальна вага багажу) n=10 пасажирів. Скласти програму, яка визначає: а) кількість пасажирів, які мають більше двох речей; б) чи є хоч один пасажир, багаж якого складається з однієї речі вагою менше 25 кг; в) число пасажирів, кількість речей яких перевершує середнє число речей всіх пасажирів; г) номер багажу, в якому середня вага однієї речі відрізняється від загальної середньої ваги однієї речі не більше ніж на 0,5 кг.

30. Задано дані про про зріст n=10 юнаків класу, впорядковані за зменшенням (немає жодної пари учнів, які мають однаковий зріст). На початку навчального року до класу вступив новий учень (його зріст не співпадає із зростом жодного з учнів класу, перевищує зріст самого низького учня і менше зросту найвищого). Скласти програму, яка визначає: а) прізвища всіх учнів, зріст яких менше росту «новенького»; б) прізвище учня, після якого слід записати прізвище «новенького», щоб впорядкованість не порушилася; в) прізвище учня, зріст якого найменше відрізняється від росту «новенького». У завданнях (а) і (б) умовний оператор не використовувати.

Контрольні запитання

1. Що таке словник? Назвіть його основні властивості.
2. Що таке кортеж? Назвіть його основні властивості.

Аудиторна робота

Приклад 12.1.

1. Створення словників

```
# наведені приклади створюють однакові словники
a=dict(one=1, two=2, three=3)
b={'one': 1, 'two': 2, 'three': 3}
c=dict(zip(['one', 'two', 'three'], [1, 2, 3]))
d=dict([('two', 2), ('one', 1), ('three', 3)])
e=dict({'three': 3, 'one': 1, 'two': 2})
print(a == b == c == d == e);print(a);print()
# Використання генераторів словників
print({string: string.upper() for string in ('one','two',
```

```
'three')})
```

Результат

```
True
{'three': 3, 'two': 2, 'one': 1}
{'three': 'THREE', 'two': 'TWO', 'one': 'ONE'}
```

2. Операції зі словниками

```
phonebook = {
    'Jack': '032-846',
    'Guido': '917-333',
    'Mario': '120-422',
    'Mary': '890-532',  # остання кома ігнорується}
# len(d) – кількість елементів
print(len(phonebook),'entries found');print()
```

d[key] – отримання значення з ключем *key*. Якщо ключ не існує, відображення реалізує спеціальний метод *__missing __ (self, key)*: якщо ключ не існує і метод *__missing_*_ не визначений, видається виняток KeyError

```
try:
    print('Mary:', phonebook['Mary'])
    print('Lumberjack:', phonebook['Lumberjack'])
except KeyError as e:
    print('No entry for', *e.args)
print()
```

d[key]=value – змінити значення або створити пару ключ-значення, якщо ключ не існує

```
phonebook['Lumberjack'] = '000-777'
```

key in d, key not in d – перевірка наявності ключа в відображенні

```
for person in ('Guido', 'Mary', 'Ahmed'):
    if person in phonebook:
        print(person, 'is in the phonebook')
    else:
        print('No entry found for', person)
print()
```

iter(d) – теж саме, що iter(d.keys())

```
print('People in the phonebook:')
for person in phonebook:
    print(person)
print()
```

copy() – створити неповну копію словника

```
phonebook_copy = phonebook.copy()
print('Phonebook:', phonebook)
print('Phonebook copy:', phonebook_copy);print()
```

clear() – видалити всі елементи зі словника

```
phonebook_copy.clear()
```

```
print('Phonebook:', phonebook)
print('Phonebook copy:', phonebook_copy);print()
```

(метод класу) **dict.fromkeys (sequence [, value])** – створює новий словник з ключами з послідовності *sequence* і заданим значенням (за замовчуванням None).

```
numbers_dict = dict.fromkeys(range(3), 42)
print(numbers_dict);print()
```

d.get(key[, default]) – безпечне отримання значення за ключем (ніколи не видає *KeyError*). Якщо ключ не знайдений, повертається значення default (за замовчуванням None).

```
for key in range(5):
    print('{}:'.format(key), numbers_dict.get(key, 0))
print()
```

d.items() – повертає об'єкт подання словника, який відповідає парам (двоелементним кортежам) вигляду (ключ, значення).

```
print('Items:', phonebook.items())
```

d.keys() – повертає об'єкт подання словника, який відповідає ключам словника.

```
print('Keys:', phonebook.keys())
```

d.values() – повертає об'єкт подання словника, який відповідає значенням

```
print('Values:', phonebook.values()); print()
```

d.pop(key[, default]) – якщо ключ *key* існує, метод *pop* видаляє елемент із словника і повертає його значення; якщо ключ не існує і задано значення *default*, то повертає дане значення, інакше видається виняток KeyError.

```
number = phonebook.pop('Lumberjack')
print('Deleted Lumberjack (was ' + number + ')')
print(phonebook);print()
```

```
>>> D={'toast': 4, 'muffin': 5, 'eggs': 3, 'ham': 1, 'spam': 2}
>>> D.pop('muffin') # видалення елементів словника за ключем
5
>>> D.pop('toast') # видаляє і повертає значення заданого ключа
4
>>> D
{'eggs': 3, 'ham': 1, 'spam': 2}
# видалення елементів списку за номером позиції
>>> L = ['aa', 'bb', 'cc', 'dd']
>>> L.pop() # видаляє і повертає останній елемент списку
'dd'
>>> L
['aa', 'bb', 'cc']
>>> L.pop(1) # видаляє і повертає елемент із заданої позиції
'bb'
>>> L
['aa', 'cc']
```

d.popitem() – видаляє довільну пару ключ-значення і повертає її. Якщо словник порожній, виникає виключення KeyError. Метод корисний для алгоритмів, які обходять словник, видаляючи вже оброблені значення (наприклад, деякі алгоритми, пов'язані з теорією графів).

```
person = phonebook.popitem()
print('Popped {} (phone: {})'.format(*person)); print()
```

d.setdefault(key[, default]) – якщо ключ *key* існує, повертає відповідне значення, інакше створює елемент з ключем *key* і значенням *default* (*default* за замовчуванням дорівнює None).

```
for person in ('Jack', 'Liz'):
    phone = phonebook.setdefault(person, '000-000')
    print('{}: {}'.format(person, phone))
print(phonebook);print()
```

d.update(mapping) – приймає інший словник або відображення, або ітерабельний об'єкт, який складається з ітерабельних об'єктів - пар ключ-значення, або іменовані аргументи. Додає відповідні елементи в словник, переписуючи елементи з існуючими ключами.

```
phonebook.update({'Alex': '832-438', 'Alice': '231-987'})
phonebook.update([('Joe', '217-531'), ('James', '783-428')])
phonebook.update(Carl='783-923', Victoria='386-486')
print(phonebook)
```

3. Використання елементів словника

Об'єкти, які повертають методи items (), keys (), values() - це об'єкти, які надають динамічне подання елементів словника (зміни у словнику автоматично відображаються і на цих об'єктах).

Операції з поданням словників:

iter (dictview) – отримання ітератора за ключами, значеннями або парами (ключ, значення); всі зображення словників при ітеруванні повертають елементи словника в однаковому порядку; при спробі змінити словник під час ітерування може виникнути виключення RuntimeError;

len (dictview) – кількість елементів в словнику;

x in dictview – перевірка існування ключа, значення або пари (ключ,-значення) в словнику.

```
dishes = {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam': 500}
keys = dishes.keys()
values = dishes.values()
# Ітерування
n = 0
for val in values:
    n += val
print(n)
# Ключі та значення ітерують у такому ж порядку
print(list(keys)); print(list(values))
# Об'єкти  динамічно відображають зміни в словнику
```

```
    del dishes['eggs'];del dishes['sausage']
    print(list(keys))
# Також вони підтримують операції із множинами
    print(keys & {'eggs', 'bacon', 'salad'})
    print(keys ^ {'sausage', 'juice'})
```

Результат

```
504
['bacon', 'eggs', 'sausage', 'spam']
[1, 2, 1, 500]
['bacon', 'spam']
{'bacon'}
{'juice', 'bacon', 'sausage', 'spam'}
```

4. Використання *collections.Counter*.

collections.Counter – це підклас *dict*, призначений для підрахунку об'єктів, які хешують; також її називають мультимножина. Елементи зберігають як ключі словника, а їх кількість – як значення.

```
    1)from collections import Counter
    2)
    counter = Counter();counter[1] += 1
    for i in range(3):
        print(counter[i])
    print()

    c=Counter('abcdeabcdabcaba')  # підрахунок кількості кожного
символа в рядку
    print(c.most_common(3))     # три найчастіших елементи
    print(sorted(c))            # всі унікальні елементи
    print(sorted(c.elements())) # всі елементи
    print(sum(c.values()))      # сума значень
    print(c['a'] )            # кількість літер 'a'

    for elem in 'shazam':      # додати нові літери
        c[elem] += 1
    print(c['a'])              # тепер у лічильнику сім літер 'a'
    del c['b']                 # видалити всі 'b'
    print(c['b'])

    d = Counter('simsalabim')  # створити новий лічильник
    c.update(d)                # додати його елементи до першого
    print(c['a'])              # тепер в ньому дев'ять 'a'

    c.clear()                   # очистити лічильник
    print(c)
    # Якщо рахунок елемента встановити «0» або зменшити до нуля,
    # він залишиться в лічильнику, поки не буде видалений явно
    c = Counter('aaabbc')
    c['b'] -= 2; print(c.most_common())
```

Результат

```
0
1
```

```
0
[('a', 5), ('b', 4), ('c', 3)]
['a', 'b', 'c', 'd', 'e']
['a', 'a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', 'c', 'c', 'c', 'd',
'd', 'e']
15
5
7
0
9
Counter()
[('a', 3), ('c', 1), ('b', 0)]
```

5. Використання lambda-виразів

```
>>> key = 'got'
>>> {'already': (lambda: 2 + 2),
... 'got': (lambda: 2 * 4),
... 'one': (lambda: 2 ** 6)}[key]()
8
```

Коли інтерпретатор створює словник, кожний з вкладених *lambda-виразів* генерує і залишає після себе функцію для подальшого використання - звернення за ключем витягує одну з цих функцій, а круглі дужки забезпечують виклик витягнутої функції. При такому підході словник перетворюється в більш універсальний засіб множинного вибору. Щоб реалізувати те ж саме без використання *lambda-виразів*, довелося б написати три окремі інструкції *def* за межами словника, в якому ці функції використовують, і посилатися на функції за їхніми іменами:

```
>>> def f1(): return 2 + 2
...
>>> def f2(): return 2 * 4
...
>>> def f3(): return 2 ** 6
...
>>> key = 'one'
>>> {'already': f1, 'got': f2, 'one': f3}[key]()
64
```

Приклад 12.2. Відомо дані про вартість кожного з *n* найменувань товарів: «число грн., число копійок». Скласти програму, яка порівнює вартість двох будь-яких найменувань товарів (визначає, який з товарів коштує найдорожче).

Крок 1. *Формуємо список кортежів із ціною товара* (a_i, b_i).

Нехай задано списки цілих випадкових чисел [a_1, ..., a_n] і [b_1, ..., b_n]. Написати програму формування списку [(a_1, b_1), ..., (a_n, b_n)]. Вивести на екран початкові та отриманий списки.

```
import random
n=10; a=[random.randint(0,25) for i in range(0,10)]
b=[random.randint(0,5) for j in range(0,10)]
print(a); print(b)
xy=zip(a,b); # кортеж
```

```
xy=list(xy); print(xy)
```

Крок 2. *Формуємо словник із назвою різних товарів та їх ціною.*

```
L=[]
for i in range(1, n+1):
 a='name'+str(i)
 L.append(a)
print(L)
D={k:v for (k,v) in zip(L,xy)}
print (D)
```

Крок 3. *Виконуємо пошук найдорожчого товару*

```
a=list(D.values()); print (a)
b=a[0][0];nomer_t=1
for i in range(1, n):
 if b<a[i][0]:
   b=a[i][0]
   nomer_t=i+1
print (b, 'грн.', nomer_t, 'товар')
```

Весь код

```
import random
n=10
a=[random.randint(0,25) for i in range(0,10)]
b=[random.randint(0,5) for j in range(0,10)]
print(a); print(b)
xy=zip(a,b); # кортеж
xy=list(xy); print(xy)

L=[]
for i in range(1, n+1):
 a='name'+str(i); L.append(a); print(L)
D={k:v for (k,v) in zip(L,xy)}
print ('СЛОВНИК')
print (D);a=list(D.values()); print (a)

b=a[0][0];nomer_t=1
for i in range(1, n):
 if b<a[i][0]:
   b=a[i][0]; nomer_t=i+1
print (b, 'грн.', nomer_t, 'товар')
```

Результат

```
СЛОВАРЬ
{'name8': (9, 0), 'name10': (13, 4), 'name4': (22, 0), 'name6': (25, 2), 'name1': (21, 0),
 'name2': (22, 0), 'name9': (2, 1), 'name5': (19, 5), 'name3': (25, 1), 'name7': (17, 0)}
[(9, 0), (13, 4), (22, 0), (25, 2), (21, 0), (22, 0), (2, 1), (19, 5), (25, 1), (17, 0)]
25 грн. 4 товар
```

б) додавання (видалення) нового запису до (зі) словника:

```
import random
n=5
a=[random.randint(0,25) for i in range(0,10)]
b=[random.randint(0,5) for j in range(0,10)]
```

```
xy=zip(a,b); # кортеж
xy=list(xy); # print(xy)
L=[]
for i in range(1, n+1):
 a='name'+str(i); L.append(a);
D={k:v for (k,v) in zip(L,xy)}
print ('СЛОВНИК'); print(D);
D['name11']=(22,0);print (D) # Додавання нового елемента в словник
del D['name11'] # видалення ключів з словника
print (D)
```

Результат

```
СЛОВАРЬ
{'name5': (16, 4), 'name2': (7, 0), 'name3': (16, 1), 'name1': (13, 1), 'name4': (17, 5)}
{'name11': (22, 0), 'name3': (16, 1), 'name1': (13, 1), 'name5': (16, 4), 'name4': (17, 5), 'name2': (7, 0)}
{'name3': (16, 1), 'name1': (13, 1), 'name5': (16, 4), 'name4': (17, 5), 'name2': (7, 0)}
```

Підказка:

```
D['name11']=(22,0) # Додавання нового елемента в словник
del D[key] # видалення ключів
D.pop(key [, default]) # видаляє ключ і повертає значення;
#якщо ключа немає, повертає default (за замовчуванням - виняток).
```

Приклад 12.3. Використання функції *zip*

```
import random
n=int(input('Кількість стовпчиків рядків  = '))
a=[random.randint(-9,9) for u1 in range(1000)]
m=[random.sample(a,n) for u in range(n)]
# де "n" - це кількість рядків, стовпчиків
answer='Матриця не ортонормована'
for i in m:
    print(i)
    for j in m:
        v = sum(x*y for x,y in zip(i,j))
        if (i==j and v==1) or (i!=j and v==0):
            answer='Матриця ортонормована'; break
print(answer)
```

Результат

```
Кількість стовпчиків/рядків  = 5
[2, -7, -6, 9, 3]
[5, 9, 1, -5, 7]
[-1, 6, 8, 0, -2]
[-3, -5, -8, -3, 5]
[9, -1, -9, -5, 3]
Матриця не ортонормована
```

Приклад 10.4. Нехай задано назви 10 країн із загальною інформацією про них (площа, населення, частина світу, де знаходиться країна). Сформувати словник із цими даними.

1. Словник з використанням кортежу

```
table ={'Germany': ('Europe','154864','13189 KM^2'),
        'Ukraine': ('Europe', '467591','6248 KM^2'),
        'Egypt': ('Africa', '7891416','1488228 KM^2'),
        'Nigeria': ('Africa', '77804','99648 KM^2'),
        'China': ('Asia', '987654321','9137462 KM^2'),
        'USA': ('North America', '193764825','1230540 KM^2'),
```

```
      'Brazil': ('South America', '123456789', '894484 KM^2'),
      'Canada': ('North America', '456852', '69874 KM^2'),
      'India': ('Asia', '87864', '987697 KM^2'),
      'Finland': ('Europe', '28613','81468 KM^2')
         }
for cr in table: # Теж саме, що й for cr in table.keys()
  print(cr, '\t', table[cr])
```

2. Словник без використання кортежу (можна вносити зміни)

```
World={
 'Germany':{'cw':'Europe','ns': '154864','sq':'13189 KM^2'},
 'Ukraine':{'cw':'Europe','ns': '467591','sq':'6248 KM^2'},
 'Egypt':{'cw':'Africa', 'ns': '7891416','sq':'1488228 KM^2'},
 'Nigeria':{'cw':'Africa','ns': '77804','sq':'99648 KM^2'},
 'China':{'cw':'Asia','ns': '987654321','sq':'9137462 KM^2'},
 'USA':{'cw':'North America','ns':'193764825','sq':'1230540 KM^2'},
 'Brazil':{'cw':'South America','ns':'123456789','sq':'894484 KM^2'},
 'Canada':{'cw': 'North America','ns': '456852','sq':'69874 KM^2'},
 'India':{'cw': 'Asia','ns': '87864','sq':'987697 KM^2'},
 'Finland':{'cw': 'Europe','ns': '28613','sq':'81468 KM^2'}
        }
l=World.keys(); l=list(l); z=[]
for i in l:
    print(i, ':', World[i])
while True:
    n = str(input('Input name of country: '))
    ans = int()
    for i in range(0,10):
        ans = l[i]
        if n == l[i]:
            break
        else:
            i += 1; continue
    key1 = 'cw'; key2 = 'ns'; key3 = 'sq'
    print(n,'is  located  in',World[n][key1],'it  has',  World[n][key2],
'population and', World[n][key3])
    z.append({World[n][key1],World[n][key2],World[n][key3]})
    print(z)
    while True:
        try:
          answer=str(input('Do you want to change items ? (y/n) '))
        except:
          raise ValueError
        if answer == 'y':
          coun = str(input('change country: '))
          city = str(input('change part of world: '))
          word = str(input('change population : '))
          sq = str(input('change squer: '))
          World[coun][key1]=city;
          World[coun][key2] = word; World[coun][key3] = sq
        elif answer == 'n':
            for i in l:
                    print(i, ':', World[i])
            break
    break
```

Результат

```
Ukraine : {'cw': 'Europe', 'sq': '6248 KM^2', 'ns': '467591'}
Brazil : {'cw': 'South America', 'sq': '894484 KM^2', 'ns': '123456789'}
USA : {'cw': 'North America', 'sq': '1230540 KM^2', 'ns': '193764825'}
Canada : {'cw': 'North America', 'sq': '69874 KM^2', 'ns': '456852'}
Egypt : {'cw': 'Africa', 'sq': '1488228 KM^2', 'ns': '7891416'}
India : {'cw': 'Asia', 'sq': '987697 KM^2', 'ns': '87864'}
Germany : {'cw': 'Europe', 'sq': '13189 KM^2', 'ns': '154864'}
China : {'cw': 'Asia', 'sq': '9137462 KM^2', 'ns': '987654321'}
Nigeria : {'cw': 'Africa', 'sq': '99648 KM^2', 'ns': '77804'}
Finland : {'cw': 'Europe', 'sq': '81468 KM^2', 'ns': '28613'}
Input name of country: USA
USA is located in North America it has 193764825 population and 1230540 KM^2
[{'193764825', '1230540 KM^2', 'North America'}]
Do you want to change items ? (y/n) y
change country: USA
change part of world: ll
change population : 22
change squer: 33
Do you want to change items ? (y/n) n
Ukraine : {'cw': 'Europe', 'sq': '6248 KM^2', 'ns': '467591'}
Brazil : {'cw': 'South America', 'sq': '894484 KM^2', 'ns': '123456789'}
USA : {'cw': 'll', 'sq': '33', 'ns': '22'}
Canada : {'cw': 'North America', 'sq': '69874 KM^2', 'ns': '456852'}
Egypt : {'cw': 'Africa', 'sq': '1488228 KM^2', 'ns': '7891416'}
India : {'cw': 'Asia', 'sq': '987697 KM^2', 'ns': '87864'}
Germany : {'cw': 'Europe', 'sq': '13189 KM^2', 'ns': '154864'}
China : {'cw': 'Asia', 'sq': '9137462 KM^2', 'ns': '987654321'}
Nigeria : {'cw': 'Africa', 'sq': '99648 KM^2', 'ns': '77804'}
Finland : {'cw': 'Europe', 'sq': '81468 KM^2', 'ns': '28613'}
```

3. БД країн у вигляді словника

```
World = {
 '1': {'co':'Germany','cw':'Europe','ns':'154864','sq':'13189 KM^2'},
 '2': {'co': 'Ukraine','cw': 'Europe','ns': '467591','sq':'6248 KM^2'},
 '3': {'co': 'Egypt','cw': 'Africa', 'ns': '7891416','sq':'1488228
KM^2'},
 '4': {'co': 'Nigeria','cw': 'Africa','ns': '77804','sq':'99648 KM^2'},
 '5': {'co': 'China','cw': 'Asia','ns': '987654321','sq':'9137462
KM^2'},
 '6': {'co': 'USA','cw': 'North America','ns': '193764825','sq':
'1230540 KM^2'},
 '7': {'co': 'Brazil','cw': 'South America','ns':
'123456789','sq':'894484 KM^2'},
 '8': {'co': 'Canada','cw': 'North America','ns': '456852','sq':'69874
KM^2'},
         '9': {'co': 'India','cw': 'Asia','ns': '87864','sq':'987697
KM^2'},
         '10':{'co': 'Finland','cw': 'Europe','ns': '28613','sq':'81468
KM^2'}
         }
l = World.keys();l = list(l); l.sort();
l.remove('10'); l.append('10')
for i in l: print(i, ':', World[i])
W_key = ('1','2','3','4','5','6','7','8','9','10')
key1 = 'cw'; key2 = 'ns'; key3 = 'sq'; key4 = 'co'
while True:
    n = str(input('Input name of country: '))
    ans = 0
    for i in range(0, 10):
        if n == World[W_key[ans]]['co']:
            break
        else:
            ans += 1; i += 1
            continue
    print(World[W_key[ans]][key4],'is located
in',World[W_key[ans]][key1],'it has',
```

```
World[W_key[ans]][key2],'population, and', World[W_key[ans]][key3])
```

Результат

```
1 : {'ns': '154864', 'cw': 'Europe', 'co': 'Germany', 'sq': '13189 KM^2'}
2 : {'ns': '467591', 'cw': 'Europe', 'co': 'Ukraine', 'sq': '6248 KM^2'}
3 : {'ns': '7891416', 'cw': 'Africa', 'co': 'Egypt', 'sq': '1488228 KM^2'}
4 : {'ns': '77804', 'cw': 'Africa', 'co': 'Nigeria', 'sq': '99648 KM^2'}
5 : {'ns': '987654321', 'cw': 'Asia', 'co': 'China', 'sq': '9137462 KM^2'}
6 : {'ns': '193764825', 'cw': 'North America', 'co': 'USA', 'sq': '1230540 KM^2'}
7 : {'ns': '123456789', 'cw': 'South America', 'co': 'Brazil', 'sq': '894484 KM^2'}
8 : {'ns': '456852', 'cw': 'North America', 'co': 'Canada', 'sq': '69874 KM^2'}
9 : {'ns': '87864', 'cw': 'Asia', 'co': 'India', 'sq': '987697 KM^2'}
10 : {'ns': '28613', 'cw': 'Europe', 'co': 'Finland', 'sq': '81468 KM^2'}
Input name of country: USA
USA is located in North America it has 193764825 population, and 1230540 KM^2
Input name of country:
```

Теоретична частина

1. СЛОВНИКИ

Списки – це впорядковані набори об'єктів, на відміну від них елементи в словниках зберігають і витягують за допомогою ключа (а не зсуву, який визначає їхню позицію). Словники (як вбудований тип даних) можуть замінити різні алгоритми пошуку і структур даних. Іноді словники грають роль записів і таблиць символів, здатні служити для подання розріджених (здебільшого порожніх) структур даних. Нижче наведено основні характеристики словників в Python:

1. Доступ до елементів за певним ключем, а не за індексом.
2. Неупорядковані колекції довільних об'єктів.
3. Змінна довжина, гетерогенність і довільна кількість рівнів вкладеності.
4. Відносяться до категорії «змінюваних відображень».
5. Таблиці посилань на об'єкти (хеш-таблиці).

Якщо списки – це масиви посилань на об'єкти, які підтримують можливість доступу до елементів за їхніми позиціями, то словники – це невпорядковані таблиці посилань на об'єкти, які підтримують доступ до елементів за ключем. Усередині словники реалізовані як хеш-таблиці (структури даних, які забезпечують високу швидкість пошуку), спочатку невеликого розміру і збільшуються за необхідності. Інтерпретатор Python використовує оптимізовані алгоритми хешування для забезпечення максимально високої швидкості пошуку ключів. Подібно списками, словники зберігають посилання на об'єкти (а не їх копії).

У табл. 12.1 наведено деякі операції, які найчастіше використовують над словниками (щоб отримати повний перелік операцій, скористайтеся функцією *dir*(dict) або *help*(dict), де *dict* – це ім'я типу). При визначенні у вигляді літералів словники записують як послідовність пар «*key:value*», розділених комами, укладених у фігурні дужки {}.

Списки і словники збільшуються в розмірах по-різному (словники зазвичай доповнюють за допомогою операції привласнення за новими ключами під час виконання програми, такий підхід не годиться для списків, які зазвичай розширюються за допомогою методу *append*).

Таблиця 12.1

Літерали словників та операції

Операция	*Интерпретация*
D = {}	Порожній словник
D = {‘spam’: 2, ‘eggs’: 3}	Словник із двох елементів
D={‘food’:{‘ham’: 1,‘egg’:2}}	Вкладення
D = dict(name=’Bob’, age=40) D = dict(zip(keyslist, valslist)) D = dict.fromkeys([‘a’, ‘b’])	Альтернативні способи створення словників: іменовані аргументи, застосування функції zip, списки ключів
D[‘eggs’] D[‘food’][‘ham’]	Доступ до елементу за ключем
‘eggs’ in D	Перевірка на входження: перевірка наявності ключа
 D.keys() D.values() D.items() D.copy() D.get(key, default) D.get (key [, default]) D1.update(D2) D.pop (key [, default]) D.popitem () D.clear () D.update ([other]) D.setdefault (key [, default])	Методи: визначає список ключів, визначає список значень, визначає пару (ключ, значення). копіювання, отримання значення за замовчуванням , повертає значення ключа; якщо ключа немає, повертає default (за замовчуванням None), злиття, видаляє ключ і повертає значення; якщо ключа немає, повертає default (за замовчуванням видає виняток), видаляє і повертає пару (ключ, значення). Якщо словник порожній, видає виняток KeyError, очищує словник оновлює словник, додаючи пари (ключ, значення) з other, повертає значення ключа, але якщо його немає, створює ключ зі значенням default (за замовчуванням None). .
len(D)	Довжина (кількість елементів)
D[key] = 42 del D[key]	Додавання / редагування ключів, видалення ключів
list(D.keys()) D1.keys() & D2.keys()	Подання словника
D = {x: x*2 for x in range(10)}	Генератори словників
classmethod dict.fromkeys (seq [, value])	створює словник з ключами з seq і значенням value (за замовчуванням None)

Порожній словник в літеральному поданні – це пуста пара дужок {}. Словники можуть вкладатися в середину інших словників, списків або кортежів. Доступ до елементів словника здійснюють за ключем, а для доступу до елементів вкладених словників використовують об'єднання серії індексів (ключів в квадратних дужках) в ланцюжок.

Коли інтерпретатор створює словник, він зберігає елементи в довільному порядку – щоб отримати значення назад, необхідно вказати ключ, з яким це значення було асоційоване. У звичайному випадку спочатку створюють словник, а потім виконують операції збереження нових ключів і звернення до

елементів за ключем:

```
>>> D={'spam': 2, 'ham': 1, 'eggs': 3} # Створення словника
>>> D['spam'] # витягування значення за ключем
2
>>> D # Випадковий порядок розташування
{'eggs': 3, 'ham': 1, 'spam': 2}
```

Тут змінній *D* присвоєно словник, в якому ключу '*spam*' відповідає цілочисельне значення «2». Для доступу до елементів словника використовують той самий синтаксис з квадратними дужками, що і при вилученні елементів списків, але в даному випадку доступ здійснюють за ключем. Остання інструкція в наведеному прикладі означає: порядок проходження ключів в словнику практично завжди відрізняється від початкового. Для забезпечення максимально високої швидкості пошуку за ключем (для хешування) ключі повинні розташовуватися в пам'яті в іншому порядку. Саме тому операції, які передбачають наявність встановленого порядку проходження елементів зліва направо (наприклад, витяг зрізу, конкатенація), непридатні до словників – вони дозволяють отримувати значення лише за ключами, а не за індексами.

Вбудована функція *len* може працювати і зі словниками – вона повертає кількість елементів в словнику (або довжину списку ключів). Оператор *in* перевірки входження дозволяє перевірити наявність ключа, а метод *keys* повертає всі ключі, наявні в словнику, у вигляді списку. Останній зручно використати для послідовної обробки словників, але при цьому ви не повинні робити будь-які припущення про порядок проходження ключів в списку. Оскільки результатом виклику методу *keys* є список, його завжди можна відсортувати:

```
>>> len(D) # кількість елементі словника
3
>>> 'ham' in D # перевірка на входження
True
>>> list(D.keys()) # створює новий список ключів
['eggs', 'ham', 'spam']
```

Оператор перевірки на входження *in* можна використати для роботи з рядками і списками, але точно так само і для роботи зі словниками. Це можливо завдяки тому, що словники визначають ітератори, які забезпечують покроковий обхід списків ключів. Виклик методу *keys* укладений у виклик функції list - list(D.keys()): метод *keys* повертає ітератор, а не список. Виклик функції list примусово виконує обхід всіх значень ітератора, що дозволяє вивести їх всі відразу. Ключі в словниках слідують в довільному порядку, який може змінюватися від версії до версії, тому не потрібно турбуватися, якщо у вас ключі будуть виведені в порядку, відмінному від того, що наведено тут.

Зміна словників. Словники, як і списки, відносяться до категорії змінних об'єктів, тому їх можна змінювати, збільшувати, зменшувати безпосередньо, не створюючи нові словники: щоб змінити або створити новий запис у словнику,

досить виконати операцію присвоювання за ключем. Інструкцію *del* також можна застосовувати до словників – вона видаляє значення, пов'язане з ключем, який грає роль індексу. Зверніть увагу на наявність вкладеного списку в наступному прикладі (значення для ключа '*ham*'). Всі типи-колекції в Python можуть вкладатися один в одного в довільному порядку:

```
>>> D
{'eggs': 3, 'ham': 1, 'spam': 2}
>>> D['ham'] = ['grill', 'bake', 'fry'] # зміна елементу
>>> D
{'eggs': 3, 'ham': ['grill', 'bake', 'fry'], 'spam': 2}
>>> del D['eggs'] # видалення елементу
>>> D
{'ham': ['grill', 'bake', 'fry'], 'spam': 2}
>>> D['brunch'] = 'Bacon' # додавання нового елементу
>>> D
{'brunch': 'Bacon', 'ham': ['grill', 'bake', 'fry'] , 'spam': 2}
```

Операція присвоювання за існуючим ключем словника призводить до зміни асоційованого з ним значення. Словники допускають виконання присвоювання за новим ключем (який раніше був відсутній), в результаті створюють новий елемент словника, як показано в попередньому прикладі для ключа '*brunch*'. Цей прийом не можна застосовувати до списків, бо в цьому випадку інтерпретатор виявляє вихід за межі списку і генерує повідомлення про помилку. Щоб збільшити розмір списку, необхідно використати такі інструменти списків, як метод append або присвоювання зрізу.

Методи словників забезпечують виконання різних операцій. Наприклад, методи словників values і items повертають список значень елементів словника і кортежі пар (*key, value*) відповідно:

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> list(D.values())
[3, 1, 2]
>>> list(D.items())
[('eggs', 3), ('ham', 1), ('spam', 2)]
```

Такі списки зручно використовувати в циклах, коли необхідно виконати обхід елементів словника. Спроба звернутися до неіснуючого елементу словника призводить до появи помилки, однак метод *get* в таких випадках повертає значення за замовчуванням (None або вказане значення). За допомогою цього методу можна реалізувати отримання значень за замовчуванням і уникнути появи помилки при зверненні до неіснуючого ключа:

```
>>> D.get('spam') # Ключ існує в словнику
2
>>> print(D.get('toast')) # Ключ не існує в словнику
None
>>> D.get('toast', 88)
88
```

Метод *update* реалізує операцію конкатенації для словників, при цьому

він не має ніякого відношення до впорядкування елементів зліва направо (для словників таке впорядкування не має сенсу). Він об'єднує ключі і значення одного словника з ключами і значеннями іншого, переписуючи значення з однаковими ключами:

```
>>> D
{'eggs': 3, 'ham': 1, 'spam': 2}
>>> D2 = {'toast':4, 'muffin':5}
>>> D.update(D2)
>>> D
{'toast': 4, 'muffin': 5, 'eggs': 3, 'ham': 1, 'spam': 2}
```

Словники мають набагато більшу кількість методів, ніж перераховано в табл. 12.1. Щоб отримати повний список, звертайтеся до інструкцій з Python.

Використання словників як «записів». Словники в Python здатні грати різні ролі. Вони здатні замінити реалізацію алгоритмів пошуку в структурах (бо операція індексування за ключем вже є операцією пошуку) і можуть подавати різні типи структурованої інформації. наприклад, словники – це один із багатьох способів опису властивостей елементів в програмах, тобто вони можуть грати ту ж роль, яку відіграють «структури» і «записи» в інших мовах програмування.

2. КОРТЕЖІ

Кортежі – це прості групи об'єктів. Вони діють як списки, за винятком того, що не допускають безпосередньої зміни (вони є незмінними) і в літеральній формі записуються як послідовність елементів в круглих дужках. Нижче коротко розглянуто їх властивості. Кортежі:

1. *Це впорядковані колекції об'єктів довільних типів*. Подібно рядкам і спискам, кортежі є колекціями об'єктів, впорядкованих за позицією (тобто вони забезпечують впорядкування свого вмісту зліва направо). Подібно списками, вони можуть містити об'єкти будь-якого типу.

2. *Забезпечують доступ до елементів за зсувом*. Подібно рядкам і списками, доступ до елементів кортежів здійснюють за зсувом (а не за ключем) – вони підтримують всі операції, які засновані на використанні зсуву, такі як індексування і витяг зрізу.

3. *Відносяться до категорії незмінних послідовностей*. Подібно рядкам і спискам, кортежі є послідовностями і підтримують багато операцій над послідовностями. Однак, подібно рядкам, кортежі є незмінними об'єктами, тому вони не підтримують жодну з операцій безпосередньої зміни, які застосовуються до списків.

4. *Мають фіксовану довжину, гетерогенні і підтримують довільну кількість рівнів вкладеності*. Оскільки кортежі є незмінними об'єктами, не можна змінити розмір кортежу, минаючи процедуру створення копії. З іншого боку, кортежі можуть зберігати інші складові об'єкти (тобто списки, словники та інші кортежі), а отже, підтримують довільну кількість рівнів вкладеності.

Масиви посилань на об'єкти. Подібно списками, кортежі простіше подати у вигляді масиву посилань на об'єкти, – кортежі зберігають покажчики

(посилання) на інші об'єкти, а операція індексування над кортежами виконується швидко. У табл. 12.2 наведено часто використовувані операції над кортежами. У програмному коді кортежі записують як послідовність об'єктів, розділених комами, укладену в круглі дужки. Порожні кортежі визначаються як пара порожніх круглих дужок ().

Таблиця 12.2

Літерали кортежів та операції

Операція	Інтерпретація
()	Порожній кортеж
T = (0,)	Кортеж з одного елементу (не вираз)
T = (0, ‘Ni’, 1.2, 3)	Кортеж із чотирьох елементів
T = 0, ‘Ni’, 1.2, 3	Ще один кортеж із чотирьох елементів Як виняток при визначенні кортежів інтерпретатор дозволяє опускати дужки, якщо синтаксично конструкція інтерпретується однозначно. Єдине місце, де круглі дужки є обов'язковими, - при передачі кортежів функціям у вигляді литералов.
T = (‘abc’, (‘def’, ‘ghi’))	Вкладені кортежі
T = tuple(‘spam’)	Створення кортежа із ітерованого об'єкта
T[i] T[i][j] T[i:j] len(T)	Індекс, індекс індексу, зріз, довжина
T1 + T2 T * 3	Конкатенація, повторення
for x in T: print(x) ‘spam’ in t2 [x ** 2 for x in T]	Обхід в циклі, перевірка входження
T.index(‘Ni’) T.count(‘Ni’)	Методы кортежів: пошук, підрахунок входжень

Розглянемо кортежі в дії. Як зазначено в табл. 12.2, кортежі не володіють методами, які є у списків (наприклад, кортежі не мають методу *append*). Проте кортежі підтримують звичайні операції над послідовностями, які застосовуються до рядків і до списків:

```
>>> (1, 2) + (3, 4) # Конкатенація
(1, 2, 3, 4)
>>> (1, 2) * 4 # Повторення
(1, 2, 1, 2, 1, 2, 1, 2)
>>> T = (1, 2, 3, 4) # Індексування, витяг зрізу
>>> T[0], T[1:3]
(1, (2, 3))
```

Особливості синтаксису визначення кортежів: коми і круглі дужки.

Другий і четвертий рядки в табл. 12.2 заслуговують додаткових пояснень. Щоб інтерпретатор розрізняв кортеж від простого виразу в дужках, то якщо необхідно отримати кортеж з одним елементом, потрібно додати кому після цього елемента, перед круглою дужкою, яка закриває:

```
>>> x = (40); x # Ціле число
40
>>> y = (40,); y # Кортеж, який містить ціле число
(40,)
```

Перетворення, методи і незмінюваність. Операції +, * і вилучення зрізу при застосуванні до кортежів повертають нові кортежі, кортежі мають скорочений набір методів. Якщо, наприклад, необхідно впорядкувати вміст кортежу, його спочатку слід перетворити в список, щоб перетворити в змінний об'єкт і отримати доступ до методу сортування або задіяти функцію *sorted*, яка приймає об'єкти будь-яких типів послідовностей (і не тільки):

```
>>> T = ('cc', 'aa', 'dd', 'bb')
>>> tmp = list(T) # Створити список із елементів кортежу
>>> tmp.sort();tmp # Відсортувати список
['aa', 'bb', 'cc', 'dd']
>>> T = tuple(tmp); T # Створити кортеж із елементів списку
('aa', 'bb', 'cc', 'dd')
>>> sorted(T) # або використати  вбудовану функцію sorted
['aa', 'bb', 'cc', 'dd']
```

де *list* і *tuple* – вбудовані функції, які використовують для перетворення в список і потім обернено в кортеж, ці функції створюють нові об'єкти, але завдяки їм створюється ефект перетворення. Для перетворення кортежів можна також використовувати генератори списків. Нижче з кортежу створюють список, причому попутно до кожного елементу додають число 20:

```
>>> T = (1, 2, 3, 4, 5)
>>> L = [x + 20 for x in T]; L
[21, 22, 23, 24, 25]
```

Генератори списків є операціями над послідовностями – вони створюють нові списки, але їх можна використати для обходу вмісту будь-яких об'єктів послідовностей, включаючи кортежі, рядки та інші списки. Кортежі мають обмежений набір методів – *index* і *count*, які діють так само, як методи списків:

```
>>> T = (1, 2, 3, 2, 4, 2) # Методи кортежів
>>> T.index(2) # Перше входження знаходиться в позиції 2
1
>>> T.index(2, 2) # наступне входження за позицією 2
3
>>> T.count(2) # визначити кількість двійок у кортежі
3
```

Правило незмінності застосовують тільки до самого кортежу, але не до об'єктів, які він містить. Наприклад, список всередині кортежу може змінюватися як зазвичай:

```
>>> T = (1, [2, 3], 4)
>>> T[1] = 'spam' # помилка: неможна змінити сам кортеж
TypeError: object doesn't support item assignment
>>> T[1][0] = 'spam'; T
# припустимо: вкладений змінюваний об'єкт можна змінити
```

```
(1, ['spam', 3], 4)
```

Комп'ютерний практикум № 13. Файли даних. Модульний принцип організації програми

Мета роботи: ознайомитися з основними положеннями роботи з файлами на мові Python. *Об'єкт дослідження* – бінарні та текстові файли, модульний принцип організації програми.

ПЛАН

1. Файл
2. ТекстовІ та бінарні файли
3. Модуль

Завдання

1. Вивчити теоретичні основи написання алгоритмів з використанням бінарного та текстового файлів. Опрацювати приклади.

2. Побудувати блок-схему алгоритму вирішення завдання.

3. Відповідно до свого варіанту

- визначити умови; за допомогою формул описати варіанти виконання необхідний дій;
- організувати введення даних з клавіатури, виведення у консоль;
- для завдань 1 і 2 реалізувати такі режими: *a*) створення нового файлу та запис даних у файл, *b*) зчитування даних з файла;
- розмістіть програмний код завдання 2 у двох різних модулях: *mymod* (містить функції) і *my_main*; перевірте свій модуль в інтерактивній оболонці, використовуючи інструкції *import*, *from* та повні імена функцій, які експортуються; перевірте свій модуль на самому собі: наприклад, *test("mymod.py")* – (приклад 13.6).

4. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних.

Варіанти

Завдання 1: бінарні файли. Використовуючи генератор випадкових чисел, заповнити список $[a_1, \ldots, a_n]$ елементами (робота №7.1) та зберегти його у бінарному файлі. Вивести на екран елементи всіх визначених у завданні файлів.

1. Компонентами бінарного файла F1 є цілі числа. Записати у новий бінарний файл F2 всі числа файлу F1, які менші за число *k* (*k* вводиться з клавіатури).
2. Компонентами бінарного файла F1 є цілі числа. Записати у новий бінарний файл F2 всі числа файлу F1, які зустрічаються один раз (порядок слідування чисел зберігається).
3. Компонентами бінарного файла F є цілі числа. Записати у середину файлу F елементи цього ж файцлу, які більші за число К.
4. Компонентами бінарних файлів F1, F2 є цілі числа. Записати у новий бінарний файл F3 спочатку елементи файлу F1, а потім файлу F2.

5. Компонентами бінарного файла F є цілі числа. Записати у початок файлу F непарні елементи цього ж файлу.

6. Компонентами бінарного файла F1 є цілі числа (від'ємні та додатні). Записати у новий бінарний файл F2 всі від'ємні числа з файлу F1, які розміщено на непарних місцях (порядок слідування чисел зберігається).

7. Компонентами бінарних файлів F1, F2 є цілі числа. Переписати зі збереженням порядку слідування елементи файлу F1 у файл F2, а елементи файлу F2 – у файл F1.

8. Компонентами бінарного файла F є цілі числа. Створити програму пошуку послідовності цілих чисел у файлі F, яку задати з клавіатури (зростання, спадання).

9. Компонентами бінарного файла F1 є цілі числа. Всі його парні елементи замінити нулями.

10. Компонентами бінарного файлу F1 є цілі числа (10 з них є додатними, а 10 – від'ємними, розміщені в довільному порядку слідування). Записати у новий бінарний файл F2 всі числа файлу F1: спочатку додатні числа, а потім - від'ємні.

11. Компонентами бінарного файла F1 є цілі числа. Записати у новий бінарний файл F2 всі числа з файлу F1, вилучаючи повторні входження одного і того ж числа.

12. Компонентами бінарного файла F1 є цілі числа (додатні та від'ємні). Записати у новий бінарний файл F2 всі числа з файлу F1, які кратні «5», а у файл F3 – від'ємні числа файлу F1, що кратні «3».

13. Компонентами бінарного файла F є цілі числа. Замінити всі його елементи, порядковий номер яких кратний «7», на нові випадкові значення.

14. Компонентами бінарного файла F1 є цілі числа (додатні та від'ємні). Записати у новий бінарний файл F2 спочатку від'ємні числа з файлу F1, потім всі нулі, а потім всі додатні.

15. Компонентами бінарного файла F є цілі числа. Записати у кінець цього файлу всі його елементи кратні «2».

16. Компонентами бінарного файла F1 є цілі числа (додатні та від'ємні). Записати у новий бінарний файл F2 всі додатні числа файлу F1, розміщені на парних місцях (порядок слідування чисел зберігається).

17. Компонентами бінарного файла F є цілі числа. Записати на початок файлу F елементи, які є дільниками максимального елемента цього ж файлу.

18. Компонентами бінарних файлів F1, F2 є цілі числа. Дописати в початок файлу F1 всі додатні числа файлу F2, а в кінець файлу F2 – від'ємні числа файлу F1 (порядок слідування чисел зберігається).

19. Компонентами бінарного файла F є цілі числа. Поміняти місцями елементи файлу F, які розміщено на парних місцях, з елементами, які розміщено на непарних.

20. Компонентами бінарного файла F1 є цілі числа (додатні та від'ємні). Записати у новий бінарний файл F2 всі числа файлу F1 (за винятком тих, які зустрічаються один раз). Порядок слідування чисел зберігається.

21. Компонентами бінарного файла F є цілі числа. Замінити кожний елемент сумою попередніх, наприкінці дописати загальну суму всіх елементів.

22. Компонентами бінарного файла F1 є цілі числа відмінні від нуля (10 з них є додатними, а 10 – від'ємними, розміщені в довільному порядку слідування). Записати у новий бінарний файл F2 спочатку 5 додатних, потім 5 від'ємних і т.д.

23. Компонентами бінарного файла F1 є цілі числа (додатні та від'ємні). Записати у новий бінарний файл F2 числа файлу F1, які більші за визначене *k*.

24. Компонентами бінарного файла F1 є цілі числа (в тому числі і нулі). Записати у новий бінарний файл F2 всі числа файлу F1 без елемента, розміщеного після першого нуля.

25. Компонентами бінарного файла F є цілі числа. Дописати у початок файлу мінімальний елемент, а в кінець – максимальний.

26. Компонентами бінарного файла F1 є цілі числа. Записати у новий бінарний файл F2 всі парні числа F1, а у файл F3 – всі непарні числа F1 (порядок слідування чисел зберігається).

27. Компонентами бінарного файла F є цілі числа. У середину цього файлу помістити елементи, які кратні «5».

28. Компонентами бінарного файла F є цілі числа. У кінець цього файлу дописати парні елементи цього ж файлу.

29. Компонентами бінарного файла F є цілі числа. Замінити максимальний його елемент сумою попередніх, а мінімальний – сумою наступних елементів.

30. Компонентами бінарних файлів F1, F2 є цілі числа. Зясувати, чи співпадають їхні елементи. Якщо ні, отримати номер першого компонента, яким ці файли різняться один від одного.

Завдання 2: текстові файли. Сформувати файл (або файли) у текстовому редакторі «Блокнот». Маємо текстовий файл (або файли).

1. Переписати його рядки в інший файл. Порядок розташування рядків у другому файлі повинен: а) збігатися з порядком рядків в заданому файлі; б) бути зворотним по відношенню до порядку рядків в заданому файлі.

2. Переписати його рядки в зворотному порядку (справа наліво) в інший файл. Порядок рядків у другому файлі повинен: а) збігатися з порядком рядків в заданому файлі; б) бути зворотним по відношенню до порядку рядків в заданому файлі.

3. Отримати текст, в якому в кінці кожного рядка з заданого файлу доданий знак оклику.

4. Переписати в інший файл ті його рядки, в яких є більше 30-ти символів.

5. Переписати в інший файл всі його рядки з заміною в них символу «0» на символ «1» і навпаки.

6. Всі парні рядки цього файлу записати в другий файл, а непарні – в третій файл. Порядок проходження рядків зберігається.

7. Два файли з однаковою кількістю рядків. Переписати зі збереженням порядку проходження рядки першого файлу в другий, а рядки другого файлу – в перший. Використовувати допоміжний файл.

8. Два файли з однаковою кількістю рядків. З'ясувати, чи співпадають їх рядки. Якщо ні, то отримати номер першого рядка, в якому ці файли відрізняються один від одного.

9. Вивести на екран: а) всі його рядки, які розпочинаються з літери «Т»; б) всі його рядки, які містять більше 30 символів; в) всі його рядки, в яких є більше трьох прогалин; г) всі його рядки, які містять в якості фрагмента заданий текст.

10. Визначити: а) кількість рядків, що починаються з літер «А» чи «а»; б) в яких є рівно п’ять літер «і».

11. Визначити і вивести: а) довжину найдовшого рядка; б) номер найдовшого рядка (якщо таких рядків декілька, то номер першого із них); в) сам найдовший рядок (якщо таких рядків декілька, то перший із них).

12. З'ясувати, чи є в ньому рядок, який розпочинається з літери «Т». Якщо так, то визначити номер першого з таких рядків.

13. Визначити і вивести: а) перший символ першого рядка; б) п'ятий символ першого рядка; в) перші 10 символів першого рядка; г) символи з s1-го по *s*2–й в першому рядку; д) перший символ другого рядка; е) *k*-й символ *n*-го рядка.

14. У кожному рядку заданого файлу перші два символи є літерами. Вивести: а) слово, утворене першими літерами кожного рядка; б) слово, утворене другими літерами кожного рядка; в) послідовність символів, утворену *s*–ми символами кожного рядка.

15. Підрахувати кількість рядків у ньому.

16. Підрахувати кількість символів в ньому.

17. Підрахувати кількість символів в кожному рядку.

18. Видалити з файлу третій рядок. Результат записати в інший файл.

19. Видалити з файлу його останній рядок. Результат записати в інший файл.

20. Видалити з файлу перший рядок, в кінці якого стоїть знак запитання. Результат записати в інший файл.

21. Додати у файл рядок з дванадцяти рисок (----------), розмістивши їх: а) після п'ятого рядка; б) після останнього з рядків, в яких немає прогалини. Якщо таких рядків немає, то новий рядок необхідно додати після всіх рядків наявного файлу. В обох випадках результат записати в інший файл.

22. Елементами файлу є окремі літери слова «кіллобайт». Отримати новий файл, в якому помилки не буде.

23. Елементами файлу є числа. Видалити з нього п'яте число. Результат записати в інший файл.

24. Елементами файлу є цілі числа. Всі парні числа записати в інший файл.

25. Елементами файлу є окремі слова. Записати в інший файл слова, які розпочинаються на літеру «о» або «а».

26. Елементами файлу є цілі числа. Видалити з нього число, записане після першого нуля (нулі у файлі обов'язково присутні). Результат записати в інший файл.

27. Два текстові файли однакового розміру, елементами яких є числа. Отримати третій файл, кожен елемент якого дорівнює: а) сумі відповідних елементів заданих файлів; б) більшому із відповідних елементів заданих файлів.

28. Два текстові файли однакового розміру, елементами яких є числа. Отримати третій файл, кожен елемент якого дорівнює: а) різниці відповідних елементів заданих файлів; б) меншому з відповідних елементів заданих файлів.

29. Два текстові файли однакового розміру, елементами яких є окремі літери. Отримати третій файл, кожен елемент якого є поєднанням відповідних літер першого і другого файлів.

30. Два текстові файли однакового розміру, елементами яких є окремі літери. Записати в третій файл всі співпадаючі елементи наявних файлів.

31. Елементами файлу є окремі символи (цифри та літери). Всі цифри цього файлу записати в другій файл, а решта символи – в третій файл. Порядок слідування зберігається.

Контрольні запитання

1. Визначити поняття «файл».

2. Назвіть види файлів, які використовують у Python.

3. Який файл називається текстовим? Яка максимальна довжина рядка в текстовому файлі? Чому в текстовому файлі використовують ознаку кінця рядка?

4. Назвіть інструкції створення, закриття та відкриття файлів. Чи можна текстовий файл відкрити одночасно для зчитування та запису?

Аудиторна робота

Приклади 13.1: файл

1. Відкриття файлу для зчитування

```
def read_file(fname):
 """Функція для зчитування файлу fname
 та виведення його вмісту на екран"""
 file=open(fname, 'r') # відкриття файлу для зчитування
 print('File '+fname+':') # виведення назви файлу
 # Зчитування вмісту файлу по рядкам
 for line in file:
  # Виведення рядка s
   print(line, end='')
file.close() # Закриття файлу
if __name__=='__main__':  read_file('data/file.txt')
```

Результат

```
File data/file.txt:
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Cum,
```

```
dicta.
Nisi culpa beatae quaerat vitae consequatur rem distinctio fugiat,
blanditiis.
Nostrum nobis inventore ipsa distinctio doloremque maxime tempore
minus quos.
Quisquam repellat, ab corporis odio impedit officiis possimus
magni similique.
Modi, tenetur reiciendis dolor ut officiis repellendus totam
tempore deserunt.
```

2. Використання функції *os.path.join* для побудови шляху до файлу

```
# Модуль, який містить функції для роботи з шляхом у файловій
системі
import os.path

def read_file(fname):
      """Функція для зчитування файла fname
      та виведення його вмісту на екран"""
      file=open(fname,'r') # відкриття файлу для зчитування
      print('File '+fname+':') # виведення назви файлу
      # зчитування вмісту файла по рядках
      for line in file:
       print(line, end='')  # виведення рядка s
      file.close() # закриття файлу

if __name__=='__main__':
 # функція os.path.join з'єднує частини шляху у файловій системі
 # необхідним роздільником
   read_file(os.path.join('data', 'file.txt'))
```

3. Запис даних у текстовий файл

```
import os.path

text='''Hello!
I am a text file. And I had been written with a Python script
before you opened me, so look up the docs and try to delete
me using Python, too.'''

def write_text_to_file(filename, text):
 """Функція для запису у файл filename рядка text"""
 f=open(filename, "w")# відкриття файла для запису
 f.write(text) # Запис рядка text у файл
 f.close()# Закриття файлу

if __name__=='__main__':
 write_text_to_file(os.path.join('data','example02.txt'),text)
```

4. Використання оператора *with* для закриття файлу

```
import os.path

filename=os.path.join('data','file.txt') #побудова імені файлу
# Оператор with закриє файл по закінченню виконання
```

```
# операторів усередині нього або виникненні виключення
with open(filename) as file: print(file.read())
```

5. Відкриття текстового файла для зчитування з вказівкою кодування

__file__ – це атрибут модуля, в якому зберігається ім'я файлу його його вихідного коду

```
with open(__file__, 'r', encoding='utf-8-sig') as file:
  for number, line in enumerate(file):
   print('{0}\t{1}'.format(number+1,line),end='')
print()
```

6. Відкриття файлу для зчитування та запису

```
import os.path
import statistics
import datetime
def calculate_stats(filename):
  with open(filename, 'r+') as file:
    numbers=[float(line) for line in file.readlines()
      if line != '\n' and not line.lstrip().startswith('#')]

        sum_ = sum(numbers)
        mean = statistics.mean(numbers)
        median = statistics.median(numbers)
        cur_time = datetime.datetime.now()
        fmt = '\n' \
              '# Статистика від {time!s}\n' \
              '# Сума:    {sum}\n' \
              '# Медіана: {median}\n' \
              '# Середнє: {mean}'

        print(fmt.format(time=cur_time,
                         mean=mean,
                         median=median,
                         sum=sum_),
              file=file)
if __name__ == '__main__':
    filename = os.path.join('data', 'example05.txt')
    calculate_stats(filename)
```

7. Відкриття файла для дозапису

```
import os.path
import datetime

log_file=os.path.join('data', 'ex06_log.txt')
with open(log_file, 'a') as log:
    print(datetime.datetime.now(), file=log)
```

8. Перезапис файла

```
import os.path
filename=os.path.join('data', 'example07.txt')
# Зчитування файла
with open(filename, 'r') as file:
```

```
    lines=file.readlines()
# Модифікація даних
lines.insert(2, 'inserted line\n')
# Перезапис файла
with open(filename, 'w') as file:
    file.writelines(lines)
```

9. Використання файлового об’єкта *io.StringIO*

```
import io

# Створення потоку
stream=io.StringIO()  # или io.StringIO('початкове значення')
stream.write('asdf in memory') # Запис даних у потік

# отримання рядка із об’єкта StringIO
print(stream.getvalue())

# Виведення поточної позиції
print('Current position:', stream.tell())
stream.seek(0) # Перехід на початок потоку
stream.write('data') # Запис даних у потік

# Виведення поточної позиції
print('Current position:', stream.tell())
print(stream.read())# Зчитування даних, які є в потоці

# Виведення поточної позиції
print('Current position:', stream.tell())
# отримання рядка із об’єкта StringIO
print(stream.getvalue())
```

Результат

```
asdf in memory
Current position: 14
Current position: 4
 in memory
Current position: 14
data in memory
```

10. Використання бінарного файлу

```
from array import array
import os.path
prefix=os.path.join('data', 'ex09_')
numbers = list(range(300, 400)) # отримання списку чисел
# Запис у текстовий файл
with open(prefix+'text.txt','w') as txt_file:
   print(numbers, file=txt_file)
# Створення масива, який підтримує buffer_protocol, із списку
numbers_array=array('i', numbers)
# Запис у бінарний файл
binary_filename=prefix+'binary.bin'
with open(binary_filename, 'wb') as bin_file:
    bin_file.write(numbers_array)
```

```
# Підготовка масиву
filesize=os.path.getsize(binary_filename)  # розмір файла
int_len=array('i').itemsize # розмір одного елемента в байтах
read_array=array('i',(0 for _ in range(filesize // int_len)))

# Зчитування із бінарного файла
with open(binary_filename, 'rb') as file:
    file.readinto(read_array)  # зчитування у масив
print(read_array) # Виведення масиву на екран
# Перевірка, чи зчитані дані відповідають початковим
print(read_array.tolist() == numbers)
```

Результат

```
array('i', [300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310,
311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323,
324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336,
337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,
350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362,
363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375,
376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388,
389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399])
True
```

11. Використання *json*. Серіалізація – десеріалізація.

Серіалізація – процес перетворення будь-якої структури даних у послідовність бітів; зворотною до операції серіалізації є операція *десеріалізації* – відновлення початкового стану структури даних з бітової послідовності.

Серіалізацію використовують для передачі об'єктів по мережі та для збереження їх у файли. Наприклад, потрібно створити розподілений додаток, різні частини якого мають обмінюватися даними зі складною структурою. У цьому випадку для типів даних, які передбачають передавати, пишеться код, який здійснює серіалізацію і десеріалізацію. Об'єкт заповнюється потрібними даними, потім викликають код серіалізації, в результаті виходить, наприклад, XML-документ. Результат серіалізації передається приймаючій стороні, наприклад, по електронній пошті або HTTP. Додаток-одержувач створює об'єкт того ж типу і викликає код десеріалізації, у результаті отримують об'єкт з тими ж даними, що були в об'єкті програми-відправника. За такою схемою працює, наприклад, серіалізація об'єктів через SOAP в Microsoft.NET.

```
import json
import os.path

data = [{
        'name': 'John',
        'age': 20,
    },
    {
        'name': 'Mary',
        'age': 19
    }]
filename = os.path.join('data', 'example10.json')
# Серіалізація
```

```
with open(filename, 'w') as file:
    json.dump(data, file)
# Десеріалізація
with open(filename, 'r') as file:
    read_data = json.load(file)
print(read_data)
```

Результат

```
[{'name': 'John', 'age': 20}, {'name': 'Mary', 'age': 19}]
```

12. Серіалізація за допомогою *pickle*. Декоратор

Декоратор reprlib.recursive_repr(fillvalue='...') – структурний шаблон проектування, призначений для динамічного підключення додаткових можливостей до об'єкта. Шаблон *Декоратор* надає гнучку альтернативу методу визначення підкласів з метою розширення функціональності.

```
import os.path
import pickle
import reprlib
class Person(object):
    """Клас, який описує людину"""
    def __init__(self, name, age, sibling=None):
        """Конструктор класу. Параметри:
            name    -- ім’я
            age     -- вік
            sibling -- брат або сестра  """
        self.name = name; self.age = age
        self.sibling = sibling
# Декоратор відслідковує рекурсивні виклики методу __repr__
# і не дає йому увійти в нескінченну рекурсію,
# повертаючи fillvalue замість викликів даного методу, які ще не завершені
    @reprlib.recursive_repr()
    def __repr__(self):
      """Подання об’єкта у рядках"""
      return 'Person({name!r}, {age!r}, {sibling!r})'.format(**self.__dict__)

def write_data(filename):
    """Функція створення і запису даних"""
    james=Person('James', 20); julia=Person('Julia', 21)
    james.sibling=julia  # створення циклічних посилань
    julia.sibling=james

    # Серіалізація списку об'єктів
    with open(filename, 'wb') as file:
    # 'wb' - запис бінарного файлу
        pickle.dump([james, julia], file)

def read_data(filename):
    """Функція зчитування і виведення даних на екран"""
    with open(filename, 'rb') as file:
        data = pickle.load(file)
```

```
        print(data)    # Виведення у консоль

    if __name__ == '__main__':
        filename = os.path.join('data', 'example11.pkl')
        write_data(filename); read_data(filename)
```

Результат

```
[Person('James',    20,    Person('Julia',    21,    ...)),
Person('Julia', 21, Person('James', 20, ...))]
```

Приклад 13.2: текстовий файл

Відкриття нового текстового файлу в режимі запису: в нього записують два рядки (що завершуються символом кінця рядка \n), після чого файл закривають. Далі цей файл відкривають в режимі зчитування і виконують зчитування рядків з нього. Третій виклик методу *readline* повертає порожній рядок – у такий спосіб методи файлів в Python повідомляють, що було досягнуто кінець файлу (порожній рядок у файлі повертають як рядок, що містить один символ нового рядка \n, а не як дійсно порожній рядок). Нижче наведено повний лістинг сеансу:

```
>>> myfile = open('myfile.txt', 'w')
# відкриття файлу (створення/очищення)
>>> myfile.write('hello text file\n') #Запис рядку тексту
16
>>> myfile.write('goodbye text file\n')
18
>>> myfile.close() # Виштовхує вихідні буфери на диск
>>> myfile = open('myfile.txt')
# відкриває файл: 'r' - за замовчуванням
>>> myfile.readline() # зчитування рядку
'hello text file\n'
>>> myfile.readline()
'goodbye text file\n'
>>> myfile.readline() # Пустий рядок: кінець файлу
```

Метод *write* повертає кількість записаних символів. Цей приклад записує два рядки тексту у файл, додаючи до кожного з них символ кінця рядка \n; методи запису не додають символ кінця рядка, тому необхідно самостійно додавати його у рядки, що виводяться (в іншому випадку наступна операція запису продовжить поточний рядок у файлі). Якщо необхідно вивести вміст файлу, забезпечивши правильну інтерпретацію символів кінця рядка, файл слід прочитати в рядок цілком, за допомогою методу *read*, і вивести:

```
>>> open('myfile.txt').read() #Прочитати файл в рядок цілком
'hello text file\ngoodbye text file\n'
>>> print(open('myfile.txt').read())
hello text file
goodbye text file
```

Якщо необхідно переглянути вміст файлу рядок за рядком, кращим вибором буде ітератор файлу:

```
    >>> for line in open('myfile'):
```

```
... print(line, end='')
hello text file
goodbye text file
```

Тут функція *open* створює тимчасовий об'єкт файлу, вміст якого автоматично буде зчитуватися ітератором і повертатися по одному рядку під час кожної ітерації циклу.

Приклад 13.3: текстовий файл

1. Записати різні об'єкти в текстовий файл

Дані завжди записують у файл у вигляді рядків, а методи запису не виконують форматування рядків:

```
>>> X,Y,Z = 43, 44, 45 # Об'єкти Python повинні записуватися
>>> S = 'Spam' # у файл тільки в вигляді рядків
>>> D = {'a': 1, 'b': 2}
>>> L = [1, 2, 3]
>>> F = open('datafile.txt', 'w') # Створює файл для запису
>>> F.write(S + '\n') # Рядки завершують символом \n
>>> F.write('%s,%s,%s\n' % (X, Y, Z)) # Перетворює числа в рядки
>>> F.write(str(L) + '$' + str(D) + '\n')
# Перетворює і розділяє символом $
>>> F.close()
```

2. Створивши файл, можна досліджувати його вміст (відкривши файл і прочитавши дані в рядок). Функція виведення дає побайтове подання вмісту, а інструкція *print* інтерпретує вбудовані символи кінця рядка, щоб забезпечити легке для зчитування відображення:

```
>>> chars = open('datafile.txt').read() # відображення рядка
>>> chars # у неформатованому вигляді
"Spam\n43,44,45\n[1, 2, 3]${'a': 1, 'b': 2}\n"
>>> print(chars) # подання, яке зручно читати
Spam
43,44,45
[1, 2, 3]${'a': 1, 'b': 2}
```

Тепер необхідно виконати зворотні перетворення, щоб отримати з рядків у текстовому файлі об'єкти мови. Інтерпретатор Python не виконує перетворення рядків у числа або в об'єкти інших типів, тому необхідно виконати відповідні перетворення, щоб можна було використати операції над цими об'єктами, такі як індексування, додавання тощо:

```
>>> F = open('datafile.txt') # відкрити файл знову
>>> line = F.readline() # Прочитати один рядок
>>> line
'Spam\n'
>>> line.rstrip() # видалити  символ кінця рядка
'Spam'
```

Тут застосовано метод *rstrip*, щоб видалити символ кінця рядка. Цей результат можна отримати за допомогою вилучення зрізу line [:– 1], але такий підхід можна використовувати, якщо ви впевнені, що всі рядки завершуються символом \n (останній рядок в файлі іноді може не містити цей символ). Ми

прочитали частину файлу, який містить рядок. Тепер прочитаємо наступний блок, в якому містяться числа, і виконаємо розбиття цього блоку (тобто виокремимо об'єкти):

```
>>> line = F.readline() # наступний рядок з файлу
>>> line # це - рядок
'43,44,45\n'
>>> parts=line.split(',') # розбити на підрядки за комами
>>> parts
['43', '44', '45\n']
```

Тут використано метод *split*, щоб розбити рядок на частини за комами (кома – символ-роздільник). В результаті отримано список рядків, кожний з яких містить окреме число. Ці рядки необхідно перетворити в цілі числа, щоб виконувати математичні операції над ними:

```
>>> int(parts[1]) # Перетворити рядок у ціле число
44
>>> numbers = [int(P) for P in parts] # Перетворити весь список
>>> numbers
[43, 44, 45]
```

Функція *int* перетворює рядок цифр в об'єкт цілого числа, а генератор списків дозволяє застосувати функцію до всіх елементів списку в одній інструкції. Для видалення символа завершення \n в кінці останнього підрядка не застосовано метод *rstrip* (*int* та інші функції перетворення ігнорують символи-роздільники, які оточують цифри). Щоб перетворити список і словник в третьому рядку файлу, можна скористатися вбудованою функцією *eval*, яка інтерпретує рядок як програмний код на Python:

```
>>> line = F.readline()
>>> line
"[1, 2, 3]${'a': 1, 'b': 2}\n"
>>> parts = line.split('$') #Розбити на рядки за символом $
>>> parts
['[1, 2, 3]', "{'a': 1, 'b': 2}\n"]
>>> eval(parts[0]) # Перетворити рядок у об'єкт
[1, 2, 3]
>>> objects=[eval(P) for P in parts]
# Теж саме для всіх рядків у списку
>>> objects
[[1, 2, 3], {'a': 1, 'b': 2}]
```

Оскільки тепер всі дані зображують собою список звичайних об'єктів, можна застосувати до них операції списків і словників.

Приклад 13.4. Сканування файлів.

Файли містять множину символів і рядків, тому їх можуть розглядати як один з типових об'єктів використання циклів. Щоб завантажити вміст файлу в рядок однією інструкцією, досить викликати метод *read*:

```
file = open('test.txt','r') #Прочитати вміст файлу в рядок
print(file.read())
```

Для завантаження файлу по частинах використовують або цикл *while*, який завершують інструкцією *break* після досягнення кінця файлу, або цикл *for*:

```
# цикл for обробляє окремо кожний символ, але завантаження вмісту
файлу в пам'ять виконують одноразово
     file = open('test.txt')
     while True:
          char = file.read(1) # зчитувати по одному символу
          if not char: break
          print(char)
     for char in open('test.txt').read():
          print(char)

# цикл while реалізує зчитування рядками або блоками
file = open('test.txt')
while True:
     line = file.readline() # зчитувати рядок за рядком
     if not line: break
     print(line, end=' ')
     # Прочитаний рядок вже містить символ \n
file=open('test.txt', 'rb')
while True:
     chunk = file.read(10) # зчитувати блоками по 10 байтів
     if not chunk: break
     print(chunk)
```

Двійкові дані зчитують блоками певного розміру. Однак в разі текстових даних зитування рядками за допомогою циклу *for* працює швидше:

```
for line in open('test.txt').readlines():
     print(line, end='')
for line in open('test.txt'):
# використання ітератора: найкращий спосіб  зчитування тексту
     print(line, end='')
```

Метод файлів *readlines* завантажує в список рядків цілий файл, тоді як при використанні ітератора файлу в кожній ітерації завантажують один рядок. Останній приклад – це найкращий спосіб роботи з текстовими файлами, він простіше і здатний працювати з файлами будь-якого розміру, тому що не завантажує цілий файл у пам'ять. Текстові файли за замовчуванням в процесі запису і зчитування відображають символи кінця рядка «\n», і виконують перетворення символів Юнікоду відповідно до кодування.

Приклад 13.5. Коли виконують операцію зчитування двійкових даних з файлу, вона повертає об'єкт типу *bytes* – послідовність коротких цілих чисел, що подають абсолютні значення байтів. Цей об'єкт нагадує звичайний рядок:

```
     >>> data=open('data.bin', 'rb').read()
     # відкриття двійкового файлу для зчитування
     >>> data # рядок bytes зберігає двійкові дані
     b'\x00\x00\x00\x07spam\x00\x08'
     >>> data[4:8] # веде себе як рядок
     b'spam'
     >>> data[4:8][0]
```

```
# Але в дійсності зберігає 8-бітові цілі числа
115
>>> bin(data[4:8][0]) # Функція bin()
'0b1110011'
```

Приклад 13.6. Скласти програму, яка підраховує кількість рядків і символів у файлі. В текстовому редакторі створіть модуль з ім'ям *mymod.py*, який експортує три імені: функцію

а) *countLines (name)*, яка зчитує вхідний файл і підраховує кількість рядків в ньому (підказка: більшу частину роботи можна виконати за допомогою методу *file.readlines*, а решту – за допомогою функції *len*).

б) *countChars (name)*, яка зчитує вхідний файл і підраховує кількість символів в ньому (підказка: метод *file.read* повертає один рядок).

в) *test(name)*, яка викликає дві попередні функції з заданим ім'ям файлу.

Ім'я файлу передають як аргумент функції. Всі три функції в модулі *mymod* повинні приймати ім'я файлу у вигляді рядка. Якщо розмір будь-якої з функцій перевищить два-три рядки, це означає, що ви робите зайву роботу.

Перевірте свій модуль в інтерактивній оболонці, використовуючи інструкцію *import* і повні імена експортованих функцій. Чи слід додати в змінну PYTHONPATH каталог, де знаходиться файл *mymod.py*? Спробуйте перевірити модуль, наприклад, таким чином *test*(*"mymod.py"*) – функція *test* відкриває файл двічі (спробуйте оптимізувати код, передаючи двом функціям підрахунку об'єкт відкритого файлу (підказка: метод *file.seek*(0) виконує переустановку вказівника на початок файлу).

Розв'язання. Необхідно сформувати файл *mymod.py* і сеанс взаємодії з інтерактивною оболонкою, як показано нижче (інтерпретатор Python може зчитувати вміст файлу цілком в список рядків, а отримати довжину кожного рядка в списку можна за допомогою вбудованої функції *len*):

mymod.py

```
def countLines(name):
 file=open(name)
 return len(file.readlines())
def countChars(name):
 return len(open(name).read())
def test(name): # або передати об'єкт файла
 return countLines(name),countChars(name) #або повернути словник
```

#main

```
>>> import mymod
>>> mymod.test('mymod.py')
(10, 291)
```

Ці функції зчитують цілий файл в пам'ять, а тому не в змозі працювати з великими файлами, які не можна вмістити в пам'яті комп'ютера (можна організувати зчитування вмісту файлу по рядках за допомогою ітератора і додавати довжину рядків кожного разу):

```
def countLines(name):
 tot = 0
```

```
 for line in open(name): tot += 1
 return tot
def countChars(name):
 tot = 0
 for line in open(name): tot += len(line)
 return tot
```

У Windows можна клацнути на файлі правою кнопкою миші і подивитися його властивості. Результат, який повертає сценарій, може відрізнятися від того, що повідомляє Windows, – інтерпретатор Python перетворює пару символів \r \n, які позначають кінець кожного рядка, в один символ \n, в результаті чого втрачають по одному байту (символу) на кожен рядок. Щоб результат сценарію в точності відповідав тому, що повідомляє Windows, файл необхідно відкрити в режимі двійкового доступу ('*rb*') або додавати до загального результату кількість рядків.

Щоб передати функціям об'єкт файлу для його відкриття лише один раз, використовують метод *seek* об'єкта файлу: він переустановлює поточну позицію в файлі у вказаний зсув. Після виклику методу *seek* наступні операції введення/виведення будуть виконуватися щодо нової позиції. Щоб переміститься у початок файлу, не закриваючи і не відкриваючи його повторно, можна викликати метод *file.seek*(0). Всі виклики методу *read* виконують зчитування з поточної позиції у файлі, тому, щоб розпочати повторне зчитування, необхідно перемістити поточну позицію на початок файлу. Нижче показано, як виглядає така реалізація:

```
def countLines(file):
 file.seek(0) # Перемістится на початок файлу
 return len(file.readlines())
def countChars(file):
 file.seek(0) # Те ж саме(перемістится на початок)
 return len(file.read())
def test(name):
 file = open(name) # Передати об’єкт файла
 return countLines(file), countChars(file)
# відкрити файл один раз
```

#main

```
>>> import mymod2
>>> mymod2.test("mymod2.py")
(11, 392)
```

Теоретичні відомості

1. ФАЙЛ

Файл – це іменована область пам'яті в комп'ютері, якою управляє операційна система (довільна послідовність елементів одного типу, довжина цих послідовностей заздалегідь не визначається, а конкретізується в процесі виконання програми; дані, що містяться у файлі, переносять на зовнішні носії).

Текстові файли прізначені для зберігання текстової інформації. Компоненти текстових файлів можуть мати змінну довжину.

Вбудована функція *open* створює об'єкт файлу, який забезпечує зв'язок з файлом, розміщеним в комп'ютері. Після виклику цієї функції можна виконувати операції зчитування і запису в зовнішній файл, використовуючи методи отриманого об'єкта. Об'єкти файлів не є ні числами, ні послідовностями або відображеннями – для роботи з файлами вони надають тільки методи. Більшість методів файлів пов'язані з виконанням операцій введення–виведення у зовнішні файли, асоційовані з об'єктом. У табл. 13.1 наведено операції над файлами, які найчастіше використовують.

Таблиця 13.1

Операції над файлами, які найчастіше використовують

Операція	Інтерпретація
output = open(r'C:\spam', 'w')	Відкриває файл для запису ('w' означає write – запис)
input = open('data', 'r')	Відкриває файл для зчитування ('r' означає read – зчитування)
input = open('data')	Відкриває файл для зчитування (режим 'r' використовують за замовчуванням)
aString = input.read()	Зчитування файлу цілком в один рядок
aString = input.read(N)	Зчитування наступних N символів (або байтів) в рядок
aString = input.readline()	Зчитування наступного текстового рядка (включаючи символ кінця рядка) в рядок
aList = input.readlines()	Зчитування файлу цілком в список рядків (включаючи символ кінця рядка)
output.write(aString)	Запис рядка символів (або байтів) у файл
output.writelines(aList)	Запис всіх рядків зі списку в файл
output.close()	Закриття файлу вручну (виконується після закінчення роботи з файлом)
output.flush()	Виштовхує вихідні буфери на диск, файл залишається відкритим
anyFile.seek(N)	Змінює поточну позицію в файлі для наступної операції, зсовуючи її на N байтів від початку файлу.
for line in open('data'): *операции над line*	Ітерації по файлу, зчитування по рядках
open('f.txt', encoding='latin-1')	Файли з текстом Юнікоду (рядки типу str)
open('f.bin', 'rb')	Файли з двійковими даними (рядки типу bytes)

Відкриття файлів. Щоб відкрити файл, програма повинна викликати функцію *open*, передавши їй ім'я зовнішнього файлу і режим роботи: як режим використовують рядок '*r*', якщо файл відкривають для зчитування (за замовчуванням), '*w*' – якщо файл відкривають для запису або '*a*' – для запису в кінець. У рядку режиму можна також зазначати інші параметри: додавання символу в рядок режиму означає 1) «*b*» – роботу з двійковими даними (відключають інтерпретацію символів кінця рядка і кодування символів Юнікоду); 2) «+» – файл відкривають для зчитування і для запису (є можливість зчитувати і записувати дані в один і той же об'єкт файлу, часто спільно з операцією позиціонування в файлі).

Обидва аргументи функції *open* повинні бути рядками. Крім того, функція може приймати третій необов'язковий аргумент, керуючий процесом буферизації виведених даних, – значення нуль означає, що вихідна інформація не буде буферизована (вона буде записуватися у зовнішній файл відразу ж, в момент виклику методу запису). Ім'я зовнішнього файлу може включати шлях до файлу, якщо шлях до файлу не вказано, передбачають, що файл розміщено в поточному робочому каталозі (тобто в каталозі, де був запущений сценарій).

Використання файлів. Як тільки отримано об'єкт файлу, можна викликати його методи для виконання операцій зчитування або запису.

Наведемо кілька основних зауважень щодо використання файлів:

1. *Для зчитування рядків краще використовувати ітератори файлів*.

Найкращий, мабуть, спосіб зчитування рядків з файлу на сьогоднішній день полягає в тому, щоб взагалі не використовувати операцію зчитування з файлу: файли мають ітератор, який автоматично зчитує інформацію з файлу рядок за рядком в контексті циклу *for*, в генераторах списків і в інших ітераційних контекстах.

2. *Вміст файлів знаходиться в рядках, а не в об'єктах.* Зверніть увагу: в табл. 13.1 показано, що дані, отримані з файлу, завжди потрапляють в сценарій у вигляді рядка. Якщо ця форма подання не підходить, необхідно виконати перетворення даних в інші типи об'єктів мови Python, а при виконанні операції запису даних у файл необхідно передавати методам сформовані рядки.

Тому при роботі з файлами треба згадати інструменти перетворення даних з рядка у число і навпаки (наприклад, *int, float, str*, а також вирази форматування рядків і метод *format*). Крім того, до складу Python входять додаткові стандартні бібліотечні інструменти, призначені для роботи з універсальним об'єктом «сховище даних» (наприклад, модуль *pickle*) і обробки упакованих двійкових даних у файлах (наприклад, модуль *struct*).

3. Виклик методу *close* є необов'язковим, він розриває зв'язок із зовнішнім файлом. Інтерпретатор Python негайно звільняє пам'ять, зайняту об'єктом, як тільки в програмі буде загублена останнє посилання на цей об'єкт. Як тільки об'єкт файлу звільняють, інтерпретатор закриває асоційований з ним файл (що відбувається також в момент завершення програми). Завдяки цьому не потрібно закривати файл вручну. З іншого боку, виклик методу *close* не зашкодить, і його рекомендують використовувати у великих системах (в момент закриття файлів звільняються ресурси операційної системи і виштовхуються вихідні буфери).

4. *Файли забезпечують буферизацію введення-виведення і дозволяють виробляти позиціонування у файлі.* За замовчуванням виведення у файли завжди виконують за допомогою проміжних буферів, тобто в момент запису тексту у файл він не потрапляє відразу ж на диск – буфери виштовхуються на диск тільки в момент закриття файлу або при виклику методу *flush*. Можна відключити механізм буферизації за допомогою додаткових параметрів функції *open*, але це може призвести до зниження продуктивності операцій введення-виведення. Файли в Python підтримують і можливість позиціонування – метод

seek дозволяє сценаріями управляти позицією зчитування і запису.

2. ТЕКСТОВІ ТА БІНАРНІ (ДВІЙКОВІ) ФАЙЛИ

В Python тип файлу визначається другим аргументом функції *open* – символ «*b*» в рядку режиму означає *binary* (двійковий). В Python існує підтримка текстових і двійкових файлів, але між цими двома типами файлів проведена чітка межа:

1. Вміст текстових файлів подають у вигляді звичайних рядків типу *str*, при цьому виконується автоматичне кодування/декодування символів Юнікоду, за замовчуванням проводиться інтерпретація символів кінця рядка.

2. Вміст бінарних файлів подають у вигляді рядків типу *bytes*, він передається програмі без будь-яких змін.

Звичайний текст і текст в Юнікоді об'єднані в один тип «рядок», адже будь-який текст можна подати в Юнікоді, включаючи ASCII та інші 8-бітові кодування. Більшості програмістам доводиться мати справу тільки з текстом ASCII, тому вони можуть користуватися базовим інтерфейсом доступу до текстових файлів і звичайними рядками. Всі рядки в Python 3.X є рядками Юнікоду, але для тих, хто використовує тільки символи ASCII, ця обставина зазвичай залишається непоміченою.

Для роботи з двійковими файлами слід використовувати рядки *bytes*, а звичайні рядки *str* – для роботи з текстовими файлами. Крім того, тому що текстові файли реалізують автоматичне перетворення символів Юнікоду, ви не зможете відкрити файл з двійковими даними в текстовому режимі – перетворення його вмісту в символи Юнікоду, швидше за все, завершиться з помилкою.

Збереження об'єктів Python за допомогою модуля pickle. Функція *eval* виконає будь-який вираз на мові Python. Якщо необхідно витягувати з файлів об'єкти Python, але не можна довіряти джерелу цих файлів, ідеальним рішенням є використати модуль *pickle*, який дозволяє зберігати у файлах будь-які об'єкти Python без необхідності виконувати їхнє перетворення. Щоб зберегти словник у файлі, наприклад, передамо його безпосередньо в функцію модуля *pickle*:

```
D = {'a': 1, 'b': 2}
F = open('datafile.pkl', 'wb')
import pickle
pickle.dump(D, F) # Модуль pickle запише у файл будь-який об'єкт
F.close()
```

Щоб потім прочитати словник назад, можна скористатися можливостями модуля *pickle*:

```
>>> F = open('datafile.pkl' 'rb')
>>> E = pickle.load(F) # Завантажує будь-які об'єкти з файлу
>>> E
{'a': 1, 'b': 2}
```

Отримали назад об'єкт–словник без необхідності виконати перетворення. Модуль *pickle* виконує перетворення об'єктів в рядок байтів і назад (тут

виконано перетворення словника в рядок):

```
>>> open('datafile.pkl', 'rb').read()
# Формат можна змінити
b'\x80\x03}q\x00(X\x01\x00\x00\x00aq\x01K\x01X\x01\x00\x00\x00bq\x
02K\x02u.'
```

У наведеному прикладі був відкритий файл, де зберігається об'єкт у двійковому режимі. В Python такі файли завжди слід відкривати саме в дійковому режимі, тому що модуль *pickle* створює і використовує об'єкти типу *bytes*. Модуль *shelve* – інструмент, який використовує модуль *pickle* для збереження об'єктів Python в файлах з доступом по ключу.

3. МОДУЛЬ

Для створення великих систем Python надає такі можливості, як модулі. Вони дозволяють розбити систему на складові. Термін «модуль» зарезервований для позначення файлів, які можуть імпортуватися іншими файлами. Кожен файл з вихідним текстом на мові Python, ім'я якого закінчується розширенням *.py*, є модулем. Інші файли можуть звертатися до програмних компонентів, які декларуються модулем, імпортуючи цей модуль. Інструкція *import* виконує завантаження іншого файлу і забезпечує доступ до його вмісту. Вміст модуля стає доступним зовнішнього світу через його атрибути. Така модульна модель є центральною ідеєю, яка лежить в основі архітектури програм на Python. Великі програми зазвичай організовані у вигляді множини файлів модулів, які імпортують і використовують функціональні можливості з інших модулів.

Один з модулів визначається як основний файл верхнього рівня, який запускає всю програму. Операція імпорту призводить до виконання програмного коду завантаження. Як наслідок, імпорт файлу є способом запустити його. Операція імпорту вимагає великих витрат обчислювальних ресурсів (в ході імпорту проводиться пошук файлів, компіляція їх в байт-код і виконання цього байт-коду).

Як організована програма. Як правило, програма на Python складається з множини текстових файлів, які містять інструкції. Програма організована як один головний файл, до якого можуть підключатися додаткові файли–модулі. Головний файл визначає, як буде рухатися основний потік виконання програми, – це той файл, який необхідно запустити, щоб розпочати роботу додатка. Файли модулів – це бібліотеки інструментальних засобів, де містяться компоненти, які використовує головний файл. Головний файл використовує інструменти, визначені у файлах модулів, а модулі використовують інструменти, визначені в інших модулях. В файлах модулів зазвичай визначені інструментальні засоби, які використовують в інших файлах. Щоб отримати доступ до визначених в модулі інструментів – атрибутів модуля (імен змінних, пов'язаних з такими об'єктами, як функції), необхідно імпортувати цей модуль: ми імпортуємо модулі і отримуємо доступ до їх атрибутів, що дозволяє використовувати їх функціональні можливості.

Комп'ютерний практикум № 14. ООП: створення класа та об'єктів–екземплярів класу

Мета роботи: ознайомитися з парадигмою об'єктно-орієнтованого програмування (ООП) на мові Python. *Об'єкт дослідження* – парадигма ООП, клас, екземпляр класу, конструктор класу.

ПЛАН

1. Парадигма об'єктно-орієнтованого програмування
2. Клас: створення класа, об'єктів-екземплярів класу
3. Конструктор класу – метод __init__.

Завдання

1. Ознайомитися з теоретичним матеріалом.
2. Відповідно до свого варіанту

- визначити умови; за допомогою формул описати варіанти виконання необхідний дій; розробити програмний додаток, який розв'язує завдання;
- організувати, якщо треба, введення даних з клавіатури і виведення у консоль;
- атрибути пов'язати з методами, які дозволяють їх змінювати
- реалізувати такі два *py* файли: *а*) створення класу, який не містить конструктор класу, в класі за допомогою атрибутів (поза функції) встановлюються декілька властивостей об'єктів; *б*) створення класу з конструктором классу (використати метод *__init__* для визначення початкових значень атрибутів об'єктів при їх створенні): для аргументів конструктора додати значення за замовчуванням;
- на основі класу створити декілька об'єктів-екземплярів.

3. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних.

Варіанти

1	Аудиторія у ВНЗ	2	Книга
3	Континент	4	Конференція
5	Автомобіль	6	Журнал
7	Овоч	8	Програмне забеспечення комп'ютера
9	Факультет інституту	10	Навчальна дисципліна
11	Газета	12	Геометрична фігура (трикутник, коло або прямокутник)
13	Місто	14	Фрукт
15	Міністерство охорони здоров'я	16	Торгівельний центр
17	Аптека	18	Операційна система (Linux, Windows, …)
19	Банк	20	Будівельні матеріали
21	Текстовий редактор	22	Лікарський препарат
23	Будинок	24	ЕОМ (комп'ютер)
25	Підприємство	26	Держава
27	Бібліотека	28	Група

Контрольні запитання

1. Назвіть парадигму об'єктно-орієнтованого програмування.
2. Яка інструкція передбачена для створення класів.
3. Для чого використовують метод конструктора *__init__*.

Аудиторна робота

Приклад 14.1. В Python все є об'єктами, в тому числі й самі класи

1. Створення екземпляра класу

```
# Оголошення порожнього класу  MyClass
class MyClass:  pass

obj = MyClass()
# Объект obj - це екземпляр класу MyClass,
# (він має тип MyClass)
print(type(obj))  # <class '__main__.MyClass'>
# MyClass – це клас, він є  об'єктом, екземпляром метакласу type
# який є абстракцією поняття типу даних
print(type(MyClass))  # <class 'type'>
# Тому з класами можна виконувати операції як із об'єктами
наприклад, копиювання
AnotherClass = MyClass
print(type(AnotherClass))
# тепер AnotherClass – це те ж саме, що і MyClass,
# і obj є екземпляром класу AnotherClass
print(isinstance(obj, AnotherClass))  # True
```

2. Всі елементи клада називають атрибутами.

Оголошення класу MyClass з двома атрибутами *int_field, str_field,* які є змінними,

```
class MyClass:
    int_field = 8
    str_field = 'a string'
# Звернення до атрибутів класу
print (MyClass.int_field); print (MyClass.str_field)
# Створення двох екземплярів  класу
object1 = MyClass (); object2 = MyClass ()
# Звернення до атрибутів класу через його екземпляри
print (object1.int_field); print (object2.str_field)
# Всі перераховані вище звернення до атрибутів насправді
відносяться
# до двох одних і тих самих  змінних
# Зміна значення атрибута класу
MyClass.int_field = 10
print (MyClass.int_field); print (object1.int_field); print
(object2.int_field)
# Однак, аналогічно до глобальних і локальних змінних,
# присвоєння значення атрибуту об'єкта не змінює значення
# атрибута класу, а веде до створення атрибута даних (нестатичного
поля)
object1.str_field = 'another string'
```

```
print(MyClass.str_field); print(object1.str_field);
print(object2.str_field)
```

3. Атрибути-дані аналогічні полям. Їх не треба описувати: як і змінні, вони створюються в момент першого присвоювання.

```
# Клас, який описує людину
class Person: pass
# Створення екземплярів  класу
alex = Person()
alex.name = 'Alex';  alex.age = 18

john = Person()
john.name = 'John'; john.age = 20

# Атрибути-дані відносять тільки до окремих екземплярів класу
# і ніяк не впливають на значення відповідних атрибутів-даних
інших екземплярів
print(alex.name, 'is', alex.age); print(john.name, 'is', john.age)
```

4. Атрибутами класу можуть бути й методи-функції

```
# Клас,  який описує  людину
class Person:
    # Перший  аргумент,  який  вказує  на поточний екземпляр
класу,
    # прийнято називати  self
    def print_info(self):
        print(self.name, 'is', self.age)

# Створення екземплярів класу
alex=Person(); alex.name='Alex'; alex.age=18

john = Person(); john.name = 'John'; john.age = 20

# Перевіримо, чим є  атрибут-функція print_info класу Person
print(type(Person.print_info))  # функція (<class 'function'>)

# Викличемо його  для об’єктів alex і john
Person.print_info(alex)
Person.print_info(john)

# Метод – функція, зв’язана з об’єктом. Всі атрибути класу, які є
# функціями, описують відповідні методи екземплярів даного класу
print(type(alex.print_info))  # метод (<class 'method'>)

# Виклик  методу print_info
alex.print_info()
john.print_info()
```

5. Початковий стан об’єкта слід створювати в методі-конструкторі __init__, який викликають автоматично після створення екземпляру класа. Його параметри вказують при створенні об’єкта.

```
# Клас, який описує людину
class Person:
    # Конструктор
    def __init__(self, name, age):
        self.name = name
        self.age = age
    # Метод з попереднього прикладу
    def print_info(self):
        print(self.name, 'is', self.age)
# Створення екземплярів классу
alex = Person('Alex', 18)
john = Person('John', 20)
# Виклик метода print_info
alex.print_info()
john.print_info()
```

6. Атрибути класу, які є функціями, – це тіж самі атрибути класу, як і змінні

```
def outer_method(self):
    print('I am a method of object', self)
class MyClass:
    method = outer_method

obj = MyClass(); obj.method()
```

7. Статичними називають методи, які є загальними для класу та усіх екземплярів класу і не мають доступ до даних екземплярів класів. Для їх створення використовують декоратор *staticmethod.* Декоратор – це спеціальна функція, яка змінює поведінку функції або класу. Для використання декоратора треба перед відповідним оголошенням вказати символ @, ім'я необхідного декоратора і список його аргументів в круглих дужках. Якщо передавати параметрів декораторові не потрібно, дужки не вказують.

```
class MyClass:
    # Оголошення атрибута класу
    class_attribute = 8

    def __init__(self):  # Конструктор
        self.data_attribute = 42

    # Статичний метод (у нього немає параметру) self,
    # оскільки він не зв’язаний з жодним із екземплярів класу
    # не має доступу до  атрибутівм-даних
    @staticmethod
    def static_method():
        print(MyClass.class_attribute)

    def instance_method(self): # звичайний  метод
        print(self.data_attribute)

if __name__ == '__main__':
    # Виклик статичного методу
    MyClass.static_method()
```

```
    # Інстанціювання об’єкта
    obj = MyClass()
    # Виклик методу
    obj.instance_method()
    # Аналогічно атрибутам класу, доступ до статичних  методів
    # можна отримати й через екземпляр класу
    obj.static_method()
```

8. Класи є об'єктами, тому крім атрибутів-функцій вони можуть мати і власні методи. Для створення методів класу використовують декоратор *classmethod*. В таких методах перший параметр прийнято називати не *self*, а *cls*. Методи класу зазвичай використовують в двох випадках: для створення
• фабричних методів, які створюють екземпляри даного класу альтернативними способами;
• статичних методів, які викликають статичні методи:
- оскільки даний клас передається як перший аргумент функції,
- не потрібно вручну вказувати ім'я класу для виклику статичного методу.

```
class Rectangle:
    """    Клас, який описує прямокутник     """
    def __init__(self, side_a, side_b):
        """  Конструктор класу
        :param side_a: перша сторона
        :param side_b: друга сторона       """
        self.side_a = side_a
        self.side_b = side_b

    def __repr__(self):
     """Метод, який повертає подання об'єкта у вигляді рядка """
      return 'Rectangle(%.1f, %.1f)' % (self.side_a, self.side_b)

class Circle:
    """ Клас, який описує коло """
    def __init__(self, radius):
        self.radius = radius
    def __repr__(self):
        return 'Circle(%.1f)' % self.radius

    @classmethod
    def from_rectangle(cls, rectangle):
        """Ми використовуємо метод класу в якості фабричного
методу,
        який створює екземпляр класу Circle з екземпляру
        класу Rectangle як коло, що вписане у цей  прямокутник.
        :param rectangle: Rectangle instance
        :return: Circle instance       """
        radius = (rectangle.side_a ** 2 + rectangle.side_b ** 2) ** 0.5 / 2
        return cls(radius)

def main():
    rectangle = Rectangle(3, 4)
```

```
    print(rectangle)
    circle1 = Circle(1)
    print(circle1)
    circle2 = Circle.from_rectangle(rectangle)
    print(circle2)

if __name__ == '__main__':  main()
```

9. Атрибути, імена яких розпочинаються, але не закінчуються, двома символами підкреслення, вважаються приватними. До них застосовують механізм «name mangling»: зсередини класу і його екземплярів до цих атрибутів можна звертатися по тому ж імені, яке було задано при оголошенні (однак до імен зліва додається підкреслення і ім'я класу). Цей механізм не передбачає захисту даних від зміни ззовні, тому що до них все одно можна звернутися, знаючи ім'я класу і те, як Python змінює імена приватних атрибутів, проте дозволяє захистити їх від випадкового перевизначення в класах-нащадках.

```
class MyClass:
    def __init__(self):
        self.__private_attribute = 42

    def get_private(self):
        return self.__private_attribute

obj = MyClass()
print(obj.get_private())  # 42
print(obj.__private_attribute)  # помилка
# print(obj._MyClass__private_attribute)  # 42
```

10. Атрибути, імена яких розпочинаються і закінчуються двома знаками підкреслення, є внутрішніми для Python і задають особливі властивості об'єктів (наприклад, рядок документування *__doc__*, атрибут *__class__*, в якому зберігають клас даного об'єкта). Серед таких атрибутів є методи. Подібні методи називають методами зі спеціальними іменами: вони задають особливу поведінку об'єктів і дозволяють перевизначати поведінку вбудованих функцій і операторів для екземплярів заданого класу. Найчастіше використовують метод-конструктор *__init__*.

```
class Complex:
    """    Комплексне число    """
    def __init__(self, real=0.0, imaginary=0.0):
        """ Конструктор
        :param real:      дійсна частина
        :param imaginary: уявна частина  """
        self.real = real; self.imaginary = imaginary

    def __repr__(self):
        """  Метод  __repr__  повертає подання об’єкта у вигляді
рядка, який має вигляд виразу, що створює  аналогічний об’єкт,
інакше містить його  опис;
        Викликають функцією repr. """
        return 'Complex(%g, %g)' % (self.real, self.imaginary)
```

```
    def __str__(self):
        """ Метод __str__ повертає подання об'єкта у вигляді
рядка; його викликають функції str, print і format.  """
        return '%g %c %gi' % (self.real,
                              '+' if self.imaginary >= 0 else '-',
                              abs(self.imaginary))

    # Арифметичні операції
    def __add__(self, other):
        """  Метод __add__ визначає  операцію додавання. """
        return Complex(self.real + other.real,
                self.imaginary + other.imaginary)

    def __neg__(self):
        """  Операція заперечення """
        return Complex(-self.real, -self.imaginary)

    def __sub__(self, other):
        """  Операція віднімання.
        Додавання і заперечення вже визначені, тому віднімання
        можна визначити через них  """
        return self + (-other)

    def __abs__(self):
        """  Модуль числа  """
        return (self.real ** 2 + self.imaginary ** 2) ** 0.5

    # Операції порівняння
    def __eq__(self, other):
     return self.real==other.real and self.imaginary==other.imaginary

    def __ne__(self, other):
        return not (self == other)

def main():
    x = Complex(2, 3.5)
    print(repr(x)); print('x =', x)
    y = Complex(5, 7)
    print('y =', y); print('x + y =', x + y)
    print('x - y =', x - y); print('|x| =', abs(x))
    print('(x == y) =', x == y)

if __name__ == '__main__':  main()
```

11. Використання спеціального методу *__new__* для реалізації такого шаблону проектування як Одинак (Singleton) (це шаблон проектування, який гарантує, що даний клас має тільки один екземпляр, і породжує його).

```
class Singleton:
  _instance = None  # атрибут, який зберігає екземпляр класу

  def __new__(cls, *args, **kwargs):
   """ Метод __new__ викликають при створенні екземпляра класу """
```

```
   # Якщо екземпляр ще не створений, то створюємо його
   if cls._instance is None:
       cls._instance = object.__new__(cls, *args, **kwargs)
       # Повертаємо екземпляр, який існує
   return cls._instance

  def __init__(self):
   self.value = 8

obj1=Singleton(); print(obj1.value)

obj2 = Singleton(); obj2.value = 42
print(obj1.value)
```

12. Використання функції __getattribute__ для приховування даних

```
class MyClass:
   def __init__(self):
        self.password = None

   def __getattribute__(self, item):
    """Метод __getattribute__ викливають при отриманні
атрибутів"""

        # Якщо поле-запит secret_field і пароль правильний
        if item == 'secret_field' and self.password == '9ea)fc':
            # то повертаємо значення
            return 'secret value'
        else:
            # инакше викликаємо метод __getattribute__ класу
object
            return object.__getattribute__(self, item)

obj = MyClass()# Створення екземпляру класу
# Розблокування секретного поля
# obj.password = '9ea)fc'
# Виведення значення secret field.
# Значення буде отримано, якщо  розкоментувати попередній
# рядок  програмного коду, інакш отримаємо помилку
print(obj.secret_field)
```

Приклад 14.2. Створимо клас з одним атрибутом поза методу й одним методом, який виводить значення цього атрибута на екран із змінами:

```
class First:
 color = "red"
 def out(self):  print (self.color + "!")
```

Створимо два об’єкти даного класу:

```
obj1 = First(); obj2 = First()
# Обидва об’єкта (obj1, obj2) мають  два однакових атрибута:
# color -  у вигляді властивості й printer (у вигляді методу)
print (obj1.color); print (obj2.color)
obj1.out(); obj2.out()
```

В результаті отримано:

```
"red"
"red"
"red!"
"red!"
```

Нехай в класі за допомогою атрибутів визначені дві властивості об'єктів: червоний колір і кругла форма. Методи можуть змінювати ці властивості в залежності від побажань тих, хто створює об'єкти.

```
class Second:
 color = "red"
 form = "circle"
 def changecolor(self,newcolor):
  self.color = newcolor
 def changeform(self,newform):
  self.form = newform

obj1 = Second(); obj2 = Second()
print (obj1.color, obj1.form) # вывод на экран "red circle"
print (obj2.color, obj2.form) # вывод на экран "red circle"

obj1.changecolor("green") # изменение цвета первого объекта
obj2.changecolor("blue")  # изменение цвет второго объекта
obj2.changeform("oval")  # изменение формы второго объекта
print (obj1.color, obj1.form) # вывод на экран "green circle"
print (obj2.color, obj2.form) # вывод на экран "blue oval"
```

Тут за замовчуванням будь-який створений об'єкт має червоний колір і круглу форму. Однак в подальшому за допомогою методів даного класу можна поміняти і колір і форму будь-якого об'єкта. В результаті об'єкти перестають бути однаковими (червоними і круглими), хоча зберігають однаковий набір властивостей (колір і форму).

Як же відбуваються зміни? – Методи крім параметра *self*, можуть мати й інші параметри, в яких передаються дані для обробки їх цим методом. Наприклад, метод *changecolor* має додатковий параметр newcolor, за допомогою якого, в метод можна передати дані про бажані кольори фігури. Далі метод змінює колір за допомогою відповідних інструкцій.

Приклад 14.3. Більшість класів використовують спеціальний метод *__init__*, який при створенні об'єкта створює йому атрибути (викликати даний метод не потрібно, він сам запускається під час виклику класу, а виклик класу відбувається, коли створюється об'єкт). Такий метод називають конструктором класу. Першим параметром у *__init__* є *self*, на місце якого підставляється об'єкт в момент його створення. Другий і наступні (якщо є) параметри замінюють аргументами, які передані в конструктор при виклику класу. Розглянемо два класи: в одному використовується конструктор, а в іншому ні. Потрібно створити два атрибути об'єкта.

```
1)class YesInit:
 def __init__(self,one,two):
```

```
  self.fname = one; self.sname = two

obj1 = YesInit("Peter","Ok")
print (obj1.fname, obj1.sname)

2)class NoInit:
 def names(self,one,two):
  self.fname = one; self.sname = two

obj1 = YesInit(); obj1.names("Peter","Ok")
print (obj1.fname, obj1.sname)
```

Результат роботи програми в обох випадках:

```
Peter Ok
```

В обох програмах у об'єкта з'являються два атрибути: *fname* і *sname*. *У першій програмі* вони приймають початкові значення при створенні об'єкта і повинні передаватися в дужках при виклику класу. Якщо атрибути повинні бути присутніми у об'єктів класу обов'язково, то використання методу *__init__* – ідеальний варіант. *У другій програмі* (без використання конструктора) атрибути створюють шляхом виклику методу *names* після створення об'єкта. В даному випадку виклик методу *names* необов'язковий, тому об'єкти можуть існувати без атрибутів *fname* і *sname*.

Зазвичай метод *__init_* передбачає передачу аргументів при створенні об'єктів, однак аргумент може не бути переданий. Наприклад, якщо в прикладі вище створити об'єкт так: *obj*1 = *YesInit*(), тобто не передати класу аргументи, то станеться помилка. Щоб уникнути подібних ситуацій, можна в методі *__init_* привласнювати параметрам значення за замовчуванням. Якщо при виклику класу були задані аргументи для даних параметрів, то вони і будуть використані, якщо ні – в тілі методу будуть використані значення за замовчуванням. Наприклад:

```
class YesInit:
 def __init__(self,one="noname",two="nametoo"):
  self.fname = one; self.sname = two
obj1 = YesInit("Sasha","Tu");obj2 = YesInit()
obj3 = YesInit("Spartak"); obj4 = YesInit(two="Harry")
print (obj1.fname,obj1.sname); print (obj2.fname,obj2.sname)
print (obj3.fname,obj3.sname); print (obj4.fname, obj4.sname)
```

Результат роботи:

```
Sasha Tu
noname nametoo
Spartak nametoo
noname Harry
```

Тут другий об'єкт створюють без передачі аргументів, тому в методі *__init__* використовують значення за замовчуванням ("*noname*", "*nametoo*"). При створенні третього і четвертого об'єктів передають по одному аргументу.

Якщо вказують значення не першого аргументу, то слід явно вказати ім'я параметра (четвертий об'єкт). Метод *__init_*_ може містити параметри як без значень за замовчуванням, так і зі значеннями за замовчуванням. В цьому випадку параметри, аргументи яких повинні бути обов'язково вказані, при створенні об'єктів вказують першими, а параметри зі значеннями за замовчуванням – після, наприклад:

```
class fruits:
 def __init__(self,w,n=0):
  self.what = w; self.numbers = n

f1 = fruits("apple",150); f2 = fruits("pineapple")
print (f1.what,f1.numbers); print (f2.what,f2.numbers)
```

Створимо клас, значення початкових атрибутів (з методу *__init__*) якого залежить від переданих аргументів при створенні об'єктів. Далі ці атрибути об'єктів, створених на основі даного класу, можна змінювати за допомогою методів.

```
class Building:
 def __init__(self,w,c,n=0):
  self.what=w; self.color=c; self.numbers=n; self.mwhere(n)

 def mwhere(self,n):
  if n <= 0:  self.where = "отсутствуют"
  elif 0 < n < 100: self.where = "малый склад"
  else: self.where = "основной склад"

 def plus(self,p):
  self.numbers = self.numbers + p
  self.mwhere(self.numbers)

 def minus(self,m):
  self.numbers=self.numbers-m
  self.mwhere(self.numbers)

m1 = Building("доски", "белые",50)
m2 = Building("доски", "коричневые", 300)
m3 = Building("кирпичи","белые")
print (m1.what,m1.color,m1.where)
print (m2.what,m2.color,m2.where)
print (m3.what,m3.color,m3.where)
m1.plus(500);print (m1.numbers, m1.where)
```

Тут значення атрибута *where* об'єкта залежить від значення атрибута *numbers*.

Теоретична частина

1. Парадигма об'єктно-орієнтованого програмування. Багато сучасних мов підтримують кілька парадигм програмування (наприклад, директивне, функціональне, об'єктно-орієнтоване). Такі мови є змішаними. До них

відносять і Python. Ключову роль в ООП відіграє множина об'єктів. Реальний світ складається з об'єктів та їхньої взаємодії між собою. В результаті взаємодії об'єкти можуть змінюватися самі або змінювати інші об'єкти. У світі умовно можна виділити різні системи, які реалізують певні цілі (змінюються з одного стану в інший). Наприклад, група на занятті – це система, яка складається з таких об'єктів, як діти, учитель, столи, комп'ютери, проектор тощо. У цієї системи можна виділити основну мету – збільшення частки знань дітей на якусь величину. Щоб домогтися цього, об'єкти системи повинні певним чином виконати взаємодію між собою.

Необхідно розуміти різницю між програмою написаною на основі структурного «стилю» і програмою в «стилі» ООП. У першому випадку, на перший план виходить логіка, розуміння послідовності виконання виразів (дій) для досягнення цілей. У другому – важливо системне мислення, вміння бачити систему в цілому, з одного боку, і розуміння ролі її частин (об'єктів), з іншого.

Свого часу Алан Кей сформулював для розробленої ним мови програмування Smalltalk кілька принципів, які описують принципи ООП:

1) об'єктно-орієнтована програма складається з об'єктів, які посилають один одному повідомлення;

2) кожний об'єкт може складатися з інших об'єктів (а може і не складатися);

3) кожний об'єкт належить певному класу (типу), який задає поведінку об'єктів, створених на його основі.

2. Клас – це опис об'єктів певного типу. На основі класів створюють об'єкти. Може існувати множина об'єктів, які належать одному класу. З іншого боку, може існувати клас без об'єктів, реалізованих на його основі. Програма, написана з використанням парадигми ООП, повинна складатися з: об'єктів, класів (опису об'єктів), взаємодій об'єктів між собою, в результаті яких змінюються їх властивості. Об'єкт в програмі можна створити лише на основі будь-якого класу. Тому, ООП має розпочинатися з проектування і створення класів. Класи можуть бути розташовані або спочатку коду програми, або імпортуватися з інших файлів – модулів (також на початку коду).

Створення класів. Для створення класів передбачена інструкція *class*, яка складається з рядка заголовка і тіла. Заголовок складається з ключового слова *class*, імені класу і, можливо, назв суперкласів в дужках. Суперкласи можуть бути відсутніми, в такому випадку дужки не потрібні. Тіло класу складається з блоку різних інструкцій. Тіло повинно мати відступ (як і будь-які вкладені конструкції в Python). Схематично клас можна подати таким чином:

```
class ІМ'Я_КЛАСУ:
 змінна = значення  ….
 def ІМ'Я_МЕТОДА(self, ...):
  self.змінна = значення …
```

У заголовку після імені класу можна вказати суперкласи (в дужках), а методи можуть бути більш складними. *Методи в класах* – це звичайні функції, за одним винятком: вони мають один обов'язковий параметр – *self*, який

потрібен для зв'язку з конкретним об'єктом. *Атрибути класу* – це імена змінних поза функцій і імена функцій. Вони успадковуються всіма об'єктами, створеними на основі даного класу, і забезпечують властивості і поведінку об'єкта. Об'єкти можуть мати атрибути, які створюються в тілі методу, якщо даний метод буде викликано для конкретного об'єкта.

Створення об'єктів. Об'єкти створюються так:

змінна = ІМ'Я_КЛАСУ()

Після такої інструкції у програмі з'являється об'єкт, доступ до якого можна отримати за ім'ям змінної, пов'язаної з ним. При створенні об'єкт отримує атрибути його класу (він має характеристики, визначені у його класі). Кількість об'єктів, які можна створити на основі певного класу, не обмежена. Об'єкти одного класу мають схожий набір атрибутів, а значення атрибутів можуть бути різними (вони схожі, але індивідуально різняться).

Self: Методи класу − це невеликі програми, призначені для роботи з об'єктами. Методи можуть створювати нові властивості (дані) об'єкта, змінювати існуючі, виконувати інші дії над об'єктами. Методу необхідно «знати», дані якого об'єкта йому потрібно буде обробляти. Для цього йому в якості першого (а іноді і єдиного) аргументу передається ім'я змінної, пов'язаної з об'єктом. Щоб в описі класу вказати об'єкт, який передається в подальшому, використовують параметр *self*. Виклик методу для конкретного об'єкта в основному блоці програми виглядає таким чином: ОБ'ЄКТ.ІМ'Я_МЕТОДА(…), де ОБ'ЄКТ – змінна, пов'язана з ним. Цей вираз перетворюється в класі, до якого відноситься об'єкт, в ІМ'Я_МЕТОДА(ОБ'ЄКТ, …) – замість параметра *self* підставляють конкретний об'єкт. Атрибути екземпляра створюються шляхом присвоювання значень атрибутам об'єкта екземпляра. Зазвичай вони створюються всередині методів класу, в інструкції *class* – присвоєнням значень атрибутам аргументу *self* (який завжди є імовірним екземпляром). Можна створювати атрибути за допомогою операції присвоєння в будь-якому місці програми, де доступне посилання на екземпляр, навіть за межами інструкції *class*.

3. Конструктор. Зазвичай всі атрибути екземплярів ініціалізують в конструкторі *__init__*. Метод конструктора *__init__* використовують для встановлення початкових значень атрибутів екземплярів і виконання інших початкових операцій (це − звичайна функція, яка підтримує і можливість визначення значень аргументів за замовчуванням, і передачу іменованих аргументів).

Комп'ютерний практикум № 15. Наслідування атрибутів класу в ООП

Мета роботи: ознайомитися з особливостями реалізації наслідування (звичайного та множинного) атрибутів класа в ООП на мові Python. *Об'єкт дослідження* – принципи ООП, клас.

ПЛАН

1. Ідеї (принципи) ООП.

Завдання

1. Ознайомитися з теоретичним матеріалом. Опрацювати приклади.
2. Відповідно до свого варіанту

- визначити умови; за допомогою формул описати варіанти виконання необхідний дій; розробити програмний додаток, який розв'язує завдання;
- організувати, якщо треба, введення даних з клавіатури і виведення у консоль;

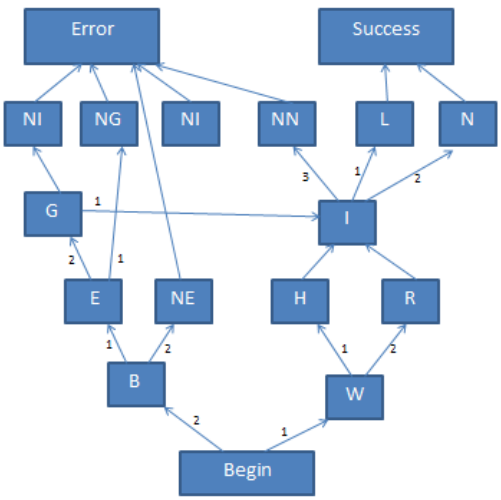
3. Використовуючи діаграми класів (на яких показують різноманітні класи, які утворюють систему, їх взаємозв'язки), створити класи і методи класів (обробники повідомлень) для опису об'єктів, наведених в індивідуальному завданні. Необхідно дотримуватись таких вимог: при створенні класів використати механізм успадкування; визначити відношення «клас/підклас».

4. Створити об'єкти для всіх елементів схеми. Поєднати елементи схеми, заповнивши атрибути-слоти, де зберігаються назви об'єктів-елементів, пов'язаних з даними, і номери вхід-вихід.

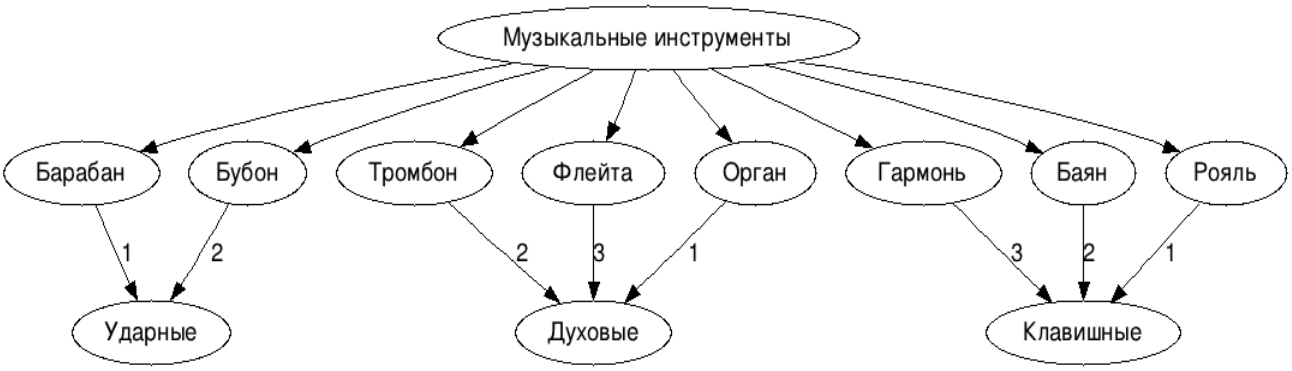
Зберегти визначення класів, об'єктів-екземплярів, методів класів – оброблювачів повідомлень з інструкцією оператору виведення у файлі.

Варіанти. Задано граф спадкування класів (цифрами показана черговість спадкування для класів, які успадковуються від декількох батьків).

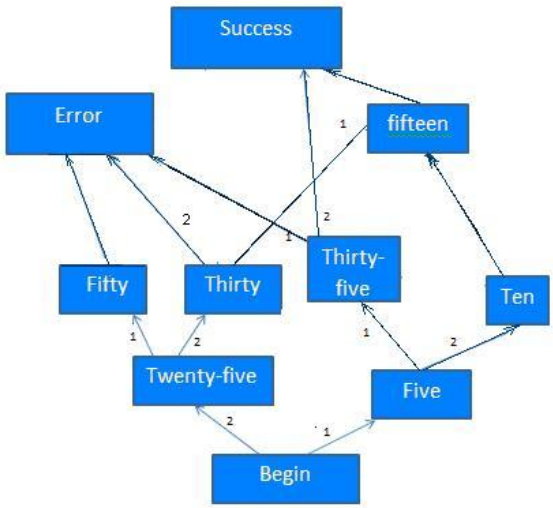
1. Для класів *Error* і *Success* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення «Error» або «Success» (успіх), відповідно. Викликати обробник для класів «Begin» і «B». Визначити, які повідомлення виводяться при цьому? Пояснити чому.



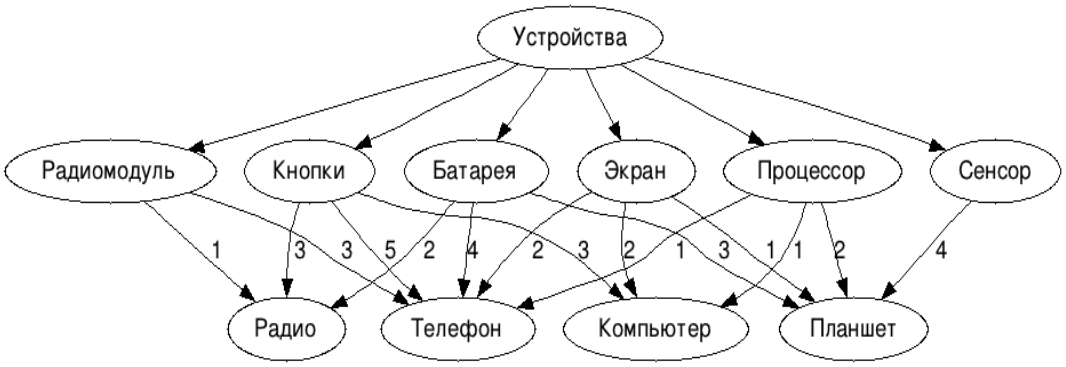
2. Для класів *Барабан, Бубон, Тромбон, Флейта, Орган, Гармонь, Баян, Рояль* задати оброблювачі повідомлень, які при зверненні до них повертають тип музичного інструменту. Викликати обробник для класів «Ударні», «Духові», «Клавішні». Визначити, які повідомлення виводяться при цьому? Пояснити чому.



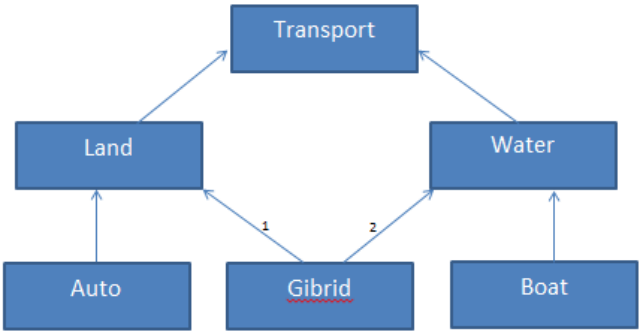
3. Для класів *Error* і *Success* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про помилку або успіх, відповідно. Викликати обробник для класів «Begin» і «Thirty□-five». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



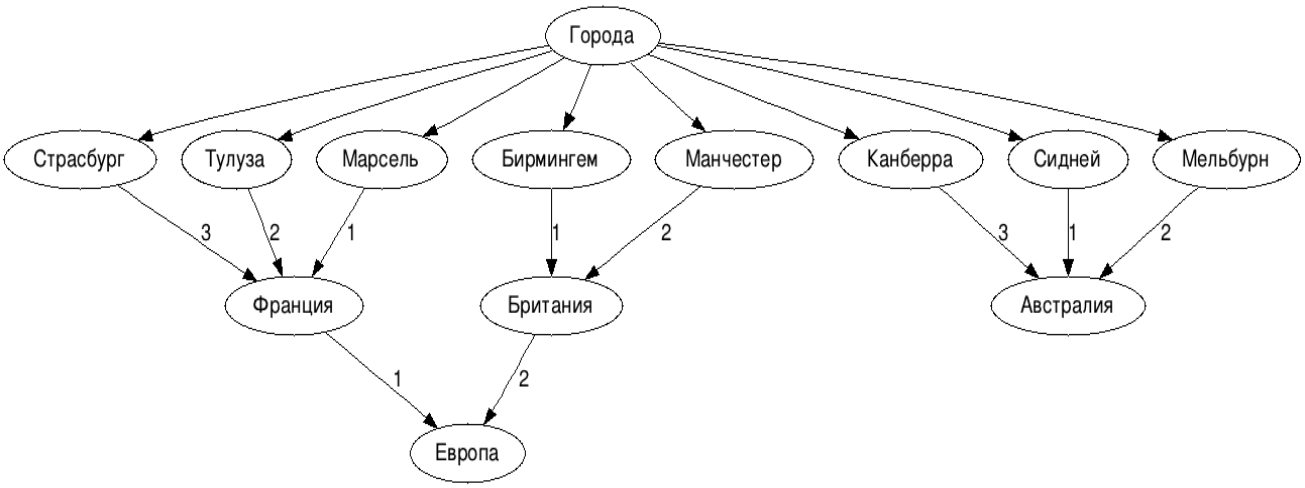
4. Для класів *Радіомодуль, Кнопки, Батарея, Екран, Процесор, Сенсор* задати оброблювачі повідомлень, які при зверненні до них повертають тип пристрою. Викликати обробник для класів «Радіо», «Телефон» і «Планшет». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



5. Для класів *Land* і *Water* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вид транспорту. Викликати обробник для класів «Boat» і «Gibrid». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



6. Для класів *Страсбург, Тулуза, Марсель, Бірмінгем, Манчестер, Канберра, Сідней, Мельбурн* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про назву міста. Викликати обробник для класів «Європа» та «Австралія». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



7. Для класів *Herbivores, Carnivores, Warm–blooded і Cool–blooded* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вигляді тварини. Викликати обробник для класів «Cobra» і «Brown bear». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.

8. Для класів 2/4, 4/8, 3/8, 4/4, 3/4, 6/8 задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вигляд розміру. Викликати обробник для класів «Соната» і «Полька». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.

9. Для класів *With_land, Without_land, For_many_families і For_one_family* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вигляд будинку. Викликати обробник для класів «Villets» і «Apartmens». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.

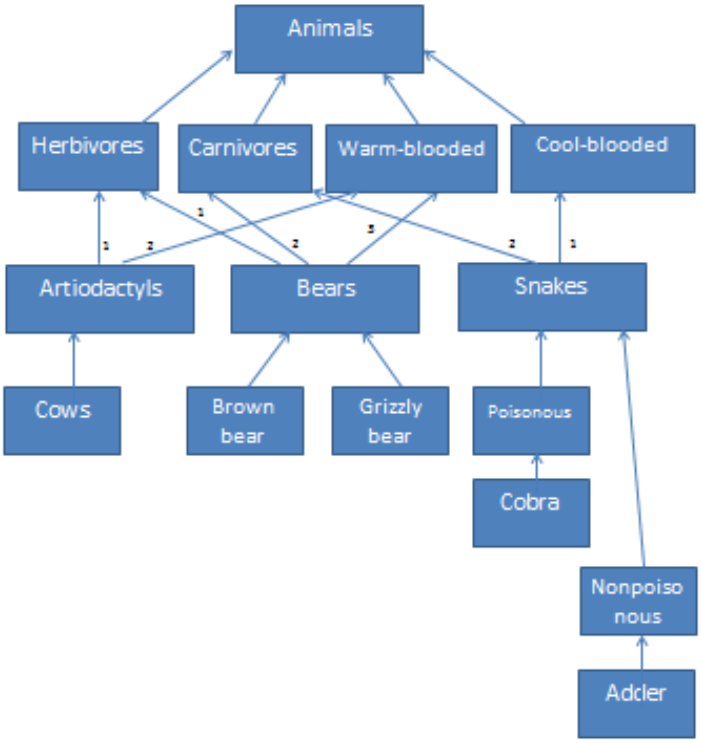


Рис. до завдання 7

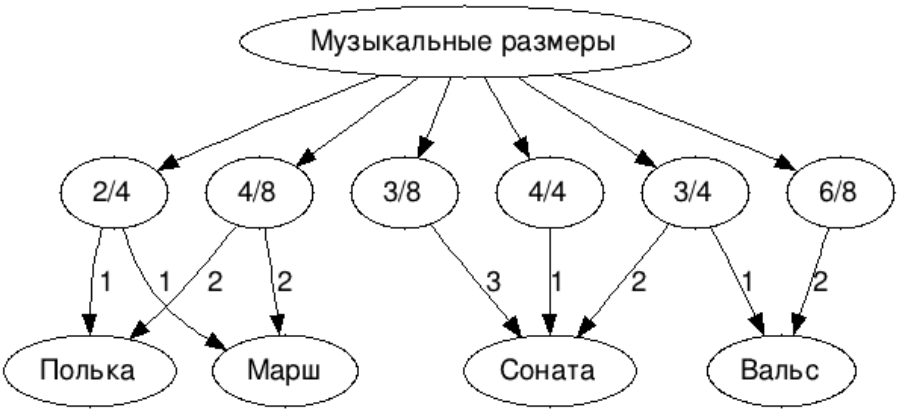


Рис. до завдання 8

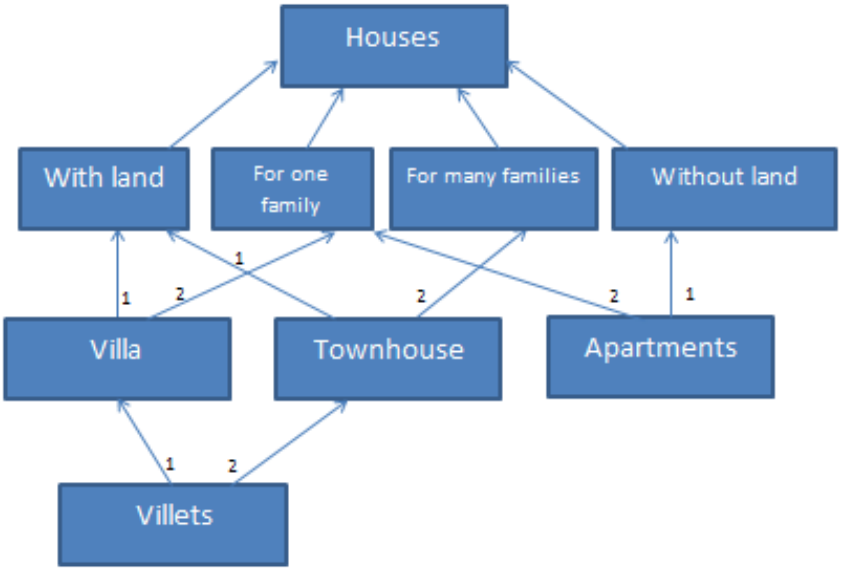
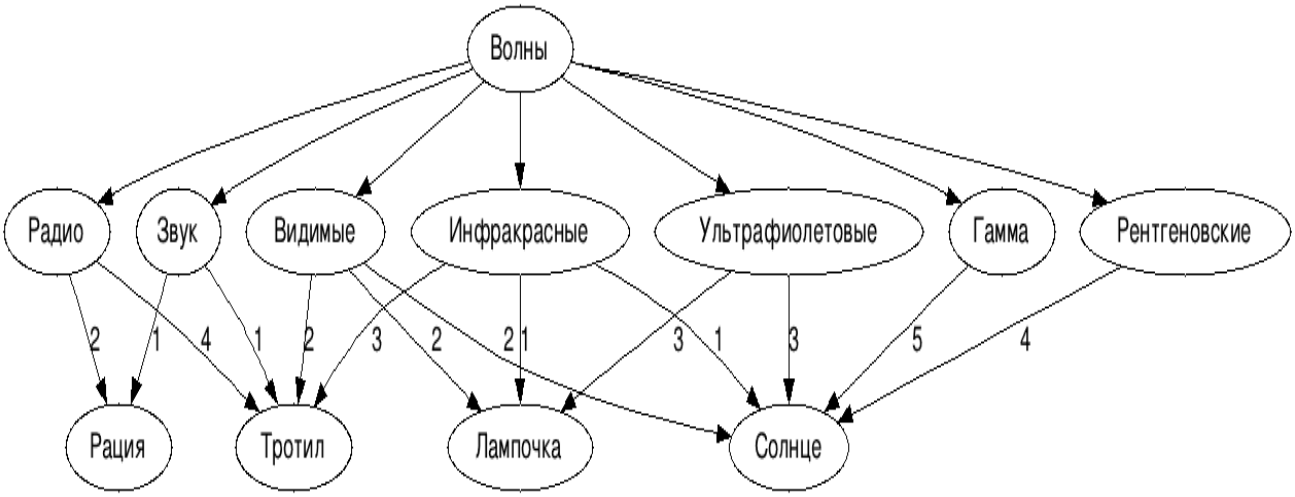
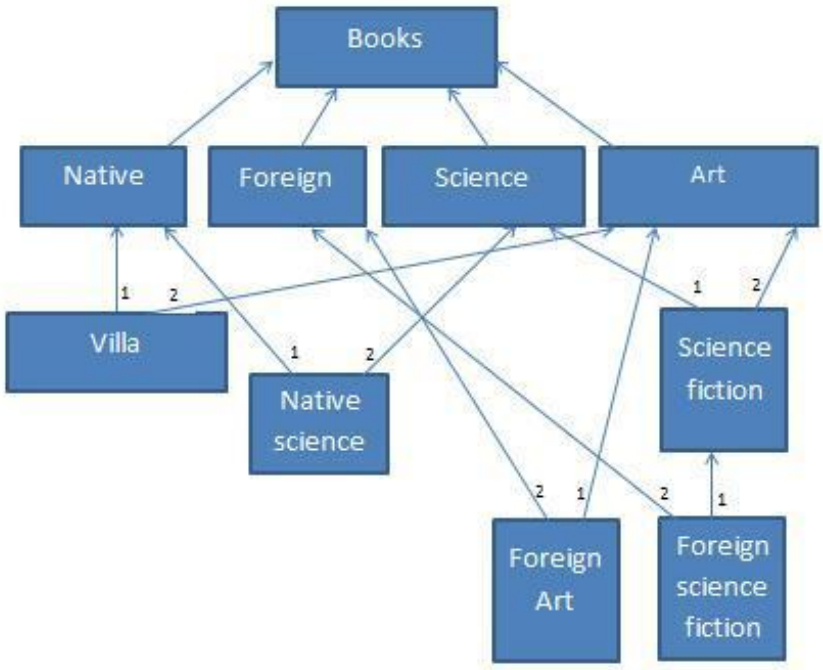


Рис. до завдання 9

10. Для класів *Радіо, Звук, Видимі, Інфрачервоні, Ультрафіолетові, Гамма, Рентгенівські* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вигляд випромінювання. Викликати обробник для класів «Лампочка» і «Рація». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.

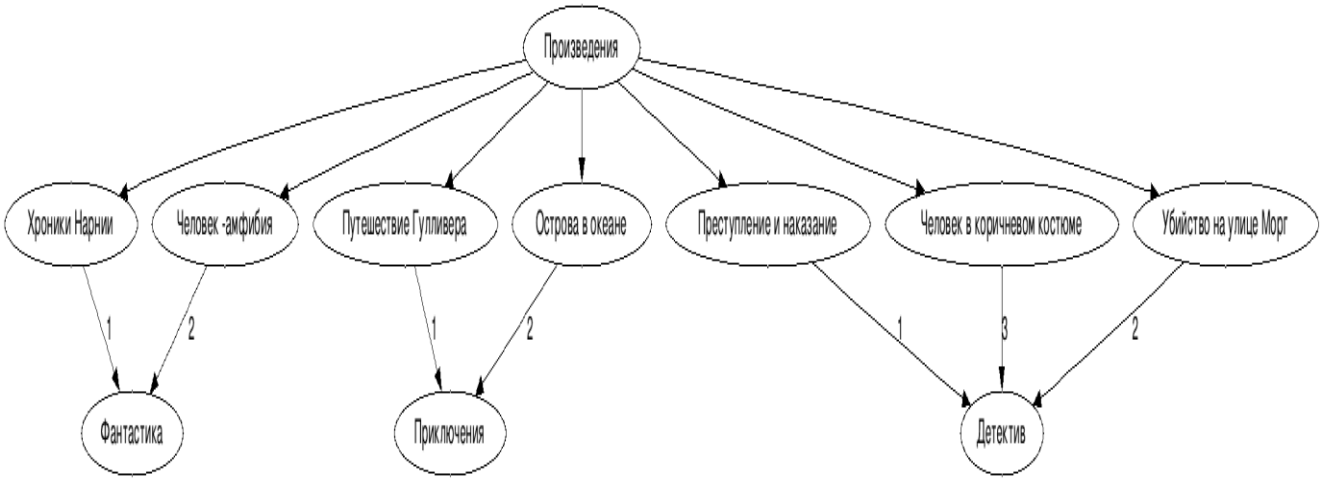


11. Для класів *Native, Foreign, Science* і *Art* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вигляд книги. Викликати обробник для класів «Foreign_science_fiction» і «Native_science». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.

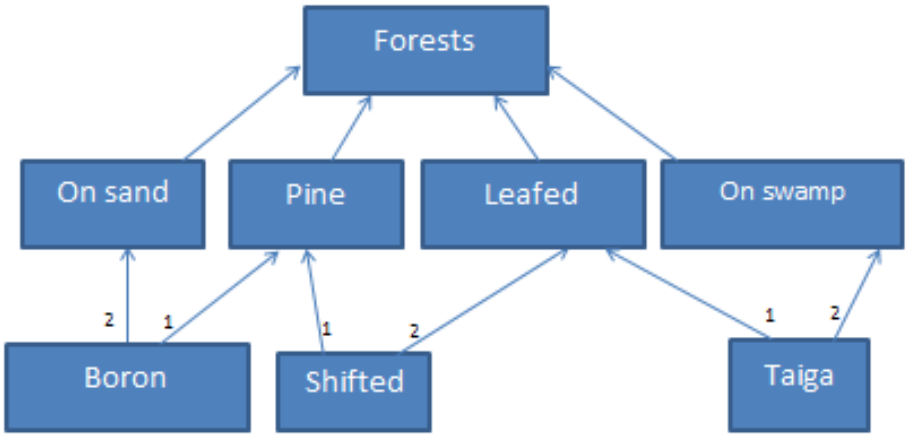


12. Для класів *Хроніки_Нарнії, Людина-амфібія, Подорож_Гуллівера, Острови_в_ океані, Злочин_і_кара, Людина_в_коричневому_костюмі, Вбивство_на_вулиці_Морг* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про назву книги. Викликати обробник для

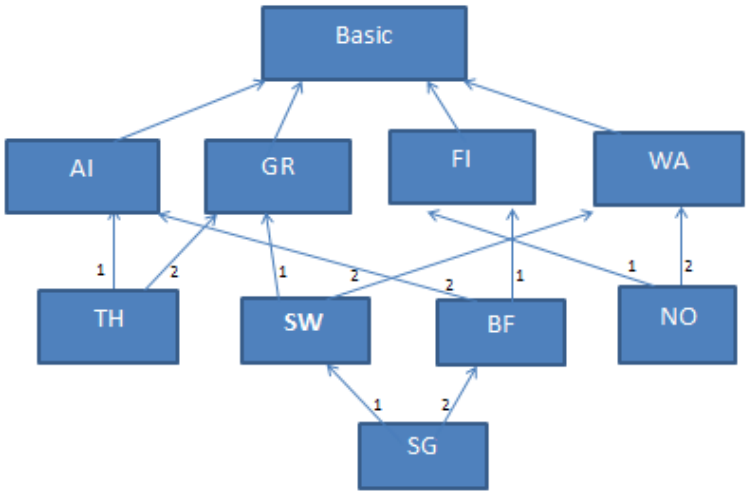
класів «Фантастика» і «Детектив». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



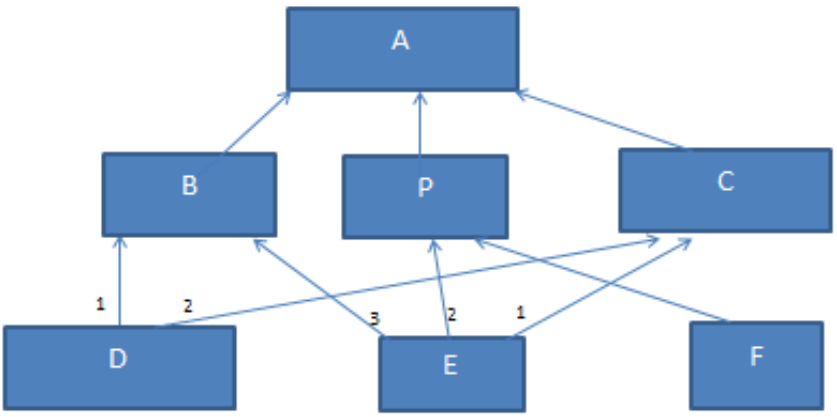
13. Для класів *On_sand*, *On_swamp*, *Pine_i_Leafed* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вигляд лісу. Викликати обробник для класів «Taiga» і «Shifted». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



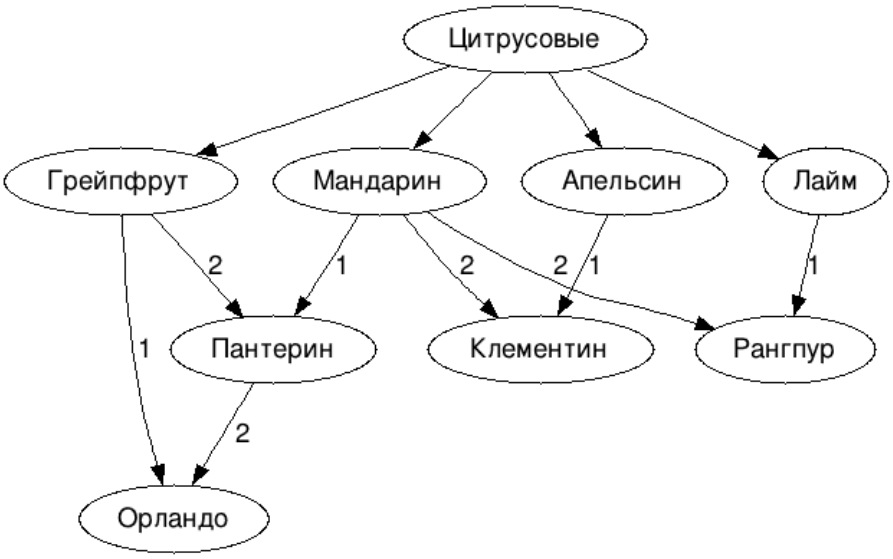
14. Для класу AI задати обробник повідомлень, який повертає повідомлення «Air», для класу GR – повідомленням «Ground», для класу FI – повідомлення «Fire», для класу WA – повідомлення «Water». Викликати обробник для класів «SG» і «NO». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



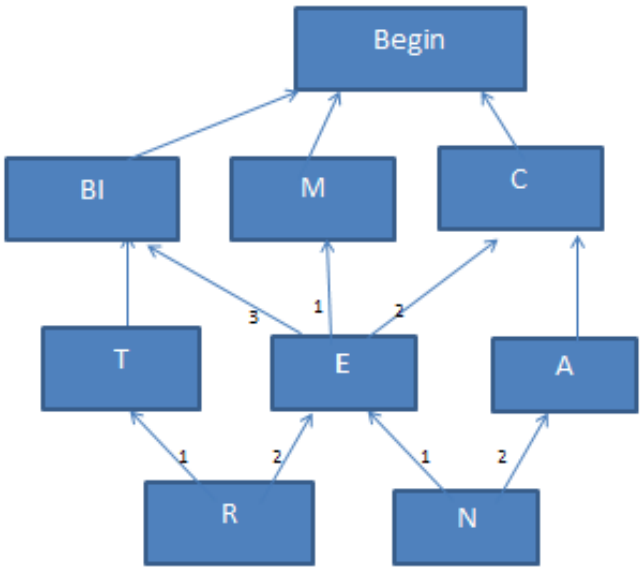
15. Для класу B задати обробник повідомлень, який повертає повідомлення «Bob», для класу P – повідомлення «Piter», для класу C – повідомлення «Chloe». Викликати обробник для класів «E» і «D». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



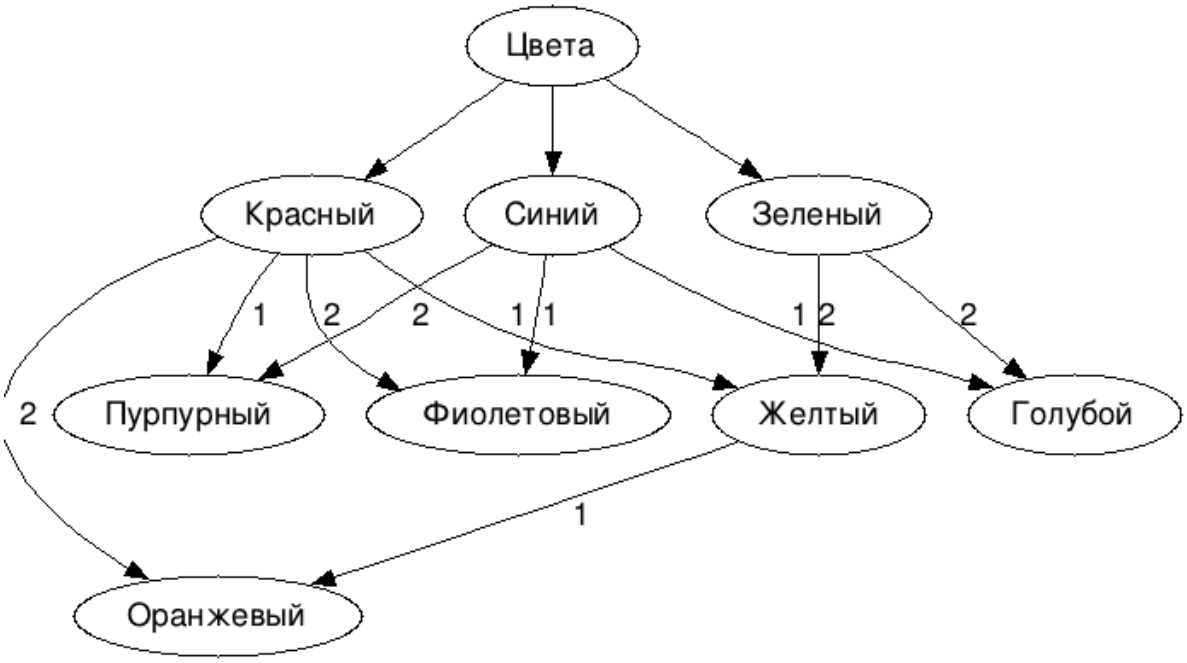
16. Для класів *Грейпфрут, Мандарин, Апельсин, Лайм* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про назву фрукта. Викликати обробник для класів «Орландо» і «Рангпур». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



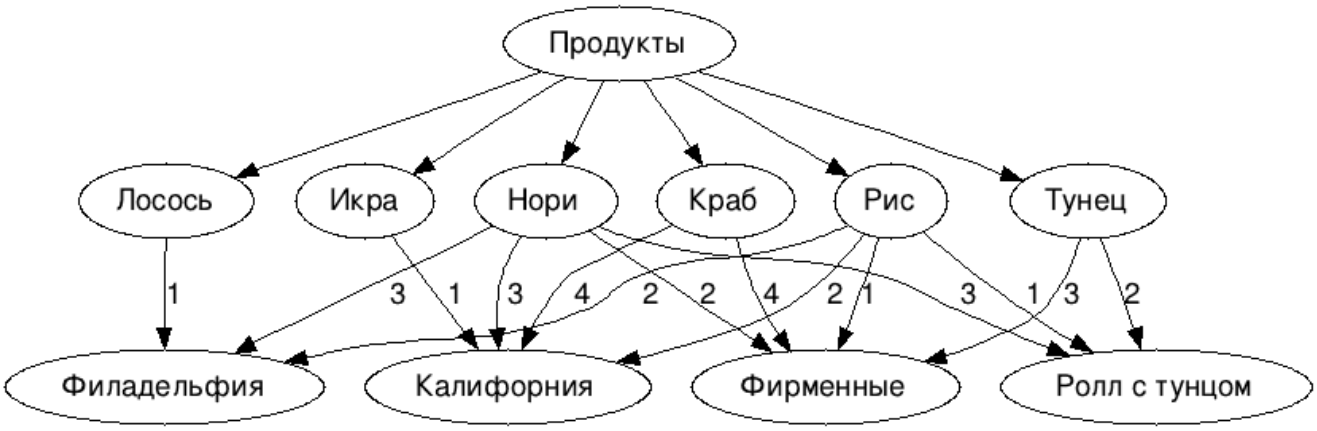
17. Для класу BI задати обробник повідомлень, який повертає повідомлення «Big», для класу M – повідомлення «Men», для класу C – повідомлення «Cat», для класу T – повідомлення «Bitter», для класу E – повідомлення «Women». Викликати обробник для класів «R» і «N». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



18. Для класів *Червоний, Синій, Зелений* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про вигляд кольору. Викликати обробник для класів «Помаранчевий» і «Фіолетовий». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.



19. Для класів *Лосось, Ікра, Норі, Краб, Рис, Тунець* задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про назву продукту. Викликати обробник для класів «Фірмові» і «Філадельфія». Визначити, які повідомлення виводяться при цьому? Пояснити, чому.

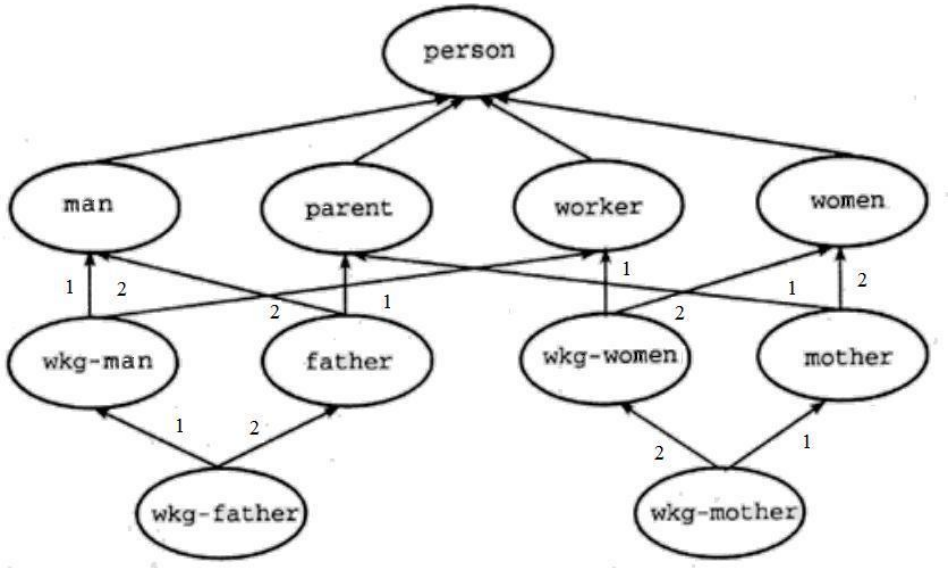


20. Описати структуру класів, зображену на схемі, таким чином, щоб дотримувався такий порядок спадкування в класах, які мають декілька суперкласів:

wkg-man: (man, worker) mother: (parent, women)
father: (parent, man) wkg-father: (wkg-man, father)
wkg-woman: (worker, women) wkg-mother: (mother, wkg-women)

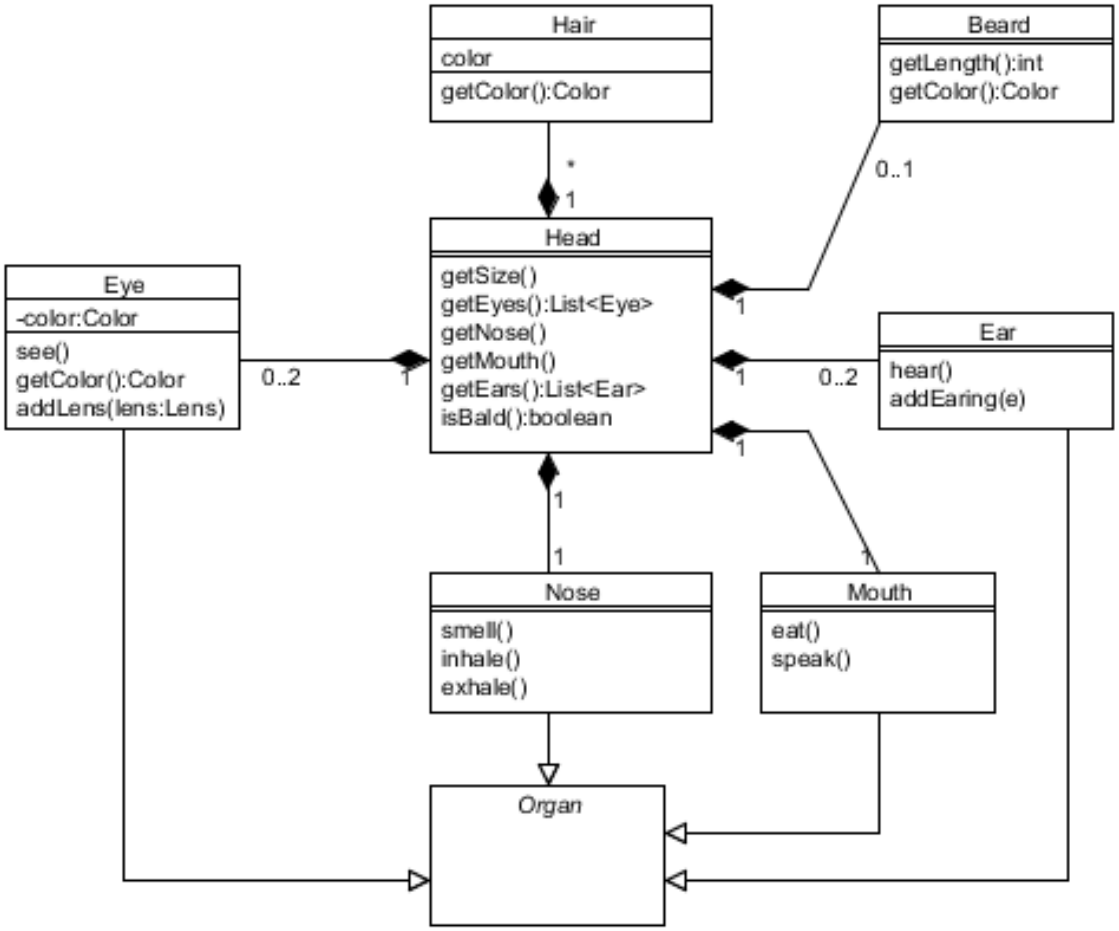
Розпочніть з класу *person*. Який вигляд матиме список передування класів *wkg-father* і *wkg-mother*?

Нехай класи мають такі переваги при виборі страв на сніданок: *man – donut; woman – croissant; parent – fruit; worker – bacon*. Закодуйте ці переваги в обробниках повідомлень цих класів таким чином, щоб клас–одержувач повідомлення повернув назву блюда, якому він надає перевагу. Сформуйте такі екземпляри класів: *Joan* – екземпляр класу *wkg-mother*, *Jim* – екземпляр класу *wkg-man*.

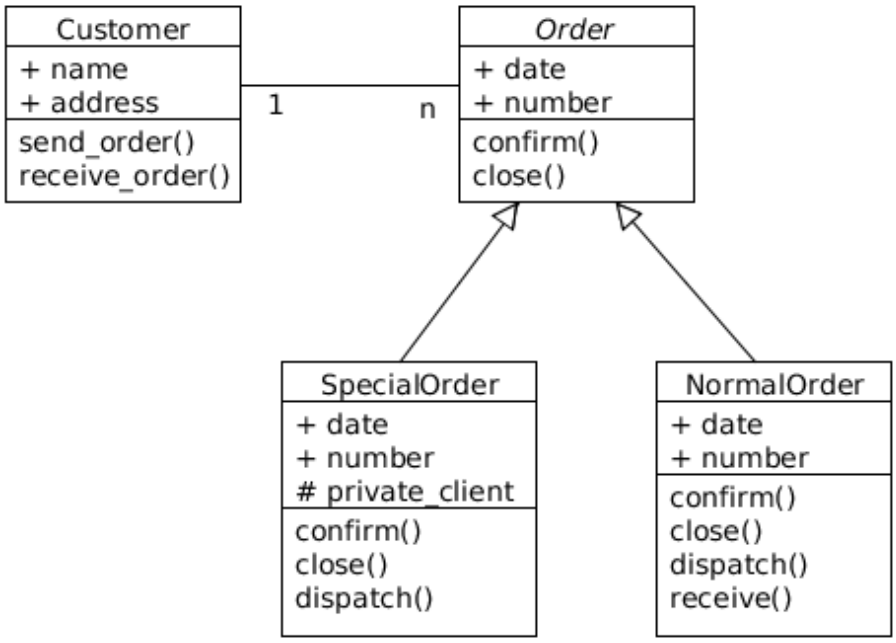


Визначити, яку відповідь нададуть екземпляри *Joan і Jim* на повідомлення ([Joan або Jim] breakfast) і чому.

21. Описати структуру класу «Голова», зображену на схемі. Для зазначених класів задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про назву органу.



22. Описати структуру системи обробки замовлень, зображену на схемі. Для зазначених класів задати оброблювачі повідомлень, які при зверненні до них повертають повідомлення про назву замовлення.



Система обробки замовлень

23–28. **Класифікація.** При необхідності сформувати Дерево класів: клас – корінь має загальний метод «*reply*», а кожен з його підкласів – свій власний

метод, який викликається методом «reply». Напишіть інструкції *class*, які імітували б сформовану модель класифікації засобами успадкування в Python.

23. *Класифікація тварин в зоології.* Вивчіть дерево класів, зображене на малюнку (клас Animal має загальний метод «reply», але кожен з класів має свій власний метод «speak», який викликається методом «reply»). Напишіть шість інструкцій *class*, які імітували б цю модель класифікації засобами успадкування в Python. Потім додайте до кожного з класів метод *speak*, яки виводив б унікальне повідомлення, і метод *reply* в суперкласі *Animal*, що є вершиною ієрархії, який просто викликав би *self.speak*, щоб вивести текст повідомлення, характерного для кожної категорії, в підкласах, розташованих нижче (це змусить розпочати пошук в дереві успадкування від екземпляру *self*).

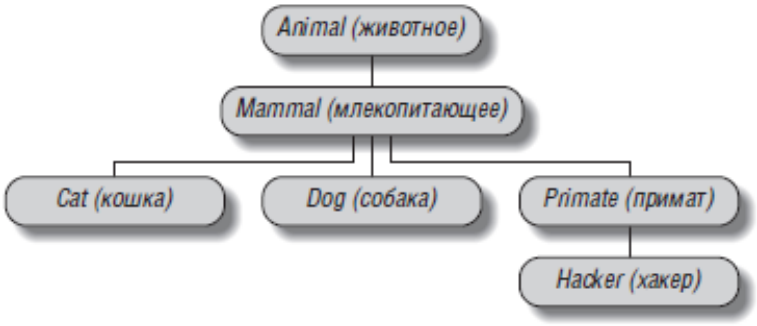


Рис. до завдання 22

Видаліть метод *speak* з класу *Hacker*, щоб для нього за замовчуванням виводилося повідомлення, яке було успадковане від класу, розташованого вище. Наприкінці класи повинні працювати таким чином:

```
>>> from zoo import Cat, Hacker
>>> spot = Cat()
>>> spot.reply() # Animal.reply; викликає Cat.speak
meow
>>> data = Hacker() # Animal.reply; викликають Primate.speak
>>> data.reply()
Hello world!
```

24. *Міністерство охорони здоров'я*. До структури Міністерства охорони здоров'я України входять такі підрозділи:

- відділ із забезпечення діяльності Міністра,
- департамент лікувально-профілактичної допомоги,
- департамент охорони материнства, дитинства та санаторного забезпечення,
- департамент кадрової політики, освіти, науки та запобігання корупції,
- департамент з реформ та розвитку галузі охорони здоров'я,
- адміністративно-господарське управління,
- управління бухгалтерського обліку,
- контрольно-ревізійний відділ,
- сектор режимно-секретної роботи.

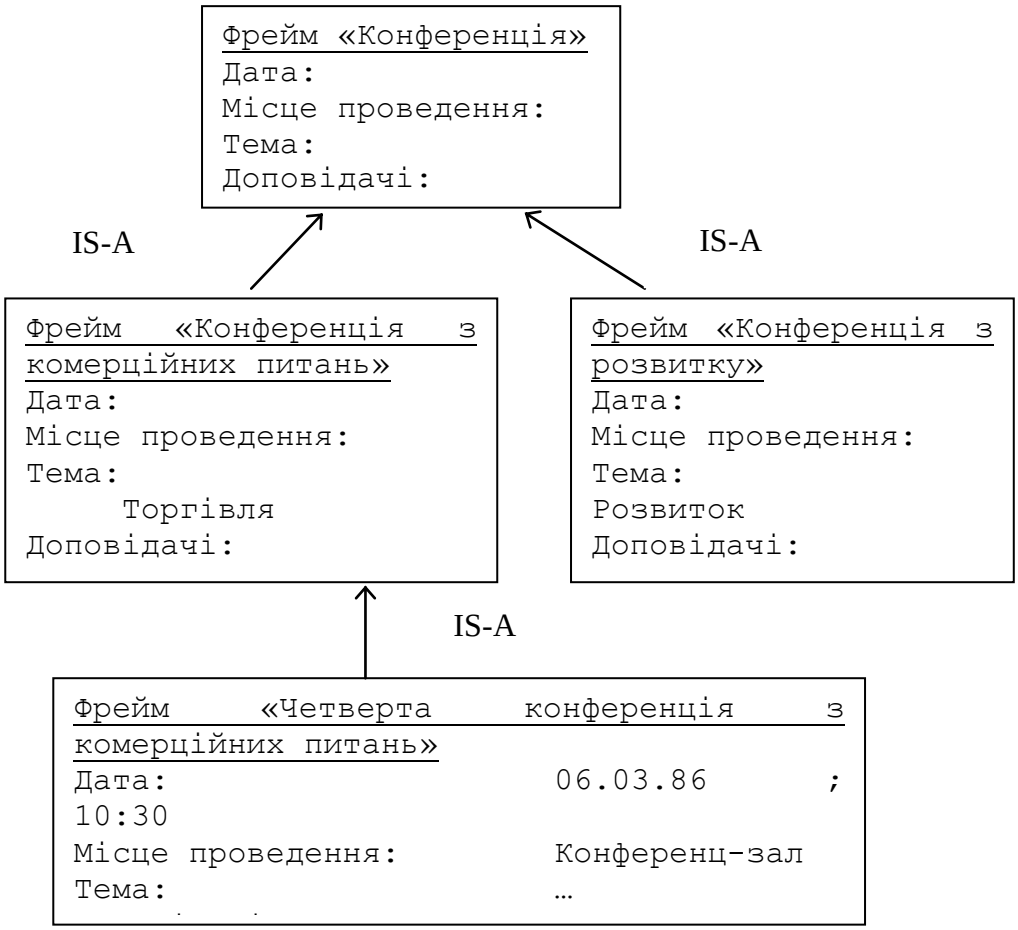
25. *Класифікація областей в Україні* (Україна – область – міста).

26. *Класифікація «*Дерево захворювань внутрішніх органів».

Рис. Фрагмент семантичної мережі «Дерево захворювань»

27. *Класифікація «*Аудиторії на факультеті: лекційні, комп'ютерні, практичні, лінгофонний, …» (аудиторія – підклас – номер)

28. «Ієрархічна структура інтелектуальної системи планування конференцій»



Контрольні запитання

1. Назвіть принципи ООП.

2. Назвіть основні переваги ООП.

3. Як можна визначити відношення «суперклас/підклас».

4. Назвіть призначення функції *issubclass/*

Аудиторна робота

Приклад 15.1 Об'єкти можна створювати як на основі класів, так і суперкласів. Створимо два класи: *Table* і *Kitchen* (другий є підкласом першого і успадковує всі його атрибути – методи __init__ і *outing*).

```
class Table:
 def __init__(self,l,w,h):
  self.long = l; self.width = w
  self.height = h
 def outing(self):
  print (self.long,self.width,self.height)
class Kitchen(Table):
 def howplaces(self,n):
  if n < 2: print ("It is not kitchen table")
  else: self.places = n
 def outplases(self):
  print (self.places)
t_room1 = Kitchen(2,1,0.5)
t_room1.outing(); t_room1.howplaces(5)
t_room1.outplases()
t_2 = Table(1,3,0.7); t_2.outing()
```

Наприкінці створено два об'єкти: *t_room*1, *t*_2. Перший об'єкт належить до класу *Kitchen* і успадковує атрибути цього класу і його суперкласу. Другий об'єкт належить класу Table; до класу Kitchen він ніякого відношення не має і тому не може звертатися до методів *howplaces* і *outplases.*

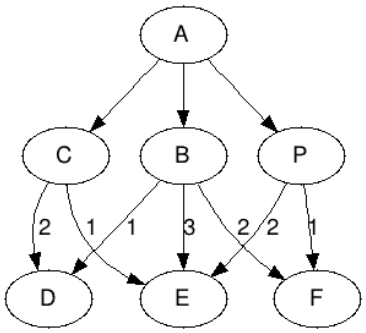
Приклад 15.2. Описати структуру класів, зображену на схемі, таким чином, щоб дотримувався заданий порядок спадкування в класах, які мають кілька суперкласів:

A ()	P (A)
B (A)	E (C, P, B)
C (A)	F (P, B)
D (B, C)	

Розпочніть з класу *A*. Нехай класи мають такі переваги при виборі:

C: Chloe; B: Bob; P: Piter.

Закодуйте ці переваги в обробниках повідомлень цих класів. Маємо такі екземпляри (*Denis* of D); (*Eelay* of E); (*Forth* of F)).




```
class A():
     def __call__(self): return 'Student '

class C(A):
     def __call__(self): return 'Chloe '
class B(A):
     def __call__(self): return 'Bob '
class P(A):
     def __call__(self): return 'Piter '

class D(B,C):
 def __init__(self):
  self.name='Denis'

print(A().__call__()+super().__call__()+'('+super(B,self).__call__()+')'
+' like '+self.name)
class E(C,P,B):
 def __init__(self):
  self.name='Eelay'
  print(A().__call__()+super().__call__()+'('+super(C,self).__call__()+
super(P,self).__call__()+')'+' like '+self.name)
class F(P,B):
 def __init__(self):
  self.name='Forth'

print(A().__call__()+super().__call__()+'('+super(P,self).__call__()+')'
+' like '+self.name)

I1=D(); I2=E(); I3=F()
```

Результат

```
Student Bob (Chloe ) like Denis
Student Chloe (Piter Bob ) like Eelay
Student Piter (Bob ) like Forth
```

Приклад 15.3. Успадкування класів

1. Успадкування класів

```
class Figure(object):
    def __init__(self, side):
        self.side = side

class Square(Figure):
    def draw(self):
        for i in range(self.side):
```

```
            print('*' * self.side)
class Triangle(Figure):
    def draw(self):
        for i in range(self.side):
            print('*' * i)

def main():
    square = Square(10); square.draw(); print()
    triangle = Triangle(5)
    triangle.draw()
if __name__ == '__main__': main()
```

2. Множинне успадкування

```
class Bird(object):
    def fly(self):
        print('I am flying.')

class Horse(object):
    def run(self):
        print('I am running.')

class Pegasus(Horse, Bird):  pass

def main():
    bird = Bird()
    horse = Horse()
    pegasus = Pegasus()
    bird.fly()
    # bird.run()  # ошибка
    print()

    # horse.fly()  # ошибка
    horse.run(); print()

    pegasus.fly(); pegasus.run()

if __name__ == '__main__': main()
```

3. Порядок дозволу методів при ромбоподібному множинному успадкуванні

```
class A(object):
    def method(self):
        print('A method')

class B(A): pass

class C(A):
    def method(self):
        print('C method')

class D(B, C):  pass

obj = D()
```

```
obj.method()  # 'C method'
```

4. Порядок дозволу методів при ромбоподібному множинному успадкуванні. Тут не вказано, що клас A успадковується від *object*. Даний клас буде успадкований від нього за замовчуванням. Результатом буде «C method». У попередньому прикладі обидві версії інтерпретатора викличуть метод класу C. При цьому відповідно до ієрархії класів саме метод класу C і повинен бути викликаний.

```
class A:
    def method(self):
        print('A method')

class B(A):  pass

class C(A):
    def method(self):
        print('C method')

class D(B, C):  pass

obj = D(); obj.method()
```

5. Якщо в заданому класі атрибут або метод був перевизначений, то доступ до відповідного атрибуту суперкласу можна отримати двома способами.

Перший спосіб – явне звернення до атрибутів суперкласу за його ім'ям. Недолік такого підходу – ускладнюється підтримка коду:

```
class Base:
  attr = 'Base attribute'
  def method(self):
    print('Base       method,       current       class       is',
self.__class__.__name__)

class Child(Base):
  attr = 'Redefined attribute'
  @staticmethod
  def get_superclass_attr():
    return Base.attr  # отримання атрибуту класа Base
  def method(self):  # перевизначення методу
    Base.method(self)  # вызов метода суперкласса
    print('Child       method,       current       class       is',
self.__class__.__name__)

def main():
    print('Base:'); print(Base.attr)
    base_instance = Base(); base_instance.method()
    print();  print('Child:'); print(Child.attr)
    print(Child.get_superclass_attr())
    child_instance = Child()
    child_instance.method()

if __name__ == '__main__':     main()
```

Другий спосіб. Існує спеціальний клас *super*, екземпляри якого є спеціальними проксі-об'єктами, прив'язаними до заданої ієрархії класів і надають доступ до атрибутів наступного класу в лінеаризації того класу, в якому був створений об'єкт *super*. Отже, за допомогою *super* можна отримати доступ до атрибутів і методів суперкласу, не називаючи його імені, причому це буде давати коректні результати навіть при використанні множинного успадкування. Виклик super().method() еквівалентний виклику super(__class__, self).method().

```
class Base(object):
  attr = 'Base attribute'
  def method(self):
    print('Base       method,       current       class       is',
self.__class__.__name__)

class Child(Base):
  attr = 'Redefined attribute'
  def get_superclass_attr(self):
   return super().attr # отримання атрибуту класа Base

  def method(self):     # перевизначення методу
    super().method()    # виклик метода суперкласу
    print('Child       method,       current       class       is',
self.__class__.__name__)
def main():
    print('Base:'); print(Base.attr)
    base_instance = Base()
    base_instance.method(); print()
    print('Child:'); print(Child.attr)
    child_instance = Child()
    print(child_instance.get_superclass_attr())
    child_instance.method()
    print()
# super можна викликати де завгодно, навіть за межами класу
    super(Child, child_instance).method()

if __name__ == '__main__':  main()
```

7. Використання *super* при множинному успадкуванні

```
class Animal(object):
  def __init__(self):
     self.can_fly = False
     self.can_run = False
  def print_abilities(self):
     print(self.__class__.__name__)
     print('Can fly:', self.can_fly)
     print('Can run:', self.can_run); print()

class Bird(Animal):
  def __init__(self):
     super().__init__()
```

```
     self.can_fly = True
class Horse(Animal):
  def __init__(self):
     super().__init__()
     self.can_run=True

class Pegasus(Horse, Bird):  pass

def main():
    bird = Bird()
    bird.print_abilities()
    horse = Horse()
    horse.print_abilities()
    pegasus = Pegasus()
    pegasus.print_abilities()

if __name__ == '__main__':  main()
```

8. Використання *super*, декоратору *gen_init* і побудова інтерпретатором лінеаризації. Декоратор *gen_init* додає автоматично згенерований конструктор, це – функція, яка приймає функцію або клас і повертає інший об'єкт, що буде прив'язаний до початкового імені. Зазвичай його використовують для зміни поведінки фукнції (шляхом створення нової функції, яка викликає початкову) або модифікації класу (наведено в даному прикладі).

Маємо таку ієрархію класів:


```
def gen_init(cls):
   """ Декоратор gen_init:
    :param cls: клас, який підлягає модифікації
    :return:    клас із доданим конструктором  """

    def init(self):
        print('Entered', cls.__name__, "constructor")
        super(cls, self).__init__()
        print('Quit', cls.__name__, "constructor")
    cls.__init__ = init
    return cls
```

```
@gen_init
class A(object):  pass
@gen_init
class B(object):  pass
@gen_init
class C(A, B):  pass
@gen_init
class D(C, B):  pass
@gen_init
class E(D):  pass

print(E.__mro__); obj = E()
```

9. Функція *isinstance* (obj, cls) перевіряє, чи є *obj* екземпляром класу *cls* або класу, який є спадкоємцем класу *cls*

```
print(isinstance(8, int))  # True
print(isinstance('str', int))  # False
print(isinstance(True, bool))  # True
print(isinstance(True, int))  # True, тому що bool - підклас int
print(isinstance('a string', object)) # True, все є об'єктами
print(isinstance(None, object))  # True, навіть None
print(isinstance(False, str))  # False
print(isinstance(int, type))
# True, будь-який клас – об'єкт-екземпляр метакласу type
print(isinstance(42, type))  # False, 42 – це не тип даних
```

10. Функція *issubclass*(cls, base) перевіряє, чи є класс *cls* спадкоємцем класу *base*.

```
print(issubclass(bool, int))  # True
print(issubclass(float, int))  # False
print(issubclass(int, float))  # False
print(issubclass(complex, type))  # False
print(issubclass(complex, object))
# True, всё наследуется от object

class Base(object): pass
class Child(object): pass

print(issubclass(Child, Base))  # True
print(issubclass(Base, object))  # True
print(issubclass(Child, object))  # True по транзитивности
```

Теоретична частина

Ідеї (принципи) ООП. Виділяють такі основні ідеї ООП як успадкування, інкапсуляція і поліморфізм:

1. У*спадкування*. Можливість виділяти загальні властивості і методи класів в один клас верхнього рівня (батьківський). Класи, які мають загального батька, різняться між собою за рахунок включення до їх складу різних додаткових властивостей (атрибутів) і методів.

2. *Інкапсуляція*. Атрибути (властивості) і методи класу класифікують на доступні зовні (опубліковані) і недоступні (захищені). Захищені атрибути можна змінити, перебуваючи поза класом. Опубліковані атрибути також називають інтерфейсом об'єкта, тому що за їх допомогою з об'єктом можна взаємодіяти. Інкапсуляція покликана забезпечити надійність програми, тому що змінити істотні для існування об'єкта атрибути стає неможливо.

3. *Поліморфізм*. Поліморфізм має на увазі заміщення атрибутів, описаних раніше в інших класах: ім'я атрибута залишається колишнім, а реалізація вже іншою. Поліморфізм дозволяє адаптувати класи, залишаючи при цьому один інтерфейс взаємодії.

Переваги ООП. Серед них можна виділити:

1. *Використання одного і того ж програмного коду з різними даними.* Класи дозволяють створювати множину об'єктів, кожен з яких має власні значення атрибутів. Немає потреби вводити множину змінних (об'єкти отримують в своє розпорядження індивідуальний простір імен). Простір імен конкретного об'єкта формується на основі класу, від якого він був створений, а також на основі усіх батьківських класів даного класу.

2. *Успадкування і поліморфізм дозволяють не писати новий код, а налаштовувати вже існуючий, за рахунок додавання і перевизначення атрибутів*. Це веде до скорочення обсягу вихідного коду.

ООП дозволяє скоротити час на написання вихідного коду, проте ООП завжди передбачає велику роль попереднього аналізу предметної області, попереднього проектування. Від правильності розв’язків на цьому попередньому етапі залежить куди більше, ніж від безпосереднього написання вихідного коду.

Особливості ООП в Python. У порівнянні з іншими поширеними мовами програмування у Python можна виділити такі особливості, пов'язані з ООП:

1. Будь-які дані (значення) – це об'єкти. Число, рядок, список, масив і ін. – все є об'єктом. Бувають об'єкти вбудованих класів (ті, що перераховані в попередньому реченні), а бувають об'єкти класів користувача (їх створює програміст). Для єдиного механізму взаємодії передбачені методи перезавантаження операторів .

2. Клас – це об'єкт з власним простором імен. Тому правильніше було вживати замість слова «об'єкт», слово «екземпляр». І говорити «екземпляр об'єкта», маючи під цим на увазі створений на основі класу саме об'єкт, і «екземпляр класу», маючи на увазі сам клас як об'єкт.

3. Інкапсуляції в Python не приділяється особливої уваги. В інших мовах програмування зазвичай не можна отримати безпосередньо доступ до властивості, описаного в класі. Для його зміни може бути передбачений спеціальний метод. В Python це легко зробити, звернувшись до властивості класу із зовні. Незважаючи на це в Python передбачені спеціальні способи обмеження доступу до змінних в класі.

Комп'ютерний практикум № 16. ООП: розробка сховища даних

Мета роботи: ознайомитися з організацією та розробити сховища даних з використанням парадигми ООП та модулів *shelve і dbm*. *Об'єкт дослідження* – сховище даних з використанням парадигми ООП.

План

1. Розробка сховища даних (Приклад 16.1)

Завдання

1. Ознайомитися з теоретичним матеріалом.

2. Відповідно до свого варіанту

- визначити умови; за допомогою формул описати варіанти виконання необхідний дій; розробити програмний додаток, який розв'язує завдання;

- організувати, якщо треба, введення даних з клавіатури і виведення у консоль;

- атрибути пов'язати з методами, які дозволяють їх змінювати

- реалізувати такі кроки:

Крок 1. Створити клас, який містить конструктор классу; на основі класу створити декілька об'єктів.

Крок 2. Додати методи, які визначають поведінку об'єктів.

Крок 3. Перевантаження операторів.

Крок 4. Адаптація поведінки за допомогою підкласів. Визначити відношення «суперклас/підклас», додати підклас, який адаптує поведінку суперкласу або/і містить унікальні методи, відсутні в суперкласі (один метод перевизначає метод суперкласу, адаптуючи його, інший метод є новим доповненням до підкласу).

Крок 5. Адаптація конструкторів (додати адаптований конструктор в підклас).

Крок 6. Збереження об'єктів в базі даних (модулі *shelve* і *dbm*). Дослідження сховища в інтерактивному сеансі.

3. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних.

Варіанти: класифікація об'єктів або визначення їх структури

№	Об'єкт	№	Об'єкт
1	2	3	4
1	Аудиторія у ВНЗ: - лекційні: №301,302,…; - комп'ютерні: № 202, 207; 403,…; - викладацькі:…	2	Книга: - фантастика: …; - пригодницька…; - навчально-наукова: …
3	Континенти(Євразія, Африка, Північна Америка, Південна Америка,Австралія,Антарктида): - в північній півкулі:….;	4	Конференція: - з комерційних питань:…; - з розвитку: …; - наукова:…:

	- в південній півкулі:…		
1	2	3	4
5	Автомобіль: - вантажний:…; - легковий:…	6	Журнал: за періодичністю виходу: - щомісячні:…; - раз на кілька місяців:…
7	Науки: - суспільні:….; - природничі: ….; - точні:…	8	Програмне забезпечення комп'ютера: - прикладне: …; - системне: операційна система, …
9	Класифікація овочів за періодом розвитку рослин: - однорічні: помідор, огірок, гарбуз, кабачок, квасоля, горох; - дворічні: морква, пастернак, буряк, цибуля, - багаторічні: ревінь, спаржа	10	Навчальна дисципліна: -базової професійної та практичної підготовки:…; -природничо-наукові:…
11	Класифікація тварин, базуючись на способі їхнього пересування: - землею:…, - водою:…, - повітрям:….	12	Геометрична фігура з кутами: - трикутник: рівносторонній, рівнобедрений,…; - чотирикутник: …; - п'ятикутник:…; - багатокутник:…
13	Класифікація міст України за областю розташування: - Київська: Київ, Фастів,…; - Одеська: Одеса,…; - Дніпропетровська: Дніпро, …; - Херсонська: Херсон,…; - Закарпатська:….; - Тернопільська: …; …	14	Фрукт: - плоди з соковитим м'якушем із насінням (апельсини, дині, ягоди, яблука); - плоди з соковитим м'якушем з однією великою центральною кісточкою (черешня, слива, персики); - сухі плоди (горіхи, боби, горох)
15	Квартира в двоповерховому будинку: розташована на - першому поверсі:….; - другому поверсі:…	16	Підрозділи Міністерства охорони здоров'я України: - департамент лікувально-профілактичної допомоги:…, - департамент охорони материнства, дитинства та санаторного забезпечення:…, - департамент кадрової політики, освіти, науки та запобігання корупції:…, - департамент з реформ та розвитку галузі охорони здоров'я:…, - адміністративно-господарське управління:…, - контрольно-ревізійний відділ
17	Антибіотики за призначенням: - антибактеріальні:…, - протипухлинні:…, - протигрибкові:…	18	Держави, розташовані: - в північній півкулі:….; - в південній півкулі:…
19	Банк: -державний: Ощадбанк,…; - приватний: …	20	Будівельні матеріали та вироби - природні (бітум і асфальт, озокерит, казеїн, очерет, торф),

			- штучні (лаки, фарби).
1	2	3	4
21	Текстовий редактор: - порядковий (рядковий): edlin, що входив у склад MS-DOS; - екранний: Блокнот, …	22	Класифікація лікарських препаратів за впливом на функції окремих систем організму: - серцево-судинну систему:…; - органи дихання:…, - органи травлення: …; …
23	Антибіотики за походженням: - природні (натуральні), - напівсинтетичні, - синтетичні	24	Комп'ютер за призначенням: -калькулятор: …; - консольний комп'ютер:…; -мінікомп'ютер:…; -мейнфрейм:…; - персональний комп'ютер:…; - робоча станція:….; - комп'ютер для операцій з функціями:…; - сервер:….; - суперкомп'ютер:….
25	Підприємство: - державне:…; - приватне: …	26	Пухлини: - злоякісні (…), - доброякісні (…)
27	Бібліотека: - міська:…; - районна:…	28	Групи кафедри БМК освітньо-кваліфікаційного рівня «бакалавр» (успішність у навчанні): - 1-ого курсу: БС-61, БС-62; - 2-ого курсу: БС-51, БС-52; - 3-ого курсу:….; - 4-ого курсу:…

Контрольні запитання

1. У чому полягає основна ідея концепції проектування, яку називають *інкапсуляцією* в ООП.

2. У чому полягає основна ідея ООП.

3. Назвіть, які ви знаєте інструменти інтроспекції (спеціальні атрибути, функції, які забезпечують доступ до внутрішньої реалізації об'єктів).

Аудиторна робота

Приклад 16.1. Завдання «Збереження і обробка інформації про людей»

Створимо два класи: 1) *Person* – клас, який подає та обробляє інформацію про людей; 2) *Manager* – адаптована версія класу *Person*, що модифікує успадковану поведінку. Попутно створимо екземпляри обох класів і протестуємо їх можливості.

Розглянемо приклад використання класів – збережемо екземпляри в сховище у вигляді об'єктно-орієнтованої бази даних, який забезпечує довгострокове їх зберігання. Завдяки цьому можна використовувати програмний код прикладу як шаблон для створення своєї власної, повноцінної бази даних.

Крок 1. Створення екземплярів. Перше завдання – створити головний клас *Person.* Назвемо файл *person.py*, класу дамо ім'я *Person*: `class Person:`

На етапі конструювання викликають метод конструктора *__init_*_ для ініціалізації новоствореного екземпляра. Передамо конструктору аргументи з даними, які будуть зберігатися екземпляром, присвоїмо їх атрибутам аргументу *self*, додамо ініціалізацію полів запису:

```
class Person:
 def __init__(self, name, job, pay):
 # Конструктор має 3 аргументи
  self.name=name # заповнення поля під час створення
  self.job=job   # self – нового екземпляру класу
  self.pay=pay
```

Аргумент *self* подає новостворений екземпляр, а аргументи *name, job, pay* перетворюються в дані, які зберігаються в об'єкті для подальшого використання. Додамо значення за замовчуванням для аргументів конструктора:

```
class Person:
 def __init__(self, name, job=None, pay=0):
     self.name=name; self.job=job; self.pay=pay
```

Отже, при створенні екземплярів класу *Person* достатньо передавати тільки значення аргументу *name*, аргументи *job, pay* є необов'язковими (вони за замовчуванням отримують значення «None» і «0»). Аргумент *self* заповнюється інтерпретатором автоматично, і в ньому передається посилання на екземпляр створеного об'єкта – процес присвоєння значень атрибутів об'єкта *self* полягає у приєднанні їх до нового екземпляру. Протестуємо код, створивши кілька екземплярів класу і переглянувши вміст їх атрибутів, створених конструктором. Для цього додамо програмний код для самоперевірки:

```
class Person:
def __init__(self, name, job=None, pay=0):
 self.name=name; self.job=job; self.pay = pay

bob = Person('Bob Smith') # Тестування класу
sue = Person('Sue Jones', job='dev', pay=100000)
# автоматично запустить __init__
print(bob.name, bob.pay); print(sue.name, sue.pay)
# атрибути в об'єктах bob, sue різняться
```

Об'єкт *bob* отримує значення атрибутів *job, pay* за замовчуванням, а для об'єкта *sue* значення всіх атрибутів вказують явно. Якщо запустити цей файл як сценарій, програмний код в кінці файлу створить два екземпляри класу і виведе значення двох атрибутів для кожного з них (*name, pay*):

```
Bob Smith 0
Sue Jones 100000
```

Щоб тести виконувалися, тільки коли файл запускають як сценарій для тестування, можна використати перевірку атрибуту *__name__* модуля. Відповідні зміни в файлі наведено нижче:

```
class Person:
 def __init__(self, name, job=None, pay=0):
  self.name=name; self.job=job; self.pay=pay
if __name__ == '__main__':
# коли файл запускають для тестування
 bob = Person('Bob Smith')
 sue = Person('Sue Jones', job='dev', pay=100000)
 print(bob.name, bob.pay); print(sue.name, sue.pay)
 print('{0} {1}' .format(bob.name, bob.pay)) # метод format
 print('%s %s' %(bob.name, bob.pay)) # вираз форматування
```

Результат запуску:

```
Bob Smith 0
Sue Jones 100000
>>> import person
```

Тепер при імпортуванні файлу *person.py* інтерпретатор створить новий клас (але не буде використовувати його). Під час запуску файлу в якості сценарію інтерпретатор створює два екземпляри класу і виводить значення двох атрибутів для кожного з них (кожен екземпляр є незалежним простором імен, тому їх атрибути можуть мати різні значення).

Крок 2. Додавання методів, які визначають поведінку об'єктів. Клас *Person* виконує функції фабрики записів – він створює записи і заповнює їх поля (атрибути екземплярів). Можна застосувати до таких записів деякі операції над об'єктами. Хоча класи і додають додатковий структурний рівень, проте більшу частину своєї роботи вони виконують за рахунок впровадження і обробки даних базових типів, таких як списки і рядки (класи є лише невеликою структурної надбудовою). Наприклад, поле *name* в наших об'єктах є звичайним рядком, тому ми в змозі витягувати прізвища людей з наших об'єктів, розбиваючи значення атрибуту прогалинами і використовуючи операцію індексування. Все це – операції над базовими типами даних:

```
>>> name = 'Bob Smith' # рядок
>>> name.split() # витягування прізвища
['Bob', 'Smith']
>>> name.split()[-1]
# або [1], якщо ім'я завжди складається з 2 компонентів
'Smith'
```

Можна збільшити зарплату, змінивши значення поля *pay*:

```
pay = 100000 # проста змінна
pay *= 1.10  # збільшено на 10%, або pay=pay*1.10
print(pay)
```

Щоб застосувати зазначені вище операції до об'єктів класу *Person*, підставимо імена *bob.name* і *sue.pay* на місце *name* і *pay*. Операції залишаться тими ж, але в якості об'єктів операцій будуть використані атрибути класу:

```
# Обробка вбудованих типів: рядки, зміна значення
class Person:
 def __init__(self, name, job=None, pay=0):
  self.name = name; self.job = job; self.pay = pay
if __name__ == '__main__':
 bob = Person('Bob Smith')
 sue = Person('Sue Jones', job='dev', pay=100000)
 print(bob.name, bob.pay); print(sue.name, sue.pay)
 print(bob.name.split()[-1]) # витягання прізвища
 sue.pay *=1.10 # підвищення зарплатні
 print(sue.pay)
```

Тут додано в кінець три нових рядки, які витягують прізвище з об'єкта *bob*, використовуючи операції над рядками і списками, і збільшують зарплату *sue*, змінюючи значення атрибута *pay* за допомогою простої числової операції. У певному сенсі об'єкт *sue* є змінним – він допускає зміну інформації про стан:

```
Bob Smith 0
Sue Jones 100000
Smith
110000.0
```

Реалізуємо концепцію проектування, яку називають *інкапсуляцією*. Її ідея полягає в тому, щоб заховати логіку операцій за інтерфейсами і тим самим домогтися, щоб кожна операція мала одну реалізацію в програмі. Якщо в подальшому буде необхідно внести зміни, модифікувати код доведеться тільки в одному місці. Реалізуємо операції над об'єктами у вигляді *методів класу*, що дозволяє застосовувати їх до будь-яких екземплярів класу. Перемістимо реалізацію двох операцій з програми до методів класу, домігшись інкапсуляції. Змінимо програмний код самоперевірки і замінимо в ньому запрограмовані операції викликами методів:

```
# Додано методи, які інкапсулюють операції
class Person:
 def __init__(self, name, job=None, pay=0):
  self.name = name; self.job = job
  self.pay = pay
 # метод «виведення прізвища»
 def lastName(self): #методи, які реалізують поведінку екземплярів
  return self.name.split()[-1] # self – екземпляр
 # метод «зміна зарплати»
 def giveRaise(self, percent):
  self.pay=int(self.pay*(1+percent)) # внесення змін
if __name__ == '__main__':
 bob = Person('Bob Smith')
 sue = Person('Sue Jones', job='dev', pay=100000)
 print(bob.name, bob.pay); print(sue.name, sue.pay)
 # використовують методи
 print(bob.lastName(),';', sue.lastName())
```

```
sue.giveRaise(.10) print(sue.pay)
```

Методи – це звичайні функції, які приєднують до класів, вони призначені для обробки екземплярів цих класів. Метод *lastName* виконує над об'єктом *self* операцію, яка раніше виконувалася над об'єктом *bob* (тут *self* – це об'єкт, який є контекстом виклику методу). Метод *lastName* (є викликом функції) обчислює значення, яке пізніше можна використати програмою, що його викликає. Метод *giveRaise* виконує над об'єктом *self* операцію, яка раніше виконувалася над об'єктом *sue*. Якщо запустити сценарій, він виведе результат:

```
Bob Smith 0
Sue Jones 100000
Smith ; Jones
110000
```

Поле *pay* (заробітна плата) в об'єкті *sue* отримує цілочисельне значення після його збільшення – всередині методу результат арифметичної операції перетворений в ціле число за допомогою вбудованої функції *int*. Можна реалізувати округлення до центів за допомогою вбудованої функції *round*(N, 2), використати тип *decimal* для забезпечення фіксованого точності або зберігати грошові суми у вигляді дійсних чисел і відображати їх із застосуванням рядка формату % .2f або {0: .2f}. У нашому прикладі ми відкидаємо центи за допомогою функції *int*.

Крок 3. Перевантаження операторів. Можна задіяти можливість перевантаження операторів, – додавши в клас метод, який виконує вбудовану операцію, коли її застосовують до екземплярів класу. Зокрема, можна реалізувати *метод перевантаження операторів* ***__str__***, який викликають, коли екземпляр перетворюють в рядок для виведення. Оскільки цей метод використовують для виведення даних про об'єкт, все, що ми отримаємо при виведенні об'єкта, є значенням методу *__str_*_ цього об'єкта, який може бути визначений в класі об'єкта або успадкований від суперкласу (методи, імена яких розпочинаються і закінчуються двома символами підкреслення, успадковуються так само, як будь-які інші). Додамо реалізацію цього методу в клас. Нижче наведено розширену версію класу, яка виводить список атрибутів при відображенні екземплярів повністю:

```
# Додано метод __str__, який реалізує виведення об’єктів повністю
class Person:
     def __init__(self, name, job=None, pay=0):
      self.name=name; self.job=job; self.pay=pay
     def lastName(self):
      return self.name.split()[-1]
     def giveRaise(self, percent):
      self.pay = int(self.pay * (1 + percent))
     def __str__(self): # доданий метод
      return ‘[Person: %s, %s]’ %(self.name, self.pay)
      # рядок для виведення
     if __name__ == ‘__main__’:
      bob = Person(‘Bob Smith’)
      sue = Person(‘Sue Jones’, job=’dev’, pay=100000)
```

```
print(bob); print(sue);
print(bob.lastName(),';', sue.lastName())
sue.giveRaise(.10); print(sue)
```

Тут в методі *__str__* для створення рядка виводу застосувано оператор форматування % (для реалізації необхідних дій класи можуть використати вбудовані типи об'єктів і операції). Змінено код самоперевірки – він виводить не окремі атрибути об'єктів, а об'єкт повністю. Якщо запустити цей сценарій, отримаємо такий результат – функції *print* викликатимуть метод *__str__*, який повертає рядки вигляду «[...]»:

```
[Person: Bob Smith, 0]
[Person: Sue Jones, 100000]
Smith ; Jones
[Person: Sue Jones, 110000]
```

Крок 4. Адаптація поведінки за допомогою підкласів. У класі *Person* задіяно більшість механізмів ООП, але не задіно адаптацію коду за рахунок успадкування. Використано наслідування – екземпляри успадковують методи свого класу. Тепер необхідно визначити відношення «суперклас/підклас», яке дозволить трохи змінити успадковану поведінку. У цьому і полягає основна ідея ООП – адаптація наявного коду дозволяє скоротити час, який витрачають на його розробку.

Створення підкласів. Застосуємо методологію ООП і адаптуємо клас *Person*, розширивши ієрархію об'єктів. Для цього визначимо підклас *Manager*, який успадковує клас *Person.* В підкласі *Manager* змінимо успадкований метод *giveRaise* на більш вузькоспеціалізовану версією:

```
class Manager(Person): # визначити підклас класу Person
```

Визначено клас з ім'ям *Manager*, який успадковує і може адаптувати суперклас *Person* (клас *Manager* схожий на клас *Person*, але реалізує свій спосіб збільшення зарплати). Нехай менеджер (екземпляр класу *Manager*) зазвичай отримує не тільки надбавку (яка передається у вигляді відсотків), але й додаткову премію (яка за замовчуванням становить 10%). Наприклад, якщо надбавка до зарплати менеджера становить 10%, то реально зарплата буде збільшена на 20%. Новий метод розпочинається, як показано нижче. Згідно з правилами, пошук в дереві успадкування закінчується, як тільки знайдено перший метод з відповідним ім'ям. Необхідно виконати операцію *giveRaise* і додати додаткову премію (викликати оригінал зі зміненими аргументами):

```
class Manager(Person): # Наслідує атрибути класу Person
 def giveRaise(self,percent,bonus=.10): #перевизначити для адапт-ї
   Person.giveRaise(self,percent+bonus) #доповнює оригінал
```

Методи класу можна викликати на основі звернення до екземпляру або звернення до класу. Виклик методу: *instance.method*(args ...) транслюється інтерпретатором в еквівалентну форму: *class.method*(instance, args ...), де клас, який містить метод, що викликають, визначають відповідно до правил пошуку

в дереві спадкування, які діють і для методів. Нижче наведено код після виконання останніх змін:

```
# Додано підклас, який адаптує поведінку суперкласу
class Person:
 def __init__(self, name, job=None, pay=0):
  self.name = name; self.job = job; self.pay = pay
 def lastName(self):
  return self.name.split()[-1]
 def giveRaise(self, percent):
  self.pay = int(self.pay*(1+percent))
 def __str__(self):
  return '[Person: %s, %s]' % (self.name, self.pay)

class Manager(Person):
 def giveRaise(self, percent, bonus=.10): # Перевизначення метода
  Person.giveRaise(self,percent+bonus) #Виклик м-ду з класу Person

if __name__ == '__main__':
 bob = Person('Bob Smith')
 sue = Person('Sue Jones',job='dev',pay=100000)
 print(bob);print(sue);print(bob.lastName(),sue.lastName())
 sue.giveRaise(.10); print(sue)
 tom = Manager('Tom Jones', 'mgr', 50000)
 # екземпляр Manager: __init__
 tom.giveRaise(.10) # Виклик адаптованої версії
 print(tom.lastName()) # Виклик успадкованого метода
 print(tom) # Виклик успадкованого __str__
```

Нижче наведено результат роботи оновленої версії коду:

```
[Person: Bob Smith, 0]
[Person: Sue Jones, 100000]
Smith ; Jones
[Person: Sue Jones, 110000]
Jones
[Person: Tom Jones, 60000]
```

Результати тестування за участю об'єктів *bob* і *sue* виглядають, як і раніше, а коли для екземпляра *tom* класу *Manager* виконують підвищення зарплати на 10%, реальне підвищення становить 20% (його зарплата збільшилася з $50K до $60), тому що адаптовану версію методу *giveRaise* в класі *Manager* викличуть тільки для цього об'єкта. При виведенні інформації про об'єкт *tom* використовують форматування, визначене в методі *__str__* класу *Person*: екземпляри класу *Manager* успадковують його, а також методи *lastName* і *__init__* від класу *Person*. Додамо в кінець коду такий текст:

```
if __name__=='__main__':
 print('--All three--')
 for object in (bob, sue, tom):
  # Обробка об'єктів узагальненим способом
  object.giveRaise(.10) #викличе метод giveRaise цього об'єту
  print(object) #викличе  метод __str__
```

Цей приклад демонструє «поліморфізм» в дії в Python – дія операції *giveRaise* залежить від того, до якого об'єкту її застосовують. Враховуючи, що вибір версії методу *giveRaise* грунтується на типі об'єкта, в результаті *sue* отримує надбавку у 10%, а *tom* – у 20%. Нижче наведено результат роботи коду:

```
Bob Smith, 0
Sue Jones, 100000
Smith Jones
Sue Jones, 110000
Jones
Tom Jones, 60000
--All three—
[Person: Bob Smith, 0]
[Person: Sue Jones, 121000]
[Person: Tom Jones, 72000]
```

Тут змінна *object* може посилатися або на екземпляр класу *Person*, або на екземпляр класу *Manager*, а інтерпретатор викличе відповідний метод *giveRaise*: 1) для об'єктів *bob*, *sue* буде викликана оригінальна версія методу з класу *Person*, 2) для об'єкта *tom* – адаптована версія з класу *Manager*.

Якщо для класу *Manager* потрібно реалізувати щось інше, можна додати в клас *Manager* унікальні методи, відсутні в класі *Person*. Нижче наведено фрагмент, в якому метод *giveRaise* перевизначає метод суперкласу, адаптуючи його, а метод *someThingElse* є новим доповненням до класу *Manager*):

```
class Person:
 def lastName(self): ...
 def giveRaise(self): ...
 def __str__(self): ...
class Manager(Person): # Наслідування
 def giveRaise(self, ...): ... # Адаптація
 def someThingElse(self, ...): ... # Розширення

tom = Manager()
tom.lastName() # Успадкований метод
tom.giveRaise() # Адаптована версія
tom.someThingElse() # Додатковий метод
print(tom) # Успадкований метод перевантаження
```

Крок 5. Адаптація конструкторів. Безглуздо вказувати значення '*mgr*' (менеджер) в аргументі *job* (посада) при створенні об'єкта класу *Manager*: цю посаду мають на увазі при назві класу. Краще заповнювати цей атрибут автоматично, при створенні екземпляра класу *Manager*. Для цього можна адаптувати роботу конструктора в класі *Manager* так, щоб він автоматично підставляв назву посади.

Перевизначимо метод *__init__* в класі *Manager*, щоб він підставляв рядок '*mgr'* автоматично. Для цього будемо викликати метод *__init__* з класу *Person* за рахунок звернення до імені класу, щоб ініціалізувати інші атрибути об'єкта. У

сценарію, який наведено нижче, створено новий конструктор для класу *Manager* і змінено виклик, який створює об'єкт *tom* (ми не передаємо йому назву посади '*mgr*'):

```
# Файл person.py - опис класів
class Person:
 def __init__(self, name, job=None, pay=0):
  self.name=name
  self.job=job
  self.pay=float(pay)
 def lastName(self):  #методи, які реалізують поведінку екземплярів
  return (self.name.split()[-1]) # self -  екземпляр
 def giveRaise(self, percent):
  self.pay=int(self.pay*(1+percent)) # внесення змін
 def __str__(self):
  return ('%s, %s, %s' %(self.name,self.job, self.pay))
  # рядок для виведення

class Manager(Person):
 def __init__(self,name,pay): # перевизначений конструктор
  Person.__init__(self,name,'mgr',pay)
  # Виклик конструктора зі значенням job='mgr'
 def giveRaise(self, percent, bonus=.10):
  Person.giveRaise(self, percent+bonus)

if __name__=='__main__':  # файл запускають для тестування
 bob = Person('Bob Smith');
 sue=Person('Sue Jones', job='dev', pay=100000)
 print(bob); print(sue);
 print(bob.lastName(),';', sue.lastName())
 sue.giveRaise(.10) # використовують методи
 print(sue)

 tom = Manager('Tom Jones', 50000)
 tom.giveRaise(.10) # Виклик адаптованої версії
 print(tom.lastName()) # Виклик успадкованого метода
 print(tom) # Виклик успадкованого __str__
```

Тут використано прийом розширення конструктора *__init__*, раніше застосований для розширення методу *giveRaise* (виклик методу із суперкласу зі зверненням до імені класу та явна передача екземпляру *self*). Для ініціалізації атрибутів екземпляру задіяно логіку конструктора класу *Person*. Таку форму виклику конструктора суперкласу з конструктора підкласу використовують в Python. Механізм успадкування, реалізований в інтерпретаторі, дозволяє відшукати тільки один метод *__init__* на етапі конструювання – найнижчий в дереві класів. Якщо під час конструювання об'єкта потрібно викликати метод *__init__*, розташований вище, його необхідно викликати вручну, зверненням через ім'я суперкласу (можна явно передати необхідні аргументи конструктору суперкласу або взагалі його не викликати). Цей сценарій виводить такий результат:

```
Bob Smith, None, 0.0
Sue Jones, dev, 100000.0
Smith ; Jones

Sue Jones, dev, 110000
Jones
Tom Jones, mgr, 60000
```

Крок 6. Збереження об'єктів у сховищі (в базі даних). Маємо систему з двох модулів, яка не тільки реалізує поставлене завдання подання інформації про людей, а й надає узагальнений інструмент відображення атрибутів, який в майбутньому можна використати і в інших програмах. Помістивши функції і класи в модулі, ми забезпечили можливість багаторазового їх використання. А організувавши програмне забезпечення у вигляді класів, ми забезпечили можливість його розширення.

Об'єкти, створені за допомогою класів, не є справжніми записами в БД: по завершенні програми всі створені екземпляри зникають – вони є звичайними об'єктами в пам'яті комп'ютера і не зберігаються на пристроях довготривалого зберігання (наприклад у файлах), тому їх не можна відновити при наступному запуску програми. Можна організувати збереження об'єктів за допомогою *сховища об'єктів*, яке дозволяє відновлювати об'єкти після того, як програма створить їх і завершить роботу. Реалізуємо можливість збереження об'єктів.

Можливість збереження об'єктів у Python забезпечують три модуля в стандартній бібліотеці: модуль

pickle перетворює довільні об'єкти на Python в рядок байтів і назад;

dbm реалізує збереження рядків у файлах, які забезпечують можливість звернення за ключем;

shelve використовує перші два модуля, дозволяючи зберігати об'єкти в файлах – сховищах, які забезпечують можливість звернення за ключем.

Модуль *pickle*: здатний перетворити будь-який об'єкт, який знаходиться в пам'яті, в рядок байтів, який потім можна використати для відновлення оригінального об'єкта. Модуль *pickle* може обробляти майже всі створювані об'єкти (списки, словники, вкладені комбінації з цих об'єктів, а також екземпляри класів). Останнє особливо важливо, тому що ця можливість дозволяє зберігати дані (атрибути) і поведінку (методи) – ця комбінація еквівалентна «записам» і «програмам». Зберігаючи об'єкти в файлі у вигляді рядків за допомогою модуля pickle, ви фактично забезпечуєте довготривале зберігання цих об'єктів: пізніше досить завантажити ці рядки і відновити з них оригінальні об'єкти.

За допомогою модуля *pickle* достатньо організувати збереження об'єктів в простих файлах і завантаження їх із файлів, однак модуль *shelve* забезпечує додаткові зручності, дозволяючи зберігати об'єкти, оброблені модулем *pickle*, за ключем. Модуль *shelve* перетворює об'єкт в рядок за допомогою модуля *pickle* і зберігає його під зазначеним ключем у файлі *dbm*. Пізніше, коли це необхідно, модуль *shelve* витягує рядок за ключем і відтворює оригінал об'єкту в пам'яті

(за допомогою модуля *pickle*). У програмі звернення до об'єктів в сховищі виглядає як звернення до елементів словника: ви звертаєтеся до об'єкту за ключем, зберігаєте його, виконуючи присвоєння за ключем, і можете використовувати інструменти словників (*len, in, dict. keys*), щоб отримати додаткову інформацію. Модуль *shelve* відображає операції зі словником на об'єкти, які зберігаються у файлі. Єдина відмінність між сховищами об'єктів і словниками полягає в тому, що сховища необхідно попередньо відкривати, а потім закрити їх після внесення змін. Таким чином, сховища можна розглядати, як найпростіші бази даних, які дозволяють зберігати і витягувати об'єкти по ключу і тим самим забезпечують збереження об'єктів між запусками програми. Сховища не підтримують можливість виконання запитів, наприклад, на мові SQL, і відчувають нестачу додаткових можливостей, якими володіють розвинуті БД (такі як обробка транзакцій), проте об'єкти, що знаходяться в сховищі, здатні використовувати всю широту можливостей Python після того, як вони будуть вилучені назад.

Модуль shelve. 6.1. *Збереження об'єктів.* Напишемо сценарій, який зберігає екземпляри класів у сховищі – файл *makedb.py*. Це – новий файл, тому в нього необхідно імпортувати класи, щоб з їх допомогою створити кілька екземплярів для подальшого збереження. Раніше для завантаження класу в інтерактивну оболонку ми використовували інструкцію *from*, але існує два способи завантаження класів з модулів, так само як функцій та інших змінних:
1) завантажує клас за допомогою інструкції *import*

```
import person # завантажує клас за допомогою інструкції import
bob = person.Person(...) # звернення до класу через ім’я модуля
```

2) завантажує клас за допомогою інструкції *from*

```
from person import Person
 bob=Person(...) # звернення до імені класу
```

Для завантаження класів в сценарій використаємо інструкцію *from* (у цьому випадку доведеться менше вводити з клавіатури). Скопіюйте фрагмент, який створює екземпляри наших класів, в новий сценарій, щоб було що зберігати. Створивши екземпляри, можна зберегти їх в сховищі. Для цього достатньо імпортувати модуль *shelve*, відкрити нове сховище, вказавши ім'я зовнішнього файлу, виконати присвоєння об'єктів за ключем і після завершення закрити сховище:

```
# Файл makedb.py: зберігає об’єкти Person у сховищі
from person import Person, Manager # імпортує класи
bob = Person(‘Bob Smith’) # створення об’єктів для зберігання
sue = Person(‘Sue Jones’, job=’dev’, pay=100000)
tom = Manager(‘Tom Jones’, 50000)
import shelve
db = shelve.open(‘persondb’) # ім’я файлу у сховищі
for object in (bob, sue, tom): # як ключ використати атрибут name
db[object.name] = object # зберегти об’єкт у сховищі
db.close() # закрити після внесення змін
```

При присвоєнні об'єктів в якості ключів використовуються значення

атрибутів *name*. Ключами в сховище можуть бути будь-які рядки, які можна було б створити із застосуванням унікальних характеристик, таких як ідентифікатор процесу і значення часу (їх можна отримати за допомогою модулів *os* і *time* стандартної бібліотеки). Одне обмеження – ключі можуть бути тільки рядками і повинні бути унікальними, тому що під кожним ключем можна зберегти тільки один об'єкт (втім, таким об'єктом може бути список або словник, що містить множину об'єктів). А ось значеннями, які зберігаються по ключу, можуть бути об'єкти будь-якого типу: це можуть бути об'єкти вбудованих типів, таких як рядки, списки, словники і екземпляри класів користувача, а також вкладені комбінації з них. Якщо при запуску сценарій нічого не виводить, це означає, що він добре працює – не передбачено виведення інформації, тільки створення і зберігання об'єктів:

6.2. *Дослідження сховища в інтерактивному сеансі*. Маємо у поточному робочому каталозі один або більше файлів, імена яких розпочинаються з «*persondb*». Реально створювані файли можуть відрізнятися в залежності від платформи, функція *shelve.open*() створює файли в поточному робочому каталозі, якщо вказане ім'я файлу не містить повний шлях. Але незалежно від того, де зберігаються ці файли, вони забезпечують доступ по ключу до подання об'єктів, створених за допомогою модуля *pickle*. Не видаляйте ці файли – вони є базою даних, яку доведеться копіювати або переміщувати, коли ви будете створювати резервні копії вашого сховища або переносити його.

Можна заглянути всередину файлів сховищ за допомогою програми Проводник Windows (Windows Explorer) або за допомогою інтерактивної оболонки Python, проте ці файли мають двійковий формат і їх вміст не має великого сенсу поза модуль *shelve*. Наша БД зберігається в трьох файлах (модуль *bsddb* є стороннім доповненням, яке поширюють з відкритими вихідними текстами):

```
# Модуль, який дозволяє отримати список файлів у каталозі:
# перевірка наявності файлів
>>> import glob
>>> glob.glob('person*')
['person.py','person.pyc','persondb.bak','persondb.dat','persondb.dir']
# Тип файлу: текстовий – для рядків, бінарний – для байтів
>>> print(open('persondb.dir').read())
'Tom Jones', (1024, 91)
...частину рядків опущено...
>>> print(open('persondb.dat', 'rb').read())
b'\x80\x03cperson\nPerson\nq\x00)\x81q\x01}q\x02(X\x03\x00\x00\x00
payq\x03K...
...частину рядків опущено...
```

Щоб перевірити результат, напишемо ще один сценарій. Нижче наведено лістинг інтерактивного сеансу, який виконує роль клієнта БД (файл *travaille.py*):

```
import shelve

db=shelve.open('persondb', 'r') # відкрити сховище
print(len(db)) # у сховищі маємо три 'записи'
```

```
print(list(db.keys())) # keys – це заголовок
bob=db['Bob Smith'] # витягти об’єкт bob за ключем
print(bob) # викличе метод __str__
print(bob.lastName()) # Викличе lastName з класу Person
for key in db: # Ітерації, витягання, виведення
 print(key, '=>', db[key])

for key in sorted(db):
 print(key,'=>',db[key])  #Ітерації через відсортований список ключів
```

Результат роботи програми:

```
3
['Sue Jones', 'Tom Jones', 'Bob Smith']
[Person: Bob Smith, 0]
Smith

Sue Jones => [Person: Sue Jones, 100000]
Tom Jones => [Person: Tom Jones, 50000]
Bob Smith => [Person: Bob Smith, 0]

Bob Smith => [Person: Bob Smith, 0]
Sue Jones => [Person: Sue Jones, 100000]
Tom Jones => [Person: Tom Jones, 50000]
```

Коли модуль *pickle* перетворює екземпляр класу, він записує атрибути екземпляру *self* разом з ім'ям класу, з якого він був створений, та ім'ям модуля, де знаходиться визначення цього класу. Коли пізніше об'єкт *bob* витягують зі сховища, інтерпретатор імпортує клас і пов'язує з ним об'єкт *bob*. Завдяки цьому після завантаження екземпляри класів автоматично знаходять поведінку свого класу. Необхідно імпортувати класи, тільки якщо треба створювати нові екземпляри, але не для роботи з існуючими.

6.3. *Оновлення об'єктів в сховищі*. Створимо останній сценарій, який оновлює екземпляри (записи) при кожному запуску, щоб переконатися, що об'єкти дійсно зберігаються (тобто при кожному запуску програми доступні їх поточні значення). Файл *updatedb.py* виводить вміст БД і збільшує зарплату одному з об'єктів при кожному запуску. Цей сценарій має масу можливостей – при виведенні об'єктів автоматично викликають реалізацію методу *__str__* і підвищення зарплати виконують викликом методу *giveRaise*:

```
# Файл updatedb.py: оновлює об’єкт класу Person в БД
import shelve

db = shelve.open(‘persondb’)
# відкрити у сховищі файл з вказаним ім’ям
for key in sorted(db): # відобразити об’єкти з БД
 print(key, ‘\t=>’, db[key]) # Виведення в необхідному форматі

sue=db[‘Sue Jones’] # витягти об’єкти за ключем
sue.giveRaise(.10)  #  змінити об’єкт у пам’яті шляхом виклику метода
```

```
db[‘Sue Jones’]=sue # присвоїти за ключем,
# щоб оновити об’єкт у сховищі
db.close() # закрити  файл після внесення змін
```

Цей сценарій виводить вміст БД, тому його можна запустити кілька разів і побачити, як змінюються об'єкти. Нижче наведено результати декількох запусків сценарію (можна спостерігати, як підвищується зарплата *sue*):

```
Bob Smith      => [Person: Bob Smith, 0]
Sue Jones      => [Person: Sue Jones, 100000]
Tom Jones      => [Person: Tom Jones, 50000]
>>>
====================== RESTART: C:/Python34/updatedb.py =
Bob Smith      => [Person: Bob Smith, 0]
Sue Jones      => [Person: Sue Jones, 110000]
Tom Jones      => [Person: Tom Jones, 50000]
>>>
====================== RESTART: C:/Python34/updatedb.py =
Bob Smith      => [Person: Bob Smith, 0]
Sue Jones      => [Person: Sue Jones, 121000]
Tom Jones      => [Person: Tom Jones, 50000]
```

Це – результат роботи модулів *shelve* і *pickle*, які входять до складу Python, і поведінки, реалізуваної в класах. Можна перевірити результати запуску сценарію за допомогою інтерактивної оболонки (еквівалент клієнта БД на основі модуля *shelve*):

```
>>> import shelve
>>> db = shelve.open(‘persondb’) # відкрити  базу даних
>>> rec = db[‘Sue Jones’] # витягти об'єкт за ключем
>>> print(rec)
[Person: job=dev, name=Sue Jones, pay=146410]
>>> rec.lastName()
‘Jones’
>>> rec.pay
146410
```

Реалізовано збереження об'єктів, створених з класів, в об'єктно-орієнтованій БД із застосуванням модуля *shelve* – простій у використанні системі, яка забезпечує можливість збереження і вилучення об'єктів за ключем.

Модуль dbm. Повторимо пункти 6.1, 6.2, 6.3, використовуючи модуль *dbm*.

```
# Файл makedb.py: зберігає об’єкти Person у сховищі
import dbm

from person import Person, Manager # імпортує класи
bob=Person('Bob Smith') # створення об’єктів для зберігання
sue=Person('Sue Jones', job='dev', pay=100000)
tom=Manager('Tom Jones', 50000)

db = dbm.open('persondb','n') # ім’я файлу у сховищі
for object in (bob,sue,tom): # як ключ використати атрибут
```

```
name
        print(object)
        db[object.name]=str(object) # зберегти об’єкт у сховищі
    print(len(db))
    db.close() # закрити після внесення змін
```

Результат:

```
Bob Smith, None, 0.0
Sue Jones, dev, 100000.0
Tom Jones, mgr, 50000.0
3
```

```
# Файл travaille.py: робота з БД
import dbm

db=dbm.open('persondb', 'n') # відкрити сховище
k=len(db) # у сховищі маємо три ‘рядки-записи’
print(list(db.keys())) # keys – це заголовок
a=list(db.keys())
while k>0: # звернення до записів БД
    f1=db[a[k-1]].decode()
    print(f1);  k-=1
print()
for key in db: # Ітерації, витягання, виведення
 print(key, '=>', db[key])
print()
for key in sorted(db):
 print(key, '=>', db[key]) # Ітерації через відсортований
 # список ключів

db.close() # закрити після внесення змін
```

Результат:

```
[b'Bob Smith', b'Tom Jones', b'Sue Jones']
Sue Jones, dev, 100000.0
Tom Jones, mgr, 50000.0
Bob Smith, None, 0.0

b'Bob Smith' => b'Bob Smith, None, 0.0'
b'Tom Jones' => b'Tom Jones, mgr, 50000.0'
b'Sue Jones' => b'Sue Jones, dev, 100000.0'
За алфавітом:
b'Bob Smith' => b'Bob Smith, None, 0.0'
b'Sue Jones' => b'Sue Jones, dev, 100000.0'
b'Tom Jones' => b'Tom Jones, mgr, 50000.0'
```

```
# Файл updatedb.py: оновлює об’єкт класу Person в БД
import dbm
from person import Person, Manager

    def Transform(a): #функция которая переводит строку в список
с разделителем ','
```

```
        list=a.split(",")
        return list

    db=dbm.open('persondb', 'n') # відкрити сховище
    k=len(db) #  у сховищі маємо три 'записи'
    print(list(db.keys())) # keys - це заголовок
    key=list(db.keys());

    # відкрити у сховищі запис файлу за вказаним ключем
    for i in sorted(key): # відобразити об'єкти з БД
      print(i, '\t=>', db[i]) # Виведення в необхідному форматі
    print()

    while k>0: # звертаємося до записів БД
      if key[k-1]==b'Sue Jones':
        sue1=db[b'Sue Jones'].decode() # витягти запис за ключем
        # f1=db[key[k-1]].decode()
        print(sue1);break
      k-=1

    l=Transform(sue1);
    ssue=Person(l[0], l[1], l[2])
    ssue.giveRaise(0.10)  #  змінити  об'єкт  у  пам'яті  шляхом
виклику метода
    db[b'Sue Jones']=str(ssue) # присвоїти за ключем, щоб оновити
сховище

    db.close() # закрити  файл після внесення змін
```

Результат:

```
[b'Bob Smith', b'Sue Jones', b'Tom Jones']
b'Bob Smith'   => b'Bob Smith, None, 0.0'
b'Sue Jones'   => b'Sue Jones, dev, 100000.0'
b'Tom Jones'   => b'Tom Jones, mgr, 50000.0'

Sue Jones,  dev, 100000.0
========== RESTART: C:\Python34\dbm\updatedb.py =
[b'Tom Jones', b'Bob Smith', b'Sue Jones']
b'Bob Smith'   => b'Bob Smith, None, 0.0'
b'Sue Jones'   => b'Sue Jones,  dev, 110000'
b'Tom Jones'   => b'Tom Jones, mgr, 50000.0'

Sue Jones,   dev, 110000.0
========== RESTART: C:\Python34\dbm\updatedb.py =
[b'Bob Smith', b'Tom Jones', b'Sue Jones']
b'Bob Smith'   => b'Bob Smith, None, 0.0'
b'Sue Jones'   => b'Sue Jones,   dev, 121000'
b'Tom Jones'   => b'Tom Jones, mgr, 50000.0'

Sue Jones,    dev, 121000.0
```

Комп'ютерний практикум № 17. Розробка додатків з графічним інтерфейсом на основі бібліотеки tkinter

Мета роботи: ознайомитися з організацією графічного інтерфейсу на основі бібліотеки *tkinter*. *Об'єкт дослідження* – модуль *tkinter*.

План

1. Графічний інтерфейс (ГІ) та його елементи. Модуль *tkinter*

Завдання

1. Ознайомитися з теоретичним матеріалом. Опрацювати приклади.

2. Відповідно до свого варіанту

- визначити умови;

- за допомогою формул описати варіанти виконання необхідний дій;

- написати програму, яка розв'язує завдання - за допомогою стандартної бібліотеки *tkinter* запрограмувати відповідний графічний інтерфейс.

- після взаємодії користувача-людини з комп'ютером організувати виведення відповідного повідомлення у потік *stdout* програми (відповідну інформацію зафіксувати у змінній).

4. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних.

Варіанти

1. **Віджет Мітка (Label).** Створити ГІ, який виводить у вікні рядок символів.

2. **Меню на основе віджетів Frame і Menubutton.** Створити головне меню з двома меню: File і Edit; в меню Edit є власне вкладене підменю.

3. **Віджети Label, Button, Frame.** Створити ГІ – рис. 17.1. При цьому створити віджет *Frame*, до якого прикріплено три інших елемента – *Label* і два *Button* – шляхом передачі об'єкта *Frame* в першому аргументі (віджет *Frame* стає батьком трьох інших віджетів). Обидві кнопки цього інтерфейсу викликають такі обробники: 1) клацання на кнопці *Hello* запускає функцію *greeting*, визначену всередині цього файлу, яка виконує виведення повідомлення в потік *stdout*; 2) клацання на кнопці *Quit* викликає стандартний метод *quit*, який віджет *win* успадковує від класу *Frame*. Нижче наведено текст, який виводиться в *stdout* при натисканні на кнопку:

```
C:\...\PP4E\Gui\Intro> python gui4.py
Hello stdout world!...
```

Рис. 17.1. Вікно із декількома віджетами

4. **Віджет Text**. Створити вікно введення рядка тексту.

5. **Віджети Toplevel і Tk**. Створити ГІ, зображений на рис. 17.2.

Рис. 17.2. Три налаштованих вікна Toplevel

6. **Віджет з використанням сітки** (метод *grid*). Побудувати набори міток і полів введення даних із двох колонок.

7. **Віджети Listbox, Scrollbar.** Створити вікно (рис. 17.3), це – фрейм Frame зі списком *Listbox* в лівій частині, що містить 30 згенерованих елементів (на п'ятому виконати клацання), пов'язаних з віджетом *Scrollbar* в правій частині, призначеним для прокрутки списку. Якщо перемістити повзунок в смузі прокрутки, список також буде прокручуватися, і навпаки.

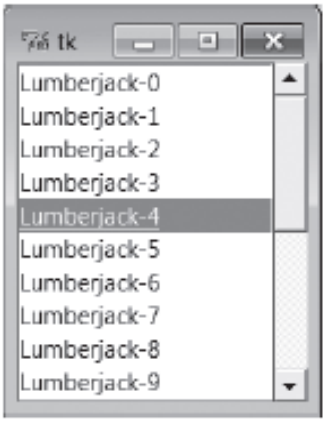


Рис. 17.3. Сценарій *scrolledlist*

8. **Віджет Message.** Створити ГІ, зображений на рис. 17.4.

9. **Віджет Entry.** Створити ГІ, зображений на рис. 17.5.

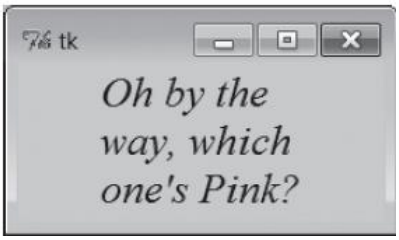


Рис. 17.4. Віджет Message

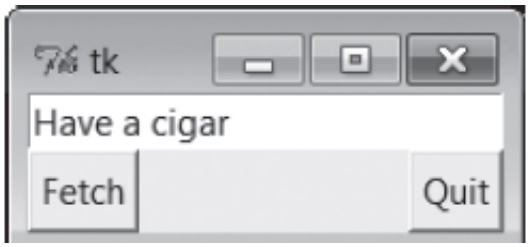


Рис. 17.5. Сценарій *entry*1 в дії

10. **Віджет Radiobutton.** Створити ГІ у вигляді групи перемикачів (рис.17.6).

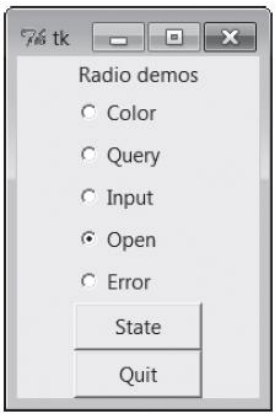


Рис. 17.6. Сценарій demoRadio в дії

11. **Віджет Entry**. Створити ГІ, зображений на рис. 17.7. Якщо натиснути кнопку *Dialog* в головному вікні, сценарій створить вікно діалогу з формою, зображеною на рис. 17.6: поля введення прикріплено до нового вікна *Toplevel*, яке містить кнопку OK, що генерує подію знищення вікна.

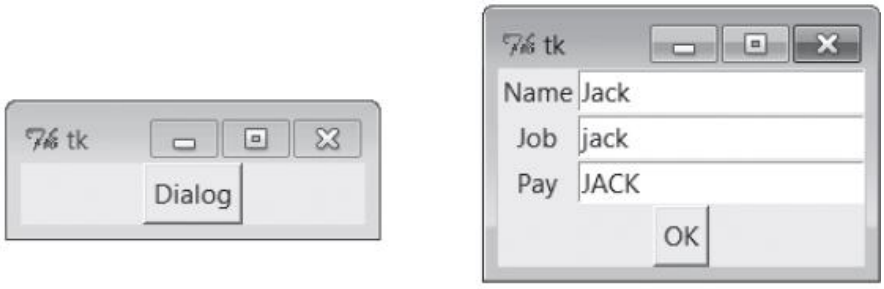


Рис. 17.7. Вікна, створені сценарієм entry2-modal

12. **Віджет Scale.** Створити ГІ, зображений на рис. 17.8. Маємо два повзунка – горизонтальний і вертикальний, зв’язані між собою через асоційовану змінну, що дозволяє їх синхронізувати.

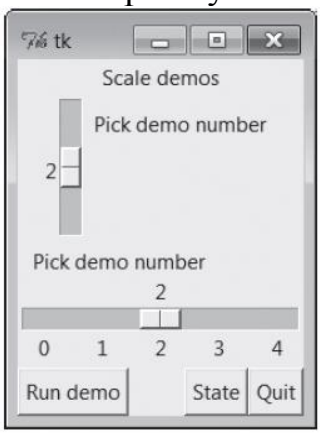


Рис. 17.8. Сценарій demoScale в дії

13. **Віджети Button.** Створити ГІ (рис. 17.9) із кнопкою, яка випадковим чином змінює свою картинку при кожному натисканні,

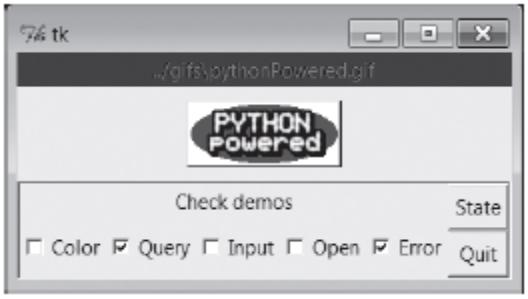


Рис. 17.9. Сценарій buttonpics в дії

14. Ієрархічна побудова ГІ. Створити вікно, в яке вбудовані екземпляри чотирьох знайомих нам демонстраційних панелей запуску діалогів (рис.17.10).

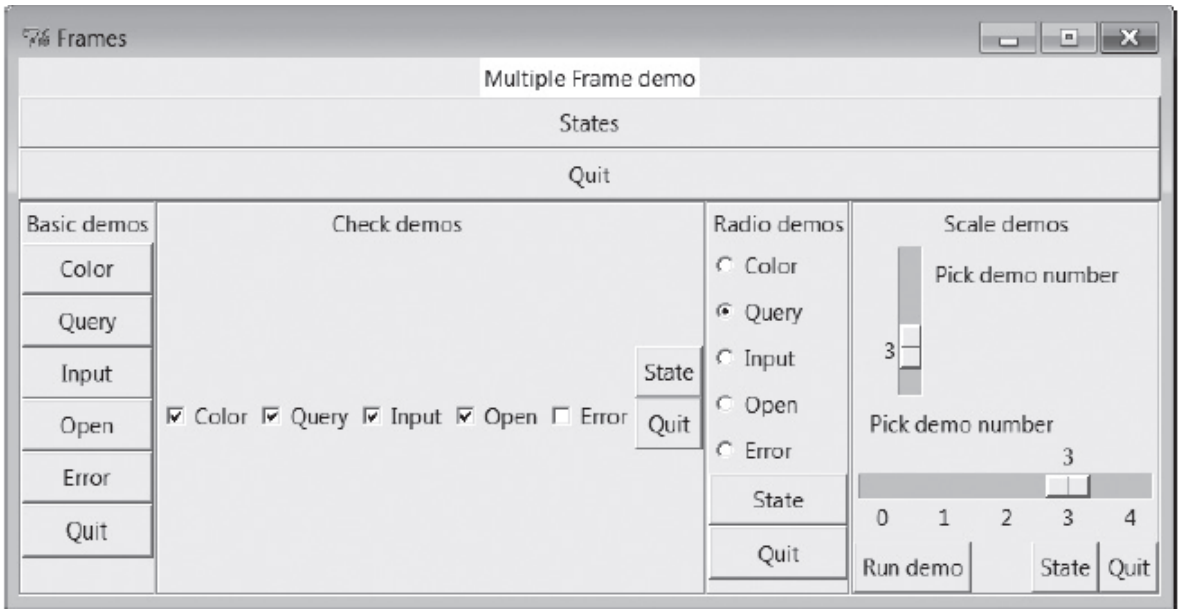


Рис. 17.10. demoAll_frm: вкладені фрейми

15. **Панель запуска демонстрації діалогів.** Створити ГІ – рис. 17.11. Сценарій створює головне вікно (17.11.а). Натискання кнопки *Quit* в цьому вікні виводить діалог (рис. 17.11.б), який блокує програму, поки користувач не клацне на одну з кнопок: 1) при натисканні кнопки Yes (або натиснути клавішу Enter) виклик діалогу повертає значення True, і сценарій виводить стандартний діалог (рис . 17.11.в). У діалозі на рис. 17.11.в користувач може натиснути тільки кнопку OK; 2) при натисканні кнопки No створиться відповідне вікно діалогу (рис. 17.11.г). Якщо в головному вікні клацнути на кнопці Spam, то буде створений стандартний діалог (рис. 17.11.д).

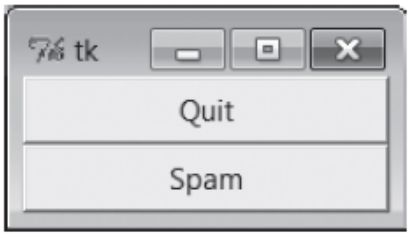


Рис. 17.11.а. Голлвне вікно dlg1: кнопки викликають появу додаткових вікон

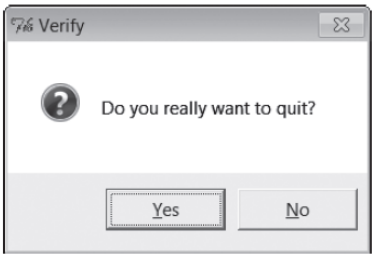


Рис. 17.11.б. Діалог *askyesno*, який виводиться сценарієм dlg1

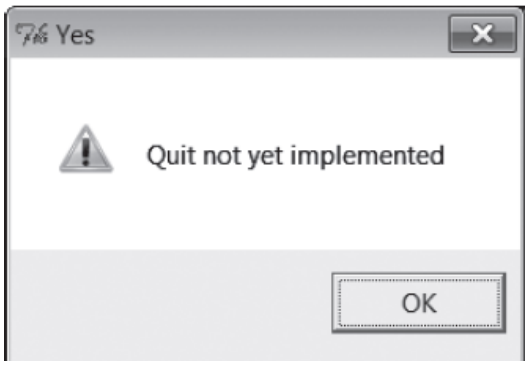


Рис. 17.11.в. Діалог showwarning, який виводиться сценарієм *dlg1*

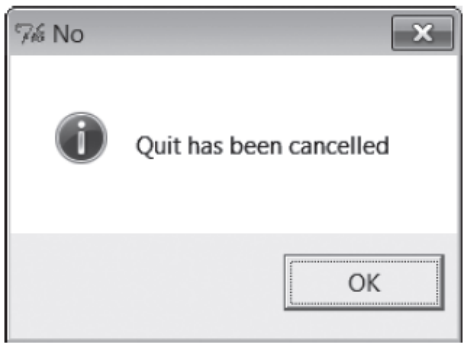


Рис. 17.11.г. Диалог showinfo, який виводиться сценарієм *dlg1*

Рис. 17.11.д. Діалог showerror, який виводиться сценарием dlg1

17. **Панель запуска демонстрації діалогів.** Створити ГІ – рис. 17.12. Вікно – це панель демонстраційних кнопок, при натисканні яких виконується управління відповідно до значень в таблиці модуля *dialogTable*. Сценарій управляється вмістом словника з модуля *dialogTable*.

Рис. 17.12. Головне вікно demoDlg

17. **Діалоги користувача** (створення модального або немодального вікон). Створити вікно з прикріпленими віджетами і додати обробник подій, який збере дані, введені користувачем (якщо вони є), і закриє вікно. Щоб зробити такий діалог модальним, необхідно передати вікну фокус введення, зробити інші вікна неактивними і очікувати події.

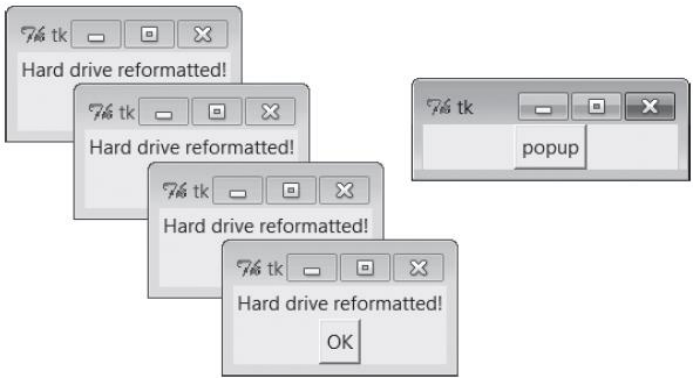


Рис. 17.13. Немодальні діалоги користувача в дії

18. **Віджет Checkbutton.** Створити ГІ – групу з п'яти прапорців, рис. 17.14. Маємо кнопку, за допомогою якої виводиться поточний стан всіх прапорців, і кнопку Quit.

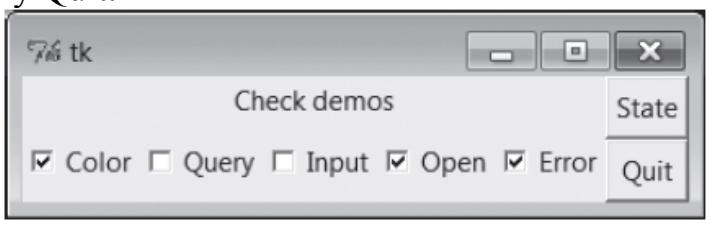


Рис. 17.14. Сценарій demoCheck в дії

19. **Віджет Entry.** Створити ГІ, зображений на рис. 17.15. Віджети Entry часто застосовують в якості полів введення при реалізації форм (кілька міток, полів введення і фреймів об'єднані в форму для введення декількох значень). Полями введення тут служать прості віджети Entry. Сценарій створює список віджетів, за допомогою якого потім будуть вилучатись їх значення. При кожному натисканні кнопки *Fetch* поточні значення беруться з усіх полів введення і виводяться у стандартний потік виводу.

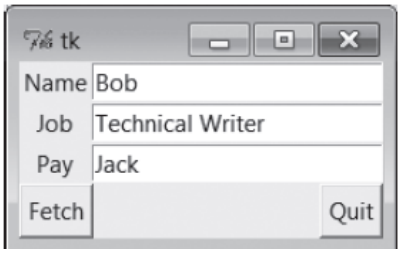


Рис. 17.15. Зовнішній вигляд форм entry2

20. **Віджет з використанням сітки: створення таблиць.** Створити вікно з масивом міток, яке складається з п'яти рядків і чотирьох стовпчиків, в якому кожна мітка виводить номер свого рядка і стовпчика (*row*, *col*).

21. **Віджет Canvas (полотно).** Створити вікно, зображене на рис. 17.16, реалізуючи прокрутку.

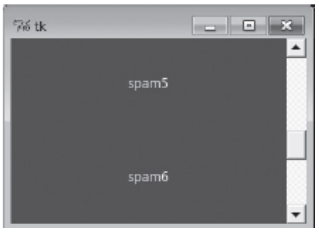


Рис. 17.16. Сценарій *scrolledcanvas* в дії

22. **Віджет з використанням сітки: створення таблиць.** Створити вікно, зображене на рис. 17.17. Маємо кнопку завантаження таблиці з файлу даних Файл містить рядки даних, розділених символом пробілу (або табуляції). При завантаженні даних з файлу автоматично змінюється розмір таблиці, щоб вмістити всі стовпчики.

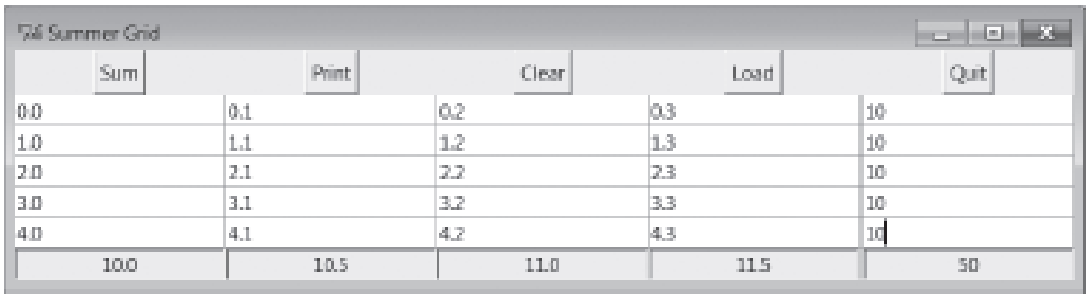


Рис. 17.17. Дані завантажують із файлу

23. **Віджет Canvas (полотно).** Створити вікно, зображене на рис. 17.18.

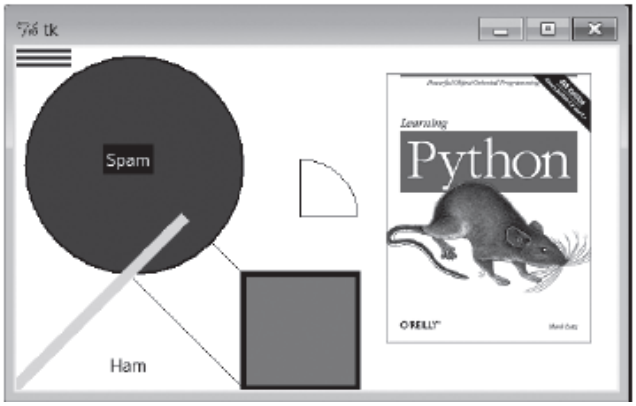


Рис. 17.18. Вікно сценарія *canvas1* із зображеннями об’єктів

24. **Віджет Text**. Створити вікно для перегляду файлу (текст файлу буде виведений в новому вікні, рис. 17.19.

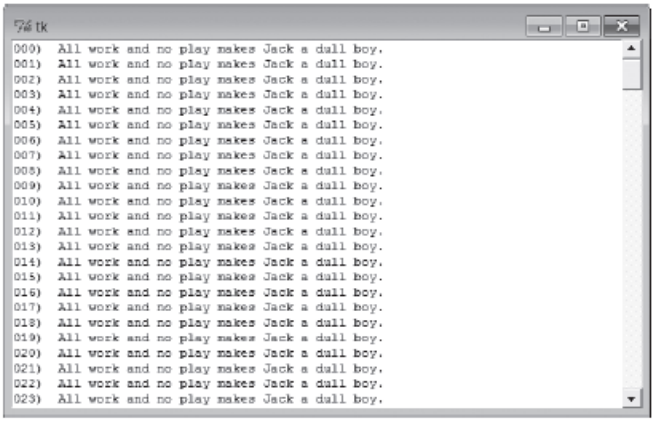


Рис. 17.19. Сценарій *scrolledtext* в дії

25. **Вікна з меню і панеллю інструментів.** Створити головне меню з двома меню: File і Edit; які мають власні вкладені підменю.

26. **Віджеты Toplevel і Tk**. Створити ГІ, зображений на рис. 17.20.

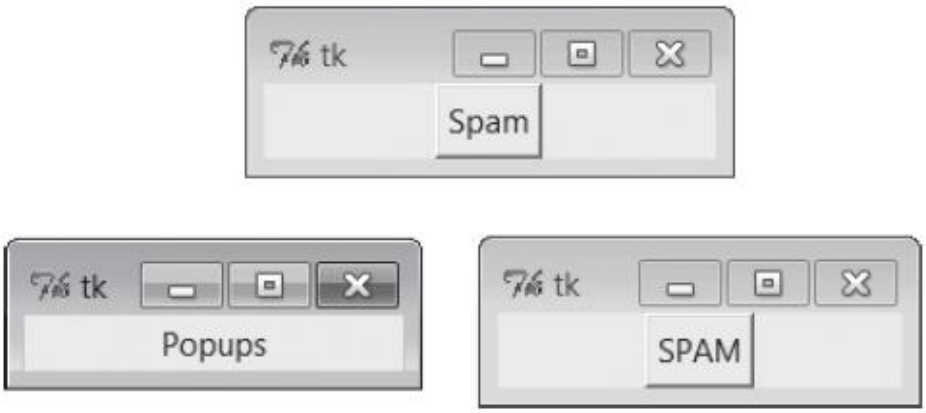
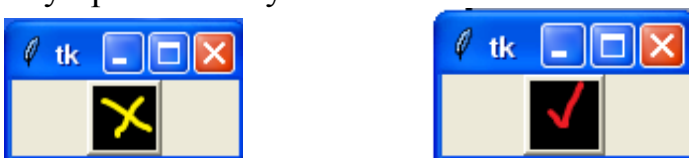


Рис. 17.20. Два вікна Toplevel і кореневе вікно

27. **Віджети Button.** Створити ГІ із кнопкою, яка випадковим чином змінює свою картинку при кожному натисканні.



28. **Виджет Кнопка (Button).** Створити ГІ, який виводить у вікні кнопку.

Контрольні запитання

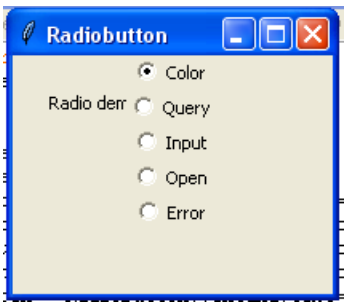
1. Для чого використовують ГІ ?
2. Назвіть основні елементи управління (віджети) бібліотеки *tkinter*.

Аудиторна робота

Приклад 17.1. Радіокнопка

```
from tkinter import *
ventana = Tk()
v = IntVar(); m = IntVar()
ventana.title("Radiobutton")
ventana.geometry("200x150")
etiquental = Label(ventana,text="Radio demos").place(x=20,y=20)
Color=Radiobutton(ventana,text="Color",variable=v,value=0).pack()
Query=Radiobutton(ventana,text="Query",variable=v,value=1).pack()
Input=Radiobutton(ventana,text="Input",variable=v,value=2).pack()
Open=Radiobutton(ventana,text="Open",variable=v,value=3).pack()
Error=Radiobutton(ventana,text="Error",variable=v,value=4).pack()
ventana.mainloop()
```

Результат



Приклад 17.2. Кнопка

1.Quitter:

```
from tkinter import *
from tkinter.messagebox import askokcancel

class Quitter(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.pack()
        widget = Button(self, text='Quit', command=self.quit)
        widget.pack(side=RIGHT, expand=YES, fill=BOTH)
    def quit(self):
        ans = askokcancel('Verify exit', "Really quit?")
        if ans: Frame.quit(self)

if __name__ == '__main__': Quitter().mainloop()
```

2.

```
from tkinter import *
from quitter import Quitter

def fetch():
    print ('Input => "%s"' % ent.get())

root = Tk()
ent = Entry(root)
ent.insert(0, 'Have a cigar')
ent.pack(side=TOP, fill=X)

ent.focus()
ent.bind('<Return>', (lambda event: fetch()))
btn = Button(root, text='Fetch', command=fetch)
btn.pack(side=LEFT)
Quitter(root).pack(side=RIGHT)
root.mainloop()
```

Результат

Input => "Have a cigar"

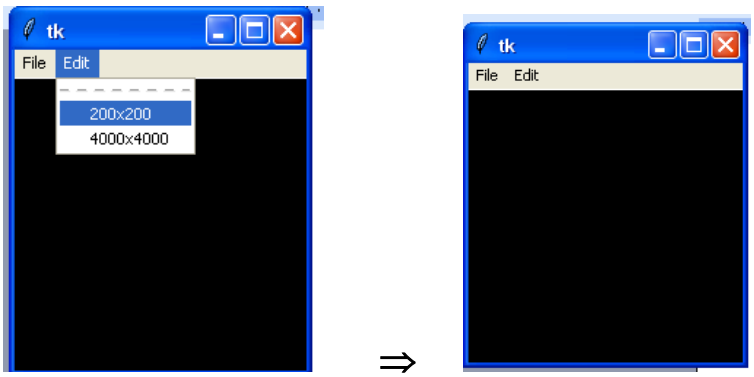
Приклад 17.3. Меню

1.

```
from tkinter import *
root = Tk()

def sq1():
```

```
        fra.config(width=200,height=200)
        print('200x200')
    def sq2():
        fra.config(width=400,height=400)
        print('4000x4000')

    fra = Frame(root,width=300,height=100,bg="Black")
    fra.pack()
    m = Menu(root)
    root.config(menu=m)
    cm = Menu(m)
    m.add_cascade(label="File",menu=cm)
    sm = Menu(m)
    m.add_cascade(label="Edit",menu=sm)
    sm.add_command(label="200x200",command=sq1)
    sm.add_command(label="4000x4000",command=sq2)
    root.mainloop()
```

Результат

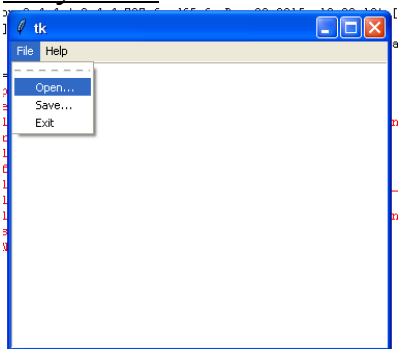


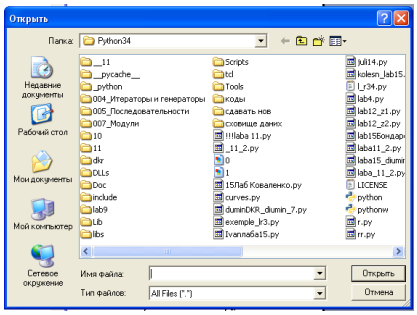
2.

```
    from tkinter import *
    from tkinter.filedialog import *
    import fileinput
    from tkinter.messagebox import *

    def close_win():
         if askyesno("Exit", "Do you want to quit?"):
              root.destroy()
    def about():
         showinfo("Editor",    "This    is    text    editor.\n(test
version)")
    def _open():
         op = askopenfilename()
         for l in fileinput.input(op):
              txt.insert(END,l)
    def _save():
         sa = asksaveasfilename()
         letter = txt.get(1.0,END)
         f = open(sa,"w")
         f.write(letter)
         f.close()
    root = Tk()
    m = Menu(root)
```

```
root.config(menu=m)
fm = Menu(m)
m.add_cascade(label="File",menu=fm)
fm.add_command(label="Open...",command=_open)
fm.add_command(label="Save...",command=_save)
fm.add_command(label="Exit",command=close_win)

hm = Menu(m)
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="About",command=about)

txt = Text(root,width=40,height=15,font="12")
txt.pack()

root.mainloop()
```

Результат

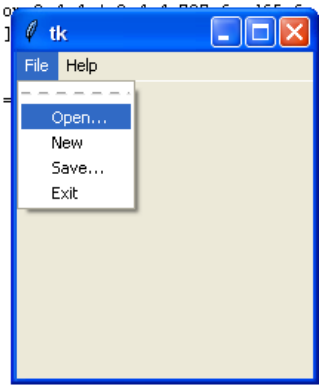


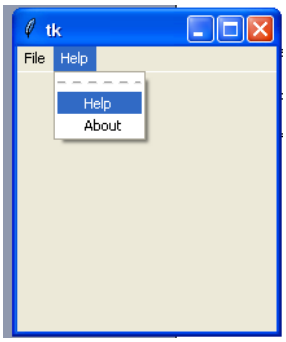


3.

```
from tkinter import * #импорт библиотеки
root = Tk() #создание диалогового окна
m = Menu(root) #создается объект Меню на главном окне
root.config(menu=m) #окно конфигурируется с указанием меню для него

fm = Menu(m) #создается пункт меню с размещением на основном меню (m)
m.add_cascade(label="File",menu=fm) #пункту располагается на основном меню (m)
fm.add_command(label="Open...") #формируется список команд пункта меню
fm.add_command(label="New")
fm.add_command(label="Save...")
fm.add_command(label="Exit")

hm = Menu(m) #второй пункт меню
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="Help")
hm.add_command(label="About")
root.main
```

Результат





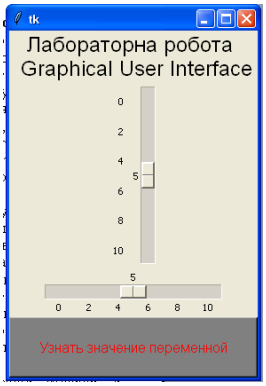
Приклад 17.4. Смуга прокрутки

1.

```
from tkinter import *
def printer(event):
    x = var.get()
    print(x)
root = Tk()
var = IntVar()
var.set(5)
sca1 = Scale(root,orient=VERTICAL,length=200,
             from_=0,to=10,tickinterval=2,resolution=1,
             variable=var)
sca2 = Scale(root,orient=HORIZONTAL,length=200,
             from_=0,to=10,tickinterval=2,resolution=1,
             variable=var)
lab=Label(root, text='Лабораторна робота \n Graphical User
Interface',
              font='Arial 18')
but=Button(root,            text='Узнать            значение
переменной',width=30,height=3,
              bg='grey',fg='red',font='Arial 12')
but.bind('<Button-1>',printer)
lab.pack()
sca1.pack()
sca2.pack()
but.pack()
root.mainloop()
```

Результат

5



2.

```
from tkinter import *
root = Tk()
myscrollbar = Scrollbar(root)
myscrollbar.pack(side=RIGHT, fill=Y)

mylistbox = Listbox(root, yscrollcommand = myscrollbar.set)
for a in range(20):
     mylistbox.insert(END,'Lumberjack-' + str(a))
mylistbox.pack(side=LEFT, fill=BOTH)
myscrollbar.config(command=mylistbox.yview)
root.mainloop()
```

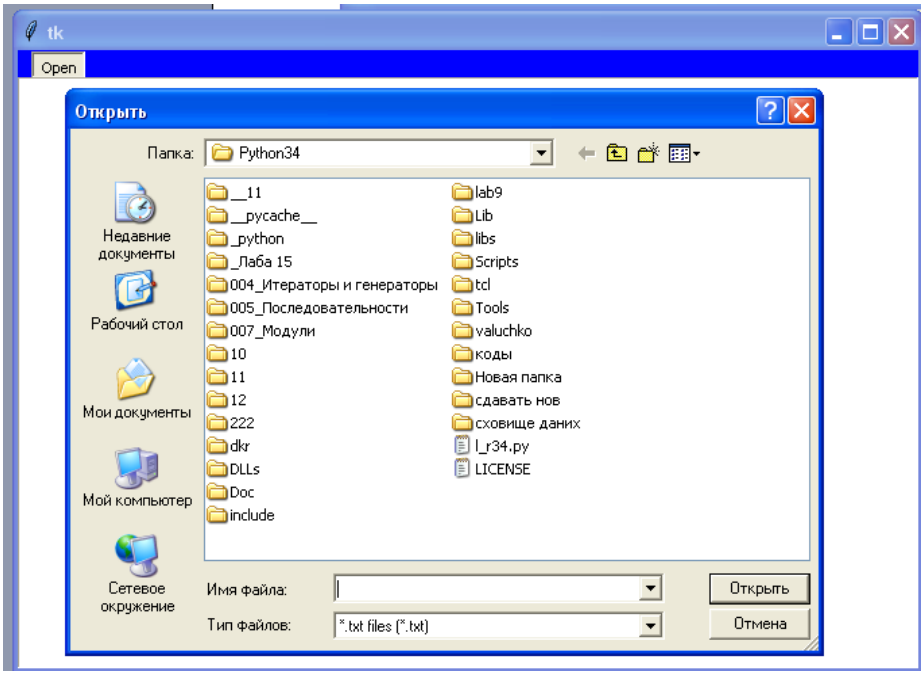
Результат

Приклад 17.5. Завантаження текстового файлу у редактор

```
from tkinter import *
import tkinter.filedialog

def LoadFile(ev):
    fn=tkinter.filedialog.Open(root,        filetypes=[('*.txt
files', '.txt')]).show()
    if fn == '':
        return
    textbox.delete('1.0', 'end')
    textbox.insert('1.0', open(fn, 'rt').read())

root = Tk()

panelFrame = Frame(root, height = 20, bg = 'blue')
textFrame = Frame(root, height = 40, width = 50)

panelFrame.pack(side = 'top', fill = 'x')
textFrame.pack(side = 'bottom', fill = 'both', expand = 1)

textbox = Text(textFrame, font='Arial 12', wrap='word')
scrollbar = Scrollbar(textFrame)
scrollbar['command'] = textbox.yview
textbox['yscrollcommand'] = scrollbar.set

textbox.pack(side = 'left', fill = 'both', expand = 1)
scrollbar.pack(side = 'right', fill = 'y')

loadBtn = Button(panelFrame, text = 'Open')
loadBtn.bind("<Button-1>", LoadFile)
```

```
loadBtn.place(x = 10, y = 1, width = 40, height = 20)
root.mainloop()
```

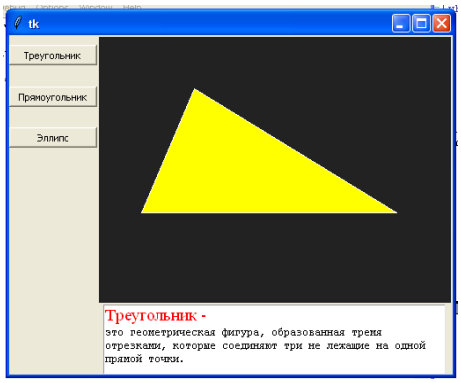
Результат



Приклад 17.6. *Геометричні фігури*. При натисканні на кнопку («Треугольник», «Прямоугольник» тощо) на полотні з'являється відповідна фігура, а в текстовому полі її визначення. При клацанні на фігурі, яка розташована на полотні, вона змінюється на іншу, що теж відноситься до тієї ж групи (наприклад, трикутники – прямокутні або рівносторонні). Внизу полотна з'являються пояснення, що характеризують чергову фігуру (наприклад, «прямокутний трикутник»). Можна отримати частини фігури (для трикутника це сторони), щоб продемонструвати, як вони утворюються

```
i = 0
def add_triangle(event):
    coords = [(50,130,290,40,170,250),(10,10,290,30,200,250),
              (30,280,330,60,300,200),(50,200,340,200,110,60)]
    colors = ['red','green','blue','yellow']
    global i
    canvas.itemconfig(t,fill=colors[i],outline='white')
    canvas.coords(t,coords[i])
    i += 1
    if i == 4:
```

```
            i = 0
def triangle():
      canvas.coords(r,(0,0,0,0))
      canvas.itemconfig(t,fill='yellow',outline='white')
      canvas.coords(t,(50,200,340,200,110,60))
      text.delete(1.0,END)
      text.insert(1.0,'Треугольник -\nэто геометрическая фигура, \
образованная тремя отрезками, которые соединяют три не лежащие \
на одной прямой точки.')
      text.tag_add('title','1.0','1.end')
      text.tag_config('title',font=('Times',14),foreground='red')

def rectangle():
      canvas.coords(t,(0,0,0,0,0,0))
      canvas.itemconfig(r,fill='lightblue',outline='white')
      canvas.coords(r,(80,50,360,200))

def ellipse():
      pass

from tkinter import *
win = Tk()

b_triangle = Button(text="Треугольник", width=15,command=triangle)
b_rectangle = Button(text="Прямоугольник",
width=15,command=rectangle)
b_ellipse = Button(text="Эллипс", width=15,command=ellipse)
canvas = Canvas(width=400,height=300,bg='#222222')
text = Text(width=55,height=5,bg='#ffffff',wrap=WORD)

t = canvas.create_polygon(0,0,0,0,0,0)
canvas.tag_bind(t,'<Button-1>',add_triangle)
r = canvas.create_rectangle(0,0,0,0)

b_triangle.grid(row=0,column=0)
b_rectangle.grid(row=1,column=0)
b_ellipse.grid(row=2,column=0)
canvas.grid(row=0,column=1,rowspan=10)
text.grid(row=11,column=1,rowspan=3)
win.mainloop()
```



Теоретична частина

Графічний інтерфейс користувача та його елементи. Модуль *tkinter*

Для більшості програмних систем графічний інтерфейс користувача (Graphical User Interface, GUI), наприклад, вікна, кнопки і меню, які використовуються при роботі з програмами, став обов'язковою частиною пакета. Програми, оснащуються компонентами GUI, що підвищують гнучкість і простоту їх використання. З точки зору реалізації, ГІ в Python спираються на використання розширень на мові С. Коли сценарій створює кнопку і меню, він в кінцевому рахунку звертається до бібліотеки мови С, а коли сценарій реагує на призначену для користувача подію, бібліотека С в кінцевому рахунку звертається назад до Python. Це може розглядатися як приклад взаємодії Python із зовнішніми бібліотеками.

Хоча в якості інструменту для створення GUI в Python найчастіше використовують бібліотеку *tkinter*, існують й інші способи програмування призначених для користувача інтерфейсів в Python. Серед інструментів для розробки ГІ, доступних програмістам на Python, можна виділити: 1) *tkinter* – бібліотеку, яка включає близько 25 основних типів віджетів, а також різні діалоги й інші інструменти; 2) *PyQt* – бібліотеку, яка містить сотні класів, тисячі функцій і методів. Бібліотека *tkinter* поширюється в складі Python як модуль стандартної бібліотеки і стала основою стандартного інтегрованого середовища розробки IDLE з ГІ. Це – єдиний набір інструментів для створення ГІ, який став частиною Python.

Графічний інтерфейс дозволяє здійснювати взаємодію людини з комп'ютером у формі діалогу з використанням вікон, меню і елементів управління (діалогових панелей, кнопок і тощо). Елементом інтерфейсу користувача є графічний примітив (віджет), що має стандартний зовнішній вигляд і виконує стандартні дії. Серед стандартного набору елементів ГІ користувача можна виділити такі елементи управління:

піктограма – схематичне зображення різних предметів і явищ; це – невелике зображення на екрані, яке використовують для ідентифікації деякого об'єкта (файлу, програми тощо); вибір і активація піктограми викликає дію, пов'язану з обраним об'єктом;

покажчик миші – елемент ГІ, призначений для візуалізації виконуваних дій, служить для виконання операцій з іншими об'єктами (натискання кнопок, перетягування вікон і зміни їх розмірів), змінює своє зображення в залежності від виконуваних дій;

іконка (icon) – піктограма, пов'язана з конкретним об'єктом, при клацанні на ній мишею ми отримуємо доступ до властивостей об'єкта або запуску його на виконання;

кнопка (button) – елемент управління, взаємодія користувача з яким обмежується однією дією – натисканням;

радіокнопка (radio button) – перемикач, який надає вибір лише одного зі списку декількох перерахованих опцій;

прапорець / перемикач (check box) – перемикач, який надає вибір декількох (навіть всіх) зі списку перерахованих опцій;

список (list box) – елемент управління, призначений для вибору деяких значень (можливо одного), із запропонованого списку;

лічильник (spinbutton) – елемент управління, призначений для зміни значення (зазвичай числового), яке вводиться в поле; він складається з поля зі значенням і двох кнопок – для збільшення або зменшення значення;

поле редагування (textbox, edit field) – елемент управління, призначений для введення і редагування даних;

повзунок – елемент управління, який дозволяє вибрати одне значення, що стоїть у впорядкованому ряді (від мінімального до максимального значення);

вікно – обмежена, обрамлена, прямокутна область екрану, в якій відображається додаток, що виконується (документ або повідомлення);

смуга прокрутки (scrollbar) – елемент управління, який відображає приховану частину вікна; смуги бувають вертикальними і горизонтальними.

ДОДАТКИ

Додаток А

Установлення та налаштування

Для запуску сценаріїв на мові Python необхідно мати інтерпретатор Python. Тому першим кроком до використання цієї мови є його установлення.

Python є програмним забезпеченням, поширюваним з відкритими вихідними текстами, його можна вільно завантажити з Мережі.

Найсвіжішу і найкращу версію Python можна отримати на офіційному веб-сайті проекту http://www.python.org. Скористайтеся посиланням Download (Завантажити) на цій сторінці і виберіть версію для своєї платформи. Тут ви знайдете файли дистрибутивів для Windows тощо.

Якщо вам цікаво альтернативні реалізації Python, пошукайте у Мережі Jython (версія Python для середовища Java) та IronPython (версія Python для середовища C # /. NET).

Установлення. Завантаживши дистрибутив Python, його необхідно встановити. Порядок установлення залежить від платформи.

Для Wimdows дистрибутив Python поставляється у вигляді інсталяційного файлу в форматі MSI – клацніть двічі на ярлику цього файлу і відповідайте на запитання натисканням кнопок Yes (Так) або Next (Далі), щоб виконати установлення з параметрами за замовчуванням.

Установка за замовчуванням включає в себе комплект документації, підтримку бібліотеки побудови графічних інтерфейсів *tkinter*, базу даних *shelve*, а також середовище розробки IDLE з графічним інтерфейсом. Зазвичай Python 3.0 встановлюють в каталог C: \ Python30, хоча можна вказати інший каталог.

Після установки в підміню «Усі програми (All Programs)», в меню кнопки Пуск (Start), з'являється додаткове меню Python, в якому є п'ять пунктів, що забезпечують швидкий доступ до найбільш типових завдань: запуск IDLE, читання документації, запуск інтерактивного сеансу, читання стандартних посібників з мови Python в веб-браузері і видалення.

Після установлення інтерпретатор Python автоматично реєструє себе в якості програми, призначеної для відкриття файлів Python клацанням миші.

Налаштування Python. Після установлення Python може знадобитися виконати деякі налаштування, які впливають на те, як Python буде виконувати ваш програмний код. Деякі особливості поведінки інтерпретатора можуть бути налаштовані за допомогою змінних оточення і параметрів командного рядка. Розглянемо змінні оточення Python. Параметри командного рядка, які вказуються при запуску програм на Python з командного рядка системи, використовують досить рідко і часто грають вузькоспеціалізовану роль.

Змінні оточення Python. Змінні оточення, іноді відомі як змінні командної оболонки або змінні DOS, визначають за межами інтерпретатора Python і тому

їх можна використати для налаштування його поведінки кожного разу, коли він запускається на комп'ютері. Інтерпретатор Python використовує різні змінні оточення, але лише деякі з них використовують досить часто. У табл. А.1. наведено основні змінні оточення, які використовують для налаштування інтерпретатора Python.

Таблиця А.1

Змінні оточення, які мають важливе значення

Змінна	Призначення
PATH (або *path*)	Шлях пошуку файлів, яки використовує система (використовується при пошуку файлу python)
PYTHONPATH	Шлях пошуку модулів Python (використовує операція імпортування)
PYTHONSTARTUP	Шлях до інтерактивного файлу запуску Python
TCL_LIBRARY, TK_LIBRARY	Змінні оточення для *tkinter* (розширення для створення графічного інтерфейсу)

PATH. Змінна *path* визначає список каталогів, де ОС буде намагатися відшукати виконувані файли програм. Зазвичай цей список повинен включати каталог, де знаходиться інтерпретатор Python (файл *python.exe* в Windows). Вам не треба налаштовувати цю змінну, якщо ви працюєте в каталозі, де розміщено інтерпретатор Python, або вводите в командному рядку повний шлях до виконуваного файлу інтерпретатора. У Windows, наприклад, налаштування змінної *path* не має значення, якщо перед запуском будь-якого програмного коду виконати команду *cd C:\Python3X* (перехід в каталог, де розташовано інтерпретатор Python), або замість команди *python* ви виконуєте команду *C:\Python3X \ python* (в команді присутній повний шлях до виконуваного файлу). Крім того, змінну оточення PATH в основному використовують для запуску команд з командного рядка – ця змінна не має значення при запуску програм клацанням миші на ярлику або з інтегрованого середовища розробки.

PYTHONPATH. Інтерпретатор Python використовує змінну PYTHONPATH під час пошуку файлів модулів, коли вони імпортуються програмами. Ця змінна містить список каталогів у форматі, що залежить від типу використовуваної платформи – в Windows каталоги в списку відокремлюють крапкою з комою. Зазвичай цей список повинен включати тільки каталоги з вашими вихідними текстами. Вміст цієї змінної оточення включають у список *sys.path* разом з каталогом, в якому знаходиться сценарій, з будь-якими каталогами, перерахованими в файлах *.pth*, і з каталогами стандартної бібліотеки. Не потрібно налаштовувати цю змінну, якщо не імпортувати модулі, які знаходяться в інших каталогах, тому що інтерпретатор завжди намагається відшукати модулі в домашньому каталозі програми. Налаштовувати цю змінну доведеться, тільки якщо будь-який модуль повинен імпортувати інший модуль, розташований в іншому каталозі. Дивіться також далі обговорення файлів *.pth*, які є альтернативою змінної PYTHONPATH.

PYTHONSTARTUP. Якщо у змінній PYTHONSTARTUP вказано повне ім'я файлу з програмним кодом на Python, інтерпретатор буде запускати цей файл автоматично кожного разу, коли запускають інтерактивний сеанс роботи з інтерпретатором, як якби інструкції з цього файлу вводилися вручну в інтерактивній командній оболонці. Цей спосіб використовують рідко, але його зручно застосовувати, коли необхідно забезпечити завантаження деяких утиліт для роботи в інтерактивній оболонці (він дозволяє заощадити час на імпортуванні вручну).

Спосіб установлення змінних оточення, які мають відношення до Python, і встановлюють значення залежать від типу комп'ютера, з яким ви працюєте. Вам не обов'язково виконувати всі ці налаштування, якщо ви працюєте в середовищі IDLE.

Нехай у вас є декілька корисних модулів в каталогах *utilities* і *package1* десь в комп'ютері, і необхідно імпортувати їх з файлів модулів, розташованих не в домашньому каталозі: щоб завантажити файл з ім'ям *spam.py* з каталогу *utilities*, необхідно виконати інструкцію: *import spam* з іншого файлу, розташованого в іншому каталозі. Для цього слід одним з можливих способів налаштувати шлях пошуку модулів, щоб включити в нього каталог, що містить файл *spam.py*. Нижче наведено декілька порад, як це можна зробити.

Змінні DOS (Windows). Якщо ви використовуєте MS-DOS або стару версію Windows, вам може знадобитися додати визначення змінних оточення в свій файл C: \ autoexec.bat і перезавантажити комп'ютер, щоб зміни вступили в силу. Команда налаштування для таких комп'ютерів має синтаксис, унікальний для DOS:

```
set PYTHONPATH=c:\pycode\utilities;d:\pycode\package1
```

Можна ввести цю команду у вікні сеансу DOS, але тоді налаштування будуть мати ефект тільки у цьому вікні. Налаштування в файлі *.bat* зберігаються постійно і є глобальними для всіх програм.

Змінні оточення в Windows. У найбільш свіжих версіях Windows, включаючи XP і Vista, є можливість встановлювати значення змінної оточення PYTHONPATH та інших змінних за допомогою графічного інтерфейсу і тим самим уникнути редагування файлів і перезавантаження комп'ютера.

У Windows XP виберіть ярлик Система (System) в меню Панель управління (Control Panel), перейдіть на вкладку Додатково (Advanced) і клацніть на кнопці Змінні середовища (Environment Variables), щоб відредагувати або додати нові змінні (PYTHONPATH – це змінна, призначена для користувача). Використайте ті ж імена змінних, значення і синтаксис, показані вище в команді налаштування для DOS. Процедуру налаштування виконують схожим чином і в Windows Vista, але при цьому вам доведеться перевірити виконання операцій. Не потрібно перезавантажувати комп'ютер, але необхідно перезапустити інтерпретатор Python, якщо до моменту внесення змін він уже був запущений – він сприймає налаштування шляху тільки під час запуску. Якщо ви працюєте у вікні програми командного рядка (Command Prompt), вам доведеться перезапустити її, щоб налаштування вступили в силу.

Реєстр Windows. Якщо ви – користувач Windows, ви можете також налаштувати шлях за допомогою редактора реєстру Windows. Виберіть пункт меню Пуск (Start) → Виконати ... (Run ...) і введіть команду *regedit*. Якщо цей інструмент редагування встановлений у вас на комп'ютері, ви зможете з його допомогою відшукати записи, які стосуються Python, і виконати необхідні зміни.

Файли шляхів. Якщо для налаштування шляху пошуку модулів ви вирішили використовувати файл *.pth*, в Windows можна створити текстовий файл з таким вмістом (файл C: \ Python3X \ *mypath.pth*):

```
c:\pycode\utilities
d:\pycode\package1
```

Інтерпретатор відшукує ці файли під час запуску. Імена каталогів у файлах шляху можуть бути абсолютними або відносними по відношенню до каталогу, де знаходиться файл шляху. Допускають використовувати кілька файлів .pth (всі каталоги, перераховані в них, будуть додані в шлях пошуку), а самі файли .pth можуть розміщуватися в будь-яких каталогах, які перевіряють автоматично, в залежності від використовуваної платформи і версії Python. Наприклад, Python N.M намагається відшукати такі файли в каталогах C: \ PythonNM і C: \ PythonNM \ Lib \ site-packages в Windows.

Параметри командного рядка інтерпретатора. Під час запуску Python з системного командного рядка можна передавати різні параметри, які керують роботою інтерпретатора. На відміну від змінних оточення, параметри командного рядка можуть змінюватися при кожному новому запуску сценарію. Повна форма команди запуску інтерпретатора версії 3.0 виглядає, як показано нижче:

```
python [-bBdEhiOsSuvVWx?] [-c command | -m module-name | script | 
-] [args]
```

У більшості випадків для запуску файлу програми з аргументами в команді вказують тільки параметри *script* і *args*. Для ілюстрації розглянемо файл сценарію *main.py*, який виводить список аргументів командного рядка, доступних сценарію у вигляді списку *sys.argv*:

```
# Файл main.py
import sys; print(sys.argv)
```

У наступній команді обидві її частини, *python* і *main.py*, можуть містити повний шлях до файлу, а три аргументи (*a b –c*) призначені для сценарію – вони доступні у вигляді списку sys.argv. Перший елемент списку *sys.argv* завжди містить ім'я файлу сценарію, якщо воно відоме:

```
c:\Python30> python main.py a b –c
# Типовий спосіб: запуск файлу сценарія
['main.py', 'a', 'b', '-c']
```

Інші параметри командного рядка дозволяють вказувати програмний код, що виконується, безпосередньо в командному рядку (*–c*), приймати його з

потоку стандартного введення (дефіс означає, що код повинен читатися з каналу або з вхідного файлу):

```
c:\Python30> python -c "print(2 ** 100)"
# код, який виконують,  в командному рядку
1267650600228229401496703205376
c:\Python30> python -c "import main"
# Імпортувати файл, щоб виконати його
 ['-c']
c:\Python30> python - < main.py a b –c # код приймають зі
['-', 'a', 'b', '-c'] # стандартного входу
c:\Python30> python - a b -c < main.py # має той же ефект,
['-', 'a', 'b', '-c'] # що і попередня команда
```

При отриманні параметра *–m* інтерпретатор намагається відшукати зазначений модуль у шляху пошуку модулів (*sys.path*) і виконати його як звичайний сценарій (як модуль *__main__*). Розширення *.py* в цьому випадку слід відкинути, оскільки тут мають на увазі ім'я модуля, а не файлу:

```
c:\Python30> python -m main a b –c
# Відшукати / запустити модуль як сценарій
['c:\\Python30\\main.py', 'a', 'b', '-c']
```

Параметр *–m* підтримує запуск модулів в пакетах з використанням синтаксису відносних шляхів, а також модулів в архівах *.zip*. Цей ключ зазвичай використовують для запуску модулів налагоджувача *pdb* і профілювальника *profile* при налагодженні сценаріїв з командного рядка, а не в інтерактивній оболонці. Однак поведінка цих модулів в такому режимі змінилася у версії 3.0 (з модуля *profile* вилучили функцію *execfile* в Python 3.0, а *pdb* виконує трасування операцій введення / виведення в модулі *io*):

```
c:\Python30> python -m pdb main.py a b -c # Налагодити сценарій
--Return--
> c:\python30\lib\io.py(762)closed()->False
-> return self.raw.closed
(Pdb) c
c:\Python30> C:\python26\python -m pdb main.py a b -c
> c:\python30\main.py(1)<module>()
-> import sys
(Pdb) c
c:\Python30> python -m profile main.py a b -c # Профілювати
сценарій
c:\Python30> python -m cProfile main.py a b -c
# Профілювальник зі зниженим споживанням ресурсів
```

Слідом за командою *python* і перед посиланням на програмний код, який потрібно запустити, є можливість вказати додаткові аргументи, що керують поведінкою самого інтерпретатора. Ці аргументи використовує сам інтерпретатор, і вони не призначені для управління поведінкою сценарію. Наприклад, параметр –O запускає інтерпретатор в оптимізованому режимі, *–u* відключає механізм буферизації стандартних потоків введення / виведення, *–i* переводить інтерпретатор в інтерактивний режим після виконання сценарію:

```
c:\Python30> python –u main.py a b -c
# Вимикає буферизацію потоків виведення
```

За додатковою інформацією про кожний із доступних параметрів командного рядка звертайтеся до інструкцій чи довідників Python, або запитайте у самого інтерпретатора, запустивши команду:

c: \ Python30> python – ?,

щоб вивести довідкову інформацію, в якій описуються всі підтримувані параметри командного рядка. Якщо вам доводиться мати справу зі складними командами запуску сценаріїв, познайомтеся з модулями *getopt* і *optparse* зі стандартної бібліотеки, які реалізують обробку більш складних параметрів командного рядка.

Комплект стандартних настанов на сьогоднішній день включає цінні рекомендації по використанню мови Python на різних платформах. Він доступний в Windows в меню кнопки Пуск(Start), після установлення Python (пункт Python Manuals – Керівництва Python) і в електронному вигляді на сайті http://www.python.org. Загляньте до керівництва «Using Python», де ви знайдете різні підказки і рекомендації, а також опис налаштувань і особливостей командного рядка для різних платформ.

Додаток Б

Модуль math

import math

Модуль math містить набір функцій для роботи з числами. Серед них:
math.ceil(X) – округлення до найближчого більшого числа,
math.copysign(X, Y) повертає число, яке має модуль такий же, як і у числа X, а знак - як у числа Y.
math.fabs(X) – модуль X, math.factorial(X) – факторіал числа X,
math.floor(X) – округлення вниз, math.fmod(X, Y) – залишок від ділення X на Y,
math.frexp(X) повертає мантису і експоненту числа,
math.ldexp(X, I) – X * 2i – функція, зворотна функції math.frexp().
math.fsum (послідовність) – сума всіх членів послідовності (еквівалент вбудованої функції sum (), але math.fsum () більш точна для чисел з плаваючою точкою),
math.isfinite(X)/ math.isinf(X) перевіряє чи є X числом/ нескінченністю,
math.isnan(X) перевіряє чи є X NaN (Not a Number – не є числом),
math.modf(X) – повертає дробову і цілу частину числа X (обидва числа мають той же знак, що і X),
math.trunc(X) усікає значення X до цілого,
math.exp(X) – eX,
math.expm1(X) – eX – 1 (при X → 0 точніш за math.exp(X) –1),
math.log(X, [base]) – логарифм X з основою base. Якщо base не вказано, обчислюють натуральний логарифм,
math.log1p(X) – натуральний логарифм (1+X) (при X→0 точніше за math.log(1+X)),
math.log10(X)/ math.log2(X) – логарифм X з основою 10/2,
math.pow(X, Y) – XY,
math.sqrt(X) – квадратний корінь з X,
math.acos(X)/ math.asin(X)/ math.atan(X) – арккосинус X/ арксинус X/ арктангенс X (X вказують у радіанах),
math.atan2(Y, X) – арктангенс Y/X (в радіанах, з урахуванням чверті, в якій розташована точка (X, Y)),
math.cos(X)/ math.sin(X) – косинус X/ синус X (X вказують у радіанах),
math.tan(X) – тангенс X (X вказують у радуанах),
math.hypot(X, Y) обчислює гіпотенузу трикутника з катетами X і Y (math.sqrt(x * x + y * y)),
math.degrees(X) конвертує радіани у градуси,
math.radians(X) конвертує градуси у радіани,
math.cosh(x)/math.sinh(x)/math.tanh(x) – гіперболічний косинус/синус/тангенс,
math.erf(X) – функція похибок.
math.erfc(X) – додаткова функція похибок (1 – math.erf(X)).
math.pi – pi = 3,1415926..., math.e – e = 2,718281...

Додаток В

Базові поняття та визначення

Алгоритм – це точне розпорядження виконавцю зробити певну послідовність дій для досягнення поставленої мети за скінчену кількість кроків.

Дані – відомості: 1) отримані шляхом вимірювання, спостереження, логічних або арифметичних операцій; і 2) подані у формі, придатній для постійного зберігання, передачі і 3) (автоматизованої) обробки.

Тип данних – характеристика набору даних, яка визначає:

- діапазон можливих значень даних з набору;
- допустимі операції, які можна виконувати над цими значеннями;
- спосіб зберігання цих значень в пам'яті.

Розрізняють: 1) прості типи даних: цілі, дійсні числа, символи, рядки, логічні величини; 2) складові типи даних: масиви, файли та ін.

Програма (згідно ГОСТ 1978190) – це дані, призначені для управління конкретними компонентами системи обробки інформації з метою реалізації певного алгоритму.

Алгоритмічна мова (мова програмування) – штучна (формальна) мова, призначена для запису алгоритмів. Мова програмування задається своїм описом, її реалізують у вигляді спеціальної програми: компілятора або інтерпретатора.

Транслятор мови програмування:

в широкому сенсі – програма, яка перетворює текст, написаний на одній мові, в текст на іншій мові.

у вузькому сенсі – програма, яка перетворює програму, написану на одній (вхідній) мові у програму, подану на іншій (вихідній) мові.

Це – програма, яка перетворює вихідний текст програми на мові програмування у машинну мову обчислювальної системи, на якій ця програма повинна виконуватися.

Інтерпретатор – транслятор, здатний паралельно переводити і виконувати програму, написану на алгоритмічній мові високого рівня.

Компілятор – програма, яка перетворює текст, написаний на алгоритмічній мові, в програму, що складається з машинних команд. Компілятор створює закінчений варіант програми на машинній мові.

Константа (в програмуванні) – елемент даних, який займає місце в пам'яті, має ім'я і певний тип, причому його значення ніколи не змінюється.

Змінна (в мовах програмування) – іменована частина пам'яті, в яку можуть поміщатися різні значення змінної. Причому в кожен момент часу змінна має одне значення (або один набір значень). У процесі виконання програми значення змінної може змінюватися. Тип змінних визначається типом даних, які вони подають.

Підпрограма – самостійна частина програми, яку розробляють незалежно від інших частин і потім викликають по імені.

Функція. Підпрограма, яка на основі деяких даних (аргументів функції) обчислює значення деякої змінної («функція повертає значення»).

Об'єкт (поняття об'єктно-орієнтованого програмування) – програмний модуль, який об'єднує в єдине ціле дані і програми, що маніпулюють даними. Об'єкт характеризується властивостями, які є параметрами об'єкта і методами, які дозволяють впливати на об'єкт і його властивості.

Метод – дія у вигляді процедури, яку виконує об'єкт (іноді кажуть - виконується над об'єктом).

Ідентифікатор – символічне ім'я об'єкта, змінної або підпрограми, які однозначно ідентифікують їх в програмі.

Вираз – конструкція на мові програмування, призначена для виконання обчислень. Вираз складається з операндів, об'єднаних знаками операцій. Розрізняють арифметичні, логічні і символьне вирази.

Операнд – константа, змінна, функція, вираз або інший об'єкт мови програмування, над яким виконують операції.

Арифметична операція – обчислювальна операція над числами. У багатьох мовах програмування визначені двомісні арифметичні операції: додавання, віднімання, множення, ділення, ділення без остачі, обчислення залишку від ділення.

Логічна операція – операція над логічними («булевими») операндами, результатом цієї операції є значення «Істина» або «Хибність». Найбільш поширеними є такі операції: багатомісне логічне додавання; багатомісне логічне множення; одномісне логічне заперечення. «Багатомісна» операція означає, що в ній може бути два і більше операндів, а в «одномісній» (або «унарній») операції бере участь лише один операнд.

Операція відношення виконує порівняння двох величин. Результатом цієї операції є «булева» змінна, яка має значення «Істина» (True або логічна 1) або «Хибність» (False або логічний 0).

Масив (масив даних). Сукупність, як правило, однотипних даних, кожне з яких ідентифікується з ім'ям масиву та індексом (індексами). Залежно від кількості індексів масиви бувають одномірні (лінійні), двомірні і т.д.

Індекс. Номер (або номери, якщо масив даних багатовимірний), які додають до імені масиву, щоб ідентифікувати кожен елемент даного масиву. Наприклад, *a*[1, 3] означає, що визначено елемент двовимірного масиву *a* з індексом (1,3): рядок – 1, стовпець – 3.

Присвоєння. Операція запису значення в змінну. У кожній мові програмування визначений оператор присвоювання. Якщо в змінну записують нове значення, старе стирається.

Цикл (циклічні обчислення) означають багаторазове виконання одних і тих же операцій. Залежно від завдання розрізняють цикли зі змінною (з лічильником, з відомою кількістю повторень) і цикли з умовою (цикл повторюють, поки не буде виконано умову завершення циклу).

Зациклення. Для циклів з умовою – ситуація, при якій умова завершення циклу ніколи не виконується.

ЛІТЕРАТУРА

1. Абрамов С. А. Начала информатики. М.: «Наука», 1990. – 256 с.

2. Абрамов С. А. и др. Задачи по программированию. – М.: «Наука», 1988. – 224 с.

3. Аленский Н. А. и др. Задачи и методические рекомендации по программированию. – Мн.: БГУ, 1990. – 67 с.

4. Вирт Н. Алгоритмы и структуры данных. – М., 1989, – 360 с.

5. Златопольский Д. М. Сборник задач по программированию. – 3–е изд., перераб. и доп. – СПб.: БХВ–Петербург, 2011. – 304 с.: ил. – (ИиИКТ)

6. Иванова Г.С. Основы программирования: Учебник для вузов. – 4–е изд., стер. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2007.-416 с.: ил. (Сер. «Информатика в техническом университете».)

7. Лутц М. Изучаем Python. 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с. , ил.

8. Лутц М. Программирование на Python, том 1, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с. , ил. [гл. 7–10]

9. Руководство пользователя OpenOffice.org 2. – Издательство: БХВ–Петербург, 2007 . – 320 с.

10. Сборник лабораторных работ с примерами решения задач по алгоритмизации и программированию на языке Си: Учебно-методическое пособие для студентов высших технических учебных заведений / Кравчук А.И., Кравчук А.С. – Мн.: Технопринт, 2002, 111 с.

11. Шапошникова С. Основы програмимиирования на Python. – Лаборатория юного линуксоида, 2011г. – 44 с.

http://younglinux.info