МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Чичкарьов Є.А., Зінченко О.В., Єльченко С.В.

Прикладне програмування на Python

Частина 1. Основи програмування на Python

Навчальний посібник

Київ – 2022

УДК 004.658

Рекомендовано на засіданні вченої ради Навчально-наукового інституту інформаційних технологій (Протокол № 2 від 12.09.2022 року)

Чичкарьов Є.А., Зінченко О.В., Єльченко С.В.

Прикладне програмування на Python. Частина 1. Основи програмування на Python.- Навчальний посібник.- Київ: ДУТ, 2022. - 160 с.

Рецензенти: **Іщеряков С.М,** кандидат технічних наук, доцент, доцент кафедри Комп'ютерних наук Державного університету телекомунікацій.
Хлапонін Ю.І., доктор технічних наук, доцент, завідувач кафедри кібербезпеки та комп'ютерної інженерії Київського національного університету будівництва та архітектури.

Навчальний посібник призначений для студентів спеціальності 122 Комп'ютерні науки всіх форм навчання з різними аспектами створення додатків різного призначення мовою Python та набуття ними практичного досвіду розробки і налагодження проектів на Python при виконанні практичних занять з дисципліни «Прикладне програмування -Python».

ЗМІСТ

ВСТУП

Python – одна найбільш затребуваною мовою програмування в останніх роках. Причина високої популярності лежить у відносній простоті python у вивченні, великому розмаїтті бібліотек та активному розвитку технологій штучного інтелекту, де використовується мова. А також у використанні Python при розробці різних продуктів машинного навчання, в аналізі даних та науково-дослідній діяльності

Цей навчальний посібник призначений для ознайомлення студентів спеціальності 122 Комп'ютерні науки всіх форм навчання з різними аспектами створення додатків різного призначення мовою python та набуття ними практичного досвіду розробки і налагодження проектів на python при виконанні практичних завдань.

Ця книга – навчальний посібник з вивчення мови програмування python. В неї надані основні концепції програмування на python, представлені основні типи даних, розглянути оператори управління виконанням програми, поняття і використання функцій і об'єктів, пакети розширення функціональності мови python для обчислювальних задач – NumPy і SciPy. Книга розрахована на початківців, які або трохи знають програмування, але не знають мови та екосистеми python.

Посібник, крім теоретичних матеріалів і прикладів, містить 8 практичних робіт з декількома варіантами завдань і контрольними питаннями.

1 Типи даних Python і робота з ними

1.1 Стиль створення змінних

Для імен змінних використовується зміїний_реєстр (англ. snake_case): наприклад, my_variable або i.

Snake case (або snake_case - зміїний_реєстр) - стиль написання складових слів, при якому кілька слів поділяються символом підкреслення (_), і не мають пробілів у запису, причому кожне слово зазвичай пишеться з маленької літери - "foo_bar", "hello_world" і т. д. Такий стиль написання використовується для іменування змінних та функцій у вихідному коді і іноді для іменування файлів на комп'ютері. Стиль mixedCase допускається в тих місцях, де вже переважає такий стиль, для збереження зворотної сумісності.

1.2 Класифікація типів даних

У Python вбудовані типи даних поділяються на 2 групи:
Скалярні (неподільні):

1. Числові типи (цілі, речові, комплексні числа);
2. Логічний тип;
3. NoneType – нетипізовані значення.

Структуровані (складові) колекції:

1. Послідовності: рядок, список, кортеж, числовий діапазон.
2. Множина.
3. Відображення: словник.

Крім того, всі об'єкти в Python відносяться до однієї з 2-х категорій:

1. Мутуючі (англ. Mutable): вміст об'єкта можна змінити після створення (наприклад, список);
2. Немутуючі (англ. Immutable): вміст об'єкта не можна змінити після створення (наприклад, рядок чи число).

Також часто використовується термінологія «змінювані» та «незмінювані» типи відповідно.

Обидва види об'єктів (об'єкти, які мутують, та об'єкти, що не мутують) мають свої переваги і недоліки. Основною перевагою типів, які не мутують, є гарантія незмінності з моменту створення: кожна ділянка коду, що використовується, має справу з копією об'єкта і не може його якимось чином змінити. Цей же принцип формує основний недолік типів, що не мутують: більша кількість споживаної пам'яті на додаткове копіювання об'єктів при необхідності внесення змін.

1.3 Створення змінних в Python

Змінна — це назва області пам'яті. Після того як ви дали ім'я цій області, з'являється можливість звертатися до даних, що зберігаються в ній. У мові Python ім'я змінної має складатися лише з цифр, літер та знаків підкреслення. І не повинно починатись із цифри.

Кожен елемент даних у Python є об'єктом певного типу чи класу. Коли, у процесі виконання програмного коду, з'являється нове значення,

інтерпретатор виділяє йому область пам'яті — тобто створює об'єкт певного типу (число, рядок або щось інше). Після цього Python записує до свого внутрішнього списку адресу цього об'єкта.

Для отримання доступу до створеного об'єкта використовуються змінні – вони дають можливість зручно працювати з об'єктами, використовуючи імена замість адрес.

Для контролю значень змінних найпростіше роздрукувати їх за допомогою функції print. Деталі використання print, форматування виведення, інші суміжні питання наведено нижче.

Функція print() друкує вказане повідомлення на екрані або іншому стандартному пристрої виводу. Повідомлення може бути рядком або будь-яким іншим об’єктом, об’єкт буде перетворено на рядок перед записом на екран.

Найпростіший приклад:

```
Роздрукувати повідомлення на екрані
print("Hello World")
```

Синтаксіс використання print()

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

Коротка інформація про пареметри

Параметр	Опис
object(s)	Будь-який об’єкт, або декілька об’єктів. Об’єкт буде перетворено на рядок перед друком.
sep='separator'	Не обов`язковий. Вказує, як розділити об’єкти, якщо їх більше одного. За замовчуванням ' '.
end='end'	Не обов`язковий. Вказує, що друкувати наприкінці. Типовим є '\n' (переведення рядка)
file	Не обов`язковий. Об’єкт із методом запису. Типовим є sys.stdout.
flush	Не обов`язковий. Логічне значення, яке вказує, чи скидається вихід (True) чи буферизується (False). Типовим значенням є False.

Таким чином, змінна в Python - це ім'я, за яким звертаються до певного об'єкту. Щоб створити нову змінну, її не потрібно заздалегідь ініціалізувати - достатньо вигадати їй ім'я і присвоїти значення через оператор =. Наприклад:

```
a = 123
print(a)
> 123
print(type(a))
<class 'int'>
print(id(a))
1827204944
```

У прикладі вище ми створили змінну, надали їй значення 123, далі

вивели її значення, тип і цифровий ідентифікатор об'єкта в пам'яті.

Треба мати на увазі, що функція id() повертає унікальний ідентифікатор зазначеного об’єкта. Усі об’єкти в Python мають власний унікальний ідентифікатор, який є постійним для цього об'єкта протягом його життя. Ідентифікатор присвоюється об’єкту під час його створення. Два об’єкти з неперекриваючим часом життя можуть мати однакове значення id(). У реалізації CPython це адреса об’єкта в пам’яті.

Важливо, що змінна в Python не зберігає значення безпосередньо - вона зберігає лише посилання на об'єкт.

Створимо найпростішу змінну: a = 100.

При цьому створюється об'єкт типу int зі значенням 100; створюється змінна a; у змінній a буде збережено адреса (посилання) на об'єкт.

Цифровий ідентифікатор – унікальна характеристика об’єкта.

Створимо копію змінної, яку було створено вище:

b = a,

і передивимось id об’єктів a та b.

```
print(id(a))
> 1827204576
print(id(b))
> 1827204576
```

Значення id однакові, тому у цьому прикладі Python не створює новий об'єкт — він просто створює змінну, яка посилається на той самий об'єкт, що й змінна a.

Механізм виконаних дій:

a = 100, створюється новий об'єкт, а змінна "a" одержує його адресу.

b = a, змінна "b" посилається на той самий об'єкт, що і змінна "a".

Припустимо, що в якийсь момент значення змінної b було змінено:

b = 500

Розглянемо id змінних a та b:

```
print(id(a))
> 1827204576
print(id(b))
> 56754272
```

У цьому прикладі Python створив новий об'єкт типу int і тепер змінна b посилається на новий об'єкт. Виконуємо присвоєння b = 500, змінна "b" посилається новий об'єкт з іншим адресою.

Розглянемо ще один приклад:

```
b = "tree"
print(id(b))
> 54134048
```

У цьому прикладі створюється новий об'єкт типу str, і змінна b посилається новий об'єкт.

Після створення нового значення змінної “b” на об'єкт типу int зі значенням 500 ніхто більше нема посилань. Отже, він більше не доступний і буде видалений збирачем сміття (тим самим звільнивши трохи пам'яті).

1.4 Числа і числові змінні в Python

У Python існує три типи чисел:

- цілі (int);
- речові (float) [з десятковою точкою];
- комплексні (complex) [складаються з дійсної та уявної частини].

Змінні числового типу створюються, коли ви присвоюєте їм значення:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Щоб перевірити тип будь-якого об’єкта в Python, використовується функцію type(), наприклад:

```
print(type(x))  # отримаємо <class 'int'>
print(type(y))  # отрамиємо <class 'float'>
print(type(z))  # отримаємо <class 'complex'>
```

Int, або integer, є цілим числом, позитивним чи негативним, без десяткових знаків, необмеженої довжини, наприклад:

```
x = 1
y = 35656222554887711
z = -3255522
print(type(x)) # отримаємо <class 'int'>
print(type(y)) # отримаємо <class 'int'>
print(type(z)) # отримаємо <class 'int'>
```

Число з плаваючою комою — це число, позитивне або негативне, що містить один або більше десяткових знаків, наприклад:

```
x = 1.10
y = 1.0
z = -35.59
print(type(x)) # отримаємо <class 'float'>
print(type(y)) # отримаємо <class 'float'>
print(type(z)) # отримаємо <class 'float'>
```

Також можна використовувати наукову нотацію чисел з плаваючою комою з літерою «e» для позначення ступеня числа 10, наприклад:

```
x = 35e3 # 35000
y = 12E4 # 120000
z = -87.7e-4 # -0.00877
print(type(x)) # отримаємо <class 'float'>
print(type(y)) # отримаємо <class 'float'>
print(type(z)) # отримаємо <class 'float'>
```

Комплексні числа записуються з використанням «j» як ознаки уявної

частини, наприклад:

```
x = 3+5j
y = 5j
z = -5j
print(type(x)) # отримаємо <class 'complex'>
print(type(y)) # отримаємо <class 'complex'>
print(type(z)) # отримаємо <class 'complex'>
```

Числа можна конвертувати з одного типу в інший за допомогою методів int(), float() і complex(), наприклад:

```
# створення змінних
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)  #1.0
print(b)  #2
print(c)  #(1+0j)

print(type(a))  # <class 'float'>
print(type(b))  # <class 'int'>
print(type(c))  # <class 'complex'>
```

Будь-яке ціле число складається з масиву цифр змінної довжини, тому в Python 3 змінну типу intможе бути записано число необмеженої довжини. Єдине обмеження довжини – це розмір оперативної пам'яті.

```
 1345233452342525235234787 77 ** 2
18096530413013891133013347014216107772438771969415729
```

Цілі числа можуть записуватися як як десяткові, а й як двійкові, вісімкові чи шістнадцяткові. Для цього перед числом потрібно написати символи:

- 0b (0B) – для двійкового подання;
- 0o (0O) – для вісімкового подання;
- 0x (0X) – для шістнадцяткового подання.

```
 print(20, type(20))  # десяткове подання
20 <class 'int'>
```

```
 print(0b10100, type(0b10100))  # двійкове подання
20 <class 'int'>

 print(0o24, type(0o24))  # вісімкове подання
20 <class 'int'>

 print(0x14, type(0x14))  # шістнадцяткове подання
20 <class 'int'>
```

Речові числа (float) також називають числами з плаваючою точкою. Це числа, що містять точку (десятковий роздільник) або експонент знак.

```
 1.5
1.5
 type(1.5)
<class 'float'>
 3.
3.0
 .5
0.5
 .4e7
4000000.0
 type(.4e7)
<class 'float'>
 4.1e-4
0.00041
```

Числа типу float - через уявлення чисел з плаваючою комою в комп'ютері неточні. Приклад:

```
 0.3 + 0.3 + 0.3 + 0.1
0.9999999999999999
```

Інформацію про точність і внутрішнє уявлення float для конкретної системи можна отримати з атрибуту sys.float_info. Він що містить інформацію про тип float. Він містить інформацію низького рівня про точність і внутрішнє представлення. Значення відповідають різним константам із плаваючою комою, визначеним у стандартному файлі заголовків float.h для мови програмування «C», наприклад:

```
 import sys
 sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15,
mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

Якщо потрібна висока точність, зазвичай використовують модулі Decimal і Fraction.

Комплексні числа являють собою пару значень типу int або float, і мають

вигляд <дійсна частина>+<уявна частина>j.

```
 1.1+2j
(1.1+2j)
 type(1.1+2j)
<class 'complex'>
```

Окремі частини комплексного числа доступні через атрибути real та imag

```
 num = 1.1+2j
 num.real, num.imag
(1.1, 2.0)
```

1.4.1 Операції з числами

Арифметичні операції

- x + y- Додавання;
- x - y- віднімання;
- x * y- множення;
- x / y- Поділ;
- x // y- Цілочисельне розподіл;
- x % y- залишок від ділення;
- x ** y- зведення в ступінь;
- -x- Зміна знака;
- abs(x)- модуль числа;
- divmod(x, y) - Повертає кортеж з результату та залишку від поділу x на y;
- pow(x, y[, z])- Зведення числа в ступінь (z - розподіл за модулем);
- round(x[, ndigits])- Округлення числа (ndigits - знаки після коми).

Порівняння чисел

- x == y- одно;
- x != y- не дорівнює;
- x > y- Більше;
- x < y- менше;
- x >= y- більше або дорівнює;
- x <= y- менше або дорівнює.

Перетворення

- int(x)- Перетворення в ціле число int;
- float(x)- Перетворення в число з плаваючою точкою float;
- complex(x)- Перетворення в комплексне число complex;
- bin(x)— ціле число у двійковий рядок;
- oct(x)- ціле число у вісімковий рядок;
- hex(x)- ціле число в шістнадцятковий рядок;
- [int(x) for x in str(123)]- Переведення цілого числа 123 до списку цифр цього числа;
- int(''.join(str(digit) for digit in [1,2,3]))- Переведення списку цифр [1,2,3] в ціле число 123;
- str(x)- Число в рядок;

1.4.2 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'])
'Holy  Grail'
 random.choice(['Life  of  Brian', 'Holy  Grail', 'Meaning  of Life'])
'Meaning  of Life'
 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.

1.5 Рядки та іх особливості

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. 1.1 наведені найбільш типові літерали рядків і операцій.

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.

Таблиця 1.1 - 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	Обхід в циклі Перевірка на входження

f-Strings: вдосконалений спосіб форматування рядків у Python

Так звані «форматовані рядкові літерали», f-рядки — це рядкові літерали, які мають f на початку та фігурні дужки, що містять вирази, які будуть замінені їхніми значеннями. Вирази обчислюються під час виконання, а потім форматуються за допомогою протоколу __format__.

Розглянемо приклад:

```
 name = "Eric"
 age = 74
 f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.'
```

Можна використовувати також велику літеру F на початку рядку.

Оскільки f-рядки обчислюються під час виконання, ви можете помістити в них будь-які дійсні вирази Python. Приклад:

```
 f"{2 * 37}"
'74'
```

1.6 Списки та їх використання

1.6.1 Властивості списків

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. Списки відносяться до категорії змінних об'єктів.

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 функції list):

```
 str([1, 2]) + "34" # Те ж, що й "[1, 2]" + "34"
'[1, 2]34'
 [1, 2] + list("34") # Те ж, що й [1, 2] + ["3", "4"]
[1, 2, '3', '4']
```

1.6.2 Методи списків.

Якщо задано список з деяким іменем, то метод обробки цього списку викликається з допомогою загальної форми

ListName.MethodName(parameters)

де

ListName – назва списку (об’єкту типу “список”);

MethodName – ім’я методу, який викликається;

parameters – перелік параметрів, розділених комою ‘ , ‘. Деякі методи не мають параметрів.

Розділювачем між іменем списку та іменем методу є символ ‘ . ‘ (крапка).

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
[[]]
```

1.7 Словники

Словники в Python – невпорядковані колекції довільних об'єктів з доступом по ключу. Їх іноді ще називають асоціативними масивами або хеш-таблицями.

Щоб працювати зі словником, його потрібно створити. Створити його можна кількома способами. По-перше, за допомогою літерала:

```
 d = {}
 d
{}
 d = {'dict': 1, 'dictionary': 2}
 d
{'dict': 1, 'dictionary': 2}
```

По-друге, за допомогою функції **dict**:

```
 d = dict(short='dict', long='dictionary')
 d
{'short': 'dict', 'long': 'dictionary'}
 d = dict([(1, 1), (2, 4)])
 d
{1: 1, 2: 4} 36
```

По-третє, за допомогою методу **fromkeys**:

```
 d = dict.fromkeys(['a', 'b'])
 d
{'a': None, 'b': None}
 d = dict.fromkeys(['a', 'b'], 100)
 d
{'a': 100, 'b': 100}
```

По-четверте, за допомогою **генераторів словників**, які дуже схожі на генератори списків.

```
 d = {a: a ** 2 for a in range(7)}
 d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Тепер спробуємо додати записи в словник і витягти значення ключів:

```
 d = {1: 2, 2: 4, 3: 9}
 d[1]
2
 d[4] = 4 ** 2
 d
{1: 2, 2: 4, 3: 9, 4: 16}
 d['1']
Traceback (most recent call last):
File "", line 1, in
d['1']
KeyError: '1'
```

Як видно з прикладу, привласнення по новому ключу розширює словник, привласнення за існуючим ключем перезаписує його, а спроба вилучення неіснуючого ключа породжує виключення. Для уникнення виключення є спеціальний метод, або можна перехоплювати виняток.

Що ж можна ще робити зі словниками? Те ж саме, що і з іншими об'єктами: є вбудовані методи, можна організувати перегляд словників поелементно (наприклад, в циклах for або while).

Методи словників

dict.clear() – очищає словник.
dict.copy() – повертає копію словника.
dict.fromkeys (seq [, value]) – створює словник з ключами з seq і значенням value (за замовчуванням None).
dict.get(key [, default]) – повертає значення ключа, але якщо його немає, не викидає виняток, а повертає default (за замовчуванням None).
dict.items() – повертає пари (ключ, значення).
dict.keys() – повертає ключі в словнику.
dict.pop(key [, default]) – видаляє ключ і повертає значення. Якщо ключа немає, повертає default (за замовчуванням кидає виняток).
dict.popitem() – видаляє і повертає пару (ключ, значення). Якщо словник порожній, кидає виняток KeyError. Пам'ятайте, що словники невпорядковані.
dict.setdefault(key [, default]) – повертає значення ключа, але якщо його немає, не кидає виняток, а створює ключ із значенням default (за замовчуванням None).
dict.update([other]) – оновлює словник, додаючи пари (ключ, значення) з other. Існуючі ключі перезаписуються. Повертає None (не новий словник!).
dict.values() – повертає значення в словнику.

1.8 Множини Python

Множина — це сукупність унікальних даних. Тобто елементи множини не можуть дублюватися.

У математиці точне визначення множини може бути абстрактним і важким для розуміння. Однак на практиці набір можна розглядати просто як чітко визначену колекцію різних об’єктів, які зазвичай називають елементами або членами .

Групування об’єктів у набір також може бути корисним у програмуванні, і Python надає для цього вбудований тип множини. Набори відрізняються від інших типів об’єктів унікальними операціями, які можна виконувати над ними.

1.8.1 Створення множин у Python

У Python множини формально створюють, поміщаючи всі елементи у фігурні дужки {}, розділяючи їх комою.

Множина може містити будь-яку кількість елементів, і вони можуть бути різних типів (ціле, з плаваючою точкою, кортеж, рядок тощо). Але набір не може мати такі елементи, як списки, множини або словники (mutable data types). Приклад:

```
# create an empty set
empty_set = set()
# check data type of empty_set
print('Data type of empty_set:', type(empty_set))

# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)

# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)

# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```

Результат:
Тип даних – порожня множина:
Студентський квиток: {112, 114, 115, 116, 118}
Голосні літери: {'u', 'a', 'e', 'i', 'o'}
Набір змішаних типів даних: {'Hello', 'Bye', 101, -2}

У наведеному вище прикладі ми створили різні типи множин, коли розміщували різні набори елементів у фігурні дужки {}.

1.8.2 Вбудовані функції і методи для роботи з елементами множин

Вбудовані функції, такі як all(), any(), enumerate(), len(), max(), min(), тощо sorted(), sum()зазвичай використовуються з наборами для виконання різних завдань.

Функція	Опис
all()	Повертає True, якщо всі елементи множини істинні (або якщо множина порожня).
any()	Повертає True, якщо будь-який елемент множини є істинним. Якщо множина порожня, повертає False.
enumerate()	Повертає об'єкт перерахування. Він містить індекс і значення для всіх елементів множини як пари.
len()	Повертає довжину (кількість елементів) у множині.
max()	Повертає найбільший елемент у множині.
min()	Повертає найменший елемент у множині.
sorted()	Повертає новий відсортований список з елементів у множині (не сортує саму множину).
sum()	Повертає суму всіх елементів у множині.

Ось список усіх методів, які доступні для множин як об’єктів:

Метод	Опис
add()	Додає елемент до множини
clear()	Вилучає всі елементи з множини
copy()	Повертає копію множини
difference()	Повертає різницю двох чи більше множин як нову множину
difference_update()	Видаляє всі елементи іншої множини з цієї множини
discard()	Вилучає елемент із множини, якщо він є її членом (нічого не робить, якщо елементу нема в складі множини)
intersection()	Повертає перетин двох множин як нову множину
intersection_update()	Оновлює множину перетином самої себе та іншої
isdisjoint()	Повертає True, якщо дві множини мають нульовий перетин
issubset()	Повертає True, якщо інша множина містить цю множину
issuperset()	Повертає True, якщо ця множина містить іншу множину

pop()	Вилучає та повертає довільний елемент множини. Викликає KeyError, якщо ця множина порожня
remove()	Вилучає елемент із множини. Якщо елемент не є її членом, викликає KeyError
symmetric_difference()	Повертає симетричну різницю двох множин як нову множину
symmetric_difference_update()	Оновлює множину за допомогою симетричної різниці між нею та іншою
union()	Повертає об’єднання множин у нову множину
update()	Оновлює множину об'єднанням себе та інших

1.9 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. 1.1: 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).

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).

Рис. 1.1. 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»

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.

1.10 Приклади

Приклад 1. Введення і друк

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, наприклад:

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(Введить рядок: ') # або string = input()
print('Ви ввели', string, 'sep="" ')
# введимо два числа і складаємо їх
n = int(input('Введить перше число: '))
m = int(input('Введить друге число: '))
print('{} + {} = {}'.format(n, m, n + m))
```

3) Функції *bin, oct, hex* повертають рядки, які зображують задане число у відповідних системах числення

```
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
```

Приклад 2. Арифметичні операції і обчислення з використанням math

1)

```
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) # зведення до ступеня
print(3.2 * 0.8 - 2 * 5 - 3**3) # арифметичний вираз
print(4 ** 0.5) # зведення до ступеню 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
```

2) 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, основа натуральних логарифмів
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))
```

```
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*]) - з одним аргументом повертає натуральний логарифм x (з основи *e*).

math.log1p(*x*) – повертає натуральний логарифм виразу *1+x* (з основи *e*).

3) Обчислити значення функції $\frac{sin\left(\frac{\pi}{2}\right)}{b\cdot\sqrt{m}}$. Значення вхідних даних *m*=4, *b*=2.

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

Результат

m=4, b=2, y=1.0

Приклад 3. Логічні операції та вирази

```
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. Обчислення деяких функцій

1) 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
```

Приклад 5. Деяки дії з рядками.

1) Створення і печать рядків

```
s1 = "Рядок 1"; s2 = 'Рядок 2'
print(s1, s2, s1+s2)
# 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) Робота з окремими символами рядку

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

3) Слайсінг рядків

```
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
```

4) Використання перевірки in
```
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" ')
```

Приклад результату:
Введіть рядок: студент
В цьому рядку немає символа "q"

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

Приклад 6. 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) # більш зручні f-рядки
```

Приклад 7 Операції зі списком

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)          # Список чисел: [1, 2, 3, 5]
print('Список символів:', char_list)  # Список символів: ['a', 'c', 'z', 'x']
print('Пустий список:', empty_list)  # Пустий список: []

# Створення списку з об'єкту, який ітерується
L = list('abc') # L = [ 'a', 'b', 'c']
# Створення списку з неперервної послідовності цілих чисел
L = list(range(1,10))
print("L = ",L) # L = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

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) # 5   [5, 7, 9, 1, 1, 2]
# Введення індексу
index = int (input ('Введіть номер елемента:'))  # вводимо 2
# Отримання відповідного елемента
element=my_list[index]
# Виведення його значення на екран
print (element)                                    # отримаємо 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) # отримаємо 7
```

4) видалення елемента списку

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

5) додавання (конкатенації) списків

```
# список L
L=[2,3,4]
# список LS
LS = ["456", 7, 3.1415]
# включити в список L список LS
L=L+LS # L = [2, 3, 4, '456', 7, 3.1415]
print("L = ",L)
```

6) Дублювання списку

```
A = [ 1, 3, 'abcde', -0.02 ] # заданий список
B = A # копіювання списку A в список B
C = A*3 # дублювання списку A в список C 3 рази (3 копії)
```

```
print("A = ", A) # A = [1, 3, 'abcde', -0.02]
print("B = ", B) # B = [1, 3, 'abcde', -0.02]
print("C = ", C) # C = [1, 3, 'abcde', -0.02, 1, 3, 'abcde', -0.02, 1, 3, 'abcde', -0.02]
```

Приклад 8. Видалення частин списку

```
# Операція del
# Створимо список
A = [ 1, 2, 3, 4, 5, 6 ]

# Сформувати новий список
B = A[0:5] # B =   [1, 2, 3, 4, 5]

# видалити елемент з індексом 2 з списку B
del B[2]

print("A = ", A)  # A = [1, 2, 3, 4, 5, 6]
print("B = ", B)  # B = [1, 2, 4, 5]

del B[7] # помилка в індексі
```

Результат:

```
IndexError: list assignment index out of range
```

```
# Операція del - видалення діапазону
# Створити список A
A = [ 'a', 'b', 'c', 'd', 'e', 'f' ]

# Сформувати новий список B
B = list(A)

# видалити символи 'b' та 'c' зі списку B
del B[1:3] # B =   ['a', 'd', 'e', 'f']

# Сформувати ще один список C зі списку A
C = list(A) # C = [ 'a', 'b', 'c', 'd', 'e', 'f' ]

# видалити зі списку C останні 3 елементи
n = len(C)
del C[n-3:n] # C =   ['a', 'b', 'c']

# відобразити результат
print("A = ", A) # A =   ['a', 'b', 'c', 'd', 'e', 'f']
print("B = ", B) # B =   ['a', 'd', 'e', 'f']
print("C = ", C) # C =   ['a', 'b', 'c']
```

Приклад 9. Робота з індексами списку

```
# Операція присвоювання окремих елементів списку
# Створити список A
A = list(range(1,10)) # A = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# виклик операції присвоєння
t1 = A[-1] # t=9 - від'ємні індекси переглядаються справа наліво
t2 = A[3] # t=4 - додатні індекси переглядаються зліва направо
t3 = A[0] # t3=1

print("t1 = ",t1) # t1 = 9
print("t2 = ",t2) # t2 = 4
print("t3 = ",t3) # t3 = 1
```

Приклад 10. Зрізи списків.

```
1) Особливості операції присвоювання зрізів
# Створити список A
A = [ 'abc', 'def', 'ghi', 23.5, 8.8, -11 ]

# Створити список B, що містить 2,3,4 елементи списку A
B = A[1:4] # B = ['def', 'ghi', 23.5]

# Створити список C, що містить два останні елементи списку A
n = len(A)
C = A[n-2:n] # C =   [8.8, -11]

print("A = ",A) # A = ['abc', 'def', 'ghi', 23.5, 8.8, -11]
print("B = ",B) # B = ['def', 'ghi', 23.5]
print("C = ",C) # C = [8.8, -11]

2) Зміна списку з допомогою зрізу
# Задано список
A = [0,0,0,0,0,0]

# Змінити список з допомогою зрізу
B = list(A) # зробити копію списку A, B = [0,0,0,0,0,0]
B[1:]=[1,1,1] # B = [0, 1, 1, 1]

C = list(A)   # C = [0,0,0,0,0,0]
C[1:3]=[1,1,1] # C = [0,1,1,1,0,0,0]

D = list(A)     # D = [0,0,0,0,0,0]
D[:3]=[1,1,1]   # D = [1,1,1,0,0,0]

E = list(A)
E[:5]=[1,1,1]   # E = [1,1,1,0]
```

```
F = list(A)
F[1:10]=[1,1,1] #F = [0,1,1,1]

print("A       = ",A)   # A       = [0, 0, 0, 0, 0, 0]
print("B[1:]   = ",B)  # B[1:]   = [0, 1, 1, 1]
print("C[1:3]  = ",C) # C[1:3]  = [0, 1, 1, 1, 0, 0, 0]
print("D[:3]   = ",D) # D[:3]   = [1, 1, 1, 0, 0, 0]
print("E[:5]   = ",E)  # E[:5]   = [1, 1, 1, 0]
print("F[1:10] = ",F) # F[1:10] = [0, 1, 1, 1]
```

3) Зріз списка – отримання значень за інтервалом індексів

```
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 - [5, 7, 9]
# Виведення елементів списка від другого до переостаннього
print(my_list[2:-2]) # [9, 1]
# Виведення ел-тів списка від 4-ого (5-ого) до 5-ого (6-ого)
print(my_list[4:5]) # [1]
```

4) 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 - [5, 9, 1]
# Виведення елементів від 2-ого (3-ього) до передостаннього з кроком 2
print(my_list[2:-2:2]) # [9]
# Виведення ел-тів списка, крім першого, в зворотному порядку
print(my_list[-1:0:-1]) # [2, 1, 1, 9, 7]
```

5) 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:]) # [9, 1, 1, 2]
# Виведення всіх елементів списка від початку до передостаннього елемента
print(my_list[:-2]) # [5, 7, 9, 1]
# Виведення всіх елементів списку в зворотному порядку
print(my_list[::-1]) # [2, 1, 1, 9, 7, 5]
```

Приклад 11. Методи списків

```
1) # метод append() - додавання елементів до списку
# заданий список
A = [ 2, 3.78, 'abcde', True ]

2)# додати 1 елемент до списку
A.append(7) # A = [2, 3.78, 'abcde', True, 7]
print("A = ", A) # результат A = [2, 3.78, 'abcde', True, 7]
# метод extend() - розширення списку
# задані списки
A = [ 2, 3.78, 'abcde', True ]
B = [ "Hello", 77, 1.84 ]

3)# розширити список A на список B
A.extend(B) # A = [2, 3.78, 'abcde', True, 'Hello', 77, 1.84]
print("A = ", A) # результат A = [2, 3.78, 'abcde', True, 'Hello', 77, 1.84]

4)# метод insert() - вставка одиночного елементу в список
# заданий список
A = [ 1, 2, 3, 4, 5]
# вставка в позицію 2 нового числа 777
A.insert(2, 777)
print("A = ", A) # A = [1, 2, 777, 3, 4, 5]

5)# метод index()
# заданий список
A = [ 'a', 'b', 'c', 'd', 'e', 'f' ]
t = A.index('c') # t = 'c'
print("t = ", t) # Результат виконання програми t = 2

6) # метод count() - кількість входжень заданого елементу в списку
# заданий список
A = [ 'a', 'b', 'c', 'd', 'e', 'f' ]
na = A.count('d') # na = 1
B = [ 1, 3, 5, 3, 2, 4 ]
nb = B.count(3) # nb = 2
print("na = ", na)
print("nb = ", nb) # Результат виконання програми na = 1, nb = 2

7) # метод sort() - сортування списку
# заданий список
A = [ 'a', 'f', 'v', 'd', 'n', 'b' ]
# сортування списку
A.sort() # результат A = ['a', 'b', 'd', 'f', 'n', 'v']
# метод sort() - сортування списку
```

```
# заданий список
C = [ 2, 3, 1, 5 ]
C.sort(reverse = True) # посортувати в порядку спадання
print("C = ", C)  # Результат виконання програми C = [5, 3, 2, 1]
8) # метод sort() - сортування списку з заданим ключем key
# заданий список рядків
S = [ "aBC", "ABCD", "ab", "ABCC", "DEFf" ]
S2 = list(S) # утворити новий список
S2.sort(key=str.upper) # посортувати з ключем key
S3 = list(S) # ще один список
S3.sort(key=str.upper, reverse=True) # посортувати з аргументами key та
reverse
print("S = ", S)
print("S2 = ", S2)
print("S3 = ", S3)
```

Результат виконання програми

```
S = ['aBC', 'ABCD', 'ab', 'ABCC', 'DEFf']
S2 = ['ab', 'aBC', 'ABCC', 'ABCD', 'DEFf']
S3 = ['DEFf', 'ABCD', 'ABCC', 'aBC', 'ab']
```

Методи pop() і remove()

```
# метод pop() – витягує елемент зі  списку і повертає його
# заданий список
A = [ 5, 3.8, True, False, "ABCD" ]

# видалити останній елемент
A.pop() # A = [5, 3.8, True, False]
print("A = ", A) # результат A = [5, 3.8, True, False]

# видалити елемент з індексом 1
A.pop(1) # A = [5, True, False]
print("A = ", A) # результат A = [5, True, False]

# метод remove()
# заданий список
A = [ 5, 3.8, True, 3.8, True, False, "ABCD" ]

# видалити перший елемент, який рівний True
A.remove(True) # результат A = [ 5, 3.8, 3.8, True, False, "ABCD" ]

# видалити перший елемент, який рівний 3.8
A.remove(3.8)

print("A = ", A) # результат A = [ 5, 3.8, True, False, "ABCD" ]
```

Приклад 12. Робота зі словниками

```
# Створення словника
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(Dict) # {1: 'Geeks', 2: 'For', 3: 'Geeks'}

# Сворення словника зі змішаними ключами
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict) # {'Name': 'Geeks', 1: [1, 2, 3, 4]}

# Створення порожнього словника
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Створення словника зі списку пар ключ-значення
Dict = dict([(1, 'Geeks'), (2, 'For')])
print("\nDictionary with each item as a pair: ")
print(Dict) # {1: 'Geeks', 2: 'For'}

# Створення вкладеного словника
Dict = {1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
print(Dict) # {1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}

# Демонстрація доступу до значень елементів словника за ключом
# Creating a Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}
# accessing a element using key
print("Accessing a element using key:")
print(Dict['name'])
# accessing a element using key
print("Accessing a element using key:")
print(Dict[1])

# Використання методу get
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}
# accessing a element using get() method
print("Accessing a element using get:")
print(Dict.get(3))  # Geeks

# Перегляд значень вкладеного словника
Dict = {'Dict1': {1: 'Geeks'}, 'Dict2': {'Name': 'For'}}
# Accessing element using key
print(Dict['Dict1'])              #{1: 'Geeks'}
```

```
print(Dict['Dict1'][1])          #Geeks
print(Dict['Dict2']['Name'])  #For
```

Приклад 13. Робота з множинами

```
# Створення списку та тьюплу, їх перетворення в множину
lis1 = [ 3, 4, 1, 4, 5 ]
tup1 = (3, 4, 1, 4, 5)
print(set(lis1))      # {1, 3, 4, 5}
print(set(tup1))     # {1, 3, 4, 5}

# Додавання елементів до множини
myset = set(["a", "b", "c"])
myset.add("d")
print(myset) # {'d', 'c', 'b', 'a'}

# Демонстрація об’єднання двох множин
people = {"Jay", "Idrish", "Archil"}
vampires = {"Karan", "Arjun"}
dracula = {"Deepanshu", "Raju"}

# використання методу union()
population = people.union(vampires)

print("Union using union() function")
print(population) # {'Karan', 'Idrish', 'Jay', 'Arjun', 'Archil'}

# об’єднання з використанням оператору "|"
population = people|dracula
print(population) # {'Deepanshu', 'Idrish', 'Jay', 'Raju', 'Archil'}

{'Karan', 'Idrish', 'Jay', 'Arjun', 'Archil'}

{'Deepanshu', 'Idrish', 'Jay', 'Raju', 'Archil'}

# Демонстрація перетину множин

set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7, 8, 9}

# Intersection using intersection() function
set3 = set1.intersection(set2)
print(set3) # {3, 4, 5}

# Intersection using "&" operator
set4 = set1 & set2
```

print(set4) # {3, 4, 5}

1.11 Практична робота

Мета роботи: ознайомитися з основними типами даних в Python, з властивостями арифметичних операцій, обчислень з використанням math, з можливостями і властивостями списків, тьюплів, словників і множин.

Варіанти завдань

1. Вивчити теоретичні основи написання додатків на мові Python. Опрацювати приклади.
2. Виконати завдання 1 (у відповідності з варіантом), 2-7 (елементи списків, множин та словників обрати самостійно).
3. Скласти звіт по роботі і захистити його.

Завдання 1. Обчислити значення змінної y по значенням інших змінних. Варіанти наведені в таблиці 1.1 (завдання 1.1 та 1.2).

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).

Таблиця 1.1. Значення змінних для обчислення y

Вар.	Завдання 1.1	Завдання 1.2
1.	$t=1,5;\ a=2;\ b=0,7;\ c=0,5;\ y = a\cos(bt\sin(t)) + c$	$x=1,5;\ a=0,5;\ b=1;\ y = e^{-ax}\sqrt{x+1} + e^{-bx}\sqrt[3]{x+1,5}$
2.	$x=1,3;\ a=2;\ b=1,2;\ y = \sqrt{a + be^{\sin x} + 1}$	$x=1,8;\ a=0,5;\ b=3,2;\ y = e^{2x}\,lg(a + x) - b^{3x}\,lg(b - x)$
3.	$x=3,2;\ a=3,7;\ b=0,5;\ y = b^x\,arctg\frac{x}{a} - \sqrt[5]{x+a}$	$x=4,1;\ a=1,7;\ c=3,2;\ b=-2,3;\ y = c \cdot e^{-a\sqrt{x}} - be^{-2\sqrt{x}}$
4.	$x=3,1;\ a=0,5;\ b=1,3;\ y = e^{-ax}\frac{x+\sqrt{x+a}}{x-\sqrt{x-b}}$	$x=-0,7;\ a=-0,5;\ b=1,2;\ y = 2^{-x}\,arctg(x+a) - 3^{-bx}\cos(x+b)$
5.	$x=3,2;\ a=1,2;\ b=0,7;\ y = 2^x\,lg\,a\,x - 3^x\,lg\,b\,x$	$l=1,8;\ a=-0,5;\ b=1,7;\ y = e^{-bl}\sin(al+b) - \sqrt{bl+a}$
6.	$x=-0,8;\ a=-0,5;\ b=2,3;\ y = \frac{bx^2-a}{e^{ax}-1}$	$x=1,5;\ a=1,5;\ b=-1,2;\ y = e^{-ax}\sqrt[3]{ax + b\sin 2x}$
7.	$x=0,6;\ a=2,7;\ b=1,7;\ y = \frac{a}{x+2}e^{-bx^2} + ln(a+bx)$	$x=1,3;\ a=0,5;\ b=3,1;\ y = \sqrt{ax\sin 2x + e^{-2x}(x+b)}$
8.	$x=0,5;\ a=0,3;\ b=0,9;\ y = \frac{a^{2x}+b^{-x}\cos((a+b)x)}{x+1}$	$x=1,9;\ a=2,1;\ b=-0,3;\ y = b\sin(ax^2\cos 2x) - 1$

Вар.	Завдання 1.1	Завдання 1.2
9.	$x=-0,8;\ a=2,3;\ b=0,75;\ y = arcsin\frac{x}{a} - e^{-bx}\sqrt{x+1}$	$x=0,5;\ a=0,5;\ b=2,9;\ y = \frac{ax+e^{-x}\cos bx}{bx-e^{-x}\sin bx+1}$
10.	$x=1,3;\ b=0,8;\ a=1,2;\ y = \frac{a^{2x}+b^{-x}\cos x^3}{x+1}$	$x=2,4;\ a=1,5;\ b=2,5;\ y = a\,arctg\frac{x}{a} - b\,arccos\frac{x}{b}$
11.	$x=5,4;\ a=2,3;\ b=0,5;\ y = 2^x\,lg\,a\cdot x - 3^x\,lg\,b\cdot x$	$x=1,9;\ a=0,3;\ b=1,9;\ y = \frac{ax+e^{-x}\cos bx}{bx-e^{-x}\sin bx+1}$
12.	$x=5,1;\ a=3,7;\ b=1,5;\ y = b^x\,arctg\frac{x}{a} - \sqrt[5]{x+a}$	$x=2,4;\ a=2,3;\ b=-2,3;\ y = \frac{x+a\sin 3x}{a+\sqrt{a+b^2\sin^2 3x}}$
13.	$t=2,5;\ a=4;\ b=0,8;\ c=0,9;\ y = a\cos(bt\sin(t)) + c$	$x=2,5;\ a=0,9;\ b=4;\ y = e^{-ax}\sqrt{x+1} + e^{-bx}\sqrt[3]{x+1,5}$
14.	$x=2,3;\ a=2;\ b=2,2;\ y = \sqrt{a+be^{\sin x}+1}$	$x=2,4;\ a=0,9;\ b=3,2;\ y = e^{2x}\,lg(a+x) - b^{3x}\,lg(b-x)$
15.	$x=3,2;\ a=3,7;\ b=0,9;\ y = b^x\,arctg\frac{x}{a} - \sqrt[5]{x+a}$	$x=4,2;\ a=2,7;\ c=3,2;\ b=-2,3;\ y = c\cdot e^{-a\sqrt{x}} - be^{-2\sqrt{x}}$
16.	$x=3,2;\ a=0,9;\ b=2,3;\ y = e^{-ax}\frac{x+\sqrt{x+a}}{x-\sqrt{x-b}}$	$x=-0,7;\ a=-0,9;\ b=2,2;\ y = 2^{-x}\,arctg(x+a) - 3^{-bx}\cos(x+b)$
17.	$x=3,2;\ a=2,2;\ b=0,7;\ y = 2^x\,lg\,a\,x - 3^x\,lg\,b\,x$	$l=2,4;\ a=-0,9;\ b=2,7;\ y = e^{-bl}\sin(al+b) - \sqrt{bl+a}$
18.	$x=-0,4;\ a=-0,9;\ b=2,3;\ y = \frac{bx^2-a}{e^{ax}-1}$	$x=2,9;\ a=2,9;\ b=-2,2;\ y = e^{-ax}\sqrt[3]{ax+b\sin 2\,x}$
19.	$x=0,6;\ a=2,7;\ b=2,7;\ y = \frac{a}{x+2}e^{-bx^2} + ln(a+bx)$	$x=2,3;\ a=0,9;\ b=3,2;\ y = \sqrt{ax\sin 2\,x + e^{-2x}(x+b)}$
20.	$x=0,9;\ a=0,3;\ b=0,9;\ y = \frac{a^{2x}+b^{-x}\cos((a+b)x)}{x+1}$	$x=2,9;\ a=2,2;\ b=-0,3;\ y = b\sin(ax^2\cos 2\,x) - 1$

Завдання 2. Виконати операції зі списками.
Створити список з 10 випадкових елементів (від -10 до 12). Збільшити значення усіх елементів списку удвічі. Знайти суму, кількість та середнє арифметичне окремо позитивних та негативних елементів та порівняти з відповідними значеннями початкового списку.

Завдання 3. Виконати операції з індексами та елементами списку.
Створити список з числами типа float (не менш 12 елементів). Для цього списку знайти:
- індекс максимального елемента списку;
- різницю між найбільшим та найменшим елементами списку;
- поміняти місцями найбільший та найменший елементи списку;

- знайти найбільший парний елемент списку;
- знайти найменший додатній елемент списку.

Завдання 4. Сортування списків, методи списків.
Побудувати список з назв найкращих пісень (не менш 10). Виконати сортування цього списку в прямом і зворотньом напрямках. Видалити найдовший та найкоротший елементи.

Завдання 5. Списки і множини.
Створити список з 10 випадкових елементів (від -10 до 12). Перевірити, чи є в ньому повторювані елементи.

Завдання 6. Операції з множинами.
Створити два списки з 10 випадкових елементів (від 1 до 9). Обислити об'єднання та перетин множин, які створено з елементів цих списків.

Завдання 7. Словники.
Створить словник з даними про людину (прізвище, ім’я, по-батькові, вік, телефон, email). Роздрукуйте окремо дані про прізвище, вік, email людини.

1.12 Контрольні запитання

1. Які основні типи даних визначені в мові Python.
2. Як ввести значення змінних з термінала?
3. Як проглянути значення змінних? Вивести ці значення на термінал?
4. Як виконувати математичні операції зі змінними?
5. Як обчислити математичні функції? Як получити значення констант?
6. Перерахуйте характеристики типу даних "список", які ви знаєте.
7. Як перевірити наявність елемента у списку?
8. Чим відрізняються методи append() та extend()?
9. Які параметри можна передавати під час зрізів списків?
10. Яким чином можна отримати доступ до значення ключа, не змінюючи при цьому словник?
11. Що може бути ключем словника? Перерахуйте структури даних та загальні вимоги до іменування.
12. Навіщо потрібен метод pop()? Які параметри може приймати? Наведіть приклад використання.
13. Охарактеризуйте методи keys(), items(), values(). Що вони повертають, якою є специфіка результуючих об'єктів?
14. Перерахуйте характеристики типу даних "множина", які ви знаєте.
15. Як перевірити наявність елемента у множині?
16. Перерахуйте методи роботи з об’єктами "множина", які ви знаєте.

2 Управління потоком виконання програм в мові програмування python.

2.1 Умовна інструкція if

2.1.1 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.

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 ліворуч потім праворуч 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'
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
```

2.1.2 Сінтаксіс інструкції 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>: # Інструкція if з умовним виразом test1
   <statements1> # Асоційований блок
elif <test2>: # Необов'язкові частини elif
   <statements2>
else: # Необов'язковий 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
```

2.1.3 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.

2.1.4 Оператор match-case

У Python 3.10 введена нова конструкція match/case, яка називається Structural pattern matching (відповідність структурі шаблону). Оператор match був введений для того, щоб бути більш ніж просто схожим на оператор switch, який є в багатьох інших мовах програмування.

У багатьох випадках конструкція match/caseможе спростити і підвищити читабельність коду Python.

Шаблони в операторах case конструкції match/case складаються з послідовностей, словників, примітивних типів даних, а також екземплярів класів. Зіставлення із зразком дозволяє програмам витягувати інформацію зі складних типів даних, переходити до структури даних та застосовувати певні дії на основі різних форм даних.

Загальний синтаксис конструкції match/case:

```
match  subject :
   case  < pattern_1 > :
      < action_1 >
   case  < pattern_2 > :
      < action_2 >
   case  < pattern_3 > :
      < action_3 >
   case  _ :
      < action_wildcard >
```

Оператор match набуває виразу subject та порівнює його значення з послідовними шаблонами, заданими як один або кілька блоків case. Зокрема, зіставлення із зразком працює наступним чином:

- використання даних з типом та формою (subject);
- оцінка subjecty заяві match;
- порівняння subject з кожним шаблоном у заяві case зверху вниз, доки збіг не буде підтверджено.
- виконання дії action, пов'язаного з шаблоном підтвердженого збігу;
- якщо точний збіг не підтверджено, то як збігаючий випадок буде використовуватися останній case з підстановним знаком '_', якщо він вказаний. Якщо точний збіг не підтверджено і case _: не існує, весь блок match не виконується.

Зверніть увагу, що більшість літералів порівнюються за рівністю. Однак синглтони: True, Falsei None порівнюються з ідентичності.

2.2 Цикли і ітерації

Ітерація означає виконання того самого блоку коду знову і знову, потенційно багато разів. Структура програмування, яка реалізує ітерацію, називається циклом.

У програмуванні існує два типи ітерації: невизначена і визначена:

- З невизначеною ітерацією кількість разів виконання циклу не вказується заздалегідь явно. Швидше, призначений блок виконується неодноразово, доки виконується певна умова.
- З визначеною ітерацією кількість разів, які буде виконано призначений блок, вказується явно під час початку циклу.

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).

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.

2.3 Цикл while

2.3.1 Загальний синтаксис оператору цикла

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.

Давайте подивимося, як інструкція Python while використовується для створення циклів. Формат найпростішого циклу while показано нижче:

```
while <expr>:
   <statement(s)>
```

<statement(s)>представляє блок, який потрібно багаторазово виконувати, часто називають тілом циклу. Це позначається відступом, як і в операторі if.

Пам’ятайте: усі керуючі структури в Python використовують відступи для визначення блоків.

Керуючий вираз <expr>, як правило, включає одну або більше змінних, які ініціалізуються перед запуском циклу, а потім змінюються десь у тілі циклу.

Коли зустрічається цикл while, <expr>спочатку оцінюється в логічному контексті . Якщо воно істинне, виконується тіло циклу. Потім <expr>знову перевіряється, і якщо все ще вірно, тіло виконується знову. Це продовжується до тих пір, поки <expr>не стане false, після чого виконання програми переходить до першого оператора поза тілом циклу.

Розглянемо приклад простого циклу:

```
n = 5
while n > 0:
   n -= 1
   print(n)
```

Отримаємо наступний результат:

```
4
3
2
1
0
```

Ось що відбувається в цьому прикладі:

- Значення n спочатку 5. Вираз у заголовку оператора while в рядку 2 є n > 0, що є істинним, тому тіло циклу виконується. Усередині тіла циклу в рядку 3 n зменшується на 1 до 4, а потім друкується.
- Коли тіло циклу завершується, виконання програми повертається до початку циклу в рядку 2, і вираз обчислюється знову. Це все ще вірно, тому тіло виконується знову та друкується 3.
- Це триває до тих пір, поки n не стане дорівнювати 0. У цей момент, коли вираз перевіряється, він є хибним, і цикл завершується. Виконання буде відновлено після першого оператора, наступного за тілом циклу, але в цьому випадку його немає.

Зауважте, що керуючий вираз циклу while перевіряється спочатку, перш ніж щось трапиться. Якщо спочатку значення false, тіло циклу взагалі ніколи не буде виконано:

```
n = 0
while n > 0:
   n -= 1
   print(n)
```

У прикладі вище, коли зустрічається цикл, n це 0. Керуючий вираз n > 0 уже хибний, тому тіло циклу ніколи не виконується.

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.

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
```

2.3.2 Оператори break і continue

У попередніх простих прикладах все тіло циклу while виконується на кожній ітерації. Але Python надає два ключові слова, які передчасно завершують ітерацію циклу:

- Інструкція Python break одразу повністю завершує цикл. Виконання програми переходить до першого оператора, наступного за тілом циклу.
- Інструкція Python continue негайно завершує поточну ітерацію циклу. Виконання переходить до початку циклу, а керуючий вираз повторно обчислюється, щоб визначити, чи буде цикл виконано знову чи завершиться.

Різниця між breaki continueпоказана на наступній діаграмі (рис. 2.1):

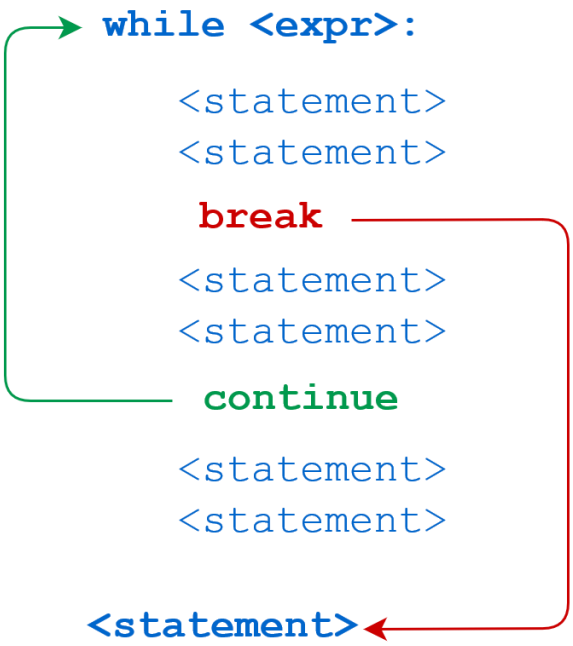

Рис. 2.1. Структурна схема роботи операторів break і continue

Ось проста програма, яка демонструє оператор break:

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('Loop ended.')
```

Результат запуску цієї програми:

```
4
3
Loop ended.
```

Коли n стає 2, оператор break виконується. Цикл повністю припиняється, і виконання програми переходить до оператора print() в рядку 7.

Примітка: Python не має конструкції do-while. Але ви можете використовувати цикл while із інструкцією break для його емуляції.

Наступний сценарій ідентичний наведеному, за винятком оператора continue замість break:

```
n = 5
while n > 0:
```

```
    n -= 1
    if n == 2:
        continue
    print(n)
print('Loop ended.')
```

Результат цієї програми виглядає так:

```
4
3
1
0
Loop ended.
```

Цього разу, коли n дорівнює 2, оператор continue спричиняє завершення цієї ітерації. Таким чином, значення n=2 не друкується. Виконання повертається до початку циклу, умова переоцінюється, і вона залишається істинною. Цикл відновлюється, завершується, коли n стає 0, як і раніше.

2.3.3 Використання else в циклі while

Python допускає додаткову пропозицію else наприкінці циклу while. Це унікальна функція Python, якої немає в більшості інших мов програмування. Синтаксис показано нижче:

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

Зазначене <additional_statement(s)> в пункті else буде виконано після завершення циклу while. Те ж саме можна зробити, розмістивши ці оператори одразу після циклу while, без else:

```
while <expr>:
    <statement(s)>
<additional_statement(s)>
```

Яка різниця? В останньому випадку, без пункту else, <additional_statement(s)> буде виконано після завершення циклу while, незважаючи ні на що.

Коли <additional_statement(s)> розміщені в пункті else, вони будуть виконані лише в тому випадку, якщо цикл завершиться «за вичерпанням», тобто якщо цикл повторюватиметься до тих пір, поки керуюча умова не стане помилковою. Якщо вийти з циклу за допомогою оператора break, пропозиція else не буде виконана.

Розглянемо такий приклад:

```
n = 5
while n > 0:
    n -= 1
```

```
    print(n)
else:
    print('Loop done.')
```
Результати виконання:
```
4
3
2
1
0
Loop done.
```
У цьому випадку цикл повторювався до тих пір, поки умова не була вичерпана: значення n стало рівним 0, тому умова n > 0 стала хибною. Оскільки цикл віджив своє «природне життя», пункт else був виконаний. Тепер розглянемо програму, де цикл while завершується з використаням break:
```
n = 5
while n > 0:
    n -= 1
    print(n)
    if n == 2:
        break
else:
    print('Loop done.')
```

Результати виконання програми:
```
4
3
2
```
Цей цикл передчасно завершується за допомогою break, тому пропозиція else не виконується.

2.3.4 Нескінченні цикли

«while True» — це циклічна конструкція на мові програмування Python, яка дозволяє блоку коду повторюватися нескінченно довго. Він часто використовується в поєднанні з оператором break, який дозволяє вийти з циклу за певних умов.

Щоб використовувати цикл «while True» у Python, ви повинні спочатку визначити умову, яка зрештою матиме значення «False» для завершення циклу. Зазвичай це робиться за допомогою логічної змінної, для якої спочатку встановлено значення «True», а потім змінено в циклі на значення «False».

Припустімо, ви пишете цикл while, який теоретично ніколи не закінчується. Розглянемо приклад:

```
while True:
    print('foo')
```

Результат виконання:

```
foo
foo
foo
  .
  .
  .
foo
foo
foo
Traceback (most recent call last):
  File "<pyshell#2>", line 2, in <module>
    print('foo')
KeyboardInterrupt
```

Цей код завершується при натисканні Ctrl+C, який генерує переривання з клавіатури. Таким чином, за допомогою while True: ініціюється нескінченний цикл, який теоретично працюватиме вічно. Роботу циклу можна припинити, використовуючи оператор break. Можна вказати кілька операторів break у циклі:

```
while True:
    if <expr1>:  # One condition for loop termination
        break
    ...
    if <expr2>:  # Another termination condition
        break
    ...
    if <expr3>:  # Yet another
        break
```

У подібних випадках, коли існує кілька причин для завершення циклу, часто краще виходити break з кількох різних місць, ніж намагатися вказати всі умови завершення в заголовку циклу.

Нескінченні цикли можуть бути дуже корисними. Просто пам’ятайте, що ви повинні переконатися, що цикл у певний момент буде розірваний, щоб він справді не став нескінченним.

2.3.5 Вкладені while цикли

Загалом, керуючі структури Python можуть бути вкладені одна в одну. Наприклад, if/elif/else умовні оператори можуть бути вкладеними:

```
if age < 18:
    if gender == 'M':
        print('son')
    else:
        print('daughter')
elif age >= 18 and age < 65:
```

```
    if gender == 'M':
        print('father')
    else:
        print('mother')
else:
    if gender == 'M':
        print('grandfather')
    else:
        print('grandmother')
```

Так само цикл while може міститися в іншому циклі while. Оператор break або continue, знайдений у вкладених циклах, застосовується до найближчого охоплюючого циклу:

```
while <expr1>:
    statement
    statement

    while <expr2>:
        statement
        statement
        break  # Applies to while <expr2>: loop

    break  # Applies to while <expr1>: loop
```

Крім того, цикли while можуть бути вкладені в оператори if/elif/elsei навпаки:

```
if <expr>:
    statement
    while <expr>:
        statement
        statement
else:
    while <expr>:
        statement
        statement
    statement

while <expr>:
    if <expr>:
        statement
    elif <expr>:
        statement
    else:
        statement

    if <expr>:
        statement
```

Насправді всі керуючі структури Python можна змішувати одна з одною в будь-якій мірі, яка вам потрібна.

2.3.6 Однолінійний запис циклу while

Як і оператор if, цикл while можна вказати в одному рядку. Якщо в блоці, який складає тіло циклу, є кілька операторів, їх можна розділити крапкою з комою (;):

```
n = 5
while n > 0: n -= 1; print(n)
```

Результати виконання:

```
4
3
2
1
0
```

Однак це працює лише з простими операторами. Ви не можете об'єднати два складних висловлювання в один рядок. Таким чином, ви можете вказати весь цикл while в одному рядку, як описано вище, і написати оператор if в одному рядку:

```
if True: print('foo')
```

```
foo
```

Але декілька синтаксичних конструкцій поєднувати в одному рядку не можна:

```
while n > 0: n -= 1; if True: print('foo')
SyntaxError: invalid syntax
```

2.3.7 Інструкція 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.

2.4 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.

2.4.1 Базовий синтаксис циклу Python for

Основний синтаксис циклу for у Python виглядає так, як описано нижче.

```
for iterator_variable in sequence_name:
    Statements
    . . .
    Statements
```

Перше слово оператора починається з ключового слова «for» , яке означає початок циклу for.

Змінна ітератора (iterator_variable) виконує ітерації по послідовності та може використовуватися в циклі для виконання різних функцій.

Далі — ключове слово «in» у Python, яке вказує змінній ітератора виконувати цикл для елементів у послідовності.

Змінна послідовності (sequence_name) може бути списком, кортежем або будь-яким іншим видом ітератора.

Приклад ітерації по символам рядку (python розглядає «рядок» як послідовність символів):

```
word="anaconda"
for letter in word:
   print (letter, end=" ")
```

Результат:

```
a n a c o n d a
```

Списки та кортежі теж є ітерованими об’єктами. Розглянемо приклад перегляду елементів списку:

```
words= ["Apple", "Banana", "Car", "Dolphin" ]
for word in words:
   print (word)
```

Результат:

```
Apple
Banana
Car
Dolphin
```

Ще один приклад ітерації за елементами кортежа:

```
nums = (1, 2, 3, 4)
```

```
sum_nums = 0
for num in nums:
    sum_nums = sum_nums + num
print(f'Sum of numbers is {sum_nums}')

# Output
# Sum of numbers is 10
```

2.4.2 Вкладені цикли for

Цикли for можуть бути вкладеними один до одного. Розглянемо приклад друку декількох елементів списку посимвольно:

```
words= ["Apple", "Banana", "Car", "Dolphin" ]
for word in words:
    #This loop is fetching word from the list
    print ("The following lines will print each letters of "+word)
    for letter in word:
        #This loop is fetching letter for the word
        print (letter, end=" ")
    print("") #This print is used to print a blank line
```

Результати друку:

```
The following lines will print each letters of Apple
A p p l e
The following lines will print each letters of Banana
B a n a n a
The following lines will print each letters of Car
C a r
The following lines will print each letters of Dolphin
D o l p h i n
```

2.4.3 Цикл for с использованием функции range()

Функція range() дозволяє генерувати послідовність чисел. Зазвичай вона використовується для повторення заданої кількості разів. За допомогою range() генерується послідовність значень змінної циклу.

Основний синтаксис:

```
for i in range(start, stop, step):
  # your code here
```

i: змінна циклу

start: необов'язковий параметр, що визначає початкове значення змінної циклу; за замовчуванням це 0.

stop: обов'язковий параметр, який визначає кінцеве значення змінної циклу.

step: необов'язковий параметр, що вказує крок зрісту змінної циклу, зі значенням за замовчуванням 1. Розглянемо приклад нижче, де ми роздруковані числа від 1 до 5 за допомогою функції діапазону з циклом for:

```
for i in range(5):
    print(i)
```

Аналогічний приклад:

```
for x in range(3):
    print("Printing:", x)

# Output
# Printing: 0
# Printing: 1
# Printing: 2
```

Нагадаємо, що функція приймає range() приймає параметри start, step, stop. У наведеному нижче прикладі стартове значення дорівнює 1, крок дорівнює 3, кінцеве значення дорівнює 10.

```
for n in range(1, 10, 3):
    print("Printing with step:", n)

# Output
# Printing with step: 1
# Printing with step: 4
# Printing with step: 7
```

2.4.4 Використання break і continue з циклом for

Оператор break використовується для передчасного виходу з циклу for. Він використовується для розриву циклу for, коли виконується певна умова. Наприклад, є список чисел і необхідно перевірити, чи присутній номер чи ні. Пов’язане з циклом рішення - перебирати список чисел і, якщо число знайдено, виходити з циклу, тому що немає потреби перебирати ті елементи, що залишилися (див. приклад нижче):

```
nums = [1, 2, 3, 4, 5, 6] #список чисел

n = 2

found = False
for num in nums:
    if n == num:
        found = True
        break

print(f'List contains {n}: {found}')

# Output
```

```
# List contains 2: True
```

Оператори continue всередині циклу for використовуються, щоб пропустити виконання тіла циклу for за певних умов. Розглянемо приклад: є список чисел і необхідно надрукувати суму додатних чисел. Ми можемо використовувати оператор continue, щоб пропустити цикл for для від'ємних чисел.

```
nums = [1, 2, -3, 4, -5, 6]

sum_positives = 0

for num in nums:
    if num < 0:
        continue
    sum_positives += num

print(f'Sum of Positive Numbers: {sum_positives}')
```

2.4.5 Використання гілкі else в циклі for

Ми можемо використовувати блок else із циклом Python for. Блок else виконується лише тоді, коли цикл for не завершується оператором break.

Розглянемо приклад: є послідовність чисел, сума якої друкується, коли всі числа парні. Сума розраховується в циклі for. Якщо присутнє непарне число, використовується оператор break, щоб передчасно завершити цикл for. Частина else виконується лише тоді, коли цикл for завершується нормально.

```
even_nums = [2, 4, 6, 8]
total = 0
for x in even_nums:
    if x % 2 != 0:
        break
    total += x
else:
    print("For loop executed normally")
    print(f'Sum of numbers {total}')
```

Буде надруковано

```
# For loop executed normally
# Sum of numbers 20
```

Якщо серед чисел у списку є від'ємне (наприклад, even_nums = [2, 4, 5, 8]), виконання циклу скінчується з використанням break і гілка else не виконується, тому значення суми не друкується.

2.4.6 Оператор pass з циклом for

Оператор pass у Python використовується для запису порожніх циклів. Pass також використовується для порожніх операторів керування, функцій і класів.

Приклад:

```
# An empty loop
for letter in 'python_is_good':
        pass
print('Last Letter :', letter)
```

2.4.7 Загальний варіант циклу for

Загальна форма циклу for Python виглядає так:

```
for <var> in <iterable>:
   <statement(s)>
```

<iterable>це набір об’єктів, наприклад, список або кортеж. Тіло <statement(s)>циклу позначається відступом, як і в усіх керуючих структурах Python, і виконується один раз для кожного елемента в <iterable>. Змінна циклу <var>приймає значення наступного елемента в <iterable>кожному циклі.

Але що саме таке iterable? У Python ітераційний означає, що об’єкт можна використовувати в ітерації. Термін використовується як:

- Прикметник: об’єкт можна описати як повторюваний.
- Іменник: об’єкт можна охарактеризувати як повторюваний.

Якщо об’єкт можна ітерувати, його можна передати вбудованій функції Python iter(), яка повертає те, що називається ітератором.

Усі типи даних, які розглянуто в зв’язку з циклами, є колекціями або ви стикалися на даний момент, які є колекціями або ітерованими типами контейнерів. До них належать типи string, list, tuple, dict, set і frozenset. Детальніше ітератори будуть розглянуті пізніше.

2.5 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 множини (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
```

2.5.1 Обробка помилок шляхом перевірки вводу

Якщо користувач введе неправильний рядок, маємо:

```
Enter text:xxx
...текст повідомлення про помилку опущено...
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
```

2.5.2 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.

2.6 Приклади

Приклад 1. Використання оператору if

```
# Функція float конвертує значення-рядок у дійсне, з яким можна виконувати
# арифметичні операції
x = float(input('Введіть перше число: '))
y = float(input('Введіть друге число: '))
# operation — рядок
operation = input('Введіть знак арифметичної операції: ')
# 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 зведення у ступінь
else:
print('Операція, яка не підтримується')
# отримаємо результат, якщо операція була припустимою
string = input('Введіть рядок: ') # або string = input()
print('Ви ввели string, sep=" " ')
# введення двох чисел
```

Приклад 2. Приклади логічних 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 3. Створення текстового меню з використанням if

```
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')
```

Приклад 4. Використання match-case (версія python не нижче 3.10)

```
# Приклад використання конструкції match-case
user = input("Write your username -: ")

# match statement starts here .
match user:
    case "Om":
        print("Om do not have access  to the database only for the api code.")
    case "John":
        print(
            "John do not have access to the database, only for the frontend code.")

    case "Richard":
        print("Richard have the access to the database")

    case _:
        print("You do not have any access to the code")

# використання гілки default
argument=1
match argument:
    case 0:
        print("zero")
    case 1:
        print("one") # буде надруковано one
    case 2:
        print("two")
    case default:
        print("something")
```

Приклад 5. 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) # Нескінченний цикл
print('Усі натуральні числа:')
n = 1
while True: # нескінчений цикл
        print(n); n += 1
```

```
4) # вихід з нескінченного циклу за командою
while True:
   print('Введіть exit, щоб завершити цикл')
   response = input('> ')
   if response == 'exit': break

5) # приклад використання continue
x = 0
while x < 10:
   x += 1
   if x == 5: # пропускаємо число 5
      continue
      print(' Поточне число дорівнює ', x)


6) # цикл while з else
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 6. Обчислення суми ряду з використанням while

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): $\frac{x}{0!}+\frac{x}{1!}+\frac{x^2}{2!}+\ldots+\frac{x^n}{n!}$

```
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)
```

Приклад 7. Обчислення елементів послідовности з використанням while

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 розглядається як int
x=a
while x!=b:
        y=math.sqrt(x); print(x,y); x+=h
```

Приклад 8. Використання функції range і циклу for

1) Параметр і приймає значення в діапазоні [0, 10)

```
for i in range(10):
        print('i =', i)
```

2) Параметр і приймає значення в діапазоні [5, 10)

```
for i in range(5, 10):
        print('i =', i)
```

3) Параметр і 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":
      print('Цикл сам був завершений')
      break
   else: # цю гілку виконують тільки якщо цикл не був перерваний
      print('Цикл виконано повністю')
print('Кінець програми')
```

5)

```
for x in range(1, 11):
        if x == 5: continue # пропускаємо число 5
        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, end=" ")
```

Результат

```
4  3  2  1  0
```

Приклад 9. Обчислення суми елементів послідовности (цикл for)

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

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 10. Обчислення елементів послідовности (цикл for)
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 розглядається як int
for i in range(a,b+1,h):
        x=i; y=math.sqrt(i); print(x,y)
```

Приклад 11. Цикл за набором елементів списку.
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
   print(x)
```

Приклад 12. Вихід з циклу за допомогою break
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
   print(x)
   if x == "banana":
      break
```

Приклад 13. Пропуск ітерації циклу за допомогою continue
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
   if x == "banana":
```

```
    continue
  print(x)
```

Приклад 14. Використання блоку else.

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
#If the loop breaks, the else block is not executed.
```

2.7 Практична робота

Мета роботи: ознайомитися з реалізацією розгалуження потоку виконання програми; алгоритмів циклічної структури на мові Python; вивчити формат умовного оператору if, операторів циклу while та for, функції range.

Завдання

1. Вивчити теоретичні основи організації циклів на мові Python. Опрацювати приклади.
2. Виконати завдання 1-4 у відповідності з варіантом.
3. Скласти звіт і захистити його по роботі.

Варіанти завдань:

Завдання 1. Скласти програму для обчислення значення rez в залежності від поставленого умови.

Для отримання результатів підібрати вихідні дані так, щоб виконувалися всі можливі галузі алгоритму. Перед виведенням отриманого результату повинно виводитися повідомлення про гілки, при проходженні якої він отриманий. Зверніть увагу, що можливий випадок, при якому вихідні дані не будуть підходити ні для одного із запропонованих умов.

1. $rez = \begin{cases} sin^2(5k + 3m\,ln|k|)\,; 0 \le k < m; \\ cos^2(5k + 3m\,ln|k|)\,; \swarrow\nearrow\ k \ge \frac{m}{2}; \end{cases}$

2. $rez = \begin{cases} ln(|2\phi - 3e^2\theta|)\,; |\phi| < 5|\theta|; \\ ln(|2\phi^2 - 3\theta|)\,; 5|\theta| < |\phi| \le 7.5|\theta|; \end{cases}$

3. $rez = \begin{cases} \sqrt{\left|2{k_1}^2 - 5{k_2}^2\right|} \cdot e^{k_1 + k_2}; 0 < k_1 \cdot k_2 \le 1; \\ \sqrt{\left|2{k_1}^2 + 5k_2\right|} \cdot e^{k_1 - k_2}; \swarrow\nearrow\ k_1 \cdot k_2 > 1; \end{cases}$

4. $rez = \begin{cases} \frac{4r+3m}{r^3+m^2} sin^2\, m^3\,; 0.5 < |r| < |m| + 0.5; \\ \sqrt{|r-m|}\, cos^3\, r^2\,; \quad |r| > |m| + 0.5; \end{cases}$

5. $rez = \begin{cases} arctg(5m^2t + 7mt^2); & m^2 + t^2 > 0.5; \\ arcsin(5m^2t + 7mt^2); & 0.1 < m^2 + t^2 \le 0.5; \end{cases}$

6. $rez = \begin{cases} sin^2(\pi n_1 + e^{n_2}); & \pi \le n_1 + n_2 < 5; \\ sin^2(\pi n_2 + n_1); & n_1 + n_2 \ge 5; \end{cases}$

7. $rez = \begin{cases} \sqrt{|3m - 5r|}e^{\frac{m}{r}}; & r \le m < 2r; \\ \sqrt{|3m + 5r|}e^{\frac{r}{m}}; & m > 2r; \end{cases}$

8. $rez = \begin{cases} tg^2(c - 2k); \swarrow & |c + k| > 2; \\ ln(|c - 2k|) - sin\frac{c}{2k}; \swarrow & 0.5 < |c + k| \le 2; \end{cases}$

9. $rez = \begin{cases} \frac{m_1 - 2m_2}{{m_1}^2 + 2{m_2}^2}; & 0.1 < |m_1 - 2m_2| \le 1; \\ 2(m_1 - m_2)e^{\frac{m_1}{m_2} - 1}; & |m_1 - 2m_2| > 1; \end{cases}$

10. $rez = \begin{cases} \sqrt{|se^2 - ne^{-2}|}; & \frac{|n|}{2} < s \le |n|; \\ \sqrt{|s - n|}\, sin^3(s + n); \swarrow & s > |n|; \end{cases}$.

11. $rez = \begin{cases} z^3 - ln(|p| + |z|); \swarrow & 0 < p \le z + 1; \\ ln(|p - z|) + cos^2 p; \swarrow & p > z + 1; \end{cases}$

12. $rez = \begin{cases} \sqrt{|x \cdot e^{sin\, x} + t \cdot e^{-2x}|}; & 3t \le x < 10t; \\ \sqrt{|x + t| \cdot e^{cos\, x}}; & x \ge 10t; \end{cases}$

13. $rez = \begin{cases} \frac{arctg7k - 5p}{2\, sin\, k^2 + 3p^2}; & k > |p|; \\ |k - p| \cdot arcctg2k; & 0.1|p| < k \le |p|; \end{cases}$

14. $rez = \begin{cases} e^{-|m+r|} + lg|m|; & r > -2m; \\ e^{|m+r|} - lg|m|; & -2.5m < r \le -2m; \end{cases}$

15. $rez = \begin{cases} e^{-\frac{|\alpha| + |\beta|}{2}} \cdot ctg\alpha; & \alpha \cdot \beta > 0.5; \\ |\alpha + \beta^2| \cdot ctg\beta; & 0.4 < \alpha \cdot \beta \le 0.5; \end{cases}$

16. $rez = \begin{cases} e^{n1+n2} \cdot \sqrt{|2n1 - 5n2^2|}; & 0 < n1 \cdot n2 \le 1; \\ e^{n1-n2} \cdot \sqrt{|2n1^2 + 5n2|}; & n1 \cdot n2 > 1; \end{cases}$

Завдання 2

Обчислити значення функції в залежності від інтервалу, в який потрапляє вводиться з клавіатури аргумент:

1. Для t∈ [0,3], де a=-0.5,b=2

$$z = \begin{cases} a\, t^2 \ln t & \text{при } 1 \le t \le 2 \\ 1 & \text{при } t<1, \\ e^{\,a\,t} \cos b\, t & \text{при } t>2, \end{cases}$$

2. Для x∈ [0,4], де a=2.3

$$f = \begin{cases} \sqrt[5]{x + a} & \text{при } x>2, \\ x & \text{при } 0.3< x \le 2, \\ \cos(x-a) & \text{при } x\le 0.3, \end{cases}$$

3. Для x∈ [0,7],
де a=-2.7,b=-0.27

$$z = \begin{cases} (a+b)/(e^x+\cos x) & \text{при } 0 \le x<2.3, \\ (a+b)/(x+1) & \text{при } 2.3\le x<5, \\ e^x+\sin x & \text{при } 7 \ge x \ge 5, \end{cases}$$

4. Для i∈ [7,12],
де a=2.2,b=0.3.

$$y = \begin{cases} a\, i^4 + b\, i & \text{при } i< 10, \\ \operatorname{tg}(i + 0.5) & \text{при } i = 10, \\ e^{2i} + \sqrt{a^2 + i^2} & \text{при } i >10, \end{cases}$$

5. Для x∈ [0.9,5],
де a=1.5

$$y = \begin{cases} \pi x^2 - 7/x^2 & \text{при } x<1.3, \\ ax^3 + 7\sqrt{x} & \text{при } 1.3 \le x<3, \\ \lg(x + 7\sqrt{x}) & \text{при } x \ge 3, \end{cases}$$

6. Для t∈ [-1.4],
де a=2.1,b=0.37.

$$z = \begin{cases} \sqrt{at^2 + b\sin t + 1} & \text{при } t<0.1, \\ at + b & \text{при } 0.1\le t< 2, \\ \sqrt{at^2 + b\cos t + 1} & \text{при } t \ge 2, \end{cases}$$

7. Для x ∈ [0,6],
де a=1.5.

$$y = \begin{cases} a\, e^{\sin x} + 2.5 & \text{при } x<0.3, \\ e^{\cos x} + a & \text{при } 0.3\le x<4, \end{cases}$$

8. Для x∈ [1,2],
де a=1.8,b=-0.5, c=3.5

$$y = \begin{cases} (\sin x)/ (a + e^x) & \text{при } x \ge 4, \\ a/x + b\, x^2 - c & \text{при } x \le 1.2, \\ (a + bx)/\sqrt{x + 1} & \text{при } x>1.2, \end{cases}$$

9. Для t∈ [1,5],
де a=2.5

$$z = \begin{cases} t\sqrt[3]{t - a} & \text{при } t>a, \\ t \sin a\, t & \text{при } t=a, \\ e^{-at}\cos a\, t & \text{при } t<a, \end{cases}$$

10. Для x∈ [0,4],
де a=1,b=3.

$$y = \begin{cases} e^{-bx}\sin b\, x & \text{при } x<2.3, \\ \cos bx & \text{при } 2.3 \le x < 3, \\ e^{-ax}\cos b\, x & \text{при } x \ge 3, \end{cases}$$

11. Для t∈ [0.5,3],
де a=1.3,b=6.5

$$z = \begin{cases} a\, t^2 - b\sqrt{t + 1} & \text{при } t<a, \\ a - b & \text{при } a \le t \le b, \\ a\, t^{2/3} - \sqrt[3]{t + 1} & \text{при } t>b, \end{cases}$$

12. Для x∈ [0,2],
де b=-2.9

$$y = \begin{cases} |e^{-2x}\sin bx| & \text{при } x >1, \\ \cos bx & \text{при } x = 1, \\ e^{-x}\cos bx & \text{при } x < 1, \end{cases}$$

13. Для x∈ [0.5,2]

$$\begin{cases} \sin(\cos a\, x) & \text{при } x >1, \end{cases}$$

де a=-0.8 $z = \begin{cases} \text{tg } ax & \text{при } x = 1, \\ a^2 x & \text{при } x < 1, \end{cases}$

14. Для x∈ [1,2], де b=1.3. $y = \begin{cases} \ln bx - 1/(bx+1) & \text{при } x < 1.3, \\ bx + 1 & \text{при } 1.3 \geq x \geq 1.7, \\ \ln bx + 1/(bx+1) & \text{при } x > 1.7, \end{cases}$

15. Для x∈ [-1,1], де a=2.5,b=-0.9. $z = \begin{cases} ax^2+bx^{2/3} & \text{при } x<0.1, \\ a\,x^2 & \text{при } x=0.1, \\ b\,x^{2/3} & \text{при } x>0.1. \end{cases}$

16. Ввести координати точки (x, y). Надрукувати, в якому квадраті або на якій осі координат знаходиться ця точка.

Завдання 3

Для кожного x, що змінюється від ***a*** до ***b*** з кроком ***h***, знайти значення функції Y (x), суми S (x) і $|Y(x)–S(x)|$ і вивести у вигляді таблиці. Значення ***a, b, h*** і ***n*** вводяться з клавіатури. Так як значення S(x) є рядом розкладання функції Y(x), при правильному рішенні значення S і Y для заданого аргументу x (для тестових значень вихідних даних) повинні збігатися в цілій частині і в перших двох-чотирьох позиціях після крапки десяткового дробу.

Роботу програми перевірити для ***a = 0,1; b = 1,0; h = 0,1;*** значення параметра ***n*** вибрати залежно від завдання.

Обчислення реалізувати за допомогою конструкцій циклу for та while. Як перевірити, чи виконано усі ітерації циклічного обчислення, або були досягнуті особливі умови виходу з циклу?

1. $S(x) = \sum_{k=0}^{n}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$, $Y(x) = sin(x)$.
2. $S(x) = \sum_{k=0}^{n}(-1)^{k+1} \frac{x^{2k}}{2k(2k-1)}$, $\boldsymbol{Y(x) = x \cdot arctg(x) - ln\sqrt{1+x^2}}$.
3. $S(x) = \sum_{k=0}^{n} \frac{cos(k\pi/4)}{k!} x^k$, $Y(x) = e^{x\,cos\frac{\pi}{4}} cos(x\,sin(\pi/4))$.
4. $S(x) = \sum_{k=0}^{n}(-1)^k \frac{x^{2k}}{(2k)!}$, $Y(x) = cos(x)$.
5. $S(x) = \sum_{k=0}^{n} \frac{cos(kx)}{k!}$, $Y(x) = e^{cos\,x} cos(sin(x))$.
6. $S(x) = \sum_{k=0}^{n} \frac{2k+1}{k!} x^{2k}$, $Y(x) = (1+2x^2)e^{x^2}$.
7. $S(x) = \sum_{k=1}^{n} \frac{x^k cos(k\pi/3)}{k}$, $Y(x) = -\frac{1}{2} ln(1 - 2x\,cos\frac{\pi}{3} + x^2)$.
8. $S(x) = \sum_{k=0}^{n} \frac{(2x)^k}{k!}$, $Y(x) = e^{2x}$.
9. $S(x) = \sum_{k=1}^{n}(-1)^{k+1} \frac{x^{2k+1}}{4k^2-1}$, $Y(x) = \frac{1+x^2}{2} arctg(x) - x/2$.
10. $S(x) = \sum_{k=0}^{n} \frac{x^{2k}}{(2k)!}$, $Y(x) = \frac{e^x+e^{-x}}{2}$.
11. $S(x) = \sum_{k=0}^{n} \frac{k^2+1}{k!} (x/2)^k$, $Y(x) = (x^2/4 + x/2 + 1)e^{x/2}$.

12. $S(x) = \sum_{k=0}^{n}(-1)^k \frac{2k^2+1}{(2k)!}x^{2k}$, $Y(x) = (1-\frac{x^2}{2})\,cos(x) - \frac{x}{2}sin(x)$.

13. $S(x) = \sum_{k=1}^{n}(-1)^k \frac{(2x)^{2k}}{(2k)!}$, $Y(x) = 2(cos^2 x - 1)$.

14. $S(x) = \sum_{k=0}^{n} \frac{x^{2k+1}}{(2k+1)!}$, $Y(x) = \frac{e^x - e^{-x}}{2}$.

15. $\mathbf{S}(x) = \sum_{k=1}^{n}(-1)^{k+1} \frac{x^{2k}}{2k(2k-1)}$, $Y(x) = -\ln\sqrt{1+x^2} + x\,arctg(x)$.

16. $S(x) = \sum_{k=0}^{n} \frac{cos(k\pi/4)}{k!}x^k$, $Y(x) = cos[\,x \cdot sin(\pi/4)]e^{x\,cos\frac{\pi}{4}}$.

Завдання 4

Обчислити значення двох функцій в n рівномірно розподілених в діапазоні $a \le x \le b$ точках. Результати оформити у вигляді таблиці.

Обчислення реалізувати за допомогою конструкцій циклу for та while.

№ п/п	a	b	n	F1(x)	F2(x)
1	0	2π	20	sin x - cos x	sin x + cos x -1
2	1	2	18	$1+2^{x+5}$	$(x-1)^3$
3	-1	5	15	$4e^{-\lvert x\rvert}-1$	cos x
4	-2	5	14	$\lvert x+10\rvert^5$	$e^{-(x+5)}$
5	0	π	16	2 sin 2x +1	$(x+5)^3(1+\sin^2 x)$
6	$-\pi$	π	20	2-cos x	$\sqrt{x+4}$
7	-1	3	20	$2^{-x}/100$	$20/(1+x^2)$
8	-4	4	12	x^3e^{2x}	$e^x \sin x$
9	1	3	15	$\sqrt{e^x-1}$	$x \ln^2 x$
10	1	4	20	$1/(1+\sqrt{x})$	$2^x/(1-4^x)$
11	0	2π	20	5-3 cos x	$\sqrt{1+sin^2 x}$
12	$-\pi$	π	18	|sinx|+|cosx|	|sin x| - |cos x|
13	0	π	16	e^{-x}+cos2x	e^{-2x}
14	1	2	12	$e^{-x}\lg\sqrt{x+1}$	x + sin x
15	2	4	10	x cos x/2	$\sqrt[3]{x} + \sqrt{2}\,e^{-x}$
16	2	4	16	$2^x \lg x - 3^x \lg x$	ctg x
17	0	5	18	$3^{-x}/50$	$x\,e^{-x} + \ln x$
18	1	2	20	$e^{2x}\ \sqrt[3]{x}$ - sin x	$10/(2+x^2)$
19	3	4	18	2^x arctg x - $\sqrt[5]{x+1}$	$e^{ax} - 2*x^2$
20	1	3	16	$5^{-x}/50$	$e^{2x} \lg x - 3^{3x}$

2.8 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)?
5. Що таке цикл, для чого його використовують?
6. Як описується та виконується циклічна інструкція while?
7. Як можна організувати нескінченні цикли? Наведіть декілька прикладів і поясніть їх. Як можна вийти з нескінченних циклів? Що відбувається при запуску нескінченного циклу?
8. Чи може оператор циклу не мати тіла? Чому?
9. Для чого служать оператори переривання break та continue?
10. Для чого використовується конструкція try-except? Наведіть приклади.

3 Створення та використання функцій користувача. Робота з лямбда-функціями.

3.1 Поняття функції

Функція в програмуванні являє собою відокремлену ділянку коду, яку можна викликати, звернувшись до нього на ім'я, яким він був названий. Під час виклику відбувається виконання команд тіла функції.

Функції дозволяють розбивати складні процеси на менші кроки.

Функції можна порівняти з невеликими програмами, які власними силами, тобто автономно, не виконуються, а вбудовуються у стандартну програму. Іноді їх так і називають - підпрограми. Інших ключових відмінностей функцій програм немає. Функції також за необхідності можуть отримувати та повертати дані. Тільки зазвичай вони їх отримують не з введення (клавіатури, файлу та ін), а з програми, що викликає. Сюди вони повертають результат своєї роботи.

3.2 Розділення простору імен

Простір імен — це область програми, в якій ідентифікатори мають значення. Як ви побачите нижче, коли викликається функція Python, для цієї функції створюється новий простір імен, який відрізняється від усіх інших просторів імен, які вже існують.

Практичний результат цього полягає в тому, що змінні можуть бути визначені та використані у функції Python, навіть якщо вони мають те саме ім’я, що й змінні, визначені в інших функціях або в основній програмі. У цих випадках не буде плутанини чи перешкод, оскільки вони зберігаються в окремих просторах імен.

Це означає, що коли ви пишете код у функції, ви можете використовувати імена змінних та ідентифікатори, не турбуючись про те, чи вони вже використовуються деінде за межами функції. Це допомагає значно мінімізувати помилки в коді.

3.3 Визначення функції

Функції Python визначаються за допомогою інструкції def, яке вводить визначення функції. За ним має йти ім'я функції та укладений у дужки список формальних параметрів аргументів. Оператори, які формують тіло функції, починаються з наступного рядка та повинні мати відступ.

```
def func_name ( param ):
   pass
```

Тут func_name - ідентифікатор, тобто змінна, яка під час інструкції def пов'язується зі значенням як об'єкта функції; param - це необов'язковий список формальних параметрів аргументів, які пов'язуються зі значеннями, що надаються під час виклику функції. У найпростішому випадку функція взагалі

може не мати параметрів. Це означає, що за її виклик вона не отримує жодних аргументів. У визначенні такої функції круглі дужки після її імені залишаються порожніми, такими ж вони повинні залишатися під час її виклику.

Першим оператором тіла функції може бути рядок документації функції. Існують інструменти, які використовують рядки документації для автоматичного створення інтерактивної чи друкованої документації або надання користувачеві інтерактивного перегляду коду. Хорошою практикою є включення рядків документації до коду.

Визначення функції - це оператор, який виконується. Його виконання пов'язує ім'я функції в поточному локальному просторі імен з об'єктом функції (оболонка навколо коду функції, що виконується). Цей об'єкт функції містить посилання на поточний глобальний простір імен, який використовуватиметься під час виклику функції.

Визначення функції не виконує тіло функції, лише обчислює аргументи, якщо вони присутні. Тіло функції виконується лише під час виклику.

Таким чином, 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.

При створенні складних программ іноді використовується інструкція pass. ЇЇ 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
```

3.4 Виклик функцій

Синтаксис для виклику функції Python такий:

```
<function_name>([<arguments>])
```

<arguments> - це значення, передані у функцію. Вони відповідають <parameters> у визначенні функції Python. Ви можете визначити функцію, яка не приймає жодних аргументів, але дужки все одно є обов'язковими. Як визначення функції, так і її виклик завжди мають містити круглі дужки, навіть якщо вони порожні.

Розглянемо простий виклик функції python без параметрів:

```
def f():
  s = '-- Inside f()'
  print(s)

print('Before calling f()')
f()
print('After calling f()')
```

Ось як працює цей код:

У рядку 1 використовується ключове слово def, щоб вказати, що визначається функція. Виконання оператора def лише створює визначення f(). Усі наступні рядки з відступом (рядки 2–3) стають частиною тіла функції f() та зберігаються як його визначення, але вони ще не виконуються.

Рядок 4 — це невеликий пробіл між визначенням функції та першим рядком основної програми. Хоча це не є синтаксично необхідним, його рекомендується мати.

Рядок 5 — це перший оператор без відступу, оскільки він не є частиною визначення f(). Це початок основної програми. Коли виконується основна програма, цей оператор виконується першим.

Рядок 6 — це виклик до f(). Зауважте, що порожні дужки завжди потрібні як у визначенні функції, так і в її виклику, навіть якщо немає параметрів або аргументів. Виконання продовжується, f()і оператори в тілі f() виконуються.

Рядок 7 є наступним рядком, який буде виконано після завершення тіла f(). Виконання повертається до цього оператора print().

Послідовність виконання (або потоку керування) для foo.рупоказана на наступній діаграмі (рис. 3.1)

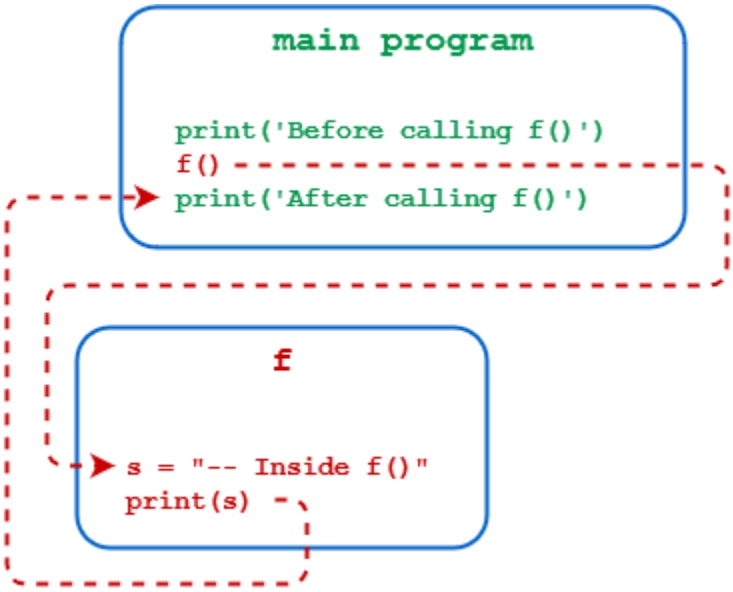

Рис. 3.1. Передача потоку керування під час виклику функції

3.5 Аргументи функції

Функція може приймати довільну кількість аргументів або взагалі не приймати їх. Також поширені функції з довільним числом аргументів, функції з позиційними та іменованими аргументами, обов'язковими та необов'язковими. Приклад:

```
def func(a, b, c=2): # c – необов’язковий аргумент
   return a + b + c
...
func(1, 2)  # a = 1, b = 2, c = 2 (за замовчанням)
5
func(1, 2, 3)  # a = 1, b = 2, c = 3
6
func(a=1, b=3)  # a = 1, b = 3, c = 2
6
func(a=3, c=6)  # a = 3, c = 6, b не визначений, тому виникає помилка
Traceback (most recent call last):
  File "", line 1, in
    func(a=3, c=6)
```

TypeError: func() takes at least 2 arguments (2 given)

Скалярні аргументи (числа, логічні значення) передаються виключно по значенню. Більш складні структури даних (списки, словники та інші) передаються за посиланням (про це нижче). Приклад:

```
def f(x,y):
   print('Локальні значення ',x,y)
   x=1 # змінюємо значення параметрів
   y=2
   return x+y
# основна програма
a, b = 2,4
print('f= ',f(a,b))
print(a,b)
```

Результат:
```
Локальні значення  2 4
f= 3
2 4
```

3.6 Списки та словники як аргументи функцій

Списки і словники - структури даних, що можна змінювати.

Якщо передавати список як параметр функції, фактично в тіло функції передається посилання на область пам'яті, зайняту списком. Тому в тілі функції можна вносити зміни до вмісту списку, переданого як параметр.

Розглянемо приклад. Функція ChangeList отримує список як параметр. Потім у тілі функції цей перелік змінюється (змінюються перший і другий елементи списку).

```
# Передача списку у функцію. Список - об'єкт, що змінюється
# Функція, яка отримує список як параметр
def ChangeList(L):
   # Змінюємо список
   L[0] = 777 # Змінюємо перший елемент списку
   L[1] = (2, 3) # Змінюємо другий елемент списку
   # Виводимо список у тілі функції
   print('ChangeList.L = ', L)
# Створюємо список у основній програмі
LL = [ [2, 3.5], True, 'abcd' ]
# Виводимо список LL для контролю
print('LL = ', LL)
# Викликаємо функцію та передаємо їй список LL
ChangeList(LL)
# Виводимо список LL після виклику функції
print('LL = ', LL)
```

Результат роботи програми
LL = [[2, 3.5], True, 'abcd']
ChangeList.L = [777, (2, 3), 'abcd']
LL = [777, (2, 3), 'abcd']

Як видно з результату, зміна списку в тілі функції призводить до зміни цього списку в коді, який викликає функцію. Це означає, що список передається за посиланням.

Аналогічний результат буде отримано й для словника.

Взагалі якщо в функцію передається аргумент (об'єкт), який є змінним (список, словник), то зміна цього аргументу всередині функції призведе до зміни аргументу в коді, що викликає. Це стосується лише зміни складових елементів об'єкта (окремий елемент словника чи списку). Якщо спробувати змінити об'єкт за його спільним ім'ям (за допомогою операції присвоєння =), то змін у коді, що викликає, не буде (передається посилання на об'єкт, а саме посилання змінити не можна).

У випадках, коли потрібно заборонити зміну об'єкта списку всередині функції, можна використовувати такі способи, як описано нижче.

Спосіб 1. Під час передачі списку у функцію можна конвертувати цей список у кортеж за допомогою операції tuple(). Після цього змінити список у тілі функції не вдасться.

Спосіб 2. Під час передачі списку в функцію можна використовувати операцію зрізу [:] (тобто в функцію фактично відправляється копія списку-параметру).

3.7 Використання функцій як об’єктів

Так як функція в Python є об'єктом, її можна присвоїти іншій змінній, як і будь-якому іншому об'єкту:

```
def hello(name):
   return f'Hello {name}.'

say = hello
```

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
```

Рядок say = hello не викликає функції. Для об'єкту функції, на який посилається hello, створюється друге ім'я say, що також вказує на нього. Тепер можна виконати об'єкт базової функції hello, викликавши say:

```
say('World')
# 'Hello World.'
```

Функціональні об'єкти та їхні імена є двома різними речами. Ось ще один доказ: можна видалити вихідне ім'я функції Hello. Інше ім'я say, як і раніше, вказує на базову функцію hello і її можна викликати через функцію say:

```
del hello
hello('World')
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# NameError: name 'hello' is not defined

say('World')
# 'Hello World.'
```

Python приєднує рядковий ідентифікатор до кожної функції під час створення. Ви можете отримати доступ до цього внутрішнього ідентифікатора за допомогою атрибуту __name__ або __qualname__:

```
say.__name__
'hello'
say.__qualname__
'hello'
```

Хоча атрибут __name__ функції say() все ще hello, це не вплине на те, як можна отримати доступ до неї з коду. Цей атрибут є просто засобом налагодження. Змінна, що вказує на функцію та ім'я самої функції дві різні речі. Якщо видалити обидва імені функції – say та hello – область пам'яті, зайнята кодом функції, буде вивільнена.

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
…(тіло функції)
func() # Виклик об’єкту
func.attr = value # Приєднання атрибуту до об’єкту
```

Оскільки функції є об'єктами, їх можна передавати як аргументи іншим функціям. Ось функція greet, яка форматує рядкове значення привітання, використовуючи переданий до неї об'єкт-функцію, а потім його друкує:

```
def f(s):
```

```
    return s # повертаємо строку-аргумент
def greet(func):
    greeting = func('Привіт! Я - програма Python')
    print(greeting)
greet(f)
```

3.8 Анонімні функції, інструкція lambda

Анонімні функції можуть містити лише один вираз, але й виконуються вони швидше. Анонімні функції створюються за допомогою інструкції Lambda. Крім цього, їх не обов'язково присвоювати змінній, як ми робили інструкцією def func(). Приклад:

```
func = lambda x, y: x + y
func(1, 2)
3
func('a', 'b')
'ab'
(lambda x, y: x + y)(1, 2)
3
(lambda x, y: x + y)('a', 'b')
'ab'
```

На відміну від звичайної функції, lambda функції не потрібна інструкція return, а в іншому ця функція поводиться так само.

3.9 Приклади

1) Об’ява і виклик функції hello_world

```
def hello_world():
        print('Hello, World!')
# Виклик функції
hello_world()
```

2) Створення функції з одним параметром

```
def print_numbers(limit):
        for i in range(limit): print(i)
# Виклик функції print_numbers, її формальний параметр limit
# при виклику замінюють фактичним параметром 10
print_numbers(10)
```

3) Створення та виклик функції

```
def hello(name):
# Якщо name - пусте, виходимо з функції
        if not name: return
        print('Hello, ', name, '!', sep='')
# Виклик функції
```

```
hello('Alex'); hello(''); hello('Python')

4) Виклик функції може бути частиною виразу
def add(a, b):
        return a + b
def sub(a, b):
        return a - b
print(add(2, 3) + sub(2, 3)) # => print((2 + 3) + (2 - 3))

5) Варианти виклику функції і присвоєння значень параметрам
def info(object, color, price):
        print('Об’єкт:', object)
        print('Колір:', color); print('Ціна:', price)
        print()
# Виклик функції, передача параметрів в прямому порядку
info('ручка', 'синій', 1)
# передача параметрів у довільному порядку
info(price=5, object='чашка', color='помаранчевий')
# можна змішувати обидва способи, але спочатку
# розташовують параметри, які передають в прямому порядку
info('кава', price=10, color='чорна')

6) Функція може приймати будь-яку кількість аргументів чи не приймати їх зовсім. Також поширені функції з довільним числом аргументів, функції з позиційними і іменованими аргументами, обов'язковими і необов'язковими.
def func(a, b, c=2): # c - необов'язковий аргумент
   return a + b + c
 func(1, 2) # a = 1, b = 2, c = 2 (за замовчуванням)
5
 func(1, 2, 3) # a = 1, b = 2, c = 3
6
 func(a=1, b=3) # a = 1, b = 3, c = 2
6
 func(a=3, c=6) # a = 3, c = 6, b не визначений - виникне помилка

7) Використання вкладених функцій
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

8) Отримання доступу до глобальної змінної
```
var = 'глобальна змінна'
def function():
    print(var)
function()
```

Результат
Глобальна змінна

9) використання локальної змінної
```
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

10) Використання об’яви global.
Об’ява 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
```

11) Приклад використання локальних і глобальних змінних

```
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
```

12) Використання об’яви nonlocal.

```
def outer_function():
        var = 8 # створення локальної змінної var
        def inner_function():
# вказує на необхідність використання зовнішньої функції
                nonlocal var
                print(var) # 8
                var=10
print(var) # 8
inner_function() # виклик внутрішньої функці
print(var) # 10

var = 0 # створення глобальної змінної var
print(var) # 0
outer_function()
print(var) # 0
```

Результат

```
0
8
8
10
0
```

3.10 Практична робота

Мета роботи: ознайомитися з принципами побудови функцій користувача на мові Python, з використанням локальних і глобальних змінних, використанням анонимних функцій.

Завдання

1. Вивчити теоретичні основи організації та використання функцій на мові Python. Опрацювати приклади.
2. Виконати завдання 1-3 у відповідності з варіантом.
3. Скласти звіт і захистити його по роботі.

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

Варианти завдань

Завдання 1. Обчислити h з використанням функцій a, b, c при заданому значенні x.

Таблиця 3.1 - Визначення функцій для обчислення

Вар	*h*	*a*	*b*	*c*	*x*
1	a^2+b^2-6c	x^2-e^{-x}	$lnx+\sqrt{x}$	cos^2x+x^5	5,4
2	$c^2+8b+10a$	$sin^2x+x^{1/4}$	$tgx-8x^3$	$x^4+2sinx^2$	1,2
3	$3a^2+4b-8$	$3x-2cos^3x$	$lnx+2e^x$	$x^{1/3}+4x-1$	0,3
4	a^3+b^2-8c	sin^3x+x^4	$\sqrt{x}-lnx$	$4x-5x^3$	1,7
5	$6b^3+4c-2$	$tgx+e^{2x}$	x^2-6x^3	$1/x-2lnx$	4,1
6	$a^2+b^2+c^2$	$e^x+e^{2x}+4$	$x-sin^3x$	x^2/cos^3x	2,4
7	$5b^3-2a+c$	$tgx-2x$	$\sqrt{x}-sinx$	$x^3/7$	5,5
8	$4a^2+5b^2$	$cosx+2x$	$x^4-2x/5$	$2x-5$	4,6
9	$3ab-4c$	sin^2x+5	$cosx^5$	$x^{1/3}+tgx$	1,6
10	c^2+5a^3-b	cos^3x-6x	$-4x^3+lnx$	$e^{2x}+4cosx$	4,6
11	$2a+4c-b^4$	e^x-2lnx	$2x-5/x$	x^5-2lnx	3,9
12	$a^2+b^2+c^2$	$2/x+x^3$	lnx^2-4x	$tgx-sin2x$	4,1
13	$(a+b)^2$	$lnx+2e^x$	$tgx+e^{2x}$	x^2-e^{-x}	3,4
14	$2ac-3cb$	$1/x-2lnx$	$cosx+2x$	$sin^2x+x^{1/4}$	1,9
15	$5c+2a^4$	x^2-2/x	$(2-x)/6$	cos^3x-2x	2,3
16	$a+b+c$	$lnx/2x$	x^3-4x	$tgx-2x$	4,2
17	$2a+3b+4c$	x^2+x^3	$lnx-x^4$	$cos^2(x-4)$	2,8
18	$a^2+b^3+c^4$	$sin^2x+x^{1/4}$	x^3+4x	e^x+2lnx	1,3
19	$a+2b+3c$	$2x-x^{1/4}$	$\sqrt{x}-2cosx$	$tgx-4x$	3,1
20	$2(a+b)-c^4$	$(x^3-x/2)^3$	$lnx-e^{2x}$	$\sqrt{x+x^3}$	2,4
21	c^2-b^3	$2x+sinx^4$	$sin(x-lnx)$	lnx^2+2x	1,1
22	$3a-4cb$	$2cosx^3$	$tgx/4$	$x/5$	3,1
23	c^5-2ab	$1/2sin^3x$	sin^6x/x^3	$x-4sin2x$	1,8
24	$6a+3b^3+c$	$cosx^x+2x$	$sin2x+tgx$	$lnx-e^{-x}$	2,1

Вар	h	a	b	c	x
25	$4abc$	$x^x-\sin x^3$	$x/2-x^5$	$2x-\sin^3 x$	4,1
26	$a^2+(b-c)^{5/3}$	$2x^{1/3}+1$	$\sin(x^2+4)$	$\ln\cos^3 x$	5,3
27	$(a+4b)^{1/3}-c^2$	$tg(2x)/4$	$\cos x^2/x^{1/5}$	e^{-2x+1}/x^2	3,8
28	$a^{1/3}+(b^3-c)$	$x+2^{3x}$	$\ln\sin^3 4x$	$\arcsin^2 x$	4,2
29	$b^3+(a-4c)^{1/5}$	$5^{3x}/(3x-1)$	$e^{-5x}+4/x$	$\cos(x^{1/3})$	2,6
30	$c^{1/5}-(b+3a)^2$	$\sqrt{\lvert x+2\rvert}+e$	$\cos x+x^2$	$arctg(x^3)$	1,3

Завдання 2. Знайти суму послідовности значень $y(x_i)=\frac{f_1(x_i)}{f_2(x_i)}$, де $a \le x_i \le b$; $\Delta x = c$; $x_i = a + \Delta x \cdot i$; $i = 0,1,\ldots$. Варіанти завдань наведені в таблиці 3.2.

Таблиця 3.2 – Варианти завдань для обчислення

Вар.	f_1	f_2	a	b	c
1	$3x-1$	$e^{-1/x}+x/(x+1)$	3	5	0,5
2	x^3-3x^2	x^4+2x^2+3	1	3	0,2
3	$e^{-x}+4x$	$\sqrt{(1+x^3+4x^2)}$	0,6	4,2	0,3
4	$\sin^2(x+4x^3)$	$(x+2x^3)\sqrt{x}$	0,5	4,8	0,2
5	$x\sin x^3-\ln 2x$	$arctgx/4+e^{-x+2}$	2	6,3	0,4
6	$x^4-\cos x$	$tgx+2x$	1	5	0,5
7	$2x+\sin^2 x$	$\sqrt{(2x+\ln x)}$	5	8	0,3
8	$\ln(4x+8)$	$e^{-x}+\sin 2x$	1	4	0,2
9	$x^3\ln(2x)$	$4x^2+6x^3-2$	0,5	6	0,3
10	$x^2+\sin^3 x$	$\cos 3x+e^{-2x}$	-2	3	0,4
11	xe^{-x}	$\sin 4x+x^3$	1,5	5	0,3
12	$\sqrt{x^2+1}$	$arctgx/5+2x$	0,6	4	0,2
13	$x^2/(3x+2)$	$\sin^2(\pi x+1)$	0,5	5,2	0,3
14	$\sqrt{2+x^3}$	$3^x/(x-2)$	1,2	6,3	0,4
15	$x^{3x+1}+8x$	$\lvert x-8\rvert+\sin x$	4	7,5	0,3
16	x^4+e^{x+3}	$x\,arctg(x/3)$	2	6,4	0,2
17	$\ln^2(x+4)$	$\sin^3(x/5)$	1	6,8	0,3
18	$e^{x-2}+x^3$	$x-\ln\lvert x-1\rvert$	0	4	0,4
19	$2\cos(x+3)$	$4x^2/(3+x^3)$	2	5	0,3
20	$\sqrt{1+x^4})$	$tg^2(x+4)-e^{-x}$	1	6	0,4
21	$3+2\sin^2(x-3)$	$4+x/10$	2	7	0,5
22	$\ln(1(1+2^x)$	$\sin^2(4x+1)$	1,5	6,8	0,4
23	$\sqrt{\lvert x\rvert}+e^{-x}$	$5arctg(4x)$	2	7	0,5
24	$\arcsin(x+2)$	$3(x-4)/(x^2+1)$	3	8	0,2

Вар.	f_1	f_2	a	b	c
25	$e^{\|x+2\|}$	$ln^2(x+4)$	-2	6	0,3
26	$(4-x)cos2x$	$\sqrt{\|x+1\|}+e^{-3x}$	1	7	0,4
27	$\sqrt{\|x\|+2^x}$	$sinx^4-4$	-2	5	0,2
28	$2^{x+4}+cos^2x$	$ln\|x+8\|$	-4	2	0,5
29	$(x+2)/sin^3x$	$\sqrt{x+tg^2x}$	1	4	0,3
30	$e^{x+3}+4x^2$	$arcsinx^3$	2	5	0,2

Завдання 3. Виконайте за варіантом:

1. Створіть просту функцію simple_func, а потім виведіть на екран її тип.
2. Напишіть функцію hello_friend, яка приймає як аргументи ім'я та прізвище користувача і повертає рядок у форматі «Привіт, {name} {surname}!». Викличте функцію та виведіть результат на екран. Передбачте використання іменованих аргументів, їх значення за замовчуванням. Продемонструйте різні варианти використання цієї функції.
3. Створіть функцію користувача, яка приймає довільну кількість аргументів і виводить їх на екран. Використовуйте цикл for, щоб вивести елементи отриманого списку. Викличте функцію, передавши їй як значення ціле число, речове число, рядок і порожній список.
4. Створіть функцію користувача, що приймає довільну кількість іменованих аргументів і виводить їх на екран у форматі «key -> value». Для виведення елементів отриманого словника використовуйте цикл for. Викличте функцію, передавши їй як значення ціле число, речове число, рядок і порожній список.
5. Створіть і викличте функцію-матрешку my_func_1, що складається з чотирьох вкладених один в одного визначень аналогічних функцій. Кожна функція повинна виводити повідомлення у форматі 'In my_func_{номер функції}.', а також містити визначення та виклик наступної вкладеної функції (в останній функції ця частина буде відсутня).
6. Створіть рекурсивну функцію recursive_func(text), яка виводитиме символи рядка, що їй передається, на екран через пробіл.
7. Напишіть функцію, яка розраховує вартість поїздки на таксі в залежності від відстані. Як аргументи функція повинна приймати іменовані параметри: км – відстань поїздки в кілометрах, мін_ціна – базовий тариф, що включає подачу таксі та перші три кілометри шляху, ціна_за_км – ціна за кожен кілометр, починаючи з четвертого. Розрахуйте та виведіть на екран вартість поїздки за 17.5 км, якщо за замовчуванням базовий тариф складає 2 у.о., а ціна за кілометр понад мінімум – 0.3 у.о.
8. Напишіть функцію, яка генеруватиме випадковий пароль. У паролі має бути від 8 до 15 символів Юнікоду з діапазонів 48-57 (цифри), 65-90 (літери латинського алфавіту у верхньому регістрі) та 97-122 (літери латинського алфавіту в нижньому регістрі). Згенеруйте та виведіть на екран три паролі.
9. Напишіть найпростішу функцію-калькулятор, яка виконуватиме з двома переданими їй числами наступні дії: додавання, віднімання, множення та поділ

із зазначеною точністю. Відповідно, функція повинна приймати два позиційні числові аргументи для чисел, один іменований рядковий аргумент op (за замовчуванням функція повинна виконувати операцію додавання) і один іменований числовий аргумент prec для необхідної точності результату (за замовчуванням три знаки після коми). У разі поділу на нуль функція повинна повертати NaN, а при спробі виконання непередбаченої операції повідомлення про помилку «Непідтримуваний тип операції!». Виведіть на екран результати викликів функції з точністю до сотих для числових виразів: -13.756 + 59.291, 599 - 783, -7/55, 7/0 та 57.75*33.2.

10. Напишіть функцію для знаходження найбільшого спільного дільника довільної кількості чисел. Потім знайдіть та виведіть на екран найбільший спільний дільник чисел 165, 435 та 300.

11. Інтернет-магазин надає послугу експрес-доставки для частини своїх товарів за ціною $10,95 за перший товар у замовленні і $2,95 за всі наступні. Напишіть функцію, яка приймає як єдиний параметр кількість товарів у замовленні і повертає загальну суму доставки. В основній програмі повинні здійснюватись запит кількості позицій у замовленні у користувача та відображатись на екрані сума доставки.

12. Написати функцію з ім'ям isInteger, що визначає, чи представляє введений рядок ціле чисельне значення.

13. Просте число є число, більше одиниці, яке без залишку ділиться лише на саму себе і одиницю. Напишіть функцію для визначення того, чи є введене число простим. Значення, що повертається, має бути або True, або False.

14. Написати функцію з ім'ям nextPrime, яка знаходить і повертає перше просте число, більше введеного числа n. Саме число n повинно передаватися в функцію як єдиний параметр.

15. Напишіть функцію, яка визначає кількості днів у конкретному місяці. Ваша функція має приймати два параметри: номер місяця у вигляді цілого числа в діапазоні від 1 до 12 та рік, що складається з чотирьох цифр. Переконайтеся, що функція коректно опрацьовує лютий високосного року.

3.11 Контрольні запитання

1. Що таке функція? Навіщо їх використовують?
2. Як створити функцію у мові Python?
3. Охарактеризуйте локальні та глобальні змінні. В чому їх відмінність?
4. Навіщо використовувати параметри під час визначення функції? Які типи параметрів існують?
5. Що таке модулі та пакети?
6. Які бувають способи підключення модулів та пакетів?
7. Що таке анонімні функції та інструкція lambda?
8. Чи можна використати модуль як самостійну програму?

4 Використання циклів та функцій користувача для роботи зі строками, списками, тьюплами, словниками та множинами.

Python надає кілька способів перебору списків; кожен має свої переваги та недоліки.

4.1. Простий цикл for

Використання циклу Python forє одним із найпростіших методів для повторення списку або будь-якої іншої послідовності (наприклад, кортежів, множин або словників).

Цикли python for є потужним інструментом, тому програмістам важливо розуміти їх універсальність. Ми можемо використовувати їх для запуску операторів, що містяться в циклі, один раз для кожного елемента в списку. Наприклад:

```
fruits = ["Apple", "Mango", "Banana", "Peach"]
for fruit in fruits:
  print(fruit)
```

Результат:

```
Apple
Mango
Banana
Peach
```

Тут цикл for було використано для виводу кожного елементу списку. В прикладі цикл викликав функцію print() чотири рази, кожного разу друкуючи поточний елемент у списку, тобто назву фрукта.

4.2. Генератори списку

Генератор списків подібний до циклу for, однак він дозволяє створити список і перебирати його в одному рядку. Завдяки своїй абсолютній простоті цей метод вважається одним із найзручніших способів ітерації списків Python. Приклад генератора списку:

```
fruits = ["Apple", "Mango", "Banana", "Peach"]
[print(fruit + " juice") for fruit in fruits]
```

Результат:

```
Apple juice
Mango juice
Banana juice
```

Peach juice

Як бачите, ми створили список фруктів так само, як і в попередньому прикладі. Однак цього разу ми використали генератор списку, щоб зробити дві речі: додати слово «сік» у кінець елемента списку та надрукувати його.

Функціональність list comprehension надає більш короткий та лаконічний синтаксис для створення списків на основі інших наборів даних. Вона має наступний синтаксис:

newlist = [expression for item in iterable (if condition)]

Синтаксис list comprehensionскладається з наступних компонентів:

iterable: джерело даних, що перебирається, в якості якого може виступати список, безліч, послідовність, або навіть функція, яка повертає набір даних, наприклад,range()

item: елемент, що витягується з джерела даних

expression: вираз, який повертає певне значення Це значення потім потрапляє в список, що генерується

condition: умова, якій повинні відповідати елементи, що витягуються з джерела даних. Якщо елемент НЕ задовольняє умову, він не вибирається. Необов'язковий параметр.

Розглянемо приклад. Припустимо, що треба вибрати зі списку всі числа, які більше 0. Варіант з використанням циклу for і перевірки чергового елементу списку:

```
numbers = [-3, -2, -1, 0, 1, 2, 3]
positive_numbers = []
for n in numbers:
    if n > 0:
        positive_numbers.append(n)

print(positive_numbers)     # [1, 2, 3]
```

Тепер змінимо цей код, застосувавши list comprehension (значно коротше):

```
numbers = [-3, -2, -1, 0, 1, 2, 3]
positive_numbers = [n for n in numbers if n > 0]
print(positive_numbers)     # [1, 2, 3]
```

Вираз [n for n in numbers if n > 0] обирає зі списку numbers кожен елемент до змінної n, якщо n більше 0, і записує значення n в результуючий список.

4.2.1 Джерело даних iterable

Як джерело даних iterableможе використовуватися будь-який об'єкт, що перебирається, наприклад, інший список, словник і т.д. Наприклад, функція range() повертає все числя від нуля до зазначеного порога (але не включаючи цей самий поріг):

```
numbers = [n for n in range(10)]
print(numbers)      # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Наприклад, така конструкція застосовується, щоб створити зі словника список. Виберемо зі словника всі ключі:

```
dictionary = {"red": "красный", "blue": "синий", "green": "зеленый"}
words = [word for word in dictionary]
print(words)    # ['red', 'blue', 'green']
```

Повернення результату

Параметр expression являє собою вираз, який повертає деяке значення. Це значення потім поміщається в список, що генерується. У прикладах вище це був поточний елемент, який витягується із джерела даних:

```
numbers = [-3, -2, -1, 0, 1, 2, 3]
new_numbers = [n for n in numbers]
print(new_numbers)      # [-3, -2, -1, 0, 1, 2, 3]
```

Так, в даному випадку параметр expression представляє елемент, що безпосередньо витягується зі списку numbers n. Але це можуть бути і складніші значення. Наприклад, повернемо подвоєне значення числа:

```
numbers = [-3, -2, -1, 0, 1, 2, 3]
new_numbers = [n*2 for n in numbers]
print(new_numbers)      # [-6, -4, -2, 0, 2, 4, 6]
```

Тут expressionpe презентує вираз n*2

Це можуть бути і складніші вирази:

```
numbers = [-3, -2, -1, 0, 1, 2, 3]
new_numbers = [n*2 if n>0 else n for n in numbers]
print(new_numbers)      # [-3, -2, -1, 0, 2, 4, 6]
```

Тут параметр expression представляє вираз n*2 if n>0 else n. Тобто буде повернкто значення n * 2, якщо n > 0, інакше буде повернуто n.

Можна expression проводити різні трансформації з даними. Наприклад, повернемо також із словника значення за ключом:

```
dictionary = {"red": "червоний", "blue": "синій", "green": "зелений"}
words = [f"{key}: {dictionary[key]}" for key in dictionary]
print(words)    # ['red: червоний', 'blue: синій', 'green: зелений']
```

4.2.2 Умова в генераторі циклу

Умова - параметр condition визначає фільтр для вибору елементів із джерела даних. Застосуємо умову для конкретизації вибірки, наприклад, виберемо лише парні числа:

```
numbers = [n for n in range(10) if n % 2 == 0]
print(numbers)      # [0, 2, 4, 6, 8]
```

Виберемо тільки ключі зі словника, довжина яких більше 3:

```
dictionary = {"red": "червоний", "blue": "синій", "green": "зелений"}
words = [f"{key}: {dictionary[key]}" for key in dictionary if len(key) > 3]
print(words)    # ['blue: синій', 'green: зелений']
```

4.3. Використання циклу for з діапазоном range()

Іншим методом циклічного перегляду списку Python є функція range() разом із циклом for. Функція range() генерує послідовність цілих чисел із наданих початкового та кінцевого індексів. Індекс стосується позиції елементів у списку. Перший елемент має індекс 0, другий елемент списку — 1 і так далі. Синтаксис функції діапазону такий:

```
range(start, stop, step)
```

Аргументи start і step необов'язкові; потрібен тільки аргумент stop. Крок визначає, чи пропускаєте ви елементи списку; за замовчуванням встановлено значення 1, тобто жоден елемент не пропускається. Якщо ви вкажете лише один параметр (тобто індекс зупинки), функція створює об’єкт діапазону, що містить усі елементи від 0 до stop-1 .

Ось приклад, який надрукує назву фрукта та його індекс у списку:

```
fruits = ["Apple", "Mango", "Banana", "Peach"]

# Constructs range object containing elements from 0 to 3
for i in range(len(fruits)):
  print("The list at index", i, "contains a", fruits[i])
```

Це призводить до наступного результату:

```
The list at index 0 contains a Apple
The list at index 1 contains a Mango
```

```
The list at index 2 contains a Banana
The list at index 3 contains a Peach
```

Дещо іншим підходом було б друкувати лише деякі фрукти на основі їх індексу. Ми б зробили це, вказавши початковий і кінцевий індекс для циклу for за допомогою функції range():

```
fruits = ["Apple", "Mango", "Banana", "Peach"]

# Constructs range object containing only 1 and 2
for i in range(1, 3):
  print(fruits[i])
```

Результат:
```
Mango
Banana
```

4.4. Використання циклу for з enumerate()

Іноді необхідно знати індекс елемента списку, з яким виконують якійсь дії. Вирішити це питання дозволяє функція enumerate(); вона додає лічильник і повертає його як щось, що називається «об’єкт перерахування». Цей об’єкт містить елементи, які можна розпакувати за допомогою простого циклу Python for. Таким чином, об’єкт enumerate зменшує накладні витрати на підрахунок кількості елементів у простій ітерації.

Розглянемо приклад:

```
fruits = ["Apple", "Mango", "Banana", "Peach"]

for index, element in enumerate(fruits):
  print(index, ":", element)
```

Запуск наведеного вище коду повертає цей список елементів та їхніх індексів:

```
0: Apple
1: Mango
2: Banana
3: Peach
```

4.5. Цикл for з лямбда-функцією

Функція Python lambda — це анонімна функція, у якій математичний вираз обчислюється, а потім повертається. Як наслідок, lambda може використовуватися як функціональний об’єкт. Давайте подивимося, як використовувати lambda, коли ми переглядаємо список.

Ми створимо цикл for, щоб перебирати список чисел, знаходити квадрат кожного числа та зберігати або додавати його до списку. Розглянемо приклад:

```
lst1 = [1, 2, 3, 4, 5]
lst2 = []

# Lambda function to square number
temp = lambda i:i**2

for i in lst1:
   # Add to lst2
   lst2.append(temp(i))

print(lst2)
```

Ми використовуємо lambda, щоб перебирати список lst1 і знаходити квадрат кожного значення. Для повторення використовується цикл for. Кожне ціле число передається за одну ітерацію; функція append() зберігає його квадрат у списку lst2.

Цей код ефективнішим за допомогою функції map():

```
lst1 = [1, 2, 3, 4, 5]

lst1 = list(map(lambda v: v ** 2, lst1))

print(lst1)
```

Після застосування наданої лямбда-функції до кожного елемента в заданому спуску map() створює об’єкт карти результатів (який є ітератором).

Обидва ці коди дають однакові результати:

```
[1, 4, 9, 16, 25]
```

4.6. Використання циклу while

Ми також можемо перебирати список Python за допомогою циклу while. У коді нижче умовою для циклу while є довжина списку; лічильник івстановлюється на нуль, потім він додає 1 кожного разу, коли цикл друкує один елемент у списку. Коли I стає більше, ніж кількість елементів у списку, цикл while завершується. Розглунемо приклад коду:

```
fruits = ["Apple", "Mango",  "Banana", "Peach"]

i = 0
```

```
while i < len(fruits):
  print(fruits[i])
  i += 1
```

Результат – роздрукування усіх елементів списку fruits:

```
Apple
Mango
Banana
Peach
```

4.7 Врахування продуктивності під час розширення списків

Коли ви створюєте список, Python виділяє достатньо місця для зберігання наданих елементів. Він також виділяє додатковий простір для розміщення майбутніх елементів. Коли ви використовуєте додатковий простір, додаючи нові елементи до цього списку за допомогою .append(), .extend() або .insert(), Python автоматично створює місце для додаткових нових елементів.

Цей процес відомий як зміна розміру, і хоча він гарантує, що список може приймати нові елементи, він вимагає додаткового процесорного часу та додаткової пам’яті. чому Що ж, щоб збільшити наявний список, Python створює новий з місцем для ваших поточних даних і додаткових елементів. Потім він переміщує поточні елементи до нового списку та додає новий елемент або елементи.

Розгляньте наступний код, щоб дослідити, як Python динамічно збільшує список:

```
from sys import getsizeof
numbers = []
for value in range(100):
   print(getsizeof(numbers))
   numbers.append(value)
...
56
88
88
88
88
120
120
120
120
184
184
...
```

У цьому фрагменті коду спочатку імпортуємо getsizeof() із модуля sys. Ця функція дозволяє отримати розмір об'єкта в байтах. Потім визначаємо порожній список для зберігання довжини списку.

Усередині циклу for отримуємо та друкуємо розмір об’єкта списку в байтах. Перша ітерація показує, що розмір порожнього списку становить 56 байт, що є базовим розміром кожного списку в Python.

Потім метод .append() додає нове значення до цього списку. Зверніть увагу, як значення довжини зростає до 88 байт. Це базовий розмір плюс 32 додаткові байти (56 + 4 × 8 = 88), які представляють чотири 8-байтові покажчики або слоти для майбутніх елементів. Це означає, що Python пішов вперед і виділив простір для чотирьох елементів, коли ви додали перший елемент.

По ходу виконання циклу довжина списку зростає до 120 байт, що становить 88 + 4 × 8 = 120. Цей крок виділяє простір для ще чотирьох елементів. Тому на екрані значення 120 отримано 4 рази.

Якщо ви простежите за результатами циклу, то помітите, що наступні кроки додають місце для восьми додаткових елементів, потім для дванадцяти, потім для шістнадцяти і так далі. Щоразу, коли Python змінює розмір списку, йому доводиться переміщувати всі елементи в новий простір, що займає значний час.

На практиці, якщо ви працюєте з невеликими списками, загальний вплив цієї внутрішньої поведінки буде незначним. Однак у критичних для продуктивності ситуаціях або коли ваші списки великі, ви можете використовувати більш ефективні типи даних, наприклад collections.deque. Наприклад, метод .append() має часову складність O(1), що означає, що додавання елемента до списку займає постійний час. Однак, коли Python має збільшити список, щоб звільнити місце для нового елемента, ця продуктивність буде трохи нижчою.

4.8 Використання функцій map( ) або filter()

В деяких випадках для роботи зі списками та тьюплами використовуються функції map() або filter(), часто разом з лямбда-функціями.

Функція map() повертає об’єкт map (який є ітератором) результатів після застосування даної функції до кожного елемента даного ітерованого об’єкту (списку, кортежу тощо). Тобто функція map() використовується, щоб застосувати необхідну функцію до кожного елементу списку або тьюплу.

Синтаксис:

map(fun, iter)

Параметри:

fun : це функція, якій map передає кожен елемент заданого iterable.

iter : це iterable, який має бути відображено.

Розглянемо приклад (перетворення кілометрів до футів у списку відстаней).

```
# Створення списку відстаней у кілометрах
```

```
kilometer = [39.2, 36.5, 37.3, 37.8]

# Перетворення відстаней у фути з використанням map()
feet = map ( lambda x: float (3280.8399) * x, kilometer)

# Перетворення feet (map-об’єкт) до списку
Feet = list(feet) # [128608.92408000001, 119750.65635, 122375.32826999998, 124015.74822]
```

Звичайно, в цьому простому прикладі перетворення списку відстаней можливе й за допомогою генератора списку.

Розглянемо інший приклад з використанням функції filter():

```
# Перетворимо елементи списку відстаней у цілі числа
feet = list ( map (int, feet))

# Оберемо зі списку тількі непарні значення
uneven = filter( lambda x: x%2, feet)

# Переглянемо тип об’єкту uneven і перетворимо його на список
print(type(uneven), list(uneven)) # <class 'filter'> [122375, 124015]
```

Метод filter() фільтрує задану послідовність за допомогою функції, яка перевіряє кожен елемент у послідовності на істину чи ні.

Синтаксис: filter(function, sequence)

Параметри:

- **function:** функція, яка перевіряє, чи є кожен елемент послідовності істинним чи ні;
- **sequence:** послідовність, яку потрібно відфільтрувати, це можуть бути набори, списки, кортежі або контейнери будь-яких ітераторів;
- **повертає:** ітератор, який уже відфільтровано.

Приклад використання filter():

```
# функція, яка фільтрує голосні
def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else:
        return False

# послідовність літер
sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']
```

```
# використовуємо функцію fun для фільтрації
filtered = filter(fun, sequence)

print('The filtered letters are:')
for s in filtered:
   print(s)
```

У цьому прикладі ми використовуємо функцію фільтра разом із функцією fun(), щоб відфільтрувати голосні зі списку Python .

4.9 Робота з функціями з довільним списком параметрів

У мові Python є можливість передавати до функції довільну кількість аргументів. Це забезпечується спеціальним синтаксисом. Довільну кількість аргументів можна описувати в інструкції def функції одним із двох способів:

- з використанням кортежу (*);
- за допомогою словника (**).

Відповідно можна викликати функцію, яка отримує довільну кількість параметрів.

Для змінної кількості аргументів у мові Python підтримуються такі режими зіставлення:

- режим отримання функцією змінної кількості аргументів Fn(*sequence) як кортежу;
- режим отримання функцією змінної кількості аргументів Fn(**name) як словника;
- режим передачі змінної кількості аргументів як кортежу: Fn(*args);
- режим передачі змінної кількості аргументів як словника: Fn(**args).

Розглянемо приклад. Функція Fn() отримує змінну кількість аргументів як кортеж з ім'ям sequence. Під час такої передачі перед ім'ям кортежу ставиться символ *.

```
# Аргументи. Режими зіставлення.
# Передача аргументу на функцію - змінна кількість аргументів
# Функція, яка отримує змінну кількість аргументів та виводить їх на екран
    def Fn(*sequence):
        # у тілі функції виводиться значення sequence
        print('sequence = ', sequence)
        return

# Виклик функції з 3 аргументами
Fn(3, 'abcd', [2, True, 3.888]) # sequence = (3, 'abcd', [2, True, 3.888])

# Виклик функції з одним аргументом - словником
Fn({1: 'A', 2: 'B', 3: 'C'}) # sequence = ({1: 'A', 2: 'B', 3: 'C'},)
```

Результат роботи програми

```
sequence = (3, 'abcd', [2, True, 3.888])
```

```
sequence = ({1: 'A', 2: 'B', 3: 'C'},)
```

Розглянемо ще один приклад. Функція Fn() отримує змінну кількість аргументів. У функцію аргументи передаються за ім'ям створених об'єктів (x, y, z).

```
# Функція, яка отримує змінну кількість аргументів та виводить їх на екран
def Fn(*sequence):
    # у тілі функції виводиться значення sequence
    print('Fn.sequence=', sequence)
    return

# Виклик функції з трьома аргументами, які мають імена
x = 2
y = 3
z = 'abcd'
Fn(x, y, z) # Fn.sequence = (2, 3, 'abcd')
```

Результат виконання програми

```
Fn.sequence = (2, 3, 'abcd')
```

4.10 Використання списку як стека або черги

Можна використовувати список Python для емуляції структури даних стека або черги за допомогою методів .append() і .pop(). Наприклад, щоб імітувати стек або структуру даних «останній прибув, першим вийшов» (LIFO), ви можете використовувати метод append() для розміщення елемента на вершині стека. Подібним чином ви можете використовувати метод pop() без аргументів, щоб витягнути елементи з верхньої частини стека:

```
stack = []
stack.append("Copy")
stack.append("Paste")
stack.append("Remove")
print(stack) # ['Copy', 'Paste', 'Remove']

print(stack.pop()) # 'Remove'
print(stack.pop()) # 'Paste'
print(stack.pop()) # 'Copy'
print(stack) # []
```

У цьому прикладі стек представлено за допомогою списку. Стек містить дії, які можна скасувати. Ви починаєте зі створення порожнього списку під назвою stack. Потім ви надсилаєте гіпотетичні дії в стек за допомогою методу append(), що додає дії в правий кінець списку.

Метод pop() повертає дії, щоб їх можна було повторити. Цей метод також видаляє дії з правого кінця списку відповідно до порядку LIFO, який розрізняє структуру даних стека.

Для емуляції черги або структури даних «першим прибув, першим вийшов» (FIFO) можна теж використовувати метод append() для розміщення елементів у кінці списку, що відомо як операція постановки в чергу. Подібним чином ви можете використовувати метод pop() з індексом 0 як аргумент для повернення та видалення елементів з лівого кінця черги, яка відома як вилучення з черги:

```
queue = []
queue.append("John")
queue.append("Jane")
queue.append("Linda")
print(queue)  # ['John', 'Jane', 'Linda']
print(queue.pop(0)) # 'John'
print(queue.pop(0)) # 'Jane'
print(queue.pop(0)) # 'Linda'
```

Цей список імітує чергу людей, які, можливо, прибувають до місця, щоб отримати певну послугу. Метод append() дозволяє додавати людей у кінець черги, коли вони прибувають. Метод pop() з аргументом 0 дозволяє обробляти людей з початку черги, коли вони повинні спілкуватись зі службою. Загалом, виконується принципу FIFO, який керує чергами.

Використовуючи список Python, ви можете швидко скористатися стандартною функціональністю списку, щоб забезпечити базові операції зі стеком і чергою, такі як push, pop, enqueue і exqueue. Однак треба мати на увазі, що хоча списки можуть допомогти імітувати стеки та черги, вони не оптимізовані для таких випадків використання. Використання списку як черги особливо погане, оскільки це може зробити чергу помітно повільною.

Більш швидкими та надійними є компоненти модуля collections.

4.11 Приклади

Приклад 1. Заміна елементу списку з використанням while

```
my_list = [1, 2, 3, 5, 7, 11] # 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
[1, 2, 3, 5, 7, 11]
Введіть індекс ел-та списку (від -6 до 5):3
Введіть нове значення заданого ел-та: 15
[1, 2, 3, 15, 7, 11]

Приклад 2. Виведення квадратів чисел зі списку (цикл for)

```
my_list = [5, 1, 5, 7, 8, 1, 0, -23] # Створення списку чисел
for x in my_list:
    print(f'{x}^2={x ** 2}', end='  ')
# 5^2=25  1^2=1  5^2=25  7^2=49  8^2=64  1^2=1  0^2=0  -23^2=529
```

Приклад 3. Створення списку чисел Фібоначчі в циклі for

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

Приклад 4. Використання генератора списку

```
1) [number**2 for number in range(1, 11)] # [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

2) numbers = ["2", "9", "5", "1", "6"]
numbers = [int(number) for number in numbers] # [2, 9, 5, 1, 6]

3) integers = [20, 31, 52, 6, 17, 8, 42, 55]
even_numbers = [number for number in integers if number % 2 == 0]
print(even_numbers) # [20, 52, 6, 8, 42]
```

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)
```

5) багатовимірний список

```
[[0 for _ in range(5)] for _ in range(5)]
```

Результат:

```
[[0, 0, 0, 0, 0],
   [0, 0, 0, 0, 0],
   [0, 0, 0, 0, 0],
```

```
    [0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0]]
```

Приклад 5. Різні варіанти ітерації за списком і використання enumerate

```
1) colors = [
"red",
"orange",
"yellow",
"green",
"blue",
"indigo",
"violet"
]
for color in colors: print(color, end="  ")
# red  orange  yellow  green  blue  indigo  violet
print()
for i in range(len(colors)): print(colors[i], end="  ")
# red  orange  yellow  green  blue  indigo  violet
print()
for i, color in enumerate(colors):
    print(f"{i} is the index of '{color}'")
# 0 is the index of 'red'
# 1 is the index of 'orange'
# 2 is the index of 'yellow'
# 3 is the index of 'green'
# 4 is the index of 'blue'
# 5 is the index of 'indigo'
# 6 is the index of 'violet'

2) numbers = ["2", "9", "5", "1", "6"]
for i, number in enumerate(numbers):
    numbers[i] = int(number)
print(numbers)  # [2, 9, 5, 1, 6]
```

Приклад 6. Паралельний перегляд списків.

```
# використовуємо функцію zip, яка дозволяє вам паралельно переглядати
# кілька списків:
integers = [1, 2, 3]
letters = ["a", "b", "c"]
floats = [4.0, 5.0, 6.0]
for i, l, f in zip(integers, letters, floats):
    print(i, l, f)
#1 a 4.0
#2 b 5.0
#3 c 6.0
```

Приклад 7. Унікальні елементи списку.

```
# варіант з переглядом списку
def get_unique_items(list_object):
    result = []
    for item in list_object:
        if item not in result:
            result.append(item)
    return result
print(get_unique_items([2, 4, 5, 2, 3, 5])) # [2, 4, 5, 3]

# варіант з використанням set
def get_unique_items2(list_object):
    result = list(set(list_object))
    return result
print(get_unique_items2([2, 4, 5, 2, 3, 5])) # [2, 4, 5, 3]
```

4.11 Практична робота

Мета роботи: ознайомитися з особливостями визначення та використання строк, одновимірних та двовимірних масивів, структурною організацією масивів та способів доступу до їх елементів, передавання ціх об'єктів в функції або повернення з них.

Завдання

1. Вивчити теоретичні основи написання додатків на мові Python. Опрацювати приклади.
2. Виконати завдання 1-4 у відповідності з варіантом.
3. Скласти звіт по роботі і захистити його.

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

Варіанти

Завдання 1. Рядки.

1. Дан рядок з 20 символів. Вивести з нього на друк тільки малі літери латинського алфавіту.
2. Вивести на друк всі рядкові, а потім все прописні букви українського і латинського алфавітів.
3. У заданому рядку підрахувати частоту появи букв «a», «b».
4. Дан текст з 60 літер. Надрукувати тільки рядкові українські букви, що входять в цей текст.
5. Дана послідовність символів, що містить символ «я». Визначити порядковий номер символу «я» в послідовності.
6. Дана послідовність символів. Визначити в ній символ, який за алфавітом передує іншим.

7. Надрукувати в алфавітному порядку всі різні малі літери, що входять в заданий текст з 100 літер.
8. Визначити, чи є задана послідовність символів в рядку симетричною: читається однаково зліва направо і справа наліво.
9. Надрукувати текст, утворений символами з порядковими номерами 56, 89, 84 і 69 та текст зі зміною регістра.
10. Дано два рядки c1 і c2, що містять до 5 цифр кожна. Обчислити арифметичне вираз c3 = (c1-c2) / (c1 + c2).
11. Обчислити суми кодів всіх букв, що входять в слова SUM і ALFA. Порівняти слова і визначити, яке з них більше.
12. Надрукувати заданий текст з видаленням з нього всіх букв b, безпосередньо перед якими знаходиться буква c.
13. Є символьний змінна d, привласнити логічної змінної T значення True, якщо значення d - цифра, і значення False в протилежному випадку.
14. Якщо в заданий текст входить кожна з букв слова key, тоді надрукувати «yes», інакше - «no».
15. Дано рядок, що містить не більше двадцяти латинських букв. Всі входження «max» в ній замінити на «min». Підрахувати кількість таких замін.
16. Дано рядок, що містить сорок латинських букв. Підрахувати всі входження «abc» в рядок і їх видалити. Вивести на екран два варіанти отриманих рядків, заповнюючи утворену «дірку» наступними буквами з додаванням в кінці прогалин і залишаючи на місці видалених символів прогалини.

Завдання 2. Одновимірні масиви (вектори)

Використовуючи генератор випадкових чисел, заповнити список [a1, …, an] 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 $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ 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, …, a_{3n}] (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, сберігаючи 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,…, п-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: $x_1=\max(x_1, y_1)$; y1=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 3. Робота зі словниками

№	Завдання
1	Відомо дані про 10 подій, які відбулися починаючи з 1930 року. Подія складається з номеру, року, місяця, числа, короткого опису. Напишіть програму, яка порівнює будь-які дві події за часом та визначає, яка з подій відбулася пізніше.
2	Відомі дані про 10 моментів часу однієї доби, а саме: години (від 0 – 23); хвилини (0 – 59) та секунди (0 – 59). Напишіть програму, яка здійснює порівняння будь-яких дох моментів часу за їх умовним

	порядковим номером і визначає, який з даних моментів відбувся пізніше.
3	Відомі прізвища 10-ти співробітників ДУТ та їх адреса. Напишіть програму, яка визначає, чи працюють у даному закладі люди з прізвищем: Іванюк, Іванець, Іваненко, Іванченко або Івашко. У разі збігу надрукувати їх адреса.
4	Відомі дані 15-ти студентів 124 гр., а саме: прізвище, ім'я, по батькові; кількість пропусків; адреса та телефон. Написати програму, яка визначає студента з максимальним числом пропусків та друкує його ім'я.
5	Відомі дані 10-ти студентів ДУТ, тобто їх прізвище, ім‘я та по батькові, дата народження, адреса проживання та телефон. Написати програму, яка визначає, чи є в університеті студенти, у яких сьогодні день народження, якщо так, то надрукувати їх імена.
6	Відомі дані про вартість, та рік випуску кожної з 10 моделей легкових автомобілів компанії Volkswagen. Написати програму, яка визначає середню вартість автомобілів, починаючи з 1970 року випуску.
7	Відомі дані про масу та об'єм 10 предметів, на основі різних матеріалів. Написати програму, яка розраховує максимальну густину матеріалу.
8	Відомі дані про число опадів за кожен день місяця та температуру повітря в ці дні. Написати програму, що обчислює, яка кількість опадів випала у виді снігу, а яка – у виді дощу (вважати, що дощ йде тоді, коли температура повітря є вищою за 0° С).
9	Відомі дані про потужність та вартість двигунів 10-ти легкових автомобілів фірми Toyota. Написати програму, яка обраховує загальну вартість автомобілів, у яких потужність їх двигуна більша за 100 кінських сил.
10	Відомі дані 10-ти країн із інформацією про їх площу, населення та ту частини світу, де вони розташовані. Напишіть програму, яка визначає, чи є серед заданих країн ті, що знаходяться в Азії, Європі або в Північній Америці. У разі збігу надрукувати їх назви.
11	Відомі дані про число студентів у кожному з 10-ти навчальних закладів та тип закладу (училище, коледж або університет). Написати програму, яка розраховує загальну кількість студентів з усіх закладів.
12	Відомі дані про ціну та тираж 10-ти фахових журналів. Написати програму, яка обчислює середню вартість журналів, тираж яких менше 1000 примірників.

13	Відомі дані про число населення (в мільйонах) та площу (в тис. км^2) 10-ти країн Світу. Написати програму, яка визначає назву тієї країни, яка має максимальну густоту населення.
14	Відомі дані про оцінки 15-ти студентів 122 гр. з чотирьох дисциплін (курсів). Написати програму, яка визначає прізвище того студента, який має мінімальний середній бал по всіх предметах.
15	Відомі дані 10-ти співробітників компанії Pixar, а саме: ім'я, прізвище, зарплата, адреса, телефон. Написати програму, яка визначає: прізвища співробітника, який має найбільшу заробітну плату.

Завдання 4. Створення функції зі змінною кількістю параметрів.
Напишіть програмну реалізацію наступних завдань:
3.1 Створить власну функцію my_max (), яка приймає будь-яку кількість чисел і повертає максимальне з них.
3.2 Створіь свою функцію my_min (), яка приймає будь-яку кількість чисел і повертає мінімальне з них.
3.3 Створіь свою функцію my_avr (), яка приймає будь-яку кількість чисел і повертає середне значення для цього множини.
3.4 Створіь свою функцію my_sum (), яка приймає будь-яку кількість чисел і повертає суму чисел.

4.12 Контрольні запитання

1. Які базові операції роботи з рядками є у мові Python?
2. Як можна одержувати зрізи рядків? Якими бувають зрізи? Наведіть приклади.
3. Які методи роботи з рядками є у мові Python?
4. Що таке одновимірний масив? Для чого використовують одномірні масиви? Як їх описують в Python?
5. Як в програмі використати значення конкретного елемента одновимірного масиву?
6. Для чого в програмах використовуються двовимірні масиви? Як їх описують в Python?
7. Скільки індексів характеризують конкретний елемент двовимірного масиву?
8. Як в програмі використати значення конкретного елемента двовимірного масиву?
9. Який індекс двовимірного масиву змінюється швидше при послідовному розміщенні елементів масиву в оперативній пам'яті?

5 Створення та використання класів та об’єктів. Конструктори класів.

5.1 Загальна інформація

5.1.1 Поняття і властивості класів

Клас — це складний користувацький тип даних. Фактично це прототип, з якого створюються об’єкти. Класи забезпечують засоби групування даних і функціональних можливостей. Створення нового класу створює новий тип об’єкта, що дозволяє створювати нові екземпляри цього типу. Кожен екземпляр класу може мати атрибути, прикріплені до нього для підтримки його стану. Екземпляри класу також можуть мати методи (визначені їхнім класом) для зміни свого стану.

Властивості класів Python:

- Класи створюються за ключовим словом class.
- Атрибути - це змінні, які належать до класу.
- Атрибути завжди загальнодоступні, і до них можна отримати доступ за допомогою оператора крапка (.). Наприклад: Мій клас.Мій атрибут.

Класи, як і модулі, приховують внутрішню будову, залишаючи на поверхні лише зовнішній "інтерфейс" для використання.

Це поєднання даних та функцій всередині однієї сутності, разом із приховуванням внутрішньої будови, називається *інкапсуляцією* і є головним принципом ООП.

При оголошенні класу в дужках можуть бути записані (одне або декілька) імена вже існуючих класів – це називається наслідуванням. В такому випадку новий клас (який називається дочірнім) успадковує всі поля та методи класів перелічених в дужках (які називаються батьківськими).

5.1.2 Об'єкти класу Python

Об’єкт – це екземпляр класу. Клас схожий на проект, тоді як екземпляр є копією класу з фактичними значеннями.

Властивості об'єкту:

- Стан: представлений атрибутами об’єкта, та відображає властивості об'єкта.
- Поведінка: вона представлена методами об’єкта, та відображає реакцію об'єкта на інші об'єкти.
- Ідентифікація: вона дає унікальне ім’я об’єкту та дозволяє одному об’єкту взаємодіяти з іншими об’єктами.

5.1.3 Створення класів та об’єктів

Класи збирають в собі набори даних (змінних) разом з наборами функцій, що на них діють. Мета полягає в тому, щоб досягти більш модульного коду за допомогою групування змінних і функцій, в невеликі вузли, що легко модифікувати.

Створення класу:

```
class ім'я_класу:
   інструкція 1
   ....
   інструкція N
```

Створення об'єкта класу:

```
об'єкт_класу = ім'я_класу()
```

Приклад створення найпростішого класу:

```
class Person:
   pass
```

У прикладі визначено клас Person, який умовно представляє людини. В даному випадку в класі не визначається жодних методів чи атрибутів. Однак оскільки в ньому має бути щось визначено, то як замінник функціоналу класу застосовується оператор **pass**. Цей оператор застосовується, коли синтаксично необхідно визначити певний код, проте ми не хочемо його, і замість конкретного коду вставляємо оператор pass.

Після створення класу, можна визначити об'єкти цього класу. Наприклад:

```
class Person:
   pass
tom = Person() # визначення об'єкта tom
bob = Person() # визначення об'єкта bob
```

Після визначення класу Person створюються два об'єкти класу Person - tom і bob. Для створення об'єкта застосовується спеціальна функція - **конструктор** , яка називається на ім'я класу і яка повертає об'єкт класу. Тобто в даному випадку виклик Person() представляє виклик конструктора. Кожен клас за замовчуванням має конструктор без параметрів:

```
tom = Person() # Person() - виклик конструктора, який повертає об'єкт класу Person
```

5.2 Методи класів

Методи класу фактично представляють функції, які визначені всередині класу та визначають його поведінку. Наприклад, визначимо клас Person з одним методом:

```
class Person: # визначення класу Person
def say_hello(self):
   print("Hello")

tom = Person()
```

```
tom.say_hello() # Hello
```

Тут визначено метод say_hello(), який умовно виконує вітання – виводить рядок на консоль. При визначенні методів будь-якого класу слід враховувати, що вони повинні приймати в якості першого параметра посилання на поточний об'єкт, який відповідно до умов називається **self** . Через це посилання всередині класу ми можемо звернутися до функціональності об'єкта. Але при самому виклик методу цей параметр не враховується.

Використовуючи ім'я об'єкта, ми можемо звернутися до його способів. Для звернення до методів застосовується нотація точки – після імені об'єкта ставиться точка і після неї йде виклик методу: об'єкт.метод([параметри методу]).

Якщо метод повинен приймати інші параметри, вони визначаються після параметра self, і за виклику подібного методу їм необхідно передати значення:

```
class Person: # визначення класу Person
def say(self, message): # метод
   print(message)
tom = Person()
tom.say("Hello DUT") # Hello DUT
```

Тут визначено метод say(). Він приймає два параметри: self і message. І другого параметра - message при виклику методу необхідно передати значення.

Через ключове слово **self** можна звертатися всередині класу до функціональності поточного об'єкта:

```
self.атрибут   # звернення до атрибуту
self.метод     # звернення до методу
```

Наприклад, визначимо два методи у класі Person:

```
class Person:
   def say(self, message):
      print(message)

   def say_hello(self):
      self.say("Hello work") # звертаємося до вище визначеного методу say

tom = Person()
tom.say_hello() # Hello work
```

При виклику методу об'єкта нам обов'язково необхідно використовувати слово **self**, якщо ми його не використовуємо:

```
def say_hello(self):
   say("Hello work") # ! Помилка
```

5.3 Конструктори

Для створення об'єкта класу використовується конструктор. Так, вище коли створювалися об'єкти класу Person, було використано конструктор за замовчуванням, який не приймає параметрів і неявно мають всі класи:

```
tom = Person()
```

Однак зручніше явно визначити в класах конструктор за допомогою спеціального методу, який називається **__init __()** (по два прочерки з кожної сторони). Наприклад, змінимо клас Person , додавши до нього конструктор:

```
class Person:
# конструктор
def __init__(self):
    print("Створення об'єкта Person")

def say_hello(self):
print("Hello")

tom = Person( ) # Створення об'єкта Person
tom.say_ hello ( ) # Hello
```

Отже, тут у коді класу Person визначено конструктор і метод say_hello(). Як перший параметр конструктор, як і методи, також приймає посилання на поточний об'єкт - self. Зазвичай конструктори застосовуються для визначення дій, які мають здійснюватися під час створення об'єкта.

Тепер під час створення об'єкта tom = Person() буде здійснено виклик конструктора __init__() з класу Person , який виведе на консоль рядок "Створення об'єкта Person".

5.4 Атрибути об'єкту

Атрибути зберігають стан об'єкта. Для визначення та встановлення атрибутів усередині класу можна застосовувати слово **self** . Наприклад, визначимо наступний клас Person :

```
class Person:
   def __init__(self, name):
      self.name = name # ім'я людини
      self.age = 1       # вік людини

tom = Person("Tom")

# Отримання значень атрибутів
print(tom.name) # Tom
print(tom.age ) # 1
```

```
# Зміна значення атрибуту
tom.age = 37
print(tom.age) # 37
```

Тепер конструктор класу Person приймає ще один параметр - name . Через цей параметр в конструктор буде передаватися ім'я людини для об’єкту, що створюється. Усередині конструктора встановлюються два атрибути - name і age (умовно ім'я та вік людини):

```
def __init__(self, name):
    self.name = name
    self.age = 1
```

Атрибуту self.name надається значення змінної name. Атрибут age набуває значення 1.

Припустимо, що створено клас MyClass, то об’єкт цього класу myobject. Коли ми викликаємо метод якогось об’єкта як myobject.method(arg1, arg2), це автоматично перетворюється Python на MyClass.method(myobject, arg1, arg2).

Приклад:

```
class UNIVStudent:
    def __init__(self, name, university):
        self.name = name
        self. university = university

    def show(self):
        print("Привіт, мене звати " + self.name+" і я" +
            " вчуся в "+self. university+".")

obj = UNIVStudent("Сергій", "ДУІКТ")
obj.show()
```

Якщо ми визначили в класі конструктор __init__, неможливо викликати конструктор за замовчуванням. При створенні об’єкту необхідно викликати явним чином конструктор __init__, до якого необхідно передати значення параметру name:

```
tom = Person("Tom")
```

Далі на ім'я об'єкта ми можемо звертатися до атрибутів об'єкта - отримувати та змінювати їх значення:

```
print(tom.name) # Отримання значення атрибута name
tom.age = 37 # Зміна значення атрибута age
```

Python дозволяє визначати атрибути динамічно поза кодом:

```
class Person:
    def __init__(self, name):
        self.name = name # ім'я людини
        self.age = 1        # вік людини

tom = Person("Tom")
tom.company = "Microsoft"
print(tom.company ) # Microsoft
```

Тут динамічно встановлюється атрибут company , який зберігає місце роботи людини. І після встановлення ми також можемо набути його значення. У той же час подібне визначення загрожує помилками. Наприклад, якщо ми спробуємо звернутися до атрибуту до його визначення, програма згенерує помилку.

Для звернення до атрибутів об'єкта всередині класу у його методах також застосовується слово self:

```
class Person:

    def __init__(self, name):
        self.name = name # ім'я людини
        self.age = 1        # вік людини

    def display_info(self):
        print(f"Name: {self.name} Age: {self.age}")

tom = Person("Tom")
tom.display_info() # Name: Tom Age: 1
```

Тут визначається метод display_info(), який виводить інформацію на консоль. І для звернення в методі до атрибутів об'єкта застосовується слово self: self.name та self.age.

5.5 Створення об'єктів

Вище створювався один об'єкт. Але так само можна створювати й інші об'єкти класу:

```
class Person:
    def __init__(self, name):
        self.name = name # ім'я людини
        self.age = 1        # вік людини

    def display_info(self):
print(f"Name: {self.name} Age: {self.age}")

tom = Person("Tom")
```

```
tom.age = 37
tom.display_info() # Name: Tom Age: 37

bob = Person("Bob")
bob.age = 41
bob.display_info() # Name: Bob Age: 41
```

В цьому прикладі створюються два об'єкти класу Person: tom і bob. Вони відповідають визначенню класу Person, мають однаковий набір атрибутів та методів, однак їхній стан буде відрізнятися.

Ще один приклад - програма з використанням класу. Обчислення середнього бала студента з трьох предметів:

```
import math

class Student():
   def GPA(self, name, e1, e2, e3) :
       self. name = name
       self.e1 = e1
       self.e2 = e2
       self.e3 = e3
       print(self. name, ' - ',((self.e1 + self.e2 + self.e3)/3))

s1 = Student()
s2 = Student()
s1.GPA('Dmytro', 5, 3, 4)
s2.GPA('Olena', 5, 4, 5)

def main():
   return 0

if __name__ == '__main__':
   main()
```

5.6 Практична робота

Мета роботи: ознайомитися з особливостями визначення та використання класів і об’єктів, сформувати навички роботи з класами і об’єктами в Python.

Завдання

1. Вивчити теоретичні основи написання об’єктно-орієнтовних додатків з використанням класів на мові Python. Опрацювати приклади.
2. Виконати завдання у відповідності з варіантом.
3. Скласти звіт по роботі і захистити його.

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

Варіанти

Завдання 1. Напишіть програму з використанням класів у відповідності до варіанту завдання.

Таблиця 5.1 – Варианти завдання 1

Вариант	Завдання
1	Написати програму, в якій реалізовано клас: «Моя Бібліотека». Даний клас повинен виконувати наступні дії: додавання та видалення книги, доступ до анотації книги (короткий опис) та пошуку по книгах за декількома параметрами, а саме: назва, автор, видання та рік випуску.
2	Написати програму, в якій реалізовано клас з двома наступними методами: 1) метод, який приймає 1 аргумент (рядковий тип даних) таповертає True або False в залежності від того, чи містить даний рядок повтори послідовностей символів довжиною починаючи з трьох. 2) метод, який повертає True або False в залежності від того, чиє рядок паліндромом. Регістрами символів можна знехтувати. Порожній рядок вважати паліндромом.
3	Написати програму, в якій реалізовано клас: «Колода Карт». Кожна карта має число і масть. Даний клас повинен забезпечити виконання наступних дій: можливість виведення карти за номером розташування у колоді, виведення всіх карт, перемішування карт, видачі випадковим чином однієї або трьох карт з колоди.
4	Написати програму, в якій реалізовано клас: «Англо-Український словник». Даний клас повинен забезпечити виконання наступних дій: можливість зберігання декількох варіантів перекладу для кожного слова та виведення всіх варіантів перекладу введеного англійського слова.
5	Написати програму, в якій реалізовано клас: «Транспортний засіб». На його основі реалізувати об'єкти: «Автомобіль», «корабель» та «літак». Даний клас повинен виконувати наступні дії: для кожного об'єкта класу повинна існувати можливість задавати та отримувати дані про його назву, модель, координати поточної локації (GPS), пройдений шлях (в км), стан, рік випуску, швидкість та вартість.

6	Розробити клас "домашня бібліотека". Реалізувати можливість роботи з довільним числом книг, пошуку по книгах за декількома параметрами (за автором, за роком видання, за жанром тощо), додавання книг у бібліотеку, видалення книг з неї, доступу до книги за номером. Написати програму, що буде демонструвати всі розроблені елементи класу.
7	Розробити клас, який наслідує функціональність стандартного типу str і містить 2 нових методи: - метод, який приймає 1 аргумент s та повертає True або False в залежності від того, чи містить рядок повтори послідовностей символів довжиною від 3 символів. - метод, який повертає True або False в залежності від того, чи є рядок паліндромом. Регістрами символів нехтувати. Порожній рядок вважати паліндромом.
8	Реалізувати клас Person, який відображає запис в книзі контактів. Клас має 4 атрибута: - surname - рядок - прізвище контакту (обов'язковий) - first_name - рядок - ім'я контакту (обов'язковий) - nickname - рядок - псевдонім (опціональний) - birth_date - об'єкт datetime.date (обов'язковий)
9	Створіть клас Triangle з методом обчислення площі трикутника за формулою Герона, обчислення периметру, перевірки існування трикутника.
10	Створіть клас Box (коробка) з полями h (висота), w (ширина), d (довжина). Створіть методи для обчислення об'єму коробки, її поверхні. Додайте поля для кольору і материалу коробки, а також методи для роботи з ними.

5.7 Контрольні питання

1. Що таке клас в Python?
2. Що таке поля та методи класу?
3. Що таке об’єкт класу?
4. Як створити новий клас в Python?
5. Як створити об'єкт класу в Python?
6. У чому полягає відмінність методу від функції?
7. Який синтаксис використовується під час звернення до атрибуту класу?
8. Поясніть роль self.
9. Який синтаксис використовується під час звернення до методу класу?
10. З якою метою створюється метод init? Напишіть синтаксис.

6 Використання наслідування та інкапсуляції. Атрибути та властивості класів. Перевизначення функціоналу базового класу. Статичні методи.

6.1 Інкапсуляція, атрибути та властивості

За умовчанням атрибути у класах є загальнодоступними, а це означає, що з будь-якого місця програми ми можемо отримати атрибут об'єкта та змінити його. Наприклад:

```
class Person:
def __init__(self, name):
        self.name = name # встановлюємо ім'я
        self.age = 1          # встановлюємо вік

    def display_info(self):
        print(f" Ім'я :{self.name}\t Вік :{self.age}")


tom = Person("Tom")
tom.name = "Людина - павук"       # змінюємо атрибут name
tom.age = -129                    # змінюємо атрибут age
tom.display_info () # Ім'я: Людина-павук Вік: -129
```

Але в даному випадку ми можемо, наприклад, надати віку або імені людини некоректне значення, наприклад, вказати негативний вік. Подібна поведінка небажана, тому постає питання контролю за доступом до атрибутів об'єкта.

З цією проблемою тісно пов'язане поняття інкапсуляції. **Інкапсуляція** є фундаментальною концепцією об'єктно-орієнтованого програмування. Вона запобігає прямому доступу до атрибутів об'єкт з коду, що викликає.

Щодо інкапсуляції безпосередньо в мові програмування Python приховати атрибути класу можна зробивши їх приватними або закритими та обмеживши доступ до них через спеціальні методи, які ще називаються **властивостями** .

Змінимо вище певний клас, визначивши в ньому властивості:

```
class Person:
    def __init__(self, name):
        self.__name = name # встановлюємо ім'я
        self.__age = __1  # встановлюємо вік

    def set_age(self, age):
        if 1 < age < 110:
            self.__age = age
        else:
```

```
            print("Неприпустимий вік")

    def get_age(self):
        return self.__age

    def get_name(self):
        return self.__name

    def display_info(self):
        print( f" Ім'я : {self.__name}\t Вік : {self.__age}")

tom = Person("Tom")
tom.display_info() # Ім'я : Tom Вік : 1
tom.set_age(-3486) # Неприпустимий вік
tom.set_age(25)
tom.display_info() # Ім'я : Tom Вік : 25
```

Для створення приватного атрибуту на початку його найменування ставиться подвійне підкреслювання, наприклад: self.__name. До такого атрибуту ми зможемо звернутися лише з того самого класу. Але не зможемо звернутися поза цим класом. Наприклад, спроба надання значення цьому атрибуту нічого не дасть:

```
tom.__age = 43
```

Тому що в даному випадку просто визначається динамічно новий атрибут __age, але це він не має нічого спільного з атрибутом self.__age.

А спроба отримати його значення командою print (tom.__age) призведе до помилки виконання (якщо раніше не було визначено змінну __age).

Однак все ж таки нам може знадобитися встановлювати вік користувача зовні. І тому створюються характеристики. Використовуючи одну властивість, ми можемо отримати значення атрибуту:

```
def get_age(self):
    return self.__age
```

Цей метод ще часто називають геттер або аксесор.

Для зміни віку визначено іншу властивість:

```
def set_age(self, age):
    if 1 < age < 110:
        self.__age = age
    else:
        print ( "Неприпустимий вік")
```

Даний метод ще називають сеттер або м'ютейтор (mutator). Тут ми вже можемо вирішити, залежно від умов, чи треба встановлювати заново вік.

Необов'язково створювати для кожного приватного атрибуту подібну пару властивостей. Так, у прикладі вище ім'я людини ми можемо встановити лише з конструктора. А для отримання визначено метод get_name.

Анотації властивостей

Вище ми розглянули, як створювати властивості. Але Python має ще один - більш елегантний спосіб визначення властивостей. Цей спосіб передбачає використання анотацій, які передуються символом @.

Для створення властивості-геттера над властивістю ставиться інструкція **@property**.

Для створення властивості-сеттера над властивістю встановлюється інструкція **ім'я_властивості_геттера.setter .**

Перепишемо клас Person з використанням анотацій:

```
class Person:
    def __init__(self, name):
        self.__name = name # встановлюємо ім'я
        self.__age = __1  # встановлюємо вік

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, age):
        if 1 < age < 110:
            self.__age = age
        else:
            print( " Неприпустимий вік ")

    @property
    def name(self):
        return self.__name

    def display_info(self):
        print( f"Ім'я :{self.__name}\t Вік :{self.__age}")

tom = Person("Tom")

tom.display_info() # Ім'я : Tom Вік : 1
tom.age = -3486  # Неприпустимий вік
print(tom.age) # 1
tom.age = 36
tom.display_info() # Ім'я : Tom Вік : 36
```

По-перше, варто звернути увагу на те, що властивість-сеттер визначається після властивості-геттера.

По-друге, і сеттер, і геттер називаються однаково - age. І оскільки геттер називається age, то над сеттером встановлюється анотація @age.setter.

Після того, що до геттера, що до сетера, ми звертаємося через вираз tom.age.

6.2 Спадкування

Спадкування дозволяє створювати новий клас на основі вже існуючого класу. Поряд з інкапсуляцією успадкування є одним із наріжних каменів об'єктно-орієнтованого програмування.

Ключовими поняттями успадкування є **підклас** та **суперклас**. **Підклас** успадковує від суперкласу всі публічні атрибути та методи. Суперклас ще називається базовим (base class) або батьківським (parent class), а підклас - похідним (derived class) або дочірнім (child class).

Синтаксис для успадкування класів виглядає так:

```
class підклас (суперклас):
   методи_підкласу
```

Наприклад, у нас є клас Person , який представляє людину:

```
class Person:
   def __init__( self, name):
      self.__name = name # ім'я людини

   @property
   def name(self):
      return self.__name

   def display_info (self):
      print(f "Name :{self.__name}")
```

Припустимо, нам потрібний клас працівника, який працює на деякому підприємстві. Ми могли б створити з нуля новий клас, наприклад, клас Employee :

```
class Employee:

   def __init__(self, name):
      self.__name = name # ім'я працівника

   @property
   def name(self):
      return self.__name
```

```
    def display_info (self):
        print( f "Name : { self.__name } ")

    def work(self):
        print(f"{self.name} works")
```

Проте клас Employee може мати самі атрибути і методи, як і клас Person, оскільки працівник - це людина. Так, у вище в класі Employee тільки додається метод works, решта коду повторює функціонал класу Person. Але щоб не дублювати функціонал одного класу в іншому, в даному випадку краще застосувати успадкування.

Отже, успадкуємо клас Employee від класу Person :

```
class Person:

    def __init__(self, name):
        self.__name = name # ім'я людини

    @property
    def name(self):
        return self.__name

    def display_info (self):
        print(f"Name :{self.__name}")

class Employee(Person):
    def work(self):
        print(f"{self.name} works")

tom = Employee("Tom")
print(tom.name) # Tom
tom.display_info () # Name: Tom
tom.work () # Tom works
```

Клас Employee повністю переймає функціонал класу Person, лише додаючи метод work(). Відповідно при створенні об'єкта Employee ми можемо використовувати успадкований від Person конструктор:

```
tom = Employee (" Tom ")
```

І також можна звертатися до успадкованих атрибутів/властивостей та методів:

```
print(tom.name) # Tom
tom.display_info () # Name: Tom
```

Однак варто звернути увагу, що для Employee НЕ доступні закриті атрибути типу __name . Наприклад, ми не можемо в методі work звернутися до приватного атрибуту self.__name :

```
def work(self):
print(f"{ self.__name} works") # ! Помилка
```

6.3 Множинне успадкування

Однією з відмінних рис Python є підтримка множинного успадкування, тобто один клас можна успадкувати від декількох класів:

```
# клас працівника
class Employee:
    def work(self):
        print("Employee works")

# клас студента
class Student:
    def study(self):
        print("Student studies")

class WorkingStudent ( Employee, Student):
# Спадкування від класів Employee та Student
    pass

# клас працюючого студента
tom = WorkingStudent()
tom.work() # Employee works
tom.study() # Student studies
```

Тут визначено клас Employee , який представляє співробітника фірми, і клас Student, який представляє студента, що навчається. Клас WorkingStudent, який представляє працюючого студента, не визначає жодного функціоналу, тому в ньому визначено оператора **pass**. Клас WorkingStudent просто успадковує функціонал від двох класів Employee та Student. Відповідно об'єкт цього класу ми можемо викликати методи обох класів.

При цьому успадковані класи можуть бути більш складними за функціональністю, наприклад:

```
class Employee:

    def __init__(self, name):
        self.__name = name

    @property
```

```
    def name(self):
        return self.__name

    def work(self):
        print(f"{self.name} works")

class Student:
    def __init__(self, name):
        self.__name = name

    @property
    def name(self):
        return self.__name

    def study(self):
        print(f"{self.name}studios")

class WorkingStudent (Employee, Student):
    pass

tom = WorkingStudent ("Tom")
tom.work () # Tom works
tom.study () # Tom studies
```

6.4 Перевизначення функціоналу базового класу

При використанні наслідування виникає питання: як щось змінити з функціоналу класу-батька? Наприклад (щодо прикладів вище), додати працівникові через конструктор новий атрибут, який буде зберігати компанію, де він працює або змінити реалізацію методу display_info. Python дозволяє перевизначити функціонал базового класу.

Розглянемо приклад зміни у класах Person та Employee в такий спосіб:

```
class Person:
    def __init__(self, name):
        self.__name = name   # имя человека

    @property
    def name(self):
        return self.__name

    def display_info(self):
        print(f"Name: {self.__name}")

class Employee(Person):
```

```
    def __init__(self, name, company):
        super().__init__(name)
        self.company = company

    def display_info(self):
        super().display_info()
        print(f"Company: {self.company}")

    def work(self):
        print(f"{self.name} works")

tom = Employee("Tom", "Microsoft")
tom.display_info()  # Name: Tom
                    # Company: Microsoft
```

Тут у класі Employee додається новий атрибут – self.company, який зберігає компанія працівника. Відповідно метод __init__() приймає три параметри: другий для встановлення імені та третій для встановлення компанії. Але якщо в базовому класі визначено конструктора за допомогою методу __init__, і ми хочемо у похідному класі змінити логіку конструктора, то в конструкторі похідного класу ми повинні викликати конструктор базового класу. Тобто, в конструкторі Employee треба викликати конструктор класу Person.

Для звернення до базового класу використовується вираз super(). Так, у конструкторі Employee виконується виклик:

```
super().__init__(name)
```

Цей вираз представлятиме виклик конструктора класу Person, який передається ім'я працівника. І це логічно. Адже ім'я працівника встановлюється у конструкторі класу Person. У самому конструкторі Employee лише встановлюємо якість company.

Крім того, в класі Employee перевизначається метод display_info() - до нього додається висновок компанії працівника. Причому ми могли визначити цей метод так:

```
def display_info(self):
    print(f"Name: {self.name}")
    print(f"Company: {self.company}")
```

Але тоді рядок виведення імені повторював код із класу Person. Якщо ця частина коду збігається з методом класу Person, то немає сенсу повторюватися, тому знову ж таки за допомогою виразу super() звертаємося до реалізації методу display_info в класі Person:

```
def display_info(self):
    super().display_info() # звернення до методу display_info у класі Person
    print(f"Company: {self.company}")
```

Потім ми можемо викликати конструктор Employee для створення об'єкта цього класу та викликати метод display_info:

```
tom = Employee("Tom", "Microsoft")
tom.display_info()
```

Консольний висновок програми:

```
Name: Tom
Company: Microsoft
```

6.5 Перевірка типу об'єкта

При роботі з об'єктами буває необхідно в залежності від їх типу виконати ті чи інші операції. І за допомогою вбудованої функції isinstance() ми можемо перевірити тип об'єкта. Ця функція приймає два параметри:

```
isinstance(object, type)
```

Перший параметр представляє об'єкт, а другий - тип, належність якого виконується перевірка. Якщо об'єкт представляє зазначений тип, функція повертає True. Наприклад, візьмемо наступну ієрархію класів Person-Employee/Student:

```
class Person:
    def __init__(self, name):
        self.__name = name # ім'я людини

    @property
    def name(self):
        return self.__name

    def do_nothing(self):
        print(f"{self.name} does nothing")

#клас працівника
class Employee(Person):

    def work(self):
        print(f"{self.name} works")

# клас студента
class Student(Person):
```

```
    def study(self):
        print(f"{self.name}studios")

def act(person):
    if isinstance(person, Student):
        person.study()
    elif isinstance(person, Employee):
        person.work()
    elif isinstance(person, Person):
        person.do_nothing()


tom = Employee("Tom")
bob = Student("Bob")
sam = Person("Sam")

act(tom) # Tom works
act(bob) # Bob studies
act(sam) # Sam does nothing
```

Тут клас Employee визначає метод work(), а клас Student – метод study. Також визначено функцію act, яка перевіряє за допомогою функції isinstance, чи є person параметром певного типу, і залежно від результатів перевірки звертається до певного методу об'єкта.

6.6 Атрибути класу

Окрім атрибутів об'єктів у класі можна визначати атрибути класів. Подібні атрибути визначаються як змінних рівня класу. Наприклад:

```
class Person:
    type = "Person"
    description = "Describes a person"


print(Person.type) # Person
print(Person.description) # Describes a person

Person.type = "Class Person"
print(Person.type) # Class Person
```

Тут у класі Person визначено два атрибути: type, який зберігає ім'я класу, та description, який зберігає опис класу.

Для звернення до атрибутів класу ми можемо використовувати ім'я класу, наприклад: Person.type, і, як і атрибути об'єкта, ми можемо отримувати та змінювати їх значення.

Подібні атрибути є спільними для всіх класових об'єктів:

```
class Person:
    type = "Person"
    def __init__(self, name):
        self.name = name

tom = Person("Tom")
bob = Person("Bob")
print(tom.type) # Person
print(bob.type) # Person

# змінимо атрибут класу
Person.type = "Class Person"
print(tom.type) # Class Person
print(bob.type) # Class Person
```

Атрибути класу можуть бути використані для таких ситуацій, коли нам потрібно визначити деякі загальні дані для всіх об'єктів. Наприклад:

```
class Person:
    default_name = "Undefined"

    def __init__(self, name):
        if name:
            self.name = name
        else:
            self.name = Person.default_name

tom = Person("Tom")
bob = Person("")
print(tom.name) # Tom
print(bob.name) # Undefined
```

У цьому випадку атрибут default_name зберігає стандартне ім'я. І якщо конструктор передано порожній рядок для імені, то атрибуту name передається значення атрибута класу default_name. Для звернення до атрибуту класу всередині методів можна використовувати ім'я класу, наприклад:

```
self.name = Person.default_name
```

Атрибут класу

Можлива ситуація, коли атрибут класу та атрибут об'єкта збігаються на ім'я. Якщо код для атрибута об'єкта не задано значення, то для нього може застосовуватися значення атрибута класу:

```
class Person:
    name = "Undefined"

    def print_name(self):
        print(self.name)

tom = Person()
bob = Person()
tom.print_name() # Undefined
bob.print_name() # Undefined

bob.name = "Bob"
bob.print_name() # Bob
tom.print_name() # Undefined
```

Тут метод print_name використовує атрибут об'єкту name, проте ніде в коді цей атрибут не встановлюється. На рівні класу заданий атрибут name. Тому при першому зверненні до методу print_name в ньому буде використовуватися значення атрибута класу:

```
tom = Person()
bob = Person()
tom.print_name() # Undefined
bob.print_name() # Undefined
```

Але далі ми можемо змінити встановити атрибут об'єкта:

```
bob.name = "Bob"
bob.print_name() # Bob
tom.print_name() # Undefined
```

Причому другий об'єкт – tom продовжить використовувати атрибут класу. І якщо змінимо атрибут класу, відповідно значення tom.name теж зміниться:

```
tom = Person()
bob = Person()
tom.print_name() # Undefined
bob.print_name() # Undefined

Person.name = "Some Person" # змінюємо значення атрибута класу
bob.name = "Bob" # встановлюємо атрибут об'єкта
bob.print_name() # Bob
tom.print_name() # Some Person
```

6.7 Статичні методи

Крім звичайних методів, клас може визначати статичні методи. Такі методи передуються інструкцією @staticmethod і ставляться загалом до класу. Статичні методи зазвичай визначають поведінку, яка залежить від конкретного об'єкта:

```
class Person:
   __type = "Person"

   @staticmethod
   def print_type():
      print(Person.__type)

Person.print_type() # Person - звернення до статичного методу через ім'я класу

tom = Person()
tom.print_type() # Person - звернення до статичного методу через ім'я об'єкта
```

У разі у класі Person визначено атрибут класу __type, який зберігає значення, загальне всім класу - назва класу. Причому оскільки назва атрибуту передує двома підкресленнями, то цей атрибут буде приватним, що захистить від неприпустимої зміни.

Також у класі Person визначено статичний метод print_type, який виводить на консоль значення атрибуту __type. Дія цього методу не залежить від конкретного об'єкта і відноситься загалом до всього класу - незалежно від об'єкта на консоль буде виводиться одне й те саме значення атрибуту __type. Тому такий метод можна зробити статичним.

6.8 Рядкове подання об'єкту

У сучасних версіях мові програмування Python, всі класи неявно мають один загальний суперклас - object і всі класи за замовчуванням успадковують його методи.

Одним із найбільш використовуваних методів класу object є метод __str__(). Python має окремий метод під назвою __str__(). який використовується для визначення того, як об’єкт класу повинен бути представлений у вигляді рядка. Його часто використовують, щоб надати об’єкту зрозуміле для людини текстове представлення, яке корисне для журналювання, налагодження або показу користувачам інформації про об’єкт. Коли об’єкт класу використовується для створення рядка за допомогою вбудованих функцій print() і str(), автоматично використовується функція __str__(). Ви можете змінити спосіб представлення об’єктів класу в рядках, визначивши метод __str__().

Наприклад, візьмемо клас Person і виведемо його рядкову виставу:

```
class Person:
```

```
    def __init__(self, name, age):
        self.name = name # встановлюємо ім'я
        self.age = age # встановлюємо вік

    def display_info(self):
        print(f"Name: {self.name} Age: {self.age}")

tom = Person("Tom", 23)
print(tom)
```

Під час запуску програма виведе щось на кшталт наступного:

<__main__.Person object at 0x10a63dc00>

Це не надто інформативна інформація про об'єкт. Ми, звичайно, можемо вийти зі становища, визначивши в класі Person додатковий метод, який виводить дані об'єкта – у прикладі вище це метод display_info.

Але є й інший вихід - визначимо в класі Person метод __str__() (по два підкреслення з кожного боку):

```
class Person:
    def __init__(self, name, age):
        self.name = name # встановлюємо ім'я
        self.age = age # встановлюємо вік

    def display_info(self):
        print(self)
        # print(self.__str__()) # або так

    def __str__(self):
        return f"Name: {self.name} Age: {self.age}"

tom = Person("Tom", 23)
print(tom) # Назви: Tom Age: 23
tom.display_info() # Name: Tom Age: 23
```

Метод __str__ повинен повертати рядок. І в цьому випадку ми повертаємо базову інформацію про людину. Якщо нам потрібно використовувати цю інформацію в інших методах класу, ми можемо використовувати вираз self.__str__()

6.9 Практична робота

Мета роботи: ознайомитися з особливостями наслідування та інкапсуляції при використання класів і об’єктів, ознайомитися зі статичними методами, сформувати навички роботи з атрибутами і властивостями класів в Python.

Завдання

Виконати завдання у відповідності з варіантом.

Варіанти

Завдання 1. Напишіть програму з використанням класів у відповідності до варіанту завдання. Для всіх реалізованих класів передбачити рядкове подання (перегрузити метод __str__).

1. Написати програму, в якій є головний клас Games зі статичним полем Year, опишіть конструктор, який надає значення полю Year, також опишіть метод getName, який повертає ім'я гри. На основі головного класу шляхом успадкування опишіть чотири класи PCGames, PS4Games, XboxGames, MobileGames. Додайте кожному класу додаткові поля та перевизначте у всіх класів метод getName.

2. Напишіть програму із порожнім класом Country. Опишіть успадковані від класу Country класи Ukraine, Canada, Germany. Додайте кожному класу поле population і опишіть метод setPopulation та getPopulation.

3. Напишіть програму, в якій є головний клас з текстовим полем. У головне класі може бути метод присвоювання значення полю: без аргументів і з одним текстовим аргументом. Об'єкт головного класу створюється передачею текстового аргументу конструктору. За підсумками головного класу створюється класу нащадок. У класі-нащадці потрібно додати числове поле. У конструктора класу-нащадка два аргументи.

4. Створить клас Mentor, який має стати батьківським класом, а від нього потрібно реалізувати спадкування класів Lecturer (лектори) та Reviewer (експерти, які перевіряють домашні завдання). Ім'я, прізвище та список закріплених курсів потрібно реалізувати на рівні батьківського класу.

5. Створить клас Student, який є нащадком класу Person. Персональні дані (Ім'я, прізвище, дата народження) є атрибутами батьківського класу. Перелік курсів, номер групи, оцінки є атрибутами класу Student.

6. Створити клас Animal та розширюючі його класи Dog, Cat, Horse. Клас Animal містить змінні food, location і методи makeNoise, eat, sleep. Метод makeNoise, наприклад, може виводити на консоль "Така тварина спить". Dog, Cat, Horse перевизначають методи makeNoise, eat. Додайте змінні до класів Dog, Cat, Horse, що характеризують лише цих тварин. Напишить програму, що характеризує поведінку тварин.

7. Реалізуйте клас Shape. Реалізуйте класи Circle, Rectangle, Triangle, які успадковують Shape. Передбачте обчислення площини та периметру фігур. Додайте атрибути, що характеризують колір заливки та контуру фігури.

8. Створити кілька класів співробітників та їх спадкоємців. Передбачити клас Person з персональними даними та кілька варіантів нащадків: робітники офісу, промислові робітники, співробітники бухгалтерії та інші. Додати необхідні методи для редагування та перегляду даних співробітників.

9. У міфах крилатий кінь Пегас народився з крові Медузи-Горгони, що потрапила на землю. А за іншою версією його батьком був Посейдон. Давайте на секунду уявимо, що батьками крилатого коня були не грецькі боги чи

медузи, а все-таки коні. У завданні необхідно створити класи Кінь і Пегас, зробивши Пегаса спадкоємцем Коня, і додавши Пегасу можливість летати.

10. Створити клас Car, який містить поля – марка автомобіля, клас автомобіля, вага, потужність двигуна, виробник. Передбачте методи start(), stop(), turnRight(), turnLeft(), які виводять на друк: "Поїхали", "Зупиняємось", "Поворот праворуч" або "Поворот ліворуч". Створити похідний від Car клас - Lorry (вантажівка), що характеризується також вантажопідйомністю кузова. Створити похідний від Car клас - SportCar, який також характеризується граничною швидкістю.

6.10 Контрольні питання

1. У чому полягає такий принцип ОВП, як інкапсуляція?
2. У чому полягає такий принцип ОВП, як наслідування? Наведіть синтаксис створення похідного класу.
3. У чому полягає такий принцип ОВП, як поліморфізм?
4. Поясніть роль статичних методів мови Python. Які методи оголошення статичних методів ви знаєте?
5. Розкажіть про методи створення закритих атрибутів та способи доступу до них.
6. З метою створюються властивості, і як відбувається звернення до них з клієнтського коду?
7. Як створити рядкове подання об’єкту?
8. Як використовувати множинне успадкування в python?

7 Робота з файлами у мові Python.

7.1 Поняття файл і класифікація операцій з ним

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 конкретизується в процесі виконання програми; дані, що містяться у файлі, переносять на зовнішні носії).

Текстові файли призначені 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. 7.1 наведено операції над файлами, які найчастіше використовують.

Таблиця 7.1 Операції над файлами

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‘): операції 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)

7.2 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).

7.3 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. 7.1 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 введеннявиведення. Файли в Python підтримують і можливість позиціонування – метод *seek* дозволяє сценаріями управляти позицією зчитування і запису.

7.4 Модуль shutil

Модуль shutil містить набір функцій високого рівня обробки файлів, груп файлів, і папок. Зокрема, доступні тут функції дозволяють копіювати, переміщати та видаляти файли та папки. Часто використовується разом з модулем os.

7.4.1 Операції над файлами та директоріями

Модуль shutils містить наступні методи:
shutil.copyfileobj (fsrc, fdst [, length]) - скопіювати вміст одного файлового об'єкта (fsrc) в інший (fdst). Необов'язковий параметр length - розмір буфера при копіюванні (щоб весь, можливо величезний, файл не читався повністю на згадку).

При цьому, якщо позиція покажчика fsrc не 0 (тобто до цього було зроблено щось на зразок fsrc.read(47)), то буде копіюватися вміст починаючи з поточної позиції, а не з початку файлу.

shutil.copyfile (src, dst, follow_symlinks=True) - копіює вміст (але не метадані) файлу src у файл dst. Повертає dst (тобто куди файл було скопійовано). src та dst це рядки - шляхи до файлів, при цьому dst має бути повним ім'ям файлу. Якщо src і dst є один і той же файл, виняток shutil.SameFileError .

Якщо dst існує, він буде перезаписаний.

Якщо follow_symlinks=False та src є посиланням на файл, то буде створено нове символічне посилання замість копіювання файлу, на який це символічне посилання вказує.

shutil.copymode (src, dst, follow_symlinks=True) - копіює права доступу з src до dst.Вміст файлу, власник і група не змінюються.

shutil.copystat (src, dst, follow_symlinks=True) - копіює права доступу, час останнього доступу, останньої зміни та прапори src в dst. Вміст файлу, власник і група не змінюються.

shutil.copy (src, dst, follow_symlinks=True) - копіює вміст файлу src у файл або папку dst.Якщо dst є директорією, файл буде скопійовано з тією самою назвою, що було у src.Функція повертає шлях до розташування нового скопійованого файлу.

Якщо follow_symlinks=False і src це посилання, dst буде посиланням.

Якщо follow_symlinks=True і src це посилання, dst буде копією файлу, на який посилається src

Метод copy() копіює вміст файлу та права доступу.

shutil.copy2 (src, dst, follow_symlinks=True) - як copy(), але намагається копіювати всі метадані.

shutil.copytree (src, dst, symlinks = False, ignore = None, copy_function = copy2, ignore_dangling_symlinks = False) - рекурсивно копіює все дерево директорій з коренем в src , повертає директорію призначення. Директорія dst має існувати. Вона буде створена разом із пропущеними батьківськими директоріями. Права та часи у директорій копіюються copystat(), файли копіюються за допомогою функції copy_function (за промовчанням shutil.copy2()).

Якщо symlinks=True, посилання в дереві src будуть посиланнями в dst і метадані будуть скопійовані настільки, наскільки це можливо. Якщо False (за промовчанням), буде скопійовано вміст і метадані файлів, на які вказували посилання.

Якщо symlinks=False і якщо файл, на який вказує посилання, не існує, буде додано виняток до списку помилок, у виключенні shutil.Error наприкінці копіювання. Можна встановити прапорець ignore_dangling_symlinks=True, щоб приховати цю помилку. Якщо ignore не None, то це має бути функція, яка приймає як аргументи ім'я директорії, в якій зараз copytree(), і список вмісту, що повертається os.listdir(). Метод copytree() викликається рекурсивно, тому ignore викликається 1 раз для кожної піддиректорії. Вона повинна повертати список об'єктів щодо поточного імені директорії (тобто підмножина елементів у другому аргументі). Ці об'єкти не будуть скопійовані.

shutil.ignore_patterns (*patterns) - функція, яка створює функцію, яка може бути використана як ignore для copytree() , ігноруючи файли та директорії, які відповідають glob -style шаблонам. Наприклад:

copytree (source , destination , ignore = ignore_patterns ('*.pyc' , 'tmp*'))

Скопіює всі файли, крім тих, що закінчуються на .pyc або починаються з tmp

shutil.rmtree(path, ignore_errors=False, onerror=None) - Видаляє поточну директорію та всі піддиректорії; path повинен вказувати на директорію, а не символічне посилання.

Якщо ignore_errors=True, помилки, що виникають в результаті невдалого видалення, будуть проігноровані. Якщо False (за замовчуванням), ці помилки будуть передаватися обробнику onerror, або, якщо його немає, виникає виняток.

На ОС, які підтримують функції на основі файлових дескрипторів, за замовчуванням використовується версія rmtree() , не вразлива до атак символічних посилань.

На інших платформах це не так: за вибраного часу та обставин "хакер" може, маніпулюючи посиланнями, видалити файли, які недоступні йому в інших обставинах.

Щоб перевірити, чи вразлива система до подібних атак, можна використовувати атрибут rmtree.avoids_symlink_attacks.

Якщо заданий onerror, це має бути функція з 3 параметрами: function, path, excinfo.

Перший параметр (function) - це функція, яка створила виняток; вона залежить від платформи та інтерпретатора. Другий параметр (path) - це шлях, що передається функції. Третій параметр (excinfo) - це інформація про виключення, що повертається sys.exc_info().

shutil.move (src, dst, copy_function=copy2) - рекурсивно переміщає файл або директорію (src) в інше місце (dst), і повертає місце призначення. Якщо dst – існуюча директорія, то src переміщається всередину директорії. Якщо dst існує, але з директорія, воно може бути перезаписано.

shutil.disk_usage (path) - повертає статистику використання дискового простору як namedtuple з атрибутами total, used і free, в байтах.

shutil.chown (path, user=None, group=None) - змінює власника та/або групу у файлу чи директорії.

shutil.which (cmd, mode=os.F_OK | os.X_OK, path=None) - повертає шлях до виконуваного файлу за заданою командою. Якщо немає відповідності з жодним файлом, то None. mode це права доступу, потрібні від файлу, за замовчуванням шукає лише виконувані.

7.4.2 Архівація

Для створення та читання архівованих та стислих файлів модуль shutil використовує досить високорівневі функції, які засновано на функціях із модулів zipfile та tarfile.

Створення архіву:

shutil.make_archive (base_name, format[, root_dir[, base_dir[, verbose[, dry_run[, owner[, group[, logger]]]]]]]) - створює архів і повертає його ім'я.
Параметри:
- **base_name** це ім'я файлу для створення, включаючи шлях, але не включаючи розширення (не потрібно писати ".zip" і т.д.).
- **format** – формат архіву.
- **root_dir** – директорія (щодо поточної), яку ми архівуємо.
- **base_dir** - директорія, в яку архівуватиметься (тобто всі файли в архіві будуть в даній папці).
- Якщо **dry_run=True**, архів не буде створений, але операції, які мали бути виконані, запишуться в logger.
- **owner** та **group** використовуються при створенні tar-архіву.
- **shutil.get_archive_formats()** - список доступних форматів для архівування.

Приклад:
```
import shutil
print(shutil.get_archive_formats())
```
Результат:
```
[('bztar', "bzip2'ed tar-file"), ('gztar', "gzip'ed tar-file"),
('tar', 'uncompressed tar file'), ('xztar', "xz'ed tar-file"), ('zip', 'ZIP file')]
```

Розпакування архіву:
shutil.unpack_archive (filename[, extract_dir[, format]]) - розпаковує архів.
Параметри:
- **filename** - повний шлях до архіву.
- **extract_dir** - те, куди витягуватиметься вміст (за замовчуванням у поточну).
- **format** - формат архіву (за замовчуванням намагається вгадати розширення файлу).
- **shutil.get_unpack_formats()** - список доступних форматів для розархівування.

7.5 Приклади

Копіювання файлу
```
import shutil
shutil.copyfile("C:\\mydoc.doc", "C:\\My Documents\\mydoc_2.doc")
```

Перейменування файлу
```
import os
os.rename("C:\\mydoc.doc\\testfile.txt",
"/home/user/test.txt")
```

Видалення файлу
```
import os
os.remove(" C:\\mydoc.doc \\testfile.txt")
```

Читання потрібного рядка з текстового файлу

Щоб прочитати рядок під певним номером - можна скористатися як стандартним читанням файлу в список, так і використовувати модуль linecache:

```
line = linecache.getline("C:\\boot.ini", 2)
# or
line = open("C:\\boot.ini").readlines()
```

Перебір файлів у каталозі

```
for filename in os.listdir("..\\plugins"):
   print(filename)
```

Перебір файлів у каталозі за маскою

```
import glob
for filename in glob.glob("..\\plugins\\*.zip"):
   print(filename)
```

Порівняння файлів

Порівнювати файли можна як за змістом, так і за їх властивостями, що значно швидше, за допомогою filecmp

```
import filecmp
similar = filecmp.cmp('C:\\file1.txt', 'C:\\file2.txt')  print(similar)
```

Приклад 1. Запис даних у текстовий файл. Створіть директорію з назвою data в середині директорії, де знаходиться даний скрипт

```
import os.path # Модуль, який містить функції для роботи з шляхом у файловій системі
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', 'example01.txt'), text)
```

Приклад 2. Відкриття файла для дозапису

```
import os.path
import datetime
text = '''This text was added in example
2!
Updated
'''
if __name__ == '__main__':
    log_file = os.path.join('data', 'example01.txt')
    with open(log_file, 'a') as log:
        print('\n', text, str(datetime.datetime.now()), file=log)
```

Приклад 3. Відкриття файлу для зчитування

```
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', 'example01.txt'))
```

7.6 Практична робота

Мета роботи: Навчитися здійснювати операції читання та запису для файлів у мові Python.

Завдання

Виконати завдання 1-2 у відповідності з варіантом.

Завдання 1. Створіть файли, у яких будуть міститися рядки з іменами студентів та їх середніми балами. Реалізуйте читання файлів, запис та дозапис у файли, пошук файлів у каталозі та пошук даних у файлі. Також реалізуйте сортування даних у файлі за середнім балом.

Завдання 2. Сформувати вихідний файл (або файли) у текстовому редакторі. Потрібно створити текстовий файл (або файли) з результатами виконання завдання.

Варіанти:

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) перший символ другого рядка; д) 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 5-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 числа. Видалити з нього п'яте число. Результат записати в інший файл.
23. Елементами файлу є цілі числа. Всі парні числа записати в інший файл.
24. 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.
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.

Завдання 3. Виконайте за варіантом.

Варі-ант	Завдання
1	Здійсніть введення двох вкладених списків у вигляді матриці [5,5] у порожній текстовий файл. Розробте програму множення двох матриць. Результуючу матрицю виведіть на екран та текстовий файл.
2	Сформуйте файл з дійсними числами. Видаліть із нього максимальний та мінімальний елементи. Виведіть результати виконання програми на екран і в новий файл.
3	Створіть файл, елементами якого будуть рядкові змінні – літери грецького алфавіту (щонайменше перші п'ять). Розробте програму,

	за допомогою якої за порядковим номером літери в алфавіті можна визначити її назву, і навпаки, за назвою літери визначити її положення в алфавіті. Результати визначення зберегти у файл.
4	Створіть файл, у якому зберігатимуться коефіцієнти квадратного рівняння. Знайдіть коріння квадратного рівняння та виведіть на екран. Додайте в файл з коефіцієтами результати рішення.
5	Сформуйте вихідний файл. З компонентів вихідного файлу цілих чисел сформуйте списки парних та непарних чисел. Визначте найбільший парний компонент файлу та найменший парний. Результат запишіть у текстовий файл.
6	Розробте програму, яка створює текстовий файл, записує в нього рядкову інформацію (кількість рядків наперед не відома, а ознакою закінчення введення є введення в кінці чергового рядка символу *). Підрахуйте кількість рядків, які містяться в цьому файлі.
7	Сформуйте вихідний файл. З компонентів вихідного файлу цілих чисел сформуйте перелік подвоєних непарних чисел. Упорядкуйте його за зростанням елементів. Результат запишіть у текстовий файл.
8	Сформуйте вихідний файл. З компонентів вихідного файлу дійсних чисел сформуйте список, записавши у нього компоненти, які у файлі до мінімального елемента, який задано, і після максимального елемента, який теж задано. Результат запишіть у текстовий файл
9	Сформуйте вихідний файл. З компонентів вихідного файлу дійсних чисел сформуйте список, подвоївши кожен його елемент через два пробіли. Результат запишіть у текстовий файл та виведіть на екран
10	Сформуйте вихідний файл. У файлі у вигляді рядків зберігаються анкети студентів. Рядок містить прізвище та ініціали, рік народження, пол. Підрахуйте, скільки анкет відноситься до осіб чоловічої статі. Підлога задана однією з підрядок «чоловік», «жинка». Результат виведіть на екран
11	Розробте програму, яка дозволяє протестувати знання студента. Послідовність запитань та варіанти відповідей повинні знаходитись у текстовому файлі. Кількість питань не менше п'яти. Текст запитання та відповідей не повинен займати більше одного рядка екрана. Програма виставляє оцінки «5» - усі правильні відповіді, «4» - понад 80 % відповідей вірні, «3» - більше 60 % відповідей вірні, «2» - менше 60 % відповідей вірні
12	Розробте програму, яка за бажанням користувача виводить таблицю перерахунку з дюймів у міліметри на екран або файл
13	Розробте програму, яка створює текстовий файл, записує в нього рядкову інформацію (кількість рядків наперед не відома, а ознакою закінчення введення є введення в кінці чергового рядка символу *). Підрахуйте кількість рядків у наявному файлі
14	Сформуйте вихідний файл. Сформуйте список позитивних чисел, що діляться на п'ять без залишку, використовуючи елементи

	вихідного списку цілих чисел. Впорядкуйте його за зменшенням елементів. Результат запишіть у текстовий файл
15	Сформуйте вихідний файл. З компонентів вихідного файлу дійсних чисел сформуйте список, записавши у нього компоненти, які у файлі між мінімальним і максимальним елементами, які задано окремо. Результат запишіть у текстовий файл
16	Створіть файл, елементами якого будуть строкові змінні. З компонентів вихідного файлу сформуйте текстовий файл, подвоївши всі літери "a". Результат виведіть на екран та в новий файл.
17	Розробте програму, яка створює файл, що містить декілька цілих чисел (кількість елементів вводить користувач) і підраховує суму елементів, що містяться в цьому файлі. Результат запишіть у текстовий файл.
18	Сформуйте вихідний файл. З компонентів вихідного файлу цілих чисел сформуйте перелік негативних чисел. Обчисліть кількість нульових компонент файлу. Результат запишіть у текстовий файл
19	Сформуйте вихідний файл. З компонентів вихідного файлу цілих чисел сформуйте список чисел, записавши лише ненульові компоненти, що перебувають після максимального елемента, який задано окремо. Результат запишіть у текстовий файл
20	Сформуйте вихідний файл. З компонентів вихідного файлу, що містить символи та цілі числа, знайдіть суму цифр, що зустрічаються в ньому. Результат запишіть у текстовий файл та виведіть на екран.
21	Сформуйте вихідний файл. Побудуйте конкатенацію (послідовний запис) вихідного файлу самого із собою. Результат запишіть у текстовий файл та виведіть на екран.
22	Розробте програму, яка записує у текстовий файл прізвище та номер телефону товариша по групі. Знайдіть кількість телефонних номерів, у яких збігаються останні три цифри. Сформуйте результуючий файл телефонів. Якщо таких номерів немає, програма повинна виводити відповідне повідомлення
23	Сформуйте вихідний файл. Видаліть текст у вихідному файлі після першої точки або після першої коми. Результат запишіть у текстовий файл та виведіть на екран.
24	Розробте програму, яка створює файл, що містить наступну введену користувачем інформацію про прочитану ним книгу: номери розділів і назви цих розділів, причому номери повинні записуватися у файл як цілі елементи, а назви - як рядкові. Кількість розділів у прочитаній книзі користувач вводить із клавіатури. Виведіть на екран вміст файлу, причому номер та назва кожного розділу повинні виводитися в окремому рядку.

25	Розробте програму, яка дозволяє знайти потрібні відомості у телефонному довіднику, що зберігається у текстовому файлі. Програма дозволяє вимагати прізвище людини та виводити її телефон. Якщо у довіднику є однакові прізвища, то програма має вивести список усіх людей, які мають ці прізвища
26	Сформуйте вихідний файл. Побудуйте конкатенацію (послідовний запис) вихідного файлу самого із собою, тільки записаного задом наперед. Результат запишіть у текстовий файл та виведіть на екран

7.7 Контрольні запитання.

1. Як здійснюється запис даних у файл у мові Python?
2. Як здійснюється читання даних з файлу у мові Python?
3. Як здійснюється копіювання файлу?
4. Який синтаксис функції open() у мові Python?
5. Які бувають режими роботи з файлами?
6. Що таке менеджер контексту with?
7. У якому вигляді зчитуються дані з файлу в програму?
8. Як скопіювати файл з однієї директорії до іншої?
9. Як переместити файл з однієї директорії до іншої?
10. Як створити архів, який містить декілька файлів?

8 Використання бібліотек python для обчислень з багатовимірними масивами. Робота з NumPy.

8.1 Бібліотека NumPy

8.1.1 Особливості NumPy

NumPy — це бібліотека Python, яка використовується для роботи з масивами. Він також має функції для роботи в області лінійної алгебри, перетворення Фур'є та матриць. NumPy був створений у 2005 році Тревісом Оліфантом. Це проект з відкритим вихідним кодом, і ви можете використовувати його вільно. NumPy означає Numerical Python.

NumPy має на меті надати об'єкт масиву, який у деяких випадках у 50 разів швидший за традиційні списки Python.

Масиви NumPy зберігаються в одному безперервному місці в пам'яті, на відміну від списків, тому процеси можуть отримувати до них доступ і маніпулювати ними дуже ефективно. Така поведінка в інформатиці називається локальністю посилання. Це головна причина, чому NumPy швидше, ніж списки. Крім того, він оптимізований для роботи з останніми архітектурами ЦП.

NumPy — це бібліотека Python, яка частково написана на Python, але більшість частин, які потребують швидкого обчислення, написані на C або C++.

Об'єкт масиву в NumPy називається ndarray, він надає багато допоміжних функцій, які роблять роботу з ним ndarrayудуже легкою.

На рис. 8.1 представлено схему взаємодії між трьома фундаментальними об'єктами, що використовуються для представлення даних в n-мірному масиві.

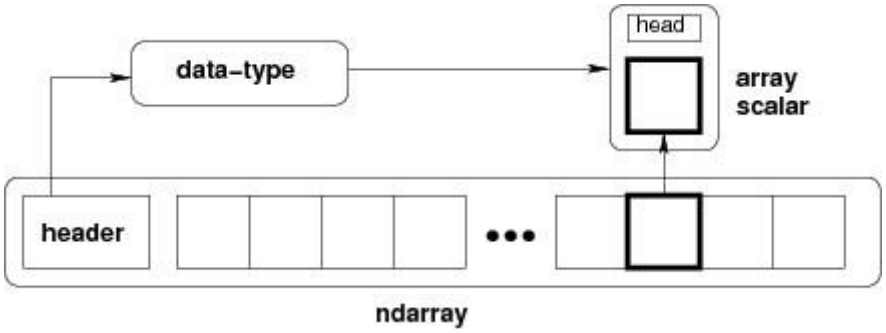

Рисунок 8.1 – Концептуальна схема організації даних у ndarray

Об'єкт ndarray містить елементи однакового типу та розміру.

Основні атрибути об'єктів ndarray:

- ndarray.ndim – кількість вимірів масиву (ранг);
- ndarray.shape – розмір масиву, кортеж натуральних чисел, що представляють довжину масиву за кожною віссю;
- ndarray.size – кількість елементів масиву;
- ndarray.dtype – об'єкт, що описує тип елементів масиву;

- ndarray.itemsize – розмір кожного елементу в байтах;
- ndarray.data – буфер, що містить фактичні елементи масиву.

Масив можна утворити за допомогою функції *array* зі звичайних переліків або кортежів:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

або більш складний (двовимірний) масив:

```
a=np.array([[2,3,4],[1,2,3]])
```

Окрім того можна утворити масив, заповнений нулями, за допомогою функції zeros, передаючи їй кількість елементів за кожною віссю. Для заповнення масиву одиницями призначена відповідна функція ones.

Функція arrange призначена для заповнення масиву значеннями в заданому інтервалі з заданим кроком:

- arange (x) – значення в інтервалі від 0 до x-1;
- arange (x, y) – значення в інтервалі від x до y;
- arange (x, y, z) – значення в інтервалі від x до y з кроком z.

Функція linspace створює масив з заданою кількістю елементів, розподілених рівномірно між значеннями x і y: linspace (x, y, num).

Арифметичні операції, функції sin, cos, exp тощо над масивами виконуються поелементно.

Функція diag() добуває діагональ або конструює діагональний масив.

8.1.2 Багатовимірні масиви

Розмір у масивах — це один рівень глибини масиву (вкладені масиви). За допомогою NumPy можна створити 0-вимірні, 1-вимірні, 2-вимірні, тривимірні і т.д. масиви. Приклад (з контролем кількості вимірів):

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)   # 0
print(b.ndim)   # 1
print(c.ndim)   # 2
print(d.ndim)   # 3
```

Нижче наведено приклад створення 5-вимірного масиву:

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)
```

```
print(arr)  #[[[[[1 2 3 4]]]]]
print('number of dimensions :', arr.ndim)  # number of dimensions : 5
```

8.1.3 Методи масивів NumPy

Об’єкти типу ndarray мають велику кількість методів, частина з яких представлена в таблиці 8.1.

Таблиця 8.1 – Методи об’єктів типу ndarray

Метод	**Опис**
ndarray.item(*args)	Копіює елемент масиву в стандартний скаляр та повертає його.
ndarray.tolist()	Повертає масив у вигляді списку.
ndarray.itemset(*args)	Вставляє скалярне значення у масив.
ndarray.tofile(fid[, sep,	Записує масив у файл в текстовому
Метод	**Опис**
format])	або двійковому вигляді.
ndarray.dump(file)	Виконує дамп масиву у вказаний файл
ndarray.dumps()	Виконує дамп масиву в рядок.
ndarray.astype(dtype[, order, casting, ...])	Повертає копію масиву, змінюючи тип.
ndarray.copy([order])	Повертає копію масиву.
ndarray.view([dtype, type])	Перетворює масив заданим чином, не змінюючи дані, та створює новий масив.
ndarray.fill(value)	Заповнює масив скалярними значеннями.
ndarray.reshape(shape[, order])	Повертає масив зі зміненими розмірами.
ndarray.resize(new_shape[, refcheck])	Змінює розмір масиву на місці.
ndarray.transpose(*axes)	Повертає транспонований масив.
ndarray.swapaxes(axis1, axis2)	Міняє місцями дві вісі масиву.
ndarray.ravel([order])	Повертає одномірний масив.
ndarray.take(indices[, axis, out, mode])	Повертає масив, сформований з елементів за заданими індексами.
ndarray.repeat(repeats[, axis])	Повторює елементи масиву.
ndarray.sort([axis, kind, order])	Сортує масив, виконуючи операцію на місці.

ndarray.argsort([axis, kind, order])	Повертає індекси елементів за розташуванням у відсортованому масиві.
ndarray.partition(kth[, axis, kind, order])	Змінює порядок елементів у масиві таким чином, що елемент зі вказаним індексом розташовується таким чином, що перед ним розташовані елементи з меншим значенням, а правіше – з рівним або більшим.
Метод	**Опис**
ndarray.argmax([axis, out])	Повертає індекси елементів з максимальним значенням за заданою віссю.
ndarray.min([axis, out])	Повертає мінімум вздовж заданої вісі.
ndarray.argmin([axis, out])	Повертає індекси мінімальних значень вздовж заданої вісі.
ndarray.ptp([axis, out])	Повертає діапазон значень (maximum - minimum) вздовж заданої вісі.
ndarray.clip(a_min, a_max[, out])	Повертає масив, значення елементів якого обмежені значеннями [a_min, a_max].
ndarray.round([decimals, out])	Округляє кожний елемент масиву.
ndarray.trace([offset, axis1, axis2, dtype, out])	Повертає суму елементів вздовж діагоналей.
ndarray.sum([axis, dtype, out])	Повертає суму елементів вздовж заданої вісі.
ndarray.cumsum([axis, dtype, out])	Повертає кумулятивну суму елементів вздовж заданої вісі.
ndarray.mean([axis, dtype, out])	Повертає середнє елементів масиву вздовж заданої вісі.
ndarray.var([axis, dtype, out, ddof])	Повертає дисперсію елементів масиву вздовж заданої вісі.
ndarray.std([axis, dtype, out, ddof])	Повертає стандартне відхилення елементів масиву вздовж даної вісі.
ndarray.prod([axis, dtype, out])	Повертає добуток елементів за заданою віссю.
ndarray.cumprod([axis, dtype, out])	Повертає кумулятивний добуток елементів за заданою віссю.

Функція numpy.mat(data, dtype=None) представляє введені дані у вигляді матриці. Об’єкт матриця має зокрема наступні атрибути, що спрощують роботу з ними:

- A – повертає результат у вигляді об’єкту ndarray;

- T – повертає транспоновану матрицю;

Створити такий об’єкт можна наступним чином:

```
import numpy
A = numpy.matrix('1.0 2.0; 3.0 4.0')
```

Бібліотека NumPy представляє нові базові типи для побудови масивів, ієрархія яких представлена на рис. 8.2. Елементи масивів мають ті ж самі атрибути і методи, що й ndarrays.

Об’єкт типу даних є екземпляром класу numpy.dtype і описує, яким чином інтерпретуються байти у блоці пам’яті фіксованого розміру.

Для створення таких об’єктів використовується конструктор:

```
dt = numpy.dtype(numpy.int32)
```

Створений об’єкт має серед інших наступні атрибути:

- dtype.type – тип об’єкту;
- dtype.char – символьний код типу;
- dtype.num – унікальне число для кожного з типів;
- dtype.itemsize – розмір елементу об’єкту даного типу;
- dtype.byteorder – порядок байтів.

Для ітерації використовується об’єкт numpy.nditer, який має ряд налаштувань, що можуть змінювати процес ітерації. Зокрема параметр flags, який може бути використано в конструкторі, може визначати інший порядок обходу масиву або режим доступу до елементів ітератора.

8.1.4 Доступ до елементів масиву

Доступ до елементів масиву базується на його індексації. Ви можете отримати доступ до елемента масиву, посилаючись на його номер індексу.

Індекси в масивах NumPy починаються з 0, тобто перший елемент має індекс 0, а другий має індекс 1 тощо. Приклад:

```
import numpy as np
arr = np.array([1, 2, 3, 4]) # одновимірний масив
print(arr[0])  # 1
arr2 = np.array([[1,2,3,4,5], [6,7,8,9,10]])  # двовимірний масив
print('2nd element on 1st row: ', arr2[0, 1]) # 2nd element on 1st dim:  2
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
# тривимірний масив
print(arr3[0, 1, 2]) # 6
# Негативне індексування використовується для доступу до масиву з кінця.
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1]) # 10
```

8.1.5 Операції зрізу даних

Операція зрізу - це зручний і поширений спосіб отримати деяке підмножина елементів, що йдуть поспіль. Зріз допомагає скоротити час

виконання підвиборки за рахунок того, що не потрібно використовувати цикли. Розглянемо приклади зрізів:

```
# Створення списку мови Python
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Конвертація створеного списку в масив Numpy
numpy_numbers = np.array(numbers)

# Зрізи
# Перші елементи списку
print(numpy_numbers[:4])     # => [0 1 2 3]

# Середина
print(numpy_numbers[2:5])   # => [2 3 4]

# Останні елементи списку
print(numpy_numbers[-3:])    # => [7 8 9]
```

Зрізи багатовимірних масивів спрощують операції зі списками:

```
# Створення списку списків
numbers_lists = [
   [0, 1, 2,],
   [3, 4, 5,],
   [6, 7, 8,],
   [9, 10, 11]
]
# Конвертація створеного списку списків у масив Numpy
numpy_numbers_lists = np.array(numbers_lists)

# Вирізання елементів із numpy.ndarray
print(numpy_numbers_lists[:2,:2])
# => [[0 1]
#     [3 4]]
```

У прикладі вище ми вирізали елементи верхнього лівого квадрата вихідної таблиці розміром 2x2. Щоб вирішити таку задачу за допомогою списків, потрібно було б писати додатковий код, витрачати більше часу та сил.

Ще одна затребувана операція з багатовимірними масивами - отримання рядків та стовпців значень. Знову використовуємо зрізи масиву numpy.ndarrayта реалізуємо завдання таким чином:

```
# Створення списку списків
numbers_lists = [
   [0, 1, 2,],
   [3, 4, 5,],
   [6, 7, 8,],
   [9, 10, 11]]

# Конвертація створеного списку списків у масив Numpy
numpy_numbers_lists = np.array(numbers_lists)
```

```
# Вирізання 0 рядка з numpy.ndarray
print(numpy_numbers_lists[0,:])   # => [0 1 2]

# Вирізання 0 рядка - ще один спосіб
print(numpy_numbers_lists[0])   # => [0 1 2]

# Вирізання 1 стовпчика з numpy.ndarray
print(numpy_numbers_lists[:,1])  # => [ 1  4  7 10]
```

8.1.6 Ітеративний обхід масивів NumPy

У роботі з масивами на python краще не використовувати цикли. Якщо є можливість, краще обходити елементи в потрібному порядку з можливим періодичним пропуском елементів – так зберігається час виконання:

```
# Створення масиву NumPy
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
numpy_numbers = np.array(numbers)

# Парні елементи масиву
print(numpy_numbers[::2])   # => [0 2 4 6 8]

# Зворотний порядок елементів масиву
print(numpy_numbers[::-1])  # => [9 8 7 6 5 4 3 2 1 0]
```

Знак кроку вказує на порядок обходу: плюс говорить про висхідний порядок обходу індексів, мінус - про зворотний. Значення кроку задає період обходу. Аналогічний синтаксис застосовується і для багатовимірних масивів:

```
# Створення масиву NumPy
numbers_lists = [
   [0, 1, 2,],
   [3, 4, 5,],
   [6, 7, 8,],
   [9, 10, 11] ]
numpy_numbers_lists = np.array(numbers_lists)

# Перестановка рядків у зворотному порядку
print(numpy_numbers_lists[::-1])
# => [[ 9 10 11]
#     [ 6  7  8]
#     [ 3  4  5]
#     [ 0  1  2]]

# Чітні стовпці
print(numpy_numbers_lists[:,::2])
# => [[ 0  2]
#     [ 3  5]
#     [ 6  8]
#     [ 9 11]]
```

8.1.7 Маскування масивів

За допомогою модуля numpy.ma можна створити маски масивів, які дозволяють виділити пропущені або невірні значення. Наприклад, один зі способів, яким такий масив можна створити:

```
import numpy.ma as ma
mx = ma.masked_array(x, mask=[0, 0, 0, 1, 0])
```

Функція copyto(dst, src, casting='same_kind', where=None) дозволяє копіювати дані з одного масиву в інший, виконуючи необхідні перетворення.

Для об'єднання масивів використовуються наступні функції:

- hstack(tup) – для горизонтального об'єднання масивів у послідовність;
- vstack(tup) – для вертикального об'єднання масивів у послідовність;
- concatenate((a1, a2, ...), axis=0) – приєднує послідовність масивів;
- column_stack(tup) – об'єднує 1-D масиви у вигляді колонок у 2-D масиви.

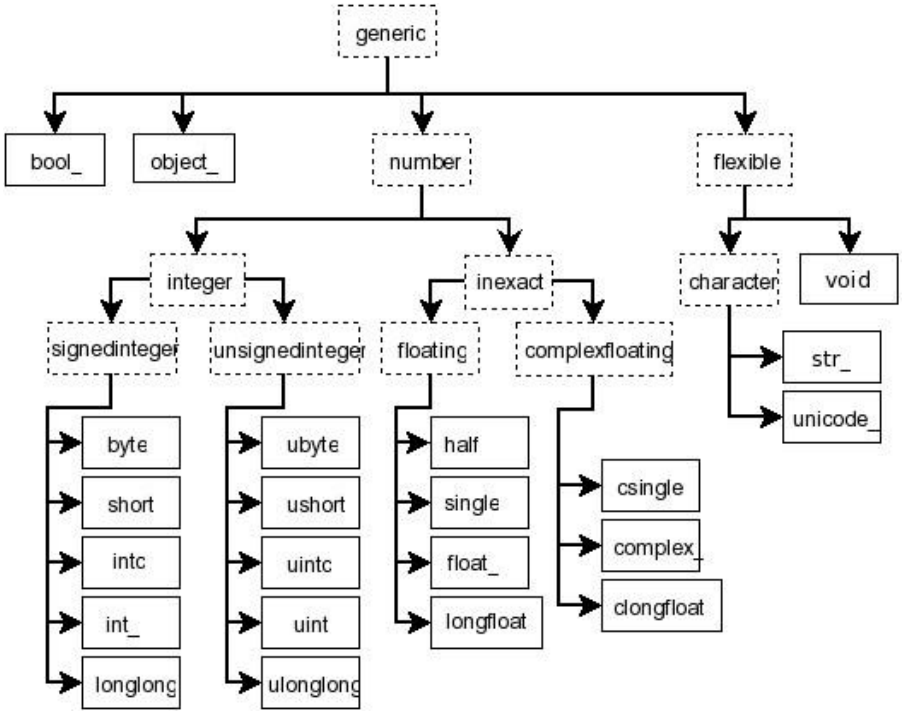

Рисунок 8.2 – Ієрархія типів об'єктів-елементів масиву

Для роз'єднання масивів використовуються наступні функції:

- array_split(ary, indices_or_sections[, axis]) – розділяє масив на підмасиви;
- hsplit(ary, indices_or_sections) – розділяє масив на підмасиви горизонтально;
- split(ary, indices_or_sections[, axis]) – розділяє масив на підмасиви;
- vsplit(ary, indices_or_sections) – розділяє масив на підмасиви вертикально.

Для додавання та вилучення елементів призначені наступні функції:

- delete(arr, obj[, axis]) – вилучає елементи з масиву вздовж заданої вісі;

– insert(arr, obj, values[, axis]) – вставляє значення вздовж заданої вісі перед заданими індексами;
– append(arr, values[, axis]) – додає значення в кінець масиву; – unique(ar[, return_index, return_inverse, ...]) – виділяє унікальні елементи масиву;
– intersect1d(ar1, ar2[, assume_unique]) – знаходить перетин двох масивів;
– setdiff1d(ar1, ar2[, assume_unique]) – знаходить різницю двох масивів у розумінні множин;
– union1d(ar1, ar2) – знаходить об'єднання двох масивів.

Логічні функції включають логічні операції (logical_and(x1, x2[, out]), logical_or(x1, x2[, out]) , logical_not(x[, out]), logical_xor(x1, x2[, out])), перевірки типу елементів масиву (iscomplex(), isreal()), порівняння масивів (array_equal(a1, a2), greater(x1, x2[, out]), greater_equal(x1, x2[, out]), less(x1, x2[, out]), less_equal(x1, x2[, out])).

Бібліотека містить набір функцій лінійної алгебри (модуль numpy.linalg):
– dot(a, b[, out]) – матричний добуток двох масивів;
– vdot(a, b) – добуток двох векторів;
– matrix_power(M, n) – підносить квадратну матрицю у заданий ступінь;
– solve(a, b) – розв'язує лінійне матричне рівняння або систему лінійних рівнянь.

8.2 Бібліотека SciPy

Бібліотека SciPy побудована на основі NumPy та надає потужний науковий інструментарій.

Бібліотека SciPy зокрема містить наступні пакети:
– спеціальних функцій (scipy.special);
– чисельного розв'язання звичайних диференційних рівнянь (scipy.integrate);
– оптимізації (scipy.optimize);
– інтерполяції (scipy.interpolate);
– перетворення Фур'є (scipy.fftpack);
– оброблення сигналів (scipy.signal);
– лінійної алгебри (scipy.linalg);
– розріджених графів (scipy.sparse.csgraph);
– просторові структури даних та алгоритми (scipy.spatial); – випадкових чисел та статистики (scipy.stats);
– багатовимірного оброблення зображень (scipy.ndimage);
– файлового вводу-виводу (scipy.io);
– включення коду C/C++ (scipy.weave); – алгоритмів кластеризації (scipy.cluster);
– розріджених матриць (scipy.sparse).

Бібліотека scipy.linalg містить функції лінійної алгебри:
– inv(a[, overwrite_a, check_finite]) – обчислює зворотну матрицю;
– solve(a, b[, sym_pos, lower, overwrite_a, ...]) – розв'язує рівняння $a\ x = b$ для x;

– det(a[, overwrite_a, check_finite]) – обчислює детермінант матриці;
– lstsq(a, b[, cond, overwrite_a, ...]) – обчислює розв'язання рівняння $Ax = b$ методом найменших квадратів.

Підпакет scipy.integrate зокрема включає наступні функції:
- quad(func, a, b[, args, full_output, ...]) – обчислює визначений інтеграл;
- dblquad(func, a, b, gfun, hfun[, args, ...]) – обчислює подвійний інтеграл;
- tplquad(func, a, b, gfun, hfun, qfun, rfun) – обчислює потрійний інтеграл;
- nquad(func, ranges[, args, opts]) – інтегрування за декількома змінними.

Пакет оптимізації містить функції локальної, глобальної оптимізації, Розенброка, лінійного програмування тощо. Функції локальної оптимізації включають:
– minimize(fun, x0[, args, method, jac, hess, ...]) – мінімізує скалярну функцію однієї або більше змінних одним з методів, представлених параметром method ('Nelder-Mead', 'Powell', 'CG', 'BFGS', 'Newton-CG', 'Anneal', 'L-BFGS-B', 'TNC', 'COBYLA', 'SLSQP', 'dogleg', 'trust-ncg', custom);
– minimize_scalar(fun[, bracket, bounds, ...]) – мінімізує скалярну функцію однієї змінної за допомогою методу, визначеного параметром method ('Brent', 'Bounded', 'Golden', custom);
– OptimizeResult – представляє результати оптимізації. Функції глобальної оптимізації включають:
  – basinhopping(func, x0[, niter, T, stepsize, ...]) – знаходить глобальний мінімум функції, використовуючи стохастичний алгоритм Basin-Hopping;
  – differential_evolution(func, bounds[, args, ...]) – знаходить глобальний мінімум багатовимірної функції за допомогою методу диференційної еволюції.

Функція curve_fit(f, xdata, ydata[, p0, sigma, ...]) використовує нелінійний метод найменших квадратів для мінімізації відхилень функції від даних.

Функція linprog(c[, A_ub, b_ub, A_eq, b_eq, bounds, ...]) мінімізує лінійну цільову функцію з обмеженнями у вигляді нерівностей та лінійних рівностей на основі лінійного програмування.

Функція leastsq(func, x0[, args, Dfun, full_output, ...]) мінімізує суму квадратів множини рівнянь.

Бібліотека інтерполяції scipy.interpolate містить сплайн-функції і класи, одновимірні і багатовимірні інтерполяційні класи, поліноміальні інтерполятори Лагранжа lagrange(x, w) і Тейлора approximate_taylor_polynomial(f, x, degree, ...) , оболонки для функцій FITPACK і DFITPACK.

Одномірна інтерполяція містить наступні функції:
– interp1d(x, y[, kind, axis, copy, ...]) – дозволяє інтерполювати одновимірну функцію;
– barycentric_interpolate(xi, yi, x[, axis]), krogh_interpolate(xi, yi, x[, der, axis]) – допоміжна функція для поліноміальної інтерполяції. Багатомірна інтерполяція включає наступні функції:

– griddata(points, values, xi[, method, ...]) – інтерполювати неструктуровані D-вимірні дані;
– LinearNDInterpolator(points, values[, ...]) – кусково-лінійний інтерполянт у N вимірах;
– NearestNDInterpolator(points, values) – інтерполяція методом найближчого сусіда в N вимірах;
– interp2d(x, y, z[, kind, copy, ...]) – інтерполювати по двовимірній сітці;
– interpn(points, values, xi[, method, ...]) – багатовимірна інтерполяція на регулярних сітках.

Одновимірні сплайни включають:
– UnivariateSpline(x, y[, w, bbox, k, s, ext]) – одновимірний згладжуючий сплайн, що відповідає даній множині точок даних;
– InterpolatedUnivariateSpline(x, y[, w, ...]) – одновимірний інтерполяційний сплайн для даної множини точок даних.

Вищенаведені класи одномірних сплайнів мають наступні методи:
– UnivariateSpline.__call__(x[, nu, ext]) – оцінює сплайн в точці *x*;
– UnivariateSpline.derivatives(x) – повертає всі похідні сплайна в точці *x*;
– UnivariateSpline.get_coeffs() – повертає коефіцієнти сплайна.

Пакет алгоритмів кластеризації містить два модуля: vq, який підтримує векторне квантування і алгоритм k-середніх, та hierarchy, який мість функції для ієрархічної кластеризації. Модуль scipy.cluster.vq включає наступні функції:
– whiten(obs) – нормалізує групу спостережень;
– kmeans(obs, k_or_guess[, iter, thresh]) – виконує алгоритм kсередніх на множині векторів спостережень, формуючи k кластерів;
– kmeans2(data, k[, iter, thresh, minit, missing]) – класифікує множину спостережень на k-кластерів, використовуючи алгоритм kсередніх.

Статистичні функції та розподіли ймовірностей містяться в підпакеті scipy.stats.

Статистичні функції підпакету включають обчислення декількох описових статистик переданого масиву describe(a[, axis, ddof]), середньогоеметричного вздовж заданої вісі gmean(a[, axis, dtype]), середньогармонічного вздовж заданої вісі hmean(a[, axis, dtype]), ексцесу (Фішера або Пірсона) множини даних kurtosis(a[, axis, fisher, bias]), моди mode(a[, axis]), перевірку множини даних на нормальний ексцес kurtosistest(a[, axis]), нормальний розподіл normaltest(a[, axis]), асиметрії на відмінність від нормального розподілу skewtest(a[, axis]), обчислення n-го моменту середнього moment(a[, moment, axis]), асиметрії skew(a[, axis, bias]), коефіцієнту варіації variation(a[, axis]) тощо.

Бібліотека scipy.ndimage містить функції багатовимірного оброблення зображень: фільтри scipy.ndimage.filters (багатовимірний фільтр з гаусівською характеристикою gaussian_filter(input, sigma[, order, ...]) і одновимірний gaussian_filter1d(input, sigma[, axis, ...]), Nвимірний фільтр Лапласа generic_laplace(input, derivative2[, ...]) тощо), фільтри Фур’є

scipy.ndimage.fourier, інтерполяції scipy.ndimage.interpolation, вимірювання scipy.ndimage.measurements, морфології scipy.ndimage.morphology.

8.3 Завдання

Мета роботи: Ознайомитись з основними можливостями бібліотеки NumPy та навчитися використовувати їх на практиці. Ознайомитись з основними можливостями бібліотеки SciPy та навчитися використовувати їх на практиці для виконання високопродуктивних наукових обчислень.

Виконати завдання 1-3.

Завдання 1.

Вирішіть без використання циклів засобами NumPy (кожен пункт вирішується в 1-2 рядки):

1) Створіть вектор з елементами від 12 до 42.
2) Створіть вектор з нулів довжини 12, але його п'ятий елемент повинен дорівнювати 1.
3) Створіть матрицю (3, 3), заповнену випадковими числами від 0 до 8.
4) Знайдіть усі позитивні числа в np.array([1,2,0,0,4,0]) і надрукуйте їх індекси.
5) Помножте матрицю розмірності (5, 3) на (3, 2).
6) Створіть матрицю (10, 10) так, щоб усі граничні елементи дорівнювали були 0, а усі внутрішні – 1.
7) Створіть рандомний вектор та відсортуйте його.

Завдання 2.

Напишіть програму, яка:

- Генерує NumPy-масив 10x10, заповнений випадковими числами від 1.01 до 9.99.
- Виводить на друк головну діагональ.
- Виводить статистичні дані про масив – середнє значення за масивом, за його стовпчиками та рядками, мінімальний та максимальний елементи за масивом, стовпчиками, рядками.

Завдання 3.

Компанія «ГрузАвто» виготовляє вантажівки на 4 заводах. Звіт про виконання виробничого плану по місяцях та заводах можна подати у вигляді масиву:

```
[
  # Січень, Лютий, Березень, Квітень, Травень, Червень, Липень, Серпень, Вересень, Жовтень, Листопад, Грудень
  [1000, 1200, 1500, 1350, 1400, 1300, 1250, 1450, 1300, 1550, 1600, 1700], # Завод 1
```

```
    [800, 900, 1000, 950, 1000, 1100, 1200, 1150, 1000, 1100, 1200, 1300],
# Завод 2
    [1200, 1300, 1250, 1400, 1500, 1600, 1650, 1700, 1600, 1550, 1500, 1400],
# Завод 3
    [900, 950, 1000, 1050, 1100, 1150, 1200, 1250, 1300, 1350, 1400, 1450],
# Завод 4
]
```

Приклад результатів розрахунку:
Загальна кількість вироблених вантажівок за рік: 61050
Завод 1 - 16600 вантажівок
Завод 2 - 12700 вантажівок
Завод 3 - 17650 вантажівок
Завод 4 - 14 100 вантажівок
Завод 3 був найпродуктивнішим
Напишіть програму, яка визначить за наведеним масивом:

- Загальну кількість випущених вантажівок.
- Кількість вантажівок, виготовлених на кожному заводі за рік.
- Найпродуктивніший завод компанії.

8.4 Контрольні запитання

1. Які засоби надаються бібліотекою NumPy?
2. Які засоби надаються бібліотекою SciPy?
3. Для чого необхідна бібліотека matplotlib?
4. Які засоби містить бібліотека matplotlib?
5. Які графіки можна побудувати за допомогою бібліотеки matplotlib?

Литература

1. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. -180 с.
2. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник для студ. спеціальності 122 "Комп'ютерні науки", спеціалізації "Інформаційні технології в біології та медицині"/А.В. Яковенко; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: 1,59 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2018. – 195 с.
3. Васильєв О. М. Програмування мовою Python. Тернопіль: Навчальна книга – Богдан, 2019. – 504с.
4. Копей В. Б. Мова програмування Python для інженерів і науковців : навч. посіб. / В. Б. Копей. - Івано-Франківськ : ІФНТУНГ, 2019. - 272 с.
5. Head First Python / Paul Barry // O'Reilly, 2015 – 494 p.
6. Beazley D. Python Cookbook: Recipes for Mastering Python 3 / David Beazley, Brian K. Jones // O'Reilly, 2013. – 706 p.
7. Swamynathan M. Mastering Machine Learning with Python in Six Steps / Manohar Swamynathan // APress, 2017 – 358 p.
8. Python Machine Learning. Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2 / Sebastian Raschka, Vahid Mirjalili // Packt Publishing, 2019. – 740 p.
9. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Ipython / W. McKinney. – 2nd. Ed. – O'Reilly Media, 2017. – 550 p.