Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни

«Вступ до спеціальності»

Одеса

2024

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
«Вступ до спеціальності»
для студентів спеціальності 113 – Прикладна математика
та 124 – Системний аналіз

курс 1, семестр 1

Затверджено на засіданні кафедри
Прикладна математика та
інформаційни технології
Протокол № 1 від 29.08.2024 р.

Одеса
2024

Методичні вказівки до виконання лабораторних робіт з дисципліни «Вступ до спеціальності» для студентів спеціальності 113 «Прикладна математика» та 124 «Системний аналіз» / Укл.: *О.Г. Нестерюк*. — Одеса: Національний університет «ОДЕСЬКА ПОЛІТЕХНІКА», 2024. – 55 с.

Укладач **О.Г. Нестерюк**, доцент

Методичні вказівки містять десять лабораторних робіт згідно плану лекційного матеріалу.

Навчальний матеріал лабораторних робіт супроводжується основними теоретичними відомостями та завданнями для обов’язкового виконання.

ЗМІСТ

ЛАБОРАТОРНА РОБОТА 1

Тема: Введення до мови програмування Python

Мета: Вивчити принципи побудови програм на мові Python. Навчиться розробляти програми, використовуючи стандартне середовище розробки мови Python; основні типи даних, команди введення та виведення.

Основні теоретичні відомості

Python – це об'єктно-орієнтована, інтерпретована, переносима мова надвисокого рівня. Програмування на Python дозволяє отримувати швидко та якісно необхідні програмні модулі.

У комплекті разом з інтерпретатором Python йде IDLE (інтегроване середовище розробки). За своєю суттю вона подібна до інтерпретатора, запущеного в інтерактивному режимі з розширеним набором можливостей (підсвічування синтаксису, перегляд об'єктів, налагодження тощо).

Для запуску IDLE у Windows необхідно перейти до папки Python у меню "Пуск" і знайти там ярлик з ім'ям "IDLE (Python 3.X XX-bit)".

Для запуску редактора програми (коду) слід виконати команду File->New File або клавіші Ctrl+N.

Будь-яка програма Python складається з послідовності допустимих символів, записаних у визначеному порядку і за певними правилами.

Склад мови Python

Програма включає:

- коментарі;
- команди;
- знаки пунктуації;
- ідентифікатори;
- ключові слова.

Коментарі

Коментарі в Python позначаються попереднім символом # і продовжуються до кінця рядка (тобто в Python всі коментарі є однорядковими), при цьому не допускається використання перед символом # лапок.

Знаки пунктуації

До алфавіту Python входить достатня кількість знаків пунктуації, які використовуються для різних цілей. Наприклад, знаки "+" або "*" можуть використовуватися для додавання та множення, а знак коми "," - для розділення параметрів функцій.

Ідентифікатори

Ідентифікатори в Python - це імена, які використовуються для позначення змінної, функції, класу, модуля або іншого об'єкта.

Ключові слова

Деякі слова мають у Python спеціальне призначення і являють собою керуючі конструкції мови.

Ключові слова в Python:

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

Типи даних

1. None (невизначене значення змінної)
2. Логічні змінні (Boolean Type)
3. Числа (Numeric Type)
    1. int – ціле число
    2. float – число з плаваючою точкою
    3. complex – комплексне число
4. Списки (Sequence Type)
    1. list – список
    2. tuple – кортеж
    3. range – діапазон
5. Рядки (Text Sequence Type)
    1. str

Введення та виведення даних

input

Введення даних здійснюється за допомогою команди

input(список введення):

```
a = input()
print(a)
```

У дужках функції можна вказати повідомлення - коментар до даних, що вводяться:

```
a = input ("Введіть кількість: ")
```

За замовчуванням input() команда сприймає вхідні дані як рядок символів. Тому, щоб ввести ціле значення, слід вказати тип даних int():

```
a = int (input())
```

Для введення дійсних чисел застосовується команда

```
a=float(input())
```

print

Виведення даних здійснюється за допомогою команди

```
print(список виводу):
```

```
a = 1
b = 2
print(a)
print(a + b)
print('сума =', a + b)
```

Існує можливість запису команд в один рядок, поділяючи їх через ;. Однак не слід часто використовувати такий спосіб, це знижує зручність читання:

```
a = 1; b = 2; print(a)
print (a + b)
print ('сума =', a + b)
```

Для команди print може бути так званий сепаратор — роздільник між елементами виведення:

```
x=2
y=5
print ( x, "+", y, "=", x+y, sep = " ")
```

Результат з'явиться з пробілами між елементами: 2 + 5 = 7

Прості арифметичні операції над числами

Операція	Значення
x + y	Додавання
x – y	Віднімання
x * y	множення
x/y	Поділ

format

Для форматованого виводу використовується формат:

Рядковий метод format() повертає відформатовану версію рядка, замінюючи ідентифікатори у фігурних дужках {}. Ідентифікатори можуть бути позиційними, числовими індексами, ключами словників, іменами змінних.

Синтаксис команди format:

поле заміни: = "{" [ім'я поля] ["!" перетворення] [":" специфікація] "}"
ім'я поля := arg_name ("." ім'я атрибута | "[" індекс "]")*
перетворення := "r" (внутрішнє уявлення) | "s" (людське уявлення)
специфікація := див. нижче

Аргументів format() може бути більше, ніж ідентифікаторів у рядку. У такому разі ті, що залишилися, ігноруються.

Ідентифікатори можуть бути індексами аргументів, або ключами.

Специфікація формату:

специфікація	:= [[fill]align][sign][#][0][width][,][.precision][type]

заповнювач	:= символ крім '{' або '}'
вирівнювання	:= "<" | ">" | "=" | "^"
знак	:= "+" | "-" | " "
ширина	:= integer
точність	:= integer
тип	:= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"

Типи даних у специфікаціях

Тип	Значення
'd', 'i', 'u'	Десяткове число.
'o'	Число у вісімковій системі числення.
'x'	Число в шістнадцятковій системі числення (літери у нижньому регістрі).
'X'	Число в шістнадцятковій системі числення (літери у верхньому регістрі).
'e'	Число з точкою, що плаває, з експонентою (експонента в нижньому регістрі).
'E'	Число з плаваючою точкою з експонентою (експонента у верхньому регістрі).
'f', 'F'	Число з плаваючою точкою (звичайний формат).
'g'	Число з плаваючою точкою. з експонентою (експонента в нижньому регістрі), якщо вона менша, ніж -4 або точності, інакше звичайний формат.
'G'	Число з плаваючою точкою. з експонентою (експонента у верхньому регістрі), якщо вона менша, ніж -4 або точності, інакше звичайний формат.
'c'	Символ (рядок одного символу або число - код символу).
's'	Рядок.
'%'	Число множиться на 100, відображається число з точкою, що плаває, а за ним знак %.

Для форматування дійсних чисел із плаваючою точкою використовується наступна команда:

print('{0:.2f}'.format(речове число))

Хід роботи

1. Ознайомитись з необхідним теоретичним матеріалом.
2. Встановити середовище розробки.
3. Відповідно до варіанта завдання скласти та налагодити програму.
4. Оформити звіт.

Завдання

Скласти програму, яка запитує дані з клавіатури, обробляє їх та виводить на консоль. Дані для введення: ПІБ студента, група, вік. Дані для виведення повинні містити назви введених полів, а також їх значення, вирівняні по горизонталі за допомогою пробілів/знаків табуляції. Кожне поле виводити окремим рядком.

Зміст звіту.

Звіт має містити такі пункти:

1.Титульний лист із темою роботи, номером варіанта.

2.Мета роботи.

3. Завдання за варіантом.

5.Текст програми із коментарями.

6.Результати роботи програми (контрольний приклад).

7.Висновки, що відображають труднощі та результати, досягнуті в ході роботи над програмою та алгоритмом, а також можливий аналіз шляхів альтернативного виконання поставленого завдання.

Контрольні запитання

1. Яки прості скалярні типи підтримує Python?
2. Яки парадигми програмування підтримує мова Python?
3. За якими правилами під час виконання коду визначається тип змінної?
4. Який буде результат int(12.5)?
5. Які типи даних розрізняють у Python?

Література

1. Свейгарт Ел. Вчимо Python, роблячи круті ігри. М.: Ексмо, 2018, - 416 с
2. Златопольський Дмитро. Основи програмування мовою Python. М.: ДМК Прес, 2018, - 396 с
3. Доусон Майкл Програмуємо на Python. СПб.: Пітер, 2019, - 416 с
4. Саммерфілд Марк Програмування на Python 3. Детальний посібник. М.: Символ-Плюс, 2009, - 608 с
5. Шоу Зед Легкий спосіб вивчити Python 3. М.: Ексмо, 2019, - 368 с
6. Мюллер Джон Поль Python для чайників. М.: Діалектика, 2019, - 416 с
7. Мусін М. Самовчитель Python v.0.2

8. Беррі Пол Вивчаємо програмування на Python. М.: Видавництво "Е", 2017. - 624с.
9. Прохоронок, Н. А. Python 3. Найнеобхідніше/Н. А. Прохоронок, В. А. Дронов. - 2-ге вид., перероб. та дод. - СПб.: БХВ-Петербург, 2019. - 608 с.:
10. Жуков Р.А. Мова програмування Python: практикум: навчань, посібник / Р.А. Жуків. - М.: ІНФРА-М, 2019. - 216с.
11. Жуков, Р. А. Мова програмування Python: практикум: навчальний посібник/Р.А. Жуків. - Москва: ІНФРА-М, 2021. - 216 с.

ЛАБОРАТОРНАЯ РОБОТА 2

Тема: Умовні оператори мови програмування Python.

Мета: Вивчити принципи побудови розгалужень програм Python. Навчиться розробляти програми, використовуючи умовні оператори.

Основні теоретичні відомості

Умовні оператори допомагають перевірити кілька значень та виконати код залежно від результату перевірки. Ви можете створювати вкладені умовні оператори, які додатково перевірятимуть дані. Також можна перевіряти кілька умов в одному операторі.

Типи змінних

Типи змінних у мові Python не оголошуються очевидно, проте вони тут присутні. Інтерпретатор Пітона розуміє, що записується в змінну і на підставі цього додає тип цієї змінної. У ході програми можна перезаписувати значення змінної, при цьому можна вказувати новий тип змінної. Наприклад, спочатку було записано тип float, але потім можна записати інший тип даних:

```
first_num = 23.2 # Тип даних float
first_num = "1" # Тип даних String
```

Цілі числа (int)

Числа Python 3 нічим не відрізняються від звичайних чисел. Вони підтримують набір найпростіших математичних операцій:

Операція	Значення
x + y	Додавання
x – y	Віднімання
x * y	множення
x/y	Поділ
x // y	Отримання цілої частини від поділу
x % y	Залишок від ділення
-x	Зміна знака числа
abs(x)	Модуль числа
divmod(x, y)	Пара (x // y, x % y)
x**y	Зведення в ступінь
pow(x, y[, z])	xy за модулем (якщо модуль заданий)

Побутові операції

Над цілими числами також можна робити бітові операції

Операція	Значення
x | y	Побітове або
x^y	Побітове виключне або
x & y	Побітове та
x << n	Бітове зрушення вліво
x >> y	Бітове зрушення вправо
~x	Інверсія бітів

Умовний оператор if, if-else, if-elif-else

if <умова1>: <оператор1>

[elif <умова2>: <оператор2>]*

[else: <оператор3>]

Конструкція if

Синтаксис оператора if виглядає так:

if логічний вираз:

команда_1

команда_2

...

команда_n

ВАЖЛИВО: блок коду, який необхідно виконати, у разі істинності виразу, відокремлюється пробілами зліва!

Конструкція if – else

if логічний вираз:

команда_1

команда_2

... команда_n

else:

команда_1

команда_2

```
    ...
    команда_n
```

Конструкція if – elif – else

Для вибору з кількох альтернатив можна використовувати конструкцію if – elif – else. Синтаксис оператора if – elif – else виглядає так:

```
if логічний вираз_1:
    команда_1
    команда_2
    ...
    команда_n
elif логічний вираз_2:
    команда_1
    команда_2
    ...
    команда_n
elif логічний вираз_3:
    команда_1
    команда_2
    ...
    команда_n
else:
    команда_1
    команда_2
    ...
    команда_n
```

приклад.

```
print('Введіть A:')
A=input()
if A < 0:
print('A < 0')
elif A == 0:
```

```
print('A == 0')
else:
print('A > 0')
```

Хід роботи

1. Ознайомитись з необхідним теоретичним матеріалом.
2. Встановити середовище розробки.
3. Відповідно до варіанта завдання скласти та налагодити програму.
4. Оформити звіт.

Завдання

Скласти програму, яка запитує дані з клавіатури, обробляє їх та виводить на консоль. Дані для введення: ПІБ студента, група, оцінка в балах. Обробка полягає у перетворенні значення поля оцінки з бальної системи в систему ECTS (A, B, C, D, E, FX, F). Дані для виведення повинні містити назви полів, а також їх значення. Кожне поле виводити окремим рядком.

Зміст звіту.

Звіт має містити такі пункти:

1.Титульний лист із темою роботи, номером варіанта.

2.Мета роботи.

3. Завдання за варіантом.

5.Текст програми із коментарями.

6.Результати роботи програми (контрольний приклад).

7.Висновки, що відображають труднощі та результати, досягнуті в ході роботи над програмою та алгоритмом, а також можливий аналіз шляхів альтернативного виконання поставленого завдання.

Контрольні запитання

1. Яки ключові слова можна використовувати для контролю за виконанням умовного розгалуження (if)?
2. У яких виподках істинність об'єкта дорівнює False?
3. Яки побітові операції існують у мові Python?
4. Як мовою Python записується логічна операція "I" (множення) x на y?

5. Який тип чи структуру даних отримуємо на виході після операції divmod()?

Література

1. Свейгарт Ел. Вчимо Python, роблячи круті ігри. М.: Ексмо, 2018, - 416 с
2. Златопольський Дмитро. Основи програмування мовою Python. М.: ДМК Прес, 2018, - 396 с
3. Доусон Майкл Програмуємо на Python. СПб.: Пітер, 2019, - 416 с
4. Саммерфілд Марк Програмування на Python 3. Детальний посібник. М.: Символ-Плюс, 2009, - 608 с
5. Шоу Зед Легкий спосіб вивчити Python 3. М: Ексмо, 2019, — 368 с
6. Мюллер Джон Поль Python для чайників. М.: Діалектика, 2019, - 416 с
7. Мусін М. Самовчитель Python v.0.2
8. Беррі Пол Вивчаємо програмування на Python. М.: Видавництво "Е", 2017. - 624с.
9. Прохоренок, Н. А. Python 3. Найнеобхідніше / Н. А. Прохоренок, В. А. Дронов. - 2-ге вид., перероб. та дод. - СПб.: БХВ-Петербург, 2019. - 608 с.:
10. Жуков Р.А. Мова програмування Python: практикум: навчальний посібник / Р.А. Жуків. - М.: ІНФРА-М, 2019. - 216с.
11. Жуков, Р.А. Мова програмування Python: практикум: навчальний посібник / Р.А. Жуків. - Москва: ІНФРА-М, 2021. - 216 с.

ЛАБОРАТОРНАЯ РОБОТА 3

Тема: Цикли мови Python програмування.

Мета: Вивчити принципи побудови циклів у програмах мовою Python. Навчиться розробляти програми за допомогою операторів циклів.

Основні теоретичні відомості

У Python існують два типи циклічних виразів:

• Цикл while

• Цикл for.

Цикл while

Інструкція while у Python повторює зазначений блок коду доти, доки зазначений у циклі логічний вираз залишатиметься істинним. Синтаксис циклу while:

```
while логічний вираз:
        команда 1
        команда 2
        ...
        команда n
```

Цикл for

Цикл for Python має здатність перебирати елементи будь-якого комплексного типу даних (наприклад, рядки або списку). Синтаксис циклу for:

```
for int in range():
        команда 1
        команда 2
        ...
        команда n
```

приклад

```
for i in ra nge (1, 6, 2):
        print (i, end='')
print('')
for i in '1 2 3':
```

```
        print (i * 2, end = '')
print('')
s = [1, 2, 3, 4, 5]
for i in s :
        s [i -1] * = s [i - l]
print(s)
```

Інструкції break, continue, pass

break

Функція break дозволяє перервати виконання циклу. Особливо це важливо у разі використання циклу while.

приклад

```
print ('Для виходу з циклу напишіть д а')
while True :
        a = input (' Напишіть тут: ')
        if a == 'так':
                break
        print('Для виходу з циклу напишіть так' )
```

continue

Інструкція continue дозволяє пропустити поточний блок та перейти до наступної ітерації циклу.

приклад

```
print('Для виходу з циклу напишіть д а ' )
while True :
        s = input ( 'Придумайте пароль(не менше 5 символів): ')
        if s = = 'na':
                break
        if len(s) <5:
                print ( 'Ненадійний пароль')
                continue
print ('Ви ввели пароль достатньої довжини')
```

pass

Інструкція pass є функцією, яка нічого не виконує і зазвичай використовується для формування логіки програми, коли згодом у тіло циклу буде вставлено блок програми.

приклад

for i in range (l, ll): pass

Результат:

Хід роботи

1. Ознайомитись з необхідним теоретичним матеріалом.
2. Відповідно до варіанта завдання скласти та налагодити програму.
3. Оформити звіт.

Завдання

Скласти програму, яка запитує дані з клавіатури, обробляє їх та виводить на консоль. Дані для введення: ПІБ студента, група, назва предмета, оцінка в балах. Обробка полягає у обчисленні середнього значення поля оцінок студента. Дані для виведення повинні містити назви полів, а також їх значення. Кожне поле виводити окремим рядком.

Зміст звіту.

Звіт має містити такі пункти:

1.Титульний лист із темою роботи, номером варіанта.

2.Мета роботи.

3. Завдання за варіантом.

5.Текст програми із коментарями.

6.Результати роботи програми (контрольний приклад).

7.Висновки, що відображають труднощі та результати, досягнуті в ході роботи над програмою та алгоритмом, а також можливий аналіз шляхів альтернативного виконання поставленого завдання.

Контрольні запитання

1. s – рядок. Що буде обчислено під час виклику програмою функції len(s)??
2. У яких виподках в мові Python використовується гілка else?
3. У яких виподках в мові Python використовується вбудований метод списку index()?

4. У яких виподках в мові Python використовується вбудований метод списку remove()?
5. У яких виподках в мові Python використовується оператор continue?

Література

1. Свейгарт Ел. Вчимо Python, роблячи круті ігри. М.: Ексмо, 2018, - 416 с
2. Златопольський Дмитро. Основи програмування мовою Python. М.: ДМК Прес, 2018, - 396 с
3. Доусон Майкл Програмуємо на Python. СПб.: Пітер, 2019, - 416 с
4. Саммерфілд Марк Програмування на Python 3. Детальний посібник. М.: Символ-Плюс, 2009, - 608 с
5. Шоу Зед Легкий спосіб вивчити Python 3. М: Ексмо, 2019, — 368 с
6. Мюллер Джон Поль Python для чайників. М.: Діалектика, 2019, - 416 с
7. Мусін М. Самовчитель Python v.0.2
8. Беррі Пол Вивчаємо програмування на Python. М.: Видавництво "Е", 2017. - 624с.
9. Прохоренок, Н. А. Python 3. Найнеобхідніше / Н. А. Прохоренок, В. А. Дронов. - 2-ге вид., перероб. та дод. - СПб.: БХВ-Петербург, 2019. - 608 с.:
10. Жуков Р.А. Мова програмування Python: практикум: навчань, посібник / Р.А. Жуків. - М.: ІНФРА-М, 2019. - 216с.
11. Жуков, Р. А. Мова програмування Python: практикум: навчальний посібник / Р.А. Жуків. - Москва: ІНФРА-М, 2021. - 216 с.

ЛАБОРАТОРНАЯ РОБОТА 4

Тема: Процедури та функції мови програмування Python.

Мета: Вивчити принципи побудови процедур та функцій у програмах мовою Python. Навчиться розробляти програми, використовуючи процедури та функції, а також наявні математичні функції.

Основні теоретичні відомості

Підпрограма - це іменований фрагмент програми, до якого можна звернутися з іншого місця програми. Підпрограми поділяються на дві категорії: процедури та функції.

Процедури.

Розглянемо синтаксис процедури:

```
def ім'я процедури(Список параметрів):
    Команди тіла процедури
```

приклад.

```
def printChar(s):
    print(s)
sim = input('введіть символ')
printChar(sim) # перший виклик, виведення введеного символу
printChar('*') # другий виклик, висновок *
```

Опції.

Функція – підпрограма, до якої можна звернутися з іншого місця програми.

Для створення функції використовується ключове слово def, після якого вказується ім'я та список аргументів у круглих дужках. Тіло функції виділяється також як тіло умови (або циклу): чотирма пробілами.

Розглянемо синтаксис функції:

```
def ім'я функції(Список параметрів):
    Команди тіла функції
    return вираз
```

приклад.

```
def sumD(n): # визначення функції з параметром
        sumD = 0
        while n! = 0:
                sumD += n % 10
                n = n // 10
        return sumD # повернення значення функції
# основна програма
print (sumD(int(input())) # виклик функції з параметром
```

Математичні операції в Python

Мова Python, завдяки наявності величезної кількості бібліотек для вирішення різноманітних обчислювальних завдань, сьогодні є конкурентом таким пакетам як Matlab та Octave. Запущений в інтерактивному режимі, він фактично перетворюється на потужний калькулятор.

Речові числа (float)

Речові числа підтримують самі операції, як і цілі. Проте (через представлення чисел у комп'ютері) дійсні числа неточні, і це може призвести до помилок.

Бібліотека (модуль) math

У стандартне постачання Python входить бібліотека math, в якій міститься велика кількість математичних функцій, що часто використовуються.

Для роботи з даним модулем його потрібно імпортувати.

Розглянемо найчастіше використовувані функції модуля math

Функція	Призначення
math.ceil(x)	Повертає найближче ціле число більше, ніж x
math.fabs(x)	Повертає абсолютне значення числа x
math.factorial(x)	Обчислює факторіал x
math.floor(x)	Повертає найближче ціле число менше, ніж x
math.exp(x)	Обчислює e**x

math.log2(x)	Логарифм на підставі 2
math.log10(x)	Логарифм на підставі 10
math.log(x[, base])	За умовчанням обчислює логарифм на основі e, додатково можна вказати основу логарифму
math.pow(x, y)	Обчислює значення x у ступені y
math.sqrt(x)	Корінь квадратний від x

Тригонометричні функції модуля math

Функція	Призначення
math.cos(x)	повертає cos числа x
math.sin(x)	повертає sin числа x
math.tan(x)	Повертає tan числа x
math.acos(x)	Повертає acos числа x
math.asin(x)	Повертає asin числа x
math.atan(x)	Повертає atan числа x

Константи:

• math.pi – число Pi.

• math.e – число e (експонента).

Хід роботи

1. Ознайомитись з необхідним теоретичним матеріалом.
2. Відповідно до варіанта завдання скласти та налагодити програму.
3. Оформити звіт.

Завдання

Скласти програму, яка запитує дані з клавіатури, обробляє їх та виводить на консоль. Для введення: початкове значення аргументу функції, кінцеве значення аргументу функції, крок зміни аргументу функції. Обробка полягає у обчисленні результату значення функції. Дані для виведення повинні містити найменування функції та її значення. Значення на кожному кроці розрахунків виводити окремим рядком. Як функцію використовувати довільно вибрані математичні функції з модуля math у кількості не менше двох різних функцій, яку оформити у вигляді функції, створеної розробником і що приймає на вхід поточне значення аргументу на поточному кроці і повертає результат обчислень, який згодом виводиться на екран.

Зміст звіту.

Звіт має містити такі пункти:

1.Титульний лист із темою роботи, номером варіанта.

2.Мета роботи.

3. Завдання за варіантом.

5.Текст програми із коментарями.

6.Результати роботи програми (контрольний приклад).

7.Висновки, що відображають труднощі та результати, досягнуті в ході роботи над програмою та алгоритмом, а також можливий аналіз шляхів альтернативного виконання поставленого завдання.

Контрольні запитання

1. Яка бібліотека забезпечує доступ до математичних функцій?
2. У яких виподках в мові Python використовується гілка else?
3. У яких виподках в мові Python використовується вбудований метод списку index()?
4. У яких виподках в мові Python використовується вбудований метод списку remove()?
5. У яких виподках в мові Python використовується оператор continue?

Література

1. Свейгарт Ел. Вчимо Python, роблячи круті ігри. М.: Ексмо, 2018, - 416 с
2. Златопольський Дмитро. Основи програмування мовою Python. М.: ДМК Прес, 2018, - 396 с
3. Доусон Майкл Програмуємо на Python. СПб.: Пітер, 2019, - 416 с
4. Саммерфілд Марк Програмування на Python 3. Детальний посібник. М.: Символ-Плюс, 2009, - 608 с
5. Шоу Зед Легкий спосіб вивчити Python 3. М: Ексмо, 2019, — 368 с
6. Мюллер Джон Поль Python для чайників. М.: Діалектика, 2019, - 416 с
7. Мусін М. Самовчитель Python v.0.2
8. Беррі Пол Вивчаємо програмування на Python. М.: Видавництво "Е", 2017. - 624с.
9. Прохоренок, Н. А. Python 3. Найнеобхідніше / Н. А. Прохоренок, В. А. Дронов. - 2-ге вид., перероб. та дод. - СПб.: БХВ-Петербург, 2019. - 608 с.:

10. Жуков Р.А. Мова програмування Python: практикум: навчань, посібник / Р.А. Жуків. - М.: ІНФРА-М, 2019. - 216с.
11. Жуков, Р. А. Мова програмування Python: практикум: навчальний посібник / Р.А. Жуків. - Москва: ІНФРА-М, 2021. - 216 с.

ЛАБОРАТОРНАЯ РОБОТА 5

Тема: Класи мови програмування Python.

Мета: Вивчити принципи побудови класів у програмах мовою Python. Навчиться розробляти програми, використовуючи принципи ООП та розробляючи власні об'єкти.

Основні теоретичні відомості

Основною «цеглинкою» ООП є клас — складний тип даних, що включає набір змінних та функцій для управління значеннями, що зберігаються в цих змінних. Змінні називають атрибутами чи властивостями, а функції методами. Клас є фабрикою об'єктів, тобто дозволяє створити необмежену кількість екземплярів, заснованих на цьому класі.

Класи.

Клас описується за допомогою ключового слова class за наступною схемою:

```
class <Назва класу>[(<Клас1>[, ..., <КласN>])]:
   [""" Рядок документування ..”””]
   <Опис атрибутів та методів>
```

Приклад:

```
class MyClass:
   """ Це рядок документування """
   print("Інструкції виконуються відразу")
input()
```

Атрибути та методи

Щоб використовувати атрибути та методи класу, необхідно створити екземпляр класу відповідно до наступного синтаксису:

```
<Примірник класу> = <Назва класу> ([<Параметри>])
```

При зверненні до методів класу використовується такий формат:

```
<Примірник класу>.<Ім'я методу>([<Параметри>])
```

Звернення до атрибутів класу здійснюється аналогічно:

<Примірник класу>. <Ім'я атрибута>

Приклад:

```
class MyClass:
    def __init__(self): # Конструктор
        self.x = 10 # Атрибут екземпляра класу
    def print_x(self): # self - це посилання на екземпляр класу
        print(self.x) # Виводимо значення атрибуту
c = MyClass() # Створення екземпляра класу
# Викликаємо метод print_x()
c.print_x() # self не вказується при викликі методу
print(c.x) # До атрибуту можна звернутися безпосередньо
```

Для доступу до атрибутів та методів можна використовувати такі функції:

getattr(<Об'єкт>, <Атрибут>[, <Знавчання за замовчуванням»])

setattr(<Об'єкт>, <Атрибут>, <Значення>)

delattr(<Об'єкт>, <Атрибут>) — видаляє атрибут, назва якого вказана у вигляді рядка;

hasattr (<об'єкт>, <Атрибут>) - перевіряє наявність вказаного атрибута. Якщо атрибут існує, функція повертає True.

Приклад:

```
class MyClass:
    def __init__(self) :
        self.x = 10
    def get_x(self):
        return self.x
c = MyClass() # Створюємо екземпляр класу
print(getattr(c, "x")) # Виведе: 10
print(getattr(c, "get_x")()) # Виведе: 10
```

```
print(getattr(c, "y", 0)) # Виведе: 0, тому що атрибут не знайдений
setattr(c, "y", 20) # Створюємо атрибут y
print.(getattr(c, "y", 0)) # Виведе: 20
delattr(c, "y") # Видаляємо атрибут y
print(getattr(c, "y", 0)) # Виведе: 0, тому що атрибут не знайдений
print(hasattr(c, "x")) # Виведе: True
print(hasattr(c, "y")) # Виведе: False
```

Усі атрибути класу в мові Python є відкритими (public), тобто доступними для безпосередньої зміни як із самого класу, так і з інших класів та з основного коду програми. Приклад:

```
class MyClass: # Визначаємо порожній клас
   pass
MyClass.x = 50 # Створюємо атрибут об'єкта класу
cl, c2 = MyClass() , MyClass() # Створюємо два екземпляри класу
cl.y = 10 # Створюємо атрибут екземпляра класу
c2.y = 20 # Створюємо атрибут екземпляра класу
print( cl.x, cl.y ) # Виведе: 50 10
print(c2.x, c2.y) # Виведе: 50 20
```

Методи__init_() та __del__()

При створенні екземпляра класу інтерпретатор автоматично викликає метод ініціалізації __init__(). В інших мовах програмування такий метод називається конструктором класу. Формат методу:

```
def __init__(self[, <Значення1>[, . .., <ЗначенняN>]]):
   <Інструкції>

<Примірник класу> = <Ім'я класу> ([<Значення1> [, ..., <ЗначенняN>]])
```

Приклад:

```
class MyClass:
   def __init__(self, valuel, value2): # Конструктор
```

```
        self.x = valuel
        self.y = value2
c = MyClass (100, 300) # Створюємо екземпляр класу
print(c.x, c.y) # Виведе: 100 300
```

Спеціальні методи

Класи підтримують такі спеціальні методи:

- __call__() – дозволяє обробити виклик екземпляра класу як виклик функції. Формат методу:

```
__call__(self[, <Параметр1>[, <ПараметрN>]])
```

Приклад:

```
class MyClass: ■
    def init (self, m):
        self.msg = m
    def call (self) :
        print(self.msg)
cl = MyClass("Значення1") # Створення екземпляра класу
c2 = MyClass("Значення2") # Створення екземпляра класу
c1() # Виведе: Значення1
c2() # Виведе: Значення2
```

- __getattr__(self, <Атрибут>) - викликається при зверненні до неіснуючого атрибута класу.

Приклад:

```
class MyClass:
    def __init__(self) :
        self.i = 20
    def __getattr__(self, attr):
        print("Викликаний метод __getattr__()")
        return 0
c = MyClass()
# Атрибут і існує
```

```
print (c.i) # Виведе: 20. Метод__getattr__() не викликається
# Атрибут s не існує
print(cs) # Виведе: Викликаний метод __getattr__()
```

- __getattribute__(self, <Атрибут>) - викликається при зверненні до будь-якого атрибуту класу. Необхідно враховувати, що використання точкової нотації (для звернення до атрибуту класу) усередині цього методу призведе до зациклювання. Щоб уникнути зациклювання, слід викликати метод __getattribute__() об'єкта object і всередині цього методу повернути значення атрибуту або порушити виняток AttributeError.

Приклад:

```
class MyClass:
   def __init__(se lf) :
      self.i = 20
   def __getattribute__(self, attr) :
      print("Викликаний метод __getattribute__()")
      return object.__getattribute__(self, attr) # Тільки так!
c = MyClass()
print(ci) # Виведе: Викликаний метод __getattribute__() 20
```

- __setattr__(self, <Атрибут>, <Значення>) - викликається при спробі привласнення значення атрибуту екземпляра класу. Якщо всередині методу необхідно надати значення атрибуту, слід використовувати словник __diet__, оскільки при застосуванні точкової нотації метод __setattr__() буде викликаний повторно, що призведе до зациклювання.

Приклад:

```
class MyClass:
   def __setattr__(se lf, attr, value ):
      print("Викликаний метод __setattr__()")
      self .__diet__[attr] = value # Тільки так!
c = MyClass()
ci = 10 # Виведе: Викликаний метод __se ta ttr__()
print(c.i) # Виведе: 10
```

- __delattr__(self, <Атрибут>) — викликається при видаленні атрибуту за допомогою інструкції del <Примірник класу>.<Атрибут>;

- __1en__(self) - викликається при використанні функції 1en(), а також для перевірки об'єкта на логічне значення за відсутності методу __bool__(). Метод має повертати позитивне ціле число.

Приклад:

```
class MyClass:
   def __len__(se lf) :
      return 50
c = MyClass()
print(len(c)) # Виведе: 50
```

- __bool__(self) - викликається під час використання функції bool();
- __int__(self) — викликається під час перетворення об'єкта на ціле число з допомогою функції int();
- __float__(self) - викликається при перетворенні об'єкта в речове число за допомогою функції float();
- __complex__(self) - викликається при перетворенні об'єкта в комплексне число за допомогою функції complex();
- __round__(self, n) - викликається під час використання функції round();
- __index__(self) - викликається при використанні функцій bin(), hex() та oct();
- __repr__(self) і __str__(self) — служать перетворення об'єкта в рядок. Метод __repr__() викликається під час виведення інтерактивної оболонці, і навіть під час використання функції repr(). Метод __str__() викликається під час виведення з допомогою функції print(), і навіть під час використання функції str(). Якщо метод __str__() відсутній, буде викликаний метод __repr__(). Як значення методи __repr__() та __str__() повинні повертати рядок.

Приклад:

```
class MyClass:
   def __init__(self, m):
      self.msg = m
   def __repr__( self ) :
      return "Викликаний метод__repr__() (0)".format(self.msg)
   def __str__ (self) :
      return "Викликано метод __str__() (0)" .format(self.msg)
c = MyClass("Значення")
print(repr(c)) # Виведе: Викликаний метод __repr__ () Значення
print (str (c)) # Виведе: Викликаний метод __str__() Значення
print(c) # Виведе: Викликаний метод __str__() Значення
```

- __hash__(self) — цей метод слід перевизначити, якщо екземпляр класу планується використовувати як ключ словника або всередині множини.

Приклад:

```
class MyClass:
   def __init__ (self, y) :
      self.x = y
   def __hash__(self) :
      return hash(self.x)
c = MyClass(10)
d = U
d[c] = "Значення"
print(d[c]) # Виведе: Значення
```

Класи підтримують ще кілька спеціальних методів, які застосовуються лише в окремих випадках

Хід роботи

1. Ознайомитись з необхідним теоретичним матеріалом.
2. Відповідно до варіанта завдання скласти та налагодити програму.
3. Оформити звіт.

Завдання

Скласти програму, яка запитує дані з клавіатури, зберігає їх у полях розробленого класу, обробляє їх та виводить на консоль. Дані для введення: ПІБ студента, група, оцінка в балах. Обробка полягає у виклик методу розробленого класу, що здійснює перетворення значення поля оцінки з бальної системи в систему ECTS (A, B, C, D, E, FX, F). Дані для виведення повинні містити назви полів, а також їх значення. Кожне поле виводити окремим рядком.

Зміст звіту.

Звіт має містити такі пункти:

1.Титульний лист із темою роботи, номером варіанта.

2.Мета роботи.

3. Завдання за варіантом.

5.Текст програми із коментарями.

6.Результати роботи програми (контрольний приклад).

7.Висновки, що відображають труднощі та результати, досягнуті в ході роботи над програмою та алгоритмом, а також можливий аналіз шляхів альтернативного виконання поставленого завдання.

Контрольні запитання

1. До чого дозволено в мовоі Python доступ через instance._class__attribute?
2. Для чого потрібен вбудований атрибут __mro__ у мові Python?
3. У яких виподках в мові Python використовується вбудований атрибут об'єкта модуля __dict__?
4. Перевірка на сумісність типів здійснюється за допомогою якого оператора?
5. Що означає позитивний результат виконання перевірки на сумісність типів оператором isinstance?

Література

1. Свейгарт Ел. Вчимо Python, роблячи круті ігри. М.: Ексмо, 2018, - 416 с
2. Златопольський Дмитро. Основи програмування мовою Python. М.: ДМК Прес, 2018, - 396 с
3. Доусон Майкл Програмуємо на Python. СПб.: Пітер, 2019, - 416 с
4. Саммерфілд Марк Програмування на Python 3. Детальний посібник. М.: Символ-Плюс, 2009, - 608 с
5. Шоу Зед Легкий спосіб вивчити Python 3. М: Ексмо, 2019, — 368 с
6. Мюллер Джон Поль Python для чайників. М.: Діалектика, 2019, - 416 с
7. Мусін М. Самовчитель Python v.0.2
8. Беррі Пол Вивчаємо програмування на Python. М.: Видавництво "Е", 2017. - 624с.
9. Прохоренок, Н. А. Python 3. Найнеобхідніше / Н. А. Прохоренок, В. А. Дронов. - 2-ге вид., перероб. та дод. - СПб.: БХВ-Петербург, 2019. - 608 с.:
10. Жуков Р.А. Мова програмування Python: практикум: навчань, посібник / Р.А. Жуків. - М.: ІНФРА-М, 2019. - 216с.
11. Жуков, Р. А. Мова програмування Python: практикум: навчальний посібник / Р.А. Жуків. - Москва: ІНФРА-М, 2021. - 216 с.

ЛАБОРАТОРНАЯ РОБОТА 6

Тема: Робота з контейнерами та файлами мови програмування Python.

Мета: Вивчити принципи побудови коконтейнерів у програмах Python. Навчиться розробляти програми, використовуючи принципи зберігання структурованих даних та зберігати їх у вигляді файлів.

Основні теоретичні відомості

Мова Python підтримує засоби для створення класів особливого призначення: ітераторів, контейнерів та перерахувань

Ітератори

Ітератори - це класи, що генерують послідовності будь-яких значень. Такі класи ми можемо задіяти, наприклад, у циклах for:

```
class Mylterator: # Визначаємо клас-ітератор
…….
it = Mylterator() # Створюємо його екземпляр
for v in it: # і використовуємо в циклі for
…….
```

Для того щоб перетворити клас на ітератор, нам слід перевизначити в ньому два спеціальні методи:

♦ __iter__(self) — говорить про те, що цей клас є ітератором (підтримує протокол, як кажуть Python-програмісти). Він повинен повертати сам екземпляр цього класу, а також при необхідності може виконувати всілякі передустановки.

Якщо в класі одночасно визначені методи __iter__() та ___getitem__(), перевага надається першому методу;

♦ __next__(self) — викликається при виконанні кожної ітерації та має повертати чергове значення із послідовності. Якщо послідовність закінчилася, у цьому методі слід порушити виключення stopiteration, яке повідомить код, що викликає, про закінчення ітерацій.

Для прикладу розглянемо клас, що зберігає рядок і на кожній ітерації повертає черговий символ, починаючи з кінця:

```
class Reversestring:
    def __init__(self, s):
```

```
        self.__s = s
    def __iter__(self):
        self.__i = 0
        return self
    def __next__(self):
        if self.__i > len(self.__s) - 1:
            raise Stopiteration
        else:
            a = self.__s[-self.__i - 1]
            self.__i = self.__i + 1
            return a
```

Контейнери

Python дозволяє створити як контейнери-послідовності, аналогічні спискам і кортежам, і контейнери-відображення, тобто. словники.

Контейнери-послідовності

Щоб клас зміг реалізувати функціональність послідовності, нам слід перевизначити у ньому такі спеціальні методи:

- __getitem__(self, <Індекс>) - викликається при вилученні елемента послідовності за його індексом за допомогою операції <Примірник класу>[<Індекс>].
- __setitem__(self, <індекс>, <значення>)— викликається у разі надання нового значення елементу послідовності із заданим індексом (операція <Примірник класу> [<Індекс>] = <Нове значення>).
- __deiitem__(self, <ключ>) — викликається у разі видалення елемента послідовності із заданим індексом за допомогою виразу del «Екземпляр кпасу>[«Ключ>].
- __contains__(self, <Значення>) — викликається під час перевірки існування заданого значення послідовності із застосуванням операторів in і not in. Метод повинен повертати True, якщо таке значення є, і False — інакше.

У класі-послідовності ми можемо додатково реалізувати функціональність ітератора, перевизначивши спеціальні методи __iter__(), __next__() та __len__(). Найчастіше так і роблять.

```
class Mutabiestring:
    def __init __( self , s ) :
        self.__s = list (s)
    # Реалізуємо функціональність ітератора
    def __iter___( self ) :
        self.__i = 0
```

```
        return self
def __next__ (self):
        if self.__i > len ( self .__s) - 1:
                raise Stoplteration
        else :
                a = self.__s [self.__i]
                self.__i = self.__i + 1
                return a
def __len__(self):
        return len f self .__s)
def __str __ ( self ) :
        return " " .join ( self .__s)
# Визначаємо допоміжний метод, який перевірятиме
# Коректність індексу
def __iscorrectindex (self, i):
if type (i) = in t or type ( i) = slice :
        if type (i) == in t and i > self.__le n __() - 1:
                raise IndexError
else :
        raise TypeError
# Реалізуємо функціональність контейнера-списку
def __getitem__( self , i ) :
        self.__iscorrectindex__( i)
        return self.__s [i]
def __setitem ( self , i , v) :
        self.__is correctindex (i)
        self.__s [i] = v
def __delitem__( self , i ) :
        self .__iscorrectindex (i)
        del self.__s [i]
def __contains__( self , v ) :
        return v in self , s
```

Контейнери-словники

Клас, що реалізує функціональність перерахування, повинен перевизначати вже знайомі нам методи: __getitem__(), __setitem__(), __delitem__() та __contains__(). Зрозуміло, у своїй слід зробити поправку те що, що замість індексів тут використовуватимуться ключі довільного типу (зазвичай, строкового).

```
class Version:
    def __init__(self, major, minor, sub):
        self.__major = major # Старша цифра
        self.__minor = minor # Молодша цифра
        self.__sub = sub # Підверсія
    def __str__(self):
        return str(self.__major) + “.” + str (self.__minor) + "." + strfself.__sub)

    # Реалізуємо функціональність словника
    def __qetitem__(self, k):
        if k == "major":
            return self.__major
        elif k = "minor":
            return self.__minor
        elif k = "sub":
            return self.__sub
        else:
            raise IndexError
    def __setitem__(self, k, v):
        if k == "major":
            self.__major = v
        elif k = "minor":
            self.__minor = v
        elif k = "sub":
            self.__sub = v
        else:
            raise IndexError
        def __delitem__(self, k):
            raise TypeError
```

```
def __contains__(self, v):
        return v = "major" or v == "minor" or v = "sub"
```

Перерахування

Перерахування - це певний самим програмістом набір будь-яких іменованих значень. Зазвичай вони застосовуються для того, щоб дати зрозумілі імена будь-яким значенням, що використовуються в коді програми, наприклад кодам помилок, що повертаються функціями Windows API.

Для створення перерахувань застосовуються два класи, визначені в модулі enum:

- Enum — базовий клас для створення класів-перерахувань, елементи яких можуть зберігати значення довільного типу. Для прикладу визначимо клас-перерахування versions, що має два елементи: V2_7 зі значенням "2.7" та V3_6 зі значенням "3.6". Зазначимо, що елементи перерахувань є атрибутами об'єкта класу.

```
from enum inport Enum
class Versions(Enum):
V2_7 = "2.7"
V3_6 = "3.6"
```

- intEnum-базовий клас для створення перерахувань, здатних зберігати лише цілочисельні значення. Запишемо приклад, який представляє код перерахування Colors з трьома елементами, що зберігають цілі числа:

```
from enum inport IntEnum
class Colors(IntEnum):
Red = 1
Green = 2
Blue = 3
```

Робота з файлами

Дуже часто потрібно зберегти будь-які дані. Якщо ці дані мають невеликий об'єм, їх можна записати до файлу.

Відкриття файлу

Перед тим, як працювати з файлом, необхідно створити об'єкт файлу за допомогою функції open(). Функція має такий формат:

open(<Шлях до файлу>[, mode=' r '] [, buffering = - l] [, encoding=None][, errors=None] [, newline=None][, closefd = True])

У першому параметрі вказується шлях до файлу. Шлях може бути абсолютним чи відносним. При вказівці абсолютного шляху Windows слід враховувати, що у Python слеш є спеціальним символом. З цієї причини слеш необхідно подвоювати або замість звичайних рядків використовувати неформатовані рядки:

Шлях визначається з урахуванням розташування поточного робочого каталогу. Відносний шлях буде автоматично перетворено на абсолютний шлях за допомогою функції abspath () з модуля os.path.

Необов'язковий параметр mode у функції open() може набувати наступних значень:

- r – лише читання (значення за замовчуванням). Після відкриття файлу вказівник встановлюється на початок файлу. Якщо файл не існує, збуджується виключення FileNotFoundError;
- r+ - читання та запис. Після відкриття файлу вказівник встановлюється на початок файлу. Якщо файл не існує, то збуджується виняток FileNotFoundError;
- w - запис. Якщо файл не існує, його буде створено. Якщо файл існує, його буде перезаписано. Після відкриття файлу покажчик встановлюється початку файла;
- w+ - Читання та запис. Якщо файл не існує, його буде створено. Якщо файл існує, його буде перезаписано. Після відкриття файлу покажчик встановлюється початку файла;
- a - запис. Якщо файл не існує, його буде створено. Запис здійснюється на кінець файлу. Вміст файлу не видаляється;
- a+ - читання та запис. Якщо файл не існує, його буде створено. Запис здійснюється на кінець файлу. Вміст файлу не видаляється;
- x - створення файлу для запису. Якщо файл вже існує, збуджується виключення FileExistsError;
- x+ — створення файлу для читання та запису. Якщо файл вже існує, збуджується виключення FileExistsError.

Після вказівки режиму може бути модифікатор:

b – файл буде відкритий у бінарному режимі. Файлові методи приймають та повертають об'єкти типу bytes;

♦ t — файл буде відкритий у текстовому режимі (за замовчуванням у Windows). Файлові методи приймають та повертають об'єкти типу str. У цьому режимі автоматично виконуватиметься обробка символу кінця рядка — так, у Windows під час читання замість символів \ r \n буде підставлено символ \n.

За замовчуванням призначається кодування, яке використовується в системі. Якщо перетворення неможливе, збуджується виняток. Вказати кодування, яке буде використовуватися під час запису та читання файлу, дозволяє параметр encoding. Наприклад запишемо дані в кодуванні UTF-8:

```
f = open(r"file.txt", "w", encoding="utf-8")
f.write("Рядок") # Записуємо рядок у файл
6
f. close () # Закриваємо файл
```

Для читання цього файлу слід явно вказати кодування під час відкриття файлу:

```
with open(r"file.txt", "r", encoding="utf-8") as f:
for line in f:
print(line)
Рядок
```

При роботі з файлами в кодуваннях UTF-8, UTF-16 і UTF-32 слід враховувати, що на початку файлу можуть бути службові символи, звані скорочено BOM (Byte Order Mark, мітка порядку байтів). Для кодування UTF-8 ці символи є необов'язковими, і в попередньому прикладі вони не були додані до файлу під час запису. Щоб додати символи BOM, у параметрі encoding слід вказати значення utf-8-sig.
Запишемо рядок у файл у кодуванні UTF-8 з BOM:

```
f = open(r"file.txt", "w", encoding="utf-8-sig")
f .write ("Рядок") # Записуємо рядок у файл
6
f. close () # Закриваємо файл
```

При використанні значень utf-16-le, utf-16-be, utf-32-le та utf-32-be маркер BOM необхідно самим додати на початок файлу, а під час читання видалити його.
У параметрі errors можна вказати рівень обробки помилок. Можливі значення: "strict" (при помилці збуджується виключення vaiueError - значення за замовчуванням), "replace" (невідомий символ замінюється символом питання або символом з кодом \ufffd), "ignore" (невідомі символи ігноруються), "xmlcharrefreplace" (невідомий символ замінюється послідовністю &#xxxx;) і "backsiashrepiace" (невідомий символ замінюється послідовністю \uxxxx).
Параметр newline визначає режим обробки символів кінця рядків. Підтримувані ним значення
такі:

- None (значення за замовчуванням) — стандартна обробка символів кінця рядка. Наприклад, у Windows під час читання символи \r\n перетворюються на символ \n, а під час запису відбувається зворотне перетворення;
- " " (порожній рядок) — обробка символів кінця рядка не виконується;
- "<спеціальний символ>" — зазначений спеціальний символ використовується для позначення кінця рядка, і жодна додаткова обробка не виконується. Як спеціальний символ можна вказати лише \r\n, \r і \n.

Методи роботи з файлами

Після відкриття файлу функція open() повертає об'єкт, за допомогою якого провадиться подальша робота з файлом. Тип об'єкта залежить від режиму відкриття файлу та буферизації. Розглянемо основні методи:

- close() – закриває файл. Оскільки інтерпретатор автоматично видаляє об'єкт, коли на нього відсутні посилання, файл можна не закривати явно в невеликих програмах. Проте явне закриття файлу є ознакою гарного стилю програмування.:

```
with open(r"file.txt", "w", encoding="cpl251") as f:
f.write("Рядок") # Записуємо рядок у файл
# Тут файл вже закрито автоматично
```

- write (<дані>) — записує дані у файл. Якщо в якості параметра вказано рядок, файл повинен бути відкритий у текстовому режимі, а якщо вказана послідовність байтів у бінарному.

```
# Текстовий режим
f = open(r"file.txt", "w", encoding="cpl251")
f.write ("Рядок1\nРядок2") # Записуємо рядок у файл
15
f.close () # Закриваємо файл
# Бінарний режим
f = open(r"file.txt", "wb")
f .write (bytes ("Рядок1\nРядок2", "cp1251"))
15
f.write ( bytearray ( " \ n CTpoKa3 " , " cp1251 " ) )
8
f.close()
```

- writelines (<Послідовність>) - записує послідовність у файл. Якщо всі елементи послідовності є рядками, файл має бути відкритим у текстовому режимі.
- writable () - повертає True, якщо файл підтримує запис, і False - в іншому
- випадку:

- read([<Кількість>]) — зчитує дані з файлу. Якщо файл відкритий у текстовому режимі, повертається рядок, і якщо у бінарному— послідовність байтів.:

```
# Текстовий режим
with open(r"file.txt", "r", encoding="cpl251") as f:
f.read()
'Рядок1\nРядок2'
# Бінарний режим
with open(r"file.txt", "rb") as f:
f.read()
b'\xdl\xf2\xf0\xee\xea\xe01\n\xdl\xf2\xf0\xee\xea\xe02'
```

- readline ([<Кількість>]) - зчитує з файлу один рядок при кожному виклику. Якщо файл відкритий у текстовому режимі, повертається рядок, і якщо в бінарному — послідовність байтів. Повертається рядок містить символ перекладу рядка.
- readlines() - зчитує весь вміст файлу до списку. Кожен елемент списку міститиме один рядок, включаючи символ перекладу.
- __next__ () — зчитує один рядок під час кожного виклику. Якщо файл відкритий у текстовому режимі, повертається рядок, і якщо у бінарному— послідовність байтів. При досягненні кінця файлу збуджується виключення stopiteration. Завдяки методу __next__() ми можемо перебирати файл рядково у циклі for. Цикл for кожної ітерації буде автоматично викликати метод __next__().
- flush() - примусово записує дані з буфера на диск;
- fileno() — повертає цілий дескриптор файлу. Значення, що повертається завжди буде більше числа 2, т.к. число 0 закріплено за стандартним введенням stdin, 1 - за стандартним виведенням stdout, а 2 - за стандартним виведенням повідомлень про помилки stderr;
- truncate ([<кількість>]) — обрізає файл до вказаної кількості символів (якщо заданий текстовий режим) або байтів (у разі бінарного режиму). Метод повертає новий розмір;
- tell() - повертає позицію покажчика щодо початку файлу у вигляді цілого числа. Зверніть увагу: у Windows метод tell вважає символ \r як додатковий байт, хоча цей символ видаляється при відкритті файлу в текстовому режимі;
- seek (<Зміщення> [, <позиція>]) — встановлює вказівник у позицію, що має задане <зсув> щодо параметра <позиція>. Як параметр <позиція> можуть бути вказані наступні атрибути з модуля io або відповідні їм значення:
  - io.SEEK_SET або 0 - початок файлу (значення за замовчуванням);
  - io.SEEK_CUR або 1 – поточна позиція покажчика. Позитивне значення зсуву викликає переміщення до кінця файлу, негативне - до початку;
  - io.SEEK_END або 2 - кінець файлу.
- seekable () - повертає True, якщо покажчик файлу можна зрушити в іншу позицію, і False - інакше;

Крім методів, об'єкти файлів підтримують кілька атрибутів:

- name - ім'я файлу;
- mode — режим, у якому було відкрито файл;

- closed — повертає True, якщо файл був закритий, і False — інакше;
- encoding— назва кодування, яке буде використовуватися для перетворення рядків перед записом у файл або під час читання. Атрибут доступний лише у текстовому режимі.
- buffer – дозволяє отримати доступ до буфера. Атрибут доступний лише у текстовому режимі. За допомогою цього об'єкта можна записати послідовність байтів у текстовий потік.

Хід роботи

1. Ознайомитись з необхідним теоретичним матеріалом.
2. Відповідно до варіанта завдання скласти та налагодити програму.
3. Оформити звіт.

Завдання

Модифікувати програму, складену у попередній лабораторній роботі №5, таким чином, щоб вона складала рейтинг студентів з кількох предметів та зберігала дані про них у файл.

Завдання лабораторної роботи №5:

Скласти програму, яка запитує дані з клавіатури, зберігає їх у полях розробленого класу, обробляє їх та виводить на консоль. Дані для введення: ПІБ студента, група, оцінка в балах. Обробка полягає у виклик методу розробленого класу, що здійснює перетворення значення поля оцінки з бальної системи в систему ECTS (A, B, C, D, E, FX, F). Дані для виведення повинні містити назви полів, а також їх значення. Кожне поле виводити окремим рядком.

Зміст звіту.

Звіт має містити такі пункти:

1.Титульний лист із темою роботи, номером варіанта.

2.Мета роботи.

3. Завдання за варіантом.

5.Текст програми із коментарями.

6.Результати роботи програми (контрольний приклад).

7.Висновки, що відображають труднощі та результати, досягнуті в ході роботи над програмою та алгоритмом, а також можливий аналіз шляхів альтернативного виконання поставленого завдання.

Література

1. Свейгарт Ел. Вчимо Python, роблячи круті ігри. М.: Ексмо, 2018, - 416 с

2. Златопольський Дмитро. Основи програмування мовою Python. М.: ДМК Прес, 2018, - 396 с
3. Доусон Майкл Програмуємо на Python. СПб.: Пітер, 2019, - 416 с
4. Саммерфілд Марк Програмування на Python 3. Детальний посібник. М.: Символ-Плюс, 2009, - 608 с
5. Шоу Зед Легкий спосіб вивчити Python 3. М: Ексмо, 2019, — 368 с
6. Мюллер Джон Поль Python для чайників. М.: Діалектика, 2019, - 416 с
7. Мусін М. Самовчитель Python v.0.2
8. Беррі Пол Вивчаємо програмування на Python. М.: Видавництво "Е", 2017. - 624с.
9. Прохоренок, Н. А. Python 3. Найнеобхідніше / Н. А. Прохоренок, В. А. Дронов. - 2-ге вид., перероб. та дод. - СПб.: БХВ-Петербург, 2019. - 608 с.:
10. Жуков Р.А. Мова програмування Python: практикум: навчань, посібник / Р.А. Жуків. - М.: ІНФРА-М, 2019. - 216с.
11. Жуков, Р. А. Мова програмування Python: практикум: навчальний посібник / Р.А. Жуків. - Москва: ІНФРА-М, 2021. - 216 с.

ЛАБОРАТОРНАЯ РОБОТА 7

Тема: Обробка виключень мови програмування Python.

Мета: Вивчити принципи побудови обробників виняткових ситуацій у програмах мовою Python. Навчиться розробляти програми, що використовують класи стандартних і винятків користувача.

Хід роботи

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Винятки — це повідомлення інтерпретатора, які збуджуються у разі виникнення помилки в програмному коді або при настанні будь-якої події. У програмі можуть зустрітися три типи помилок:

- синтаксичні - це помилки в імені оператора або функції, відсутність закриває або відкриває лапок і т. д. - тобто помилки в синтаксисі мови. Як правило, інтерпретатор попередить про наявність такої помилки, а програма не виконуватиметься зовсім.
- логічні - це помилки в логіці програми, які можна виявити лише за результатами її роботи. Як правило, інтерпретатор не попереджає про наявність такої помилки, і програма успішно виконуватиметься, але результат її виконання виявиться не тим, на який ми розраховували. Виявити та виправити логічні помилки дуже важко;
- помилки часу виконання – це помилки, які виникають під час роботи програми. Причиною є події, які не передбачені програмістом.

Необхідно зауважити, що у Python винятки збуджуються не тільки при виникненні помилки, але і як повідомлення про настання будь-яких подій.

Інструкція try...except...else...finally

Для обробки винятків призначено інструкцію try. Формат інструкції:

try:

<Блок, у якому перехоплюються винятки>

[except [<Виключення 1> [as <Об'єкт виключення>]] :

<Блок, що виконується у разі виключення>

[...

except [<ВиключенняN>[as <Об'єкт виключення>]]:

<Блок, що виконується у разі виключення>]]

[else:

<Блок, який виконується, якщо виняток не виникло>]

[finally:

<Блок, що виконується у будь-якому випадку>]

Інструкції, в яких перехоплюються винятки, мають бути розташовані всередині блоку try. У блоці except у параметрі <Виключення 1> вказується клас виключення, що обробляється. Наприклад, обробити виняток, що виникає при розподілі на нуль, можна так:

```
try: # Перехоплюємо винятки
   x = 1 / 0 # Помилка: розподіл на 0
except ZeroDivisionError: # Вказуємо клас виключення
   print("Обробили поділ на 0")
   x = 0
print(x) # Виведе: 0
```

Якщо у блоці try виник виняток, керування передається блоку except. Якщо виняток не відповідає зазначеному класу, управління передається наступному блоку except. Якщо жоден блок except відповідає винятку, то виняток «спливає» до оброблювача вищого рівня. Якщо виняток у програмі взагалі ніде не обробляється, він передається оброблювачу за замовчуванням, який зупиняє виконання програми та виводить стандартну інформацію про помилку. Таким чином, в обробнику може бути кілька блоків except з різними класами винятків. Крім того, один обробник можна вкласти до іншого.

В інструкції except можна вказати відразу кілька винятків, записавши їх через кому всередині круглих дужок. Отримати інформацію про виключення, що обробляється, можна через другий параметр в інструкції except:

```
try :
   x = 1 / 0 # Помилка поділу на 0
except (NameError, IndexError, ZeroDivisionError) as err :
   print (err.__class__.__name__) # Назва класу виключення
   print (err) # Текст повідомлення про помилку
```

Для отримання інформації про виключення можна скористатися функцією exc_info() з модуля sys, яка повертає кортеж із трьох елементів: типу виключення, значення та об'єкта з трасувальною інформацією. Перетворити ці значення на вигляд, що легко читається, дозволяє модуль traceback. Приклад:

```
import sys, traceback
try:
   x = 1/0
except ZeroDivisionError:
   Type, Value, Trace = sys.exc_info()
   print("Type: ", Type)
   print("Value:", Value)
   print("Trace:", Trace)
   print("\n", ”print_exception()",center(40, "-"))
   traceback.print_exception(Type, Value, Trace, limit=5, file=sys.stdout)
   print("\n", "print_tb()".center(40, "-"))
   traceback.print_tb(Trace, limit=l, file=sys.stdout)
   print("\n", "format_exception()”.center(40, "-"))
   print(traceback.format_exception(Type, Value, Trace, limit=5))
   print("\n", "format_exception_only()".center(40, "-"))
   print(traceback.fonnat_exception_only(Type, Value))
```

Якщо в інструкції except не вказано клас виключення, такий блок перехоплюватиме всі винятки. Насправді слід уникати порожніх інструкцій except, т. до. можна перехопити виняток, що є лише сигналом системі, а чи не помилкою.

Якщо в обробнику є блок else, то інструкції всередині цього блоку будуть виконані тільки при відсутності помилок. При необхідності виконати якісь завершальні дії незалежно від того, виник виняток чи ні, слід скористатися блоком finally.

Необхідно зауважити, що за наявності виключення та відсутності блоку except інструкції всередині блоку finally будуть виконані, але виняток не буде оброблено. Воно продовжить «випливання» до оброблювача вищого рівня. Якщо користувальницький обробник відсутній, керування передається обробникові за замовчуванням, який перериває виконання програми та виводить повідомлення про помилку.

Інструкція with...as

Мова Python підтримує протокол менеджерів контексту. Цей протокол гарантує виконання завершальних дій (наприклад, закриття файлу) незалежно від того, чи відбулося виключення всередині блоку коду чи ні. Для роботи з протоколом менеджерів контексту призначено інструкцію with . . .as. Інструкція має такий формат:

with <Вираз 1>[as <Змінна>][, . . . , <Вираз N>[as <Змінна>]] :
  <Блок, у якому перехоплюємо винятки>

Спочатку обчислюється <вираз 1>, який має повертати об'єкт, який підтримує протокол. Цей об'єкт повинен підтримувати два методи: __enter__() та __exit__(). Метод __enter__() викликається після створення об'єкта. Значення, яке повертається цим методом, присвоюється змінною, вказаною після ключового слова as. Якщо змінна не вказана, значення, що повертається, ігнорується. Формат методу __enter__():

__enter__(self)

Далі виконуються інструкції всередині тіла with with. Якщо під час виконання виник виняток, то управління передається методу __exit__(). Метод має такий формат:

__exit__(self, <Тип виключення>, <Значення>, <Об'єкт traceback>)

Приклад:

```
class MyClass:
  def __enter__(self):
    print("Викликаний метод enter()")
    return self
  def __exit__ (self, Type, Value, Trace):
    print("Викликаний метод __exit__()")
    if Type is None: # Якщо виняток не виник
      print("Виняток не виникло")
    else: # Якщо виник виняток
      print("Value =", Value)
      return False # False - виняток не оброблено
                   # True — виняток опрацьовано
print("Послідовність за відсутності виключення:")
with MyClass():
  print("Блок усередині with")
print("\nПослідовність за наявності виключення:")
with MyClass() as obj:
```

```
print("Блок усередині with")
raise TypeError("Виключення TypeError")
```

Класи вбудованих винятків

Всі вбудовані винятки в мові Python є класами. Ієрархія вбудованих класів винятків схематично виглядає так:

```
BaseException
  SystemExit
  Keyboardlnterrupt
  GeneratorExit
  Exception
    Stoplteration
    ArithmeticError
      FloatingPointError, OverflowError, ZeroDivisionError
    Assert ionError
    AttributeError
    BufferError
    EOFError
    ImportError
    LookupError
      IndexError, KeyError
    MemoryError
    NameError
      UnboundLocalError
    OSError
      BlockinglOError
      ChildProcessError
      ConnectionError
        BrokenPipeError, ConnectionAbortedError,
        ConnectionRefusedError, ConnectionResetError
      FileExistsError
      FileNotFoundError
      InterruptedError
      IsADirectoryError
```

NotADirectoryError

PermissionError

ProcessLookupError

TimeoutError

RecursionError

ReferenceError

RuntimeError

NotImplementedError

SyntaxError

IndentationError

TabError

SystemError

TypeError

ValueError

UnicodeError

UnicodeDecodeError, UnicodeEncodeError

UnicodeTranslateError

Warning

BytesWarning, DeprecationWarning, FutureWarning, Importwarning,

PendingDeprecationWarning, ResourceWarning, RuntimeWarning,

SyntaxWarning, UnicodeWarning, UserWarning

Основна перевага використання класів для обробки винятків полягає у можливості вказівки базового класу для перехоплення всіх винятків відповідних похідних класів. Наприклад, для перехоплення поділу на нуль ми використовували клас ZeroDivisionError, але якщо замість нього вказати базовий клас ArithmeticError, перехоплюватимуться винятки класів FloatingPointError, OvertlowError та ZeroDivisionError.

Розглянемо основні класи вбудованих винятків:

- BaseException - є класом найвищого рівня та базовим для всіх інших класів винятків;
- Exception — базовий клас для більшості вбудованих у Python винятків. Саме його, а не BaseException необхідно успадковувати при створенні користувача класу виключення;
- AssertionError - порушується інструкцією assert;
- AttributeError - спроба звернення до неіснуючого атрибуту об'єкта;
- EOFError - збуджується функцією input () при досягненні кінця файлу;
- ImportError — Неможливо імпортувати модуль або пакет;
- IndentationError - неправильно розставлені відступи в програмі;

- IndexError - зазначений індекс не існує у послідовності;
- KeyError - вказаний ключ не існує у словнику;
- KeyboardInterrupt - натиснута комбінація клавіш <Ctrl>+<C>;
- MemoryError - інтерпретатору суттєво не вистачає оперативної пам'яті;
- NameError – спроба звернення до ідентифікатора до його визначення;
- NotImplementedError — має порушуватись в абстрактних методах;
- OSError - базовий клас для всіх винятків, що збуджуються у відповідь на виникнення помилок в операційній системі (відсутність файлу, що запрошується, брак місця на диску тощо);
- OverflowError - число, що вийшло в результаті виконання арифметичної операції, занадто велике, щоб Python зміг його обробити;
- RecursionError - перевищено максимальну кількість проходів рекурсії;
- RuntimeError - некласифікована помилка часу виконання;
- StopIteration - збуджується методом __next__() як сигнал про закінчення ітерацій;
- SyntaxError - синтаксична помилка;
- SystemError — помилка у самій програмі інтерпретатора Python;
- TabError - у вихідному коді програми зустрівся символ табуляції, використання якого для створення відступів неприпустимо;
- TypeError - тип об'єкта не відповідає очікуваному;
- UnboundLocalError - всередині функції змінної присвоюється значення після звернення до однойменної глобальної змінної;
- UnicodeDecodeError - помилка перетворення послідовності байтів у рядок;
- UnicodeEncodeError — помилка перетворення рядка на послідовність байтів;
- UnicodeTranslationError — помилка перетворення рядка на інше кодування;
- ValueError - переданий параметр не відповідає очікуваному значенню;
- ZeroDivisionError - спроба поділу на нуль.

Винятки користувача

Для збудження винятків користувача призначені дві інструкції: raise і assert.

Інструкція raise збуджує заданий виняток. Вона має кілька варіантів формату:

raise <Примірник класу>

raise <Назва класу>

raise <Примірник або назва класу> from <Об'єкт виключення>

raise

Приклад:

raise ValueError ("Опис виключення")

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

raise ValueError("Опис виключення")

ValueError: Опис виключення

```
try:
   raise ValueError("Опис виключення")
except ValueError as msg:
   print(msg) # Виведе: Опис виключення
```

Як виняток можна вказати екземпляр користувача класу.

Клас Exception підтримує всі необхідні методи виведення повідомлення про помилку. Тому в більшості випадків достатньо створити порожній клас, який успадковує клас Exception. Приклад:

```
class MyError(Exception): pass
try:
   raise MyError("Опис виключення")
except MyError as err:
   print(err) # Виведе: Опис виключення
```

У другому варіанті формату інструкції raise у першому параметрі задається об'єкт класу, а не екземпляр:

```
try:
   raise ValueError # Еквівалентно: raise ValueErrorO
except ValueError:
   print("Повідомлення про помилку")
```

У третьому варіанті формату інструкції raise у першому параметрі задається екземпляр класу або просто назва класу, а у другому параметрі вказується об'єкт виключення. І тут об'єкт виключення зберігається в атрибуте__cause__. При обробці вкладених винятків ці дані використовуються для виведення інформації не тільки про останній виняток, але і про початкове виключення. Приклад:

```
try:
   x = 1/0
except Exception as err:
   raise ValueError() від err
```

Четвертий варіант формату інструкції raise дозволяє повторно порушити останній виняток і зазвичай застосовується в коді, що йде за інструкцією except.

Інструкція assert збуджує виключення AssertionError, якщо логічний вираз повертає значення False. Інструкція має такий формат:

assert <Логічний вираз>[, <Дані>]

Інструкція assert еквівалентна наступному коду:

```
if __debug__:
   if not <Логічний вираз>:
      raise AssertionError(<Дані>)
```

Якщо під час запуску програми використовується прапорець -O, то змінна __debug__ матиме хибне значення. Так можна видалити всі інструкції assert з байт-коду.
Приклад:

```
try:
   x = -3
   assert x >= 0, "Повідомлення про помилку"
except AssertionError as err:
   print(err) # Виведе: Повідомлення про помилку
```

Хід роботи

1. Ознайомитись з необхідним теоретичним матеріалом.
2. Відповідно до варіанта завдання скласти та налагодити програму.
3. Оформити звіт.

Завдання

Модифікувати програму, складену в попередній лабораторній роботі №6, таким чином, щоб вона перевіряла дані, що вводяться з клавіатури на коректність і, у разі виявлення помилки, генерувала виняткові ситуації запитувала їх повторно шляхом обробки згенерованих виняткових ситуацій.

Завдання лабораторної роботи №6:

Модифікувати програму, складену у попередній лабораторній роботі №5, таким чином, щоб вона складала рейтинг студентів з кількох предметів та зберігала дані про них у файл.

Завдання лабораторної роботи №5:

Скласти програму, яка запитує дані з клавіатури, зберігає їх у полях розробленого класу, обробляє їх та виводить на консоль. Дані для введення: ПІБ студента, група, оцінка в балах. Обробка полягає у виклик методу розробленого класу, що здійснює перетворення значення поля оцінки з бальної системи в систему ECTS (A, B, C, D, E, FX, F). Дані для виведення повинні містити назви полів, а також їх значення. Кожне поле виводити окремим рядком.

Зміст звіту.

Звіт має містити такі пункти:

1.Титульний лист із темою роботи, номером варіанта.

2.Мета роботи.

3. Завдання за варіантом.

5.Текст програми із коментарями.

6.Результати роботи програми (контрольний приклад).

7.Висновки, що відображають труднощі та результати, досягнуті в ході роботи над програмою та алгоритмом, а також можливий аналіз шляхів альтернативного виконання поставленого завдання.

Контрольні запитання

1. До чого використовується в Python вбудована функція dir() без аргументів?
2. Що містить вбудований атрибут __dict__ у мові Python?
3. Який код потрібно використовувати, щоб відкрити файл c:\scores.txt для читання?
4. Чим відрізняються модулі Pickle та cPickle?
5. Який код потрібно використовувати, щоб записати інформацію у файл?

Література

1. Свейгарт Ел. Вчимо Python, роблячи круті ігри. М.: Ексмо, 2018, - 416 с
2. Златопольський Дмитро. Основи програмування мовою Python. М.: ДМК Прес, 2018, - 396 с
3. Доусон Майкл Програмуємо на Python. СПб.: Пітер, 2019, - 416 с

4. Саммерфілд Марк Програмування на Python 3. Детальний посібник. М.: Символ-Плюс, 2009, - 608 с
5. Шоу Зед Легкий спосіб вивчити Python 3. М: Ексмо, 2019, — 368 с
6. Мюллер Джон Поль Python для чайників. М.: Діалектика, 2019, - 416 с
7. Мусін М. Самовчитель Python v.0.2
8. Беррі Пол Вивчаємо програмування на Python. М.: Видавництво "Е", 2017. - 624с.
9. Прохоренок, Н. А. Python 3. Найнеобхідніше / Н. А. Прохоренок, В. А. Дронов. - 2-ге вид., перероб. та дод. - СПб.: БХВ-Петербург, 2019. - 608 с.:
10. Жуков Р.А. Мова програмування Python: практикум: навчань, посібник / Р.А. Жуків. - М.: ІНФРА-М, 2019. - 216с.
11. Жуков, Р. А. Мова програмування Python: практикум: навчальний посібник / Р.А. Жуків. - Москва: ІНФРА-М, 2021. - 216 с.