ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

СУЧАСНІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

**методичні рекомендації
до виконання практичних робіт**

Запоріжжя 2025

metinvest polytechnic

УДК 004.41(072)
С89

Рекомендовано Науково-методичною радою ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол №8 від 27.06.2025 р)

Укладачі:
Малигіна С.В., канд. техн. наук, доцент,

Бережна О.В., д-р техн. наук, професор

Рецензент:
Гетьман І.А.., канд. техн. наук, доцент

С89 Сучасні технології програмування : методичні рекомендації до виконання практичних робіт / уклад.: С. В. Малигіна, О. В. Бережна. Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2025. 46 с.

У методичних рекомендаціях наведено теоретичні відомості до виконання практичних робіт, приклади виконання, контрольні питання для самоперевірки здобутих знань, вимоги до оформлення, критерії оцінювання, перелік рекомендованої літератури.

УДК 004.41(072)

ЗМІСТ

ВСТУП

У даних рекомендаціях подано мету роботу, короткі теоретичні відомості, детальний алгоритм виконання з прикладами, питання до самоперевірки знань і завдання.

Якщо практична робота містить варінти завдань, то студент обирає завдання відповідно до його позиції в списку академічної групи. Практичні роботи виконуються кожним студентом самостійно.

Перед виконанням практичного завдання здобувачу необхідно повторити відповідні розділи теоретичного курсу згідно з лекційними записами та навчальною літературою. Основні теоретичні відомості стануть при цьому в нагоді.

Належним чином оформлений та завантажений в Moodle у відповідному місці звіт є документом, що підтверджує виконання студентом практичної роботи.

Зміст звіту з практичної роботи:

- титульний аркуш, оформлений у відповідності до додатку А;
- завдання на практичну роботу;
- вихідні дані за варіантом;
- алгоритм реалізації завдання;
- отримані результати;
- висновки щодо роботи

За 20 хвилин до закінчення заняття здобувач(ка) має представити викладачу результати практичної роботи, за необхідності внести виправлення та отримати бали за роботу. Оцінка за роботу на практичному занятті оголошується наприкінці заняття і може бути оскаржена відразу ж. Критерії оцінювання роботи на практичному занятті доводяться до відома здобувачів викладачем на першому практичному занятті.

ПРАКТИЧНА РОБОТА 1

ОСНОВИ ПРОЦЕДУРНОГО ПРОГРАМУВАННЯ В PYTHON

Мета роботи: ознайомитися із визначенням, організацією та обробкою масивів, а також основними структурами даних мови Python: рядками, множинами, словниками та кортежами.

Основні теоретичні відомості

Процедурне програмування — це підхід до розробки програм, у якому вирішення задачі організовано як послідовність команд або виклики функцій.

Головні ознаки:

- поділ програми на функції (підпрограми),
- використання змінних, умов і циклів,
- відсутність чіткої прив'язки до об'єктів (на відміну від ООП).

У Python цей підхід реалізується просто і гнучко завдяки динамічній типізації та читабельному синтаксису.

Мова Python дозволяє ефективно реалізовувати процедурну логіку за допомогою:

- функцій (def),
- умов (if, elif, else),
- циклів (for, while),
- базових і складених типів даних.

Масиви (списки) в Python

У Python роль масивів виконують списки (list), які є гнучкими та підтримують змішані типи даних.

Одновимірні масиви (списки), наприклад:

```
numbers = [1, 2, 3, 4, 5]
print(numbers[2])  # Виведе: 3
```

Елементи нумеруються з нуля.
Доступ: *array[index*]

Двовимірний список (список списків):

```
matrix = [
    [1, 2, 3],
    [4, 5, 6]
```

```
]
print(matrix[1][2])  # Виведе: 6
```

Доступ до елементів: **matrix[рядок][стовпець]**

Можна використовувати для представлення таблиць, сіток, зображень у пікселях тощо.

Обробка масивів:

- Ітерація: **for element in list:**
- Зміна елементів: **list[i] = new_value**
- Додавання: **list.append(x)**
- Видалення: **list.remove(x**) або **del list[i]**

Цикл для обробки елементів:

```
for row in matrix:
    for val in row:
        print(val, end=" ")
```

Рядки (str)

Рядок — це незмінна (immutable) послідовність символів.

```
text = "Привіт, Python!"
 print(text[0]) # 'П'
```

Методи:

- **text.lower(), text.upper()**
- **text.split(","), text.strip()**
- **len(text)** — довжина рядка

Рядки підтримують «слайси»

```
print(name[1:4])                    #                    yth
```

Кортежі (tuple)

Кортеж — це незмінна послідовність елементів.

```
point = (10, 20)
print(point[0]) # 10
```

Кортежі ефективніші за списки для фіксованих даних. Використовуються для:

- повернення кількох значень з функцій,
- збереження координат або пар значень.

Словники (dict)

Словник — це структура даних, яка зберігає пари **ключ-значення**, тобто це асоціативний масив, де значення зв'язуються з унікальними ключами. Основні операції над словником – збереження з указаним ключем та виведення за ним значення. Можна також видалити пари **key:value** за допомогою інструкції **del**. Під час зберігання нового значення із ключем, що вже використовується, старе значення видаляється. При спробі прочитати значення за ключем, якого немає в словнику, генерується виняткова ситуація **KeyError**.

Метод **keys()** для словника повертає список усіх використовуваних ключів у довільному порядку (якщо необхідно, щоб список був упорядкований, застосовують до нього метод sort()).

```
student = {
    "ім'я": "Марія",
    "вік": 20,
    "спеціальність": "Інформатика"
}
print(student["вік"])  # 20
```

Методи:

- dict.keys(), dict.values(), dict.items()
- Додавання: dict["new_key"] = value
- Перевірка наявності: 'key' in dict

Словники широко використовуються у конфігураціях, JSON-даних, аналізі тексту.

Множини (set)

Множина — це **невпорядкована** колекція **унікальних елементів**.

```
unique_numbers = {1, 2, 3, 2}
 print(unique_numbers) # {1, 2, 3}
```

Для множин визначені такі операції:

+ – об'єднання множин;

– – різниця множин;

* – перетин множин;

= – перевірка еквівалентності двох множин;

<> – перевірка нееквівалентності двох множин;

<= – перевірка, чи є ліва множина підмножиною правої множини;

> = – перевірка, чи є права множина підмножиною лівої множини;

in – перевірка, чи входить елемент, який зображено зліва, в множину, вказану справа.

Результатом операції об'єднання, різниці або перетину є відповідна множина, інші операції дають результат логічного типу. Наприклад:

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)  # об'єднання
print(a & b)  # перетин
print(a - b)  # різниця
```

Приклади використання в процедурному коді.
Приклад 1

```
def максимальний_елемент(список):
    return max(список)

numbers = [5, 1, 8, 2]
print(максимальний_елемент(numbers))  # 8
```

Приклад 2.

```
def додати_оцінку(журнал, студент, оцінка):
    if студент in журнал:
        журнал[студент].append(оцінка)
    else:
        журнал[студент] = [оцінка]

оцінки = {}
додати_оцінку(оцінки, "Андрій", 90)
додати_оцінку(оцінки, "Андрій", 85)
print(оцінки)  # {'Андрій': [90, 85]}
```

Приклад 3.

```
def average(numbers):
    return sum(numbers) / len(numbers)

def build_matrix(rows, cols):
    return [[0 for _ in range(cols)] for _ in range(rows)]

scores = [90, 85, 78, 92]
print("Середнє значення:", average(scores))
```

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()  # 0.844515910136422
random.randint(1, 10)  # 9
random.choice(['Life of Brian', 'Holy Grail', 'Life'])
# 'Life'
random.choice([1, 2, 3, 4])  # 3
```

До складу модуля *random* входять такі функції:

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

Таким чином, масиви, списки, кортежі, словники та множини — базові, але потужні засоби представлення даних у Python.

Використання процедури (функцій) забезпечують повторне використання, чистоту і логічність коду, а самі структури використовуються, наприклад, для змінних послідовностей (списки), для фіксованих наборів (кортежі), для текстових задач (рядки), для зберігання асоціативних пар (словники).

Завдання на практичну роботу

Запустити на виконання приклади 1-3., отримати результати, оформити звіт з виконання.

Порядок виконання роботи і опрацювання результатів:

Приклад 1.

Функція *range*

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

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

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

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

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

Приклад 2.

Нехай згенеровано список із цілих випадкових чисел та нулів [*a*1, …, *an*]. Написати програму визначення елементів, розміщених після першого нульового. Вивести на екран початковий та отриманий списки

```
import random
 arr = random.sample(range(-6, 6), 12)       print("our
 random list: ", arr)
flag = 0
while flag == 0:
     if 0 in arr:  # перевіряємо чи є 0 в списку first_zero_index =
         arr.index(0)                    # індекс першого 0 flag = 1;
    else:
         print("we have not at list one zero in list: ")
         arr1 = []
       if flag == 1:
        for i in arr[first_zero_index:]: arr1.append(i)
print(arr1)
```

Приклад 3.

Операції з множинами (*set; frozenset,* яка діє так само, як функція *set*, але формує *незмінну множину*)

```
my_set = {4, 5, 1, 2}
 # Кількість елементів множини
print('len({}) = {}'.format(my_set, len(my_set)))

  # Перевірка входження елемента
print(4 in my_set)
print(3 not in my_set)
print(9 in my_set)

# Чи перетинаються множини
print({3, 4, 5}.isdisjoint({8, 1, 0}))
print({3, 4, 5}.isdisjoint({1, 2, 3}))

# Перевірка включення однієї множини
print({1,     7,     9}.issubset({1,    2,     3, 7, 9}))
print({1,     7,     9} <= {1, 2, 3,    7,     9})
print({1,     7,     9, 2, 3} <= {1,    2,     3, 7, 9})

 # Перевірка чіткого включення
print({1, 7, 9} < {1, 2, 3, 7, 9})
print({1, 7, 9, 2, 3} < {1, 2, 3, 7, 9})

# Перевірка включення однієї множини в іншу
print({1,     2,     3,     4}.issuperset({1, 2}))
print({1,     2,     4,     4} >= {1, 2})
print({1,     2,     3,     4} >= {1, 2, 3, 4})

 # Перевірка чіткого включення
print({1, 2, 4, 4} > {1, 2})
print({1, 2, 3, 4} > {1, 2, 3, 4})

 # Об'єднання множин
print({1, 3}.union({2, 3, 4}))
print({1, 3} | {2, 3, 4})

# Перетин множин
print({1, 3}.intersection({2, 3, 4}))
print({1, 3} & {2, 3, 4})
```

```
# Різниця множин
print({1, 2, 3, 4}.difference({3, 4, 5}))
print({1, 2, 3, 4} - {3, 4, 5})
# Симетрична різниця
print({1, 2, 3, 4}.symmetric_difference({3, 4, 5, 6}))
print({1, 2, 3, 4} ^ {3, 4, 5, 6})

# Копіювання множин
my_set = set('chars')
copy = my_set.copy ()
print(copy)
```

Питання для самоперевірки

1) Що таке одновимірний масив, двовимірні масиви? Для чого їх використовують? Як їх описують в Python?
2) Для чого призначена функція range?
3) Як видалити елемент зі списку за значенням?
4) Як визначається кортеж, що містить один елемент?
5) Охарактеризуйте структуру даних "словник".
6) Які операції можна виконувати над множинами?
7) Назвіть літерали створення порожнього списку та словника?
8) Як створити порожню множину?

ПРАКТИЧНА РОБОТА 2

РОЗРОБКА ПРОГРАМ З ВИКОРИСТАННЯМ ПРОЦЕДУР І ФУНКЦІЙ

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

Основні теоретичні відомості

Оголошення функцій

Функція визначається за допомогою ключового слова ***def:***

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).

Тіло функції часто містить інструкцію ***return***:

```
def <name>(arg1, arg2,... argN):
        ...
   return <value>
```

Параметри та аргументи

Функції можуть приймати значення — аргументи, і повертати результат.

```
def добуток(a, b):
    return a * b
print(добуток(3, 4))  # Виведе: 12
```

Типи аргументів:

- **позиційні** (використовуються за порядком),
- **іменовані** (вказуються з іменами: dobutok(b=4, a=3)),
- **за замовчуванням** (def f(x=5): ...)
- **змінна кількість** (*args, **kwargs)

Глобальні та локальні змінні

- **Локальні** — створені всередині функції; недоступні за її межами.
- **Глобальні** — оголошені поза межами функції та доступні в будь-якому місці. Наприклад:

```
x = 10
def show():
    x = 5  # локальна змінна
    print(x)
show()     # 5
print(x)   # 10
```

Функції lambda

Lambda-вирази. В Python існує можливість створювати об'єкти функцій у формі виразів. Через схожість з аналогічною можливістю в мові LISP вона отримала назву *lambda*. Ця назва походить з мови програмування LISP, в якій ця назва була запозичена з лямбда–числення. Однак в Python це – ключове слово, яке вводить вираз синтаксично.

Подібно інструкції *def* цей вираз створює функцію, яку викличуть пізніше (але на відміну від інструкції *def*, вираз повертає функцію, а не зв'язує її з ім'ям).

Саме тому *lambda*-вирази іноді називають анонімними (тобто безіменними) функціями. Їх часто використовують, як спосіб отримати вбудовану функцію або відкласти виконання фрагмента програмного коду.

У загальному вигляді *lambda*- вираз складається з ключового слова *lambda*, за яким слідують один або більше аргументів (як список аргументів у круглих дужках у заголовку інструкції *def*) і далі, слідом за двокрапкою, розташовано вираз:

Результатом *lambda*–виразу є такі ж об'єкти функцій, які утворюють інструкції *def*, але тут є кілька відмінностей:

1) lambda – це вираз, а не інструкція;

2) тіло *lambda* – це не блок інструкцій, а один вираз.

lambda *argument1, argument2,... argumentN*:
вираз, який використовує аргументи

Переваги використання функцій

- **Повторне використання коду**.
- **Краще структурування** програми.
- **Тестування та налагодження** окремих функцій.
- **Можливість розбиття складних задач** на менші підзадачі.

Побудова програми з використанням функцій

Структура типової програмної системи з функціями:

```
def зчитати_дані():
    # зчитування даних
    ...
def обчислити_результат(дані):
    # обробка
    ...
def вивести_результат(результат):
    ...
def main():
    дані = зчитати_дані()
    результат = обчислити_результат(дані)
…вивести_результат(результат)
main()  # Точка входу
```

Декомпозиція завдань

Процедури та функції дають змогу реалізувати **декомпозицію** — поділ складної задачі на простіші блоки, наприклад:

Задача: Обчислити середнє значення чисел із введеного списку.

Декомпозиція:

- функція введення списку;
- функція знаходження середнього значення;
- функція виведення результату.

. Вбудовані та користувацькі функції

- **Вбудовані** (built-in): print(), len(), sum(), type()…
- **Користувацькі**: створюються програмістом за потреби.

Рекурсія

Функція може викликати саму себе — це називається **рекурсією**. Важливо мати **умову завершення**, щоб уникнути нескінченних викликів.

```
def факторіал(n):
    if n == 0:
        return 1
    return n * факторіал(n - 1)
```

Приклад анонімної функції (lambda):

```
# Додавання двох чисел
f = lambda a, b: a + b
print(f(3, 7))  # 10
# Квадрат числа
квадрат = lambda x: x**2
print(квадрат(5))  # 25
# Застосування в сортуванні
слова = ["яблуко", "груша", "банан"]
слова.sort(key=lambda x: len(x))
print(слова)  # ['груша', 'банан', 'яблуко']
```

Приклад . Описати функцію, яка обчислює множину значень

$$y = \sqrt[n]{x} \quad (x>0,\ n>0),$$

для цього використати рекурентну формулу Ньютона:

$$y_{k+1} = \frac{k-1}{k} y_k + \frac{x}{k \cdot y_k^{k-1}}, \quad k = 0, 1, \ldots, \ y_0 = \frac{x+n-1}{2}, \text{ я}$$

яка має місце для y0>0. Необхідну точність оцінюють співвідношенням

$$|y_{n+1} - y_n| \leq \varepsilon.$$

```
import itertools
import math
def my_sqrt(a,n,Epsilon):
        term1=(a+n-1)/2
    term2=(2/3)*term1 +(a/(3*term1**2))
    k=1
      while (abs(term2- term1) > Epsilon):
          term1= term2; term2=((k-1)/k)* term1 +a/(k*(term1**(k-1)))
              k+=1
```

```
        return term2

print(" x ", ": "," y ", " : "," yt", " : "," error " )
print(" ---------------------------------------------------- ")
Epsilon=0.0001
a=0.1;b=1
for i in itertools.count(start=a, step=0.2):
        if i>b: break
        n=3
        y= my_sqrt(i,n,Epsilon)
        y1=i**(1/n) # точне значення функції
        error=abs(y-y1)
        print('%.2f' %i, ":", '%.4f' %y, " : ", '%.4f' %y1, " :", '%.4f' %error)
```

```
 x  :   y   :   yt  :   error

0.10 : 0.9850  :  0.4642  : 0.5208
0.30 : 0.9891  :  0.6694  : 0.3197
0.50 : 0.9918  :  0.7937  : 0.1981
0.70 : 0.9942  :  0.8879  : 0.1063
0.90 : 0.9968  :  0.9655  : 0.0313
```

Завдання на практичну роботу

Описати функцію обчислення значень функції (визначити *m* значень заданої функції *f*(*x*) на відрізку [*a*, *b*]). Результат вивести на екран у вигляді табл.

Стовпчики таблиці:

1 – значення xi;

2 – значення функції f(xi), яке обчислено з використанням функцій інтерпретатора (модуль math);

3 – значення функції, яке обчислено за допомогою розкладання в ряд із точністю ε;

4 – точність обчислень;

5 – кількість ітерацій, необхідних для досягнення заданої точності.

Значення *a*, *b*, ε ввести з клавіатури.

Кількість точок на відрізку [a, b] складає не менше 10 (розбиття може бути рівномірним).

Необхідну точність оцінюють співвідношенням $|y_{n+1} - y_n| \le \varepsilon$

Використати для побудови розкладання елементарних функцій в ряд Тейлора.Довідкова інформація в таблиці 1.

Таблиця 1 – Довідкова інформація

Розкладання елементарних функцій в ряд Тейлора	Функція
1	2
$x-\frac{x^3}{3!}+\frac{x^5}{5!}-\ldots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}+\ldots$	$\sin(x),\ x\in\Re$
$1+\frac{x}{1!}+\frac{x^2}{2!}+\ldots+\frac{x^n}{n!}+\ldots$	$e^x,\ x\in\Re$
$x-\frac{x^2}{2!}+\frac{x^4}{4!}-\ldots+(-1)^n\frac{x^{2n}}{(2n)!}+\ldots$	$\cos(x),\ x\in\Re$
$x+\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots+\frac{x^{2n-1}}{(2n-1)!}+\ldots$	$\mathrm{sh}(x),\ x\in\Re$
$x+\frac{x^2}{2!}+\frac{x^4}{4!}+\ldots+\frac{x^{2n}}{(2n)!}+\ldots$	$\mathrm{ch}(x),\ x\in\Re$
$x-\frac{x^2}{2}+\frac{x^3}{3}+\ldots+(-1)^{n-1}\frac{x^n}{n}+\ldots$	$\ln(1+x),\ x\in(-1;\ 1]$
$x-\frac{x^3}{3}+\frac{x^5}{5}-\ldots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)}+\ldots$	$\mathrm{arctg}(x),\ x\in[-1;\ 1]$
$-x-\frac{x^2}{2}-\frac{x^3}{3}-\ldots-\frac{x^n}{n}-\ldots$	$\ln(1-x),\ x\in(-1;\ 1)$
$1-x+x^2-\ldots+(-1)^n x^n+\ldots$	$\frac{1}{1+x},\ x\in(-1;\ 1)$
$1+x+x^2+\ldots+x^n+\ldots$	$\frac{1}{1-x},\ x\in(-1;\ 1)$
$1+\alpha x+\frac{\alpha(\alpha-1)}{2!}x^2+\ldots+\frac{\alpha(\alpha-1)\ldots(\alpha-n+1)}{n!}x^n+\ldots$	$(1+x)^\alpha,\ x\in(-1;\ 1)$

Продовження таблиці 1.

Розкладання елементарних функцій в ряд Тейлора	Функція
1	2
$1-\frac{1}{2}x+\frac{3}{8}x^2-\frac{5}{16}x^3+\ldots+(-1)^{n-1}\frac{(2n-1)!!x^n}{(2n)!!}+\ldots,$	$\frac{1}{\sqrt{1+x}},\ x\in(-1;\ 1)$
$e^{2x}=1+2x+\frac{(2x)^2}{2!}+\frac{(2x)^3}{3!}+\ldots=\sum_{n=0}^{\infty}\frac{(2x)^n}{n!}$	e^{2x}
$e^{x^2}=1+x^2+\frac{x^4}{2!}+\frac{x^6}{3!}+\ldots=\sum_{n=0}^{\infty}\frac{x^{2n}}{n!}$	e^{x^2}
$1+(1+x)+\frac{(1+x)^2}{2!}+\frac{(1+x)^3}{3!}+\ldots=\sum_{n=0}^{\infty}\frac{(1+x)^n}{n!}$	$e^{(1+x)}$
$x\sin x=\frac{1}{1!}x^2-\frac{1}{3!}x^4+\frac{1}{5!}x^6-\frac{1}{7!}x^8+\ldots=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+2}$	x*sin(x)
$1+3x+3x^2+x^2$	$(1+x)^3$
$x^{1/2}+x^{5/2}+x^{9/2}+\ldots$	$\frac{\sqrt{x}}{1-x^2}$

Варіанти

1. $f(x) = e^{2x}$
2. $f(x) = e^{(1+x)}$
3. $f(x) = \dfrac{\sqrt{x}}{1-x^2}$
4. $f(x) = \dfrac{1}{\sqrt{1+x}}$
5. $f(x) = e^{4x}$
6. $f(x) = \dfrac{1}{1-x}$
7. $f(x) = \dfrac{1}{1+x}$
8. $f(x) = ln(1+x)$
9. $f(x) = ln(1-x)$
10. $f(x) = e^{x}$

Питання для самоперевірки

1.Як створити функцію у мові Python?
2.Що таке модулі та пакети?
3.Як бувають способи підключення модулів та пакетів?
4.Що таке анонімні функції та інструкція ***lambda***?
5.Який синтаксис оголошення функції?
6.Що таке позиційні та непозиційні аргументи функції?
7 Як створити функцію зі значеннями за замовчуванням?
8. Який порядок передачі аргументів у функцію, якщо вона містить позиційні та непозиційні аргументи та аргументи із значеннями за замовчуванням?
9. Для чого икористовується інструкція return? Чи обов’язково вона присутня у функції?
10. Що повертає функція, якщо в її тілі відсутня інструкція ***return?***

ПРАКТИЧНА РОБОТА 3

МЕХАНІЗМИ ОБРОБКИ ВИНЯТКІВ

Мета роботи: ознайомитися з механізмом обробки виняткових ситуацій.

Основні теоретичні відомості

Виняток (exception) — це подія, яка виникає під час виконання програми та порушує нормальний потік її виконання. Наприклад:

- поділ на нуль,
- звертання до неіснуючого індексу,
- відсутність файла.

```
try:
    # код, який може викликати помилку
except ExceptionType:
    # обробка помилки
```

Ключові компоненти наведені в таблиці 2.

Таблиця 2 - Ключові компоненти

Компонент	Призначення
try	Блокує код, у якому можуть виникнути винятки
except	Визначає, як реагувати на помилку
else	Виконується, якщо не було винятку
finally	Виконується завжди (навіть якщо сталася помилка)

Існує три типи помилок в програмі:

Синтаксичні – це помилки в імені оператора або функції, невідповідність закриваючих та відкриваючих лапок і т.д. Тобто помилки в синтаксисі мови. Як правило, інтерпретатор попередить про наявність помилки, а програма не виконуватиметься зовсім.

Семантичні – це помилки в логіці роботи програми, які можна виявити тільки за результатами роботи скрипта. Як правило, інтерпретатор не попереджає про наявність помилки. А програма буде виконуватися, оскільки не містить синтаксичних помилок. Такі помилки досить важко виявити і виправити.

Помилки часу виконання – це помилки, які виникають під час роботи скрипта. Причиною є події, які не передбачені програмістом. Класичним прикладом служить ділення на нуль

Наприклад:

```
try:
    x = int(input("Введіть число: "))
    y = 10 / x
except ZeroDivisionError:
    print("Помилка: Ділення на нуль!")
except ValueError:
    print("Помилка: Невірне значення!")
```

Створення власного винятку

Для визначення власних винятків необхідно використовувати класи, які детально розглядатимуться далі. Будь- який виняток є класом, зокрема, нащадком класу Exception.

Наприклад:

```
class CustomError(Exception):
    pass

def перевіритu(x):
    if x < 0:
        raise CustomError("x не може бути від'ємним")
```

Окрім стандартного використання блоків ***try-except***, є кілька інших можливостей для обробки помилок::

1. Множинні блоки `except`

```
try:
    num = int(input("Введіть число: "))
    result = 10 / num
except ValueError:
    print("Помилка: введено нечислове значення.")
except ZeroDivisionError:
    print("Помилка: ділення на нуль неможливе.")
```

2. Обробка кількох винятків в одному блоці

```
try:
    num = int(input("Введіть число: "))
    result = 10 / num
except (ValueError, ZeroDivisionError) as e:
    print(f"Помилка: {e}")
```

3. Використання блоку `else`

```
try:
    num = int(input("Введіть число: "))
    result = 10 / num
except ZeroDivisionError:
    print("Помилка: ділення на нуль.")
except ValueError:
    print("Помилка: введено неправильний тип даних.")
else:
    print(f"Результат: {result}")
```

4. Блок **`finally`**

```
try:
    file = open("test.txt", "r")
    data = file.read()
except FileNotFoundError:
    print("Файл не знайдено.")
finally:
    file.close()
    print("Файл закрито.")
```

5. Власні винятки

```
class CustomError(Exception):
    def __init__(self, message):
        super().__init__(message)

try:
    raise CustomError("Це власний виняток!")
except CustomError as e:
    print(f"Обробка власного винятку: {e}")
```

6. Перекидання винятків (`raise`)

```
try:
    raise ValueError("Симуляція помилки.")
except ValueError as e:
    print(f"Помилка оброблена: {e}")
    raise  # Піднімаємо виняток повторно
```

7. Контекстний менеджер `(with)`

```
try:
    with open("test.txt", "r") as file:
        data = file.read()
except FileNotFoundError:
    print("Файл не знайдено.")
```

8. Логування винятків

```
import logging

logging.basicConfig(level=logging.ERROR, filename="errors.log")

try:
    result = 10 / 0
except ZeroDivisionError as e:
    logging.error("Помилка ділення на нуль: %s", e)
```

Завдання на практичну роботу

Створення власного винятку

Приклад.

```
def integer_division():
    try:
        # Введення чисел
        n = int(input("Введіть чисельник (ціле число): "))
        d = int(input("Введіть знаменник (ціле число): "))

        # Виконання ділення
        result = n / d
        print(f"Результат ділення: {result}")

    # Обробка ділення на нуль
    except ZeroDivisionError:
        print("Помилка: ділення на нуль недопустиме.")

    # Обробка некоректного введення (нечислове значення)
    except ValueError:
        print("Помилка: введене значення має бути цілим числом.")

    # Обробка інших неочікуваних винятків
    except Exception as e:
```

```
        print(f"Неочікувана помилка: {e}")

    finally:
        print("Програма завершила роботу.")

# Виклик функції
integer_division()
```

Пояснення:

Блок try: Програма намагається виконати ділення цілих чисел, введених користувачем.

Обробка винятків:

- ***ZeroDivisionError***: Виникає при спробі ділення на нуль.
- ***ValueError***: Виникає при введенні некоректного значення (наприклад, тексту замість числа).
- ***Exception***: Обробка будь-яких інших неочікуваних помилок.

Блок finally: Виконується завжди, незалежно від того, чи виникли помилки.

Результат: Якщо всі введені дані коректні, програма виводить результат ділення.

Питання для самоперевірки

1. Як можна запобігти аварійному завершенню програми?
2. Які блоки коду записують у блок try? А які у except?
3. Поняття помилки.
4. Що таке виняткові ситуації і, яким чином здійснюється їх оброблення у Python?
5. Атрибути винятків, ініціювання винятків.
6. Для чого використовується гілка finally в інструкції try?
7. Чи можна гілку finally поєднувати з гілками except?
8. Чи можна перехоплювати різні класи помилок в одному блоці try/except?

ПРАКТИЧНА РОБОТА 4

РОЗРОБКА ПРОГРАМ З ВИКОРИСТАННЯМ КЛАСІВ В PYTHON

Мета роботи: ознайомитись з принципами реалізації об’єктно-орієнтованого програмування у мові Python та навчитись використовувати його для розроблення програмного забезпечення.

Основні теоретичні відомості

Розробка програм з використанням класів у Python, також відома як об'єктно-орієнтоване програмування (ООП), дозволяє структурувати код, представляючи реальні сутності як об'єкти, що містять дані (атрибути) та функції для роботи з цими даними (методи). Це робить код більш організованим, модульним та легким для підтримки.

Основні поняття ООП в Python:

- Клас (Class):

Шаблон або опис для створення об'єктів. Він визначає структуру та поведінку об'єктів.

- Об'єкт (Object):

Екземпляр класу. Це конкретна реалізація класу, що має певні значення атрибутів.

- Атрибути (Attributes):

Змінні, що зберігають дані про об'єкт. Наприклад, для класу "Людина" атрибутами можуть бути ім'я, вік, зріст.

- Методи (Methods):

Функції, визначені всередині класу, які описують дії, що можуть виконуватися над об'єктами цього класу.

- Конструктор (Constructor):

Спеціальний метод, який автоматично викликається при створенні нового об'єкта. У Python конструктор - це метод __init__.

- self:

Параметр, що посилається на поточний екземпляр класу. Він дозволяє звертатися до атрибутів та методів об'єкта всередині класу.

Приклад створення класу та об'єкта:

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
```

```
        self.breed = breed
        self.is_happy = True

    def bark(self):
        print("Гав! Мене звати", self.name)

    def change_mood(self, mood):
        if mood == "happy":
            self.is_happy = True
        elif mood == "sad":
            self.is_happy = False
my_dog = Dog("Борис", "Такса") # Створення об'єкта
print(f"Мій собака: {my_dog.name}, порода: {my_dog.breed}")
my_dog.bark() # Виклик методу
my_dog.change_mood("sad")
print(f"Собака щасливий: {my_dog.is_happy}")
```

Класові методи

Методи класу приймають клас в якості параметра, його прийнято позначати як cls. Він вказує на клас, а не на об'єкт цього класу. При декларації методів цього виду використовують декоратор classmethod.

Методи класу прив'язані до самого класу, а не його екземпляра. Вони можуть міняти стан класа, що відобразиться на усіх об'єктах цього класу, але не можуть міняти конкретний об`єкт.

Статичні методи

Статичні методи декларуються за допомогою декоратора @staticmethod. Їм не потрібно певного першого аргумента (ні self, ні cls).

Статичні методи можна викликати через об'єкт класу. Насправді статичному методу ніякі спеціальні аргументи (self чи cls) не передаються. Тобто статичні методи не можуть отримати доступ до параметрів класу чи об'єкта. Вони працюють тільки з тими даними, які їм передаються як аргументи.

Переваги використання класів:

- **Модульність та організація коду:**

ООП дозволяє розбити складні задачі на менші, більш керовані частини.

- **Повторне використання коду:**

Класи можна використовувати для створення багатьох об'єктів, що заощаджує час і зменшує дублювання коду.

- **Зручність у підтримці та розширенні:**

Зміни в одному класі не впливають на інші, що полегшує супровід програми.

- **Реалізація складної логіки:**
  ООП дозволяє моделювати складні системи та взаємодії між об'єктами

Завдання на практичну роботу

1.Ознайомитися з практичним використанням класу на прикладі роботи програми за наступними умовами.

2. Внести зміни до коду, застосувавши свої дані.

3. Оформити звіт

Розробити клас для представлення відомостей про успішність студента. Об'єкт класу має містити поля для збереження імені студента та балів, отриманих ним за виконання лабораторних робіт та лекційних занять.

Забезпечити наступні методи класу: конструктор, який приймає рядок ім'я студента та словник, що містить налаштування курсу у наступному форматі:

1) кількість лекційних занять в курсі;

2) максимально можлива кількість балів за здачу однієї лабораторної роботи;

3) кількість лабораторних робіт в курсі; метод, за допомогою якого вносяться оцінки за лабораторну роботу, який приймає параметри оцінку та номер лабораторної роботи; метод, за допомогою якого вносяться дані про кількість відвіданих лекцій; метод, який повертає дійсне число (суму балів студента за проходження курсу).

```
class StudentPerformance:
    def __init__(self, name, course_settings):
        """
        Конструктор класу.
        name: рядок, ім'я студента.
        course_settings: словник з налаштуваннями курсу:
            - "num_lectures": кількість лекційних занять у курсі.
            - "max_lab_score": максимально можлива кількість балів за одну лабораторну роботу.
            - "num_labs": кількість лабораторних робіт у курсі.
```

```
        """
        self.name = name
        self.course_settings = course_settings
        self.lab_scores = [0] * course_settings["num_labs"]
        self.attended_lectures = 0
    def add_lab_score(self, score, lab_number):
        """
        Внести оцінку за лабораторну роботу.
        score: оцінка.
        lab_number: номер лабораторної роботи (починається з 1).
        """
        if 1 <= lab_number <= self.course_settings["num_labs"]:
            self.lab_scores[lab_number - 1] = min(score, self.course_settings["max_lab_score"])
        else:
            raise ValueError("Невірний номер лабораторної роботи.")
    def add_attended_lectures(self, num_lectures):
        """
        Внести кількість відвіданих лекцій.
        num_lectures: кількість лекцій, які студент відвідав.
        """
        self.attended_lectures = min(num_lectures, self.course_settings["num_lectures"])
    def get_total_score(self):
        """
        Повернути загальну кількість балів студента за курс.
        """
        lecture_score = self.attended_lectures  # Кількість балів за лекції пропорційна відвідуванню.
        total_score = sum(self.lab_scores) + lecture_score
        return total_score
# ***Приклад використання***
if __name__ == "__main__":
    # Налаштування курсу
    course_settings = {
        "num_lectures": 20,
        "max_lab_score": 10,
        "num_labs": 5
    }
```

```
# Створення об'єкта класу
student = StudentPerformance("Олена Петренко", course_settings)

# Внесення оцінок
student.add_lab_score(8, 1)
student.add_lab_score(9, 2)
student.add_lab_score(10, 3)
# Внесення відвіданих лекцій
student.add_attended_lectures(18)
# Виведення загального балу
print(f"Загальна кількість балів студента {student.name}: {student.get_total_score()}")
```

```
Загальна кількість балів студента Олена Петренко: 45

=== Code Execution Successful ===
```

Питання для самоперевірки

1) Як створити клас у мові Python?
2) Що таке інкапсуляція?
3) Що таке об’єкт?
4) Що таке атрибут?
5) Для чого призначений конструктор класу?
6) Який обов’язковий аргумент приймає метод класу?
7) Що таке self?
8) Як оголосити метод класу статичним?
9) Як оголосити метод класу класовим методом?
10) Яка різниця між статичним і класовим методом?

ПРАКТИЧНА РОБОТА 5

РОЗРОБКА ПРОГРАМ З ІЄРАРХІЄЮ КЛАСІВ. ОРГАНІЗАЦІЯ КЛАСІВ З ВИКОРИСТАННЯМ УСПАДКУВАННЯ В PYTHON

Мета роботи: ознайомитися з особливостями реалізації наслідування атрибутів класу в ООП на мові Python

Основні теоретичні відомості

Інкапсуляція, наслідування та поліморфізм - це три основні принципи об'єктно-орієнтованого програмування (ООП). Вони допомагають організовувати код у більш структурований, гнучкий та повторно використовуваний спосіб.

Інкапсуляція полягає в об'єднанні даних (атрибутів) та методів, які працюють з цими даними, в межах одного класу. Вона також передбачає приховування деталей реалізації об'єкта від зовнішнього доступу, що забезпечує захист даних та спрощує використання об'єктів.

Наслідування дозволяє створювати нові класи (нащадки) на основі вже існуючих (батьківських). Нащадки успадковують атрибути та методи батьківського класу, а також можуть додавати власні унікальні характеристики та поведінку.

Поліморфізм (від грец. "багатоформність") дозволяє об'єктам різних класів, які можуть бути пов'язані між собою через наслідування, реагувати на один і той самий метод по-різному. Це забезпечує гнучкість та можливість обробки об'єктів різних типів через єдиний інтерфейс.

Таким чином:

- **Інкапсуляція**

приховує деталі реалізації, захищає дані та спрощує використання об'єктів.

- **Наслідування**

дозволяє створювати нові класи на основі існуючих, повторно використовуючи код та розширюючи функціональність.

- **Поліморфізм**

забезпечує гнучкість та можливість роботи з об'єктами різних типів через єдиний інтерфейс.

Ці три принципи ООП, разом з абстракцією, є фундаментом для створення складних, добре структурованих та підтримуваних програм.

Сетери, гетери і делетери

- **Гетери (getters):**

Це функції, які повертають значення приватного поля класу. Вони дозволяють отримати доступ до даних об'єкта, не розкриваючи внутрішню реалізацію класу. Зазвичай, ім'я гетера починається з "get", наприклад, get_age(), get_name().

- **Сетери (setters):**

Це функції, які змінюють значення приватного поля класу. Вони дозволяють контролювати, як і які значення можуть бути присвоєні полю. Зазвичай, ім'я сетера починається з "set", наприклад, set_age(new_age), set_name(new_name).

- **Делетери:**

Це методи, які видаляють поле з об'єкта. В деяких мовах, як наприклад Python, для видалення використовується ключове слово del

Використання гетерів та сетерів дозволяє:

- ✓ приховувати деталі реалізації класу від зовнішнього світу, що підвищує безпеку та зручність використання;
- ✓ обмежувати доступ до полів, дозволяючи змінювати їх лише через спеціально визначені методи;
- ✓ перевіряти коректність даних перед їх присвоєнням полю;
- ✓ виконувати додаткові дії при отриманні або зміні значення поля.

Приклад:

```
class Person:
    def __init__(self, name, age):
        self._name = name  # Приватне поле
        self._age = age    # Приватне поле

    def get_name(self):
        return self._name

    def set_name(self, new_name):
        if isinstance(new_name, str) and len(new_name) > 0:
            self._name = new_name
        else:
            print("Некоректне ім'я")

    def get_age(self):
        return self._age

    def set_age(self, new_age):
        if isinstance(new_age, int) and new_age >= 0:
            self._age = new_age
```

```
        else:
            print("Некоректний вік")

    def delete_name(self):
        del self._name

person = Person("Іван", 30)
print(person.get_name())
        person.set_name("Петро")
print(person.get_name())
person.set_age(35)
print(person.get_age())
person.delete_name()
```

Завдання на практичну роботу

1.Ознайомитися з практичним використанням класу на прикладі роботи програми за наступними умовами.

2. Внести зміни до коду за прикладом, застосувавши свої дані.

*Створіть клас **Bakery** з атрибутом **ingredient** та дочірній клас **Cake**, який буде виконувати метод **display** (виводить на екран фразу «*ingredient cupcake*»). В дочірньому класі створіть об'єкт **cupcake** і задайте через нього значення атрибуту **name** (наприклад, raspberry або blackberry) та зверніться до методу **display.***

3. Оформити звіт

Приклад завдання:

Створіть клас ***Animal*** з атрибутом name та методом ***eat***, та дочірній клас ***Dog***, який буде виконувати метод ***display*** (виводить на екран фразу «My name is….»). В дочірньому класі створіть об'єкт ***labrador*** і задайте через нього значення атрибуту ***name*** та зверніться до методів ***eat*** і ***display.***

Приклад рішення:

```
class Animal:
    # Атрибути і методи батьківського класу
    name = ""
    def eat(self):
        print("I can eat")

# Наслідуємо від класу Animal
class Dog(Animal):
    # Новий метод у дочірньому класі
```

```
        def display(self):
            # Доступ до атрибуту name батьківського класу за допомогою self
            print("My name is ", self.name)

    # Створюємо об’єкт дочірнього класу
    labrador = Dog()

    # Отримуємо доступ до атрибуту та методу батьківського класу
    labrador.name = "Rohu"
    labrador.eat()

    # Викликаємо метод дочірнього класу
    labrador.display()
```

Результат:

```
I can eat
My name is Rohu
```

Питання для самоперевірки

1. Що означає множинне спадкування?
2. В чому різниця між статичними методами і методами класу.
3. Що означає лінеаризація?
4. Як визначити, чи є певний клас підкласом іншого класу?
5. Як в Python оголосити атрибут прихованим?
6. Як отримати доступ до прихованого атрибуту?
7. Що таке властивість класу?
8. Як в Python оголосити один клас нащадком іншого?
9. Що таке лінеаризація?
10. Для чого призначений клас super?

ПРАКТИЧНА РОБОТА 6

РОБОТА З БАЗОЮ ДАНИХ ІЗ PYTHON-ПРОГРАМИ

Мета роботи**:** ознайомитися з організацією та розробити сховища даних з використанням парадигми ООП

Основні теоретичні відомості

Реляційна база даних – це набір таблиць із даними.

Таблиця – це прямокутна матриця, що складається з рядків та стовпців. Таблиця визначає відношення (relation).

Рядок – запис, що складається з полів-стовпців. У кожному полі може міститися деяке значення або спеціальне значення NULL (порожньо). У таблиці може бути довільна кількість рядків. Для реляційної моделі порядок розташування рядків не важливий і його не визначено.

Кожний стовпець у таблиці має власне ім'я й тип.

При програмуванні на Python доступ до бази даних не складніший за доступ до інших джерел даних (файлів, мережевих об'єктів). Для демонстрації обрано СКБД SQLite, що працює як під Unix, так і під Windows. Крім установлення власне SQLite (сайт http://sqlite.org) та модуля з'єднання з Python (http://pysqlite.org), останнє необхідно для старих версій Python, у версії 2.6 ця СКБД йде як убудований модуль, якогось додаткового налаштування виконувати не потрібно, тому що SQLite зберігає дані бази в окремому файлі. Тому відразу братися за створення таблиць, занесення до них даних та виконання запитів не можна. Обрана СКБД (у силу своєї "легкості") має одну істотну особливість – за одним невеликим винятком, СКБД SQLite не звертає уваги на типи даних (вона зберігає всі дані у вигляді рядків), тому модуль розширення sqlite3 для Python виконує додаткову роботу з перетворення типів.

Python добре кластеризує дані та дозволяє спростити їх обробку, що полегшує роботу.

Для роботи з базою даних на Python необхідно виконати наступні кроки:

1. Вибір та встановлення бібліотеки:

- Для роботи з різними базами даних використовуються різні бібліотеки. Наприклад, для SQLite використовується вбудована бібліотека sqlite3. Для MySQL - mysql.connector. Для PostgreSQL - psycopg2.
- Встановіть необхідну бібліотеку за допомогою pip: pip install <назва_бібліотеки>.

2. Встановлення з'єднання з базою даних:

- Створіть об'єкт підключення, використовуючи відповідну функцію з бібліотеки.
- Вкажіть параметри підключення: ім'я бази даних, ім'я користувача, пароль, хост та інші (якщо потрібно).

3. Виконання запитів:

- Створіть об'єкт курсора за допомогою методу cursor() об'єкта підключення.
- Виконайте SQL-запит за допомогою методу execute() об'єкта курсора.
- Результати запиту можна отримати за допомогою методів fetchone(), fetchmany(), або fetchall().

4. Обробка результатів:

- Отримані дані можуть бути оброблені у відповідності до вашої логіки.

5. Закриття з'єднання:

- Після завершення роботи з базою даних, закрийте з'єднання за допомогою методу close() об'єкта підключення.

Приклад роботи з базою даних SQLite:

```
import sqlite3

# Встановлення з'єднання
conn = sqlite3.connect('mydatabase.db')
cursor = conn.cursor()

# Створення таблиці
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY,
        name TEXT,
        age INTEGER
    )
''')

# Додавання даних
cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ('Іван', 30))
conn.commit() # Застосовуємо зміни
# Запит даних
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()
```

```
# Обробка результатів
for row in rows:
        print(row)
    # Закриття з'єднання
conn.close()
```

Завдання на практичну роботу

1. Ознайомитися з теоретичним матеріалом.
2. Створити та запустити на виконання приклад 1.
3. Додати в таблицю ще 3 записи. Отримати результат.
4. Оформити звіт

Приклад 1. Збереження даних в базі даних

```
import sqlite3


# Клас для працівників
class Worker:
    def __init__(self, **kwargs):
        self.name = kwargs.get('name')
        self.position = kwargs.get('position')
        self.work_from_date = kwargs.get('work_from_date')
        self.birth_date = kwargs.get('birth_date')


    def get_data(self):
        return (self.name, self.position, self.work_from_date,
self.birth_date)  # Виправлений return


# Створення екземпляра класу Worker
w1 = Worker(name='Petrov Vadym', position='manager',
work_from_date='09-12-2019', birth_date='18-02-1990')


conn = sqlite3.connect("staff_db.db")
cursor = conn.cursor()
```

```
# Створення таблиці
cursor.execute("""CREATE TABLE IF NOT EXISTS staff
(name TEXT, position TEXT, work_from_date TEXT, birth_date TEXT)""")

# Додавання записів
cursor.execute("INSERT INTO staff VALUES ('Ivanov Ivan', 'director', '07-10-2019', '02-12-1980')")
cursor.execute("INSERT INTO staff VALUES (?, ?, ?, ?)", w1.get_data())

# Масове вставлення
all_workers = [
    ('Borysov Mykola', 'engineer', '23-11-1976', '04-12-2019'),
    ('Pavlyik Inna', 'secretary', '12-09-1991', '22-01-2020'),
    ('Kolodych Leonid', 'engineer', '16-08-1986', '13-01-2020')
]
cursor.executemany("INSERT INTO staff VALUES (?, ?, ?, ?)", all_workers)

conn.commit()

# Виведення всіх записів
print("Записи в таблиці бази даних:")
cursor.execute("SELECT * FROM staff")
print(cursor.fetchall())
# Редагування запису для конкретного працівника
cursor.execute("UPDATE staff SET position = 'main engineer' WHERE name = 'Kolodych Leonid'")
conn.commit()
```

```
# Виводимо список всіх інженерів
print("Список всіх engineers:")
cursor.execute("SELECT * FROM staff WHERE position=?", ("engineer",))
print(cursor.fetchall())

# Виведення всіх записів із сортуванням
print("Список всіх записів в таблиці:")
for row in cursor.execute("SELECT rowid, * FROM staff ORDER BY name"):
    print(row)

conn.close()
```

Питання для самоперевірки

1. Дайте визначення поняттям "реляційна база даних", "таблиця", "рядок".
2. Які СКБД підтримує Python?
3. Що таке курсор (cursor)?

ПРАКТИЧНА РОБОТА 7

ПОБУДОВА ГРАФІКІВ МАТЕМАТИЧНИХ ФУНКЦІЙ У МОВІ PYTHON

Мета роботи: набуття навичок роботи з бібліотекою Matplotlib для візуалізації даних

Основні теоретичні відомості

Завантаження та установка http://matplotlib.org/

У сервісі https://ed-info.github.io/epython/ також доступна бібліотека matplotlib

Matplotlib – бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою (3D графіка також підтримується). Отримувані зображення можуть бути використані як ілюстрації в публікаціях. Зображення, які генеруються в різних форматах, можуть бути використані в інтерактивній графіці, наукових публікаціях, графічному інтерфейсі користувача, веб-додатках, де потрібно будувати діаграми (англ. plotting).

Бібліотека Matplotlib побудована на принципах ООП, але має процедурний інтерфейс ***pylab***, який надає аналоги команд MATLAB.

Пакет підтримує багато видів графіків і діаграм:

- Графіки (line plot)
- Діаграми розсіювання (scatter plot)
- Стовпчасті діаграми (bar chart) і гістограми (histogram)
- Секторні діаграми (pie chart)
- Діаграми «Стовбур-листя» (stem plot)
- Контурні графіки (contour plot)
- Поля градієнтів (quiver)
- Спектральні діаграми (spectrogram)

Набір підтримуваних форматів зображень, векторних і растрових, можна отримати словника **FigureCanvasBase.filetypes**.

Типові підтримувані формати: **EPS, EMF, JPEG, PDF, PNG, PostScript, RGBA, SVG, SVGZ, TIFF**

Налаштування вигляду графіків

Строковий аргумент ***r-.*** відповідальний за те, що змінився вигляд кривої. За замовчуванням цей аргумент ***b-*** що означає синю (blue) суцільну лінію.

Допустимі наступні значення (табл..3):

Таблиця 3 – варіанти налаштування вигляду графіку.

b, blue	синій колір
c, cyan	блакитний колір
g, green	зелений колір
k, black	чорний колір
r, red	червоний колір
w, white	білий колір
y, yellow	жовтий колір
-	суцільна лінія
--	штрихова лінія
-.	штрих-пунктирна лінія
:	пунктирна лінія

За замовчуванням легенда розташовується в правому верхньому куті, але можна її і перенести за рахунок аргументу ***loc***. Цьому аргументу можна привласнювати і чисельне значення, але звичайно легше сприймається рядок. У таблиці 4 нижче приводяться можливі варіанти

Таблиця 4 – Варіанти розташування легенди

Місце	String	Code
кращий варіант	best	0
вгорі справа	upper right	1
вгорі зліва	upper left	2
внизу справа	lower left	3
внизу зліва	lower right	4
справа	right	5
посередині зліва	center left	6
посередині справа	center right	7
посередині внизу	lower center	8
посередині вгорі	upper center	9
посередині	center	10

Маркери

Найбільш часто в наукових дослідженнях і журналах призводять графіки, що відрізняються один від одного саме маркерами, тому і в **matplotlib** для їх позначення є безліч способів:

- **. точковий маркер;**
- **, точки, розміром з піксель;**
- **o кола;**
- **v трикутники носом вниз;**

^ трикутники носом догори;
> трикутники дивляться вправо;
< трикутники дивляться вліво;
s квадрати;
p п’ятикутнки;
*** зірочки;**
h шестикутники;
H повернені шестикутники;
+ плюси;
x хрестики;
D ромби;
d вузькі ромби;
| вертикальні зарубки.

Додаткові аргументи plot()

В один аргумент можна поставити відразу три параметра: першим вказуємо колір, другим – стиль лінії, третім – тип маркера (табл 5).

Таблиця 5 - .Додаткові аргументи plot()

Keyword argument	**Що міняє**
color або c	колір лінії
linestyle	стиль лінії, використовуються позначення, показані вище
linewidth	товщина лінії у вигляді *float*-числа
marker	вид маркера
markeredgecolor	колір краю (edge) маркера
markeredgewidth	товщина краю маркера
markerfacecolorh	колір маркера
markeresize	Розмір маркера

Можна також вносити зміни у відмітки на осях координат. Робиться це за допомогою функцій ***xticks()*** і ***yticks(),*** в які передаються один або два списки значень: або просто список згаданих значень

Для нанесення сітки існує команда: ***plt.grid(True)***

Для того, щоб одну або декілька осей виставити в логарифмічному масштабі застосовуються команди ***plt.semilogx()*** и ***plt.semilogy().***

Збереження файлу

Файл зберігається в тій же директорії з ім'ям і розширенням, зазначеним в першому аргументі.

Види графіків модуля MatPlotLib

https://matplotlib.org/stable/plot_types/index.html

Наприклад:
plot.scatter - точки
plot.bar - стовпці
plot.barh - рядки
plot.pie - круг
plot.hist - гістограма

Завдання на практичну роботу

1.Ознайомитися з лекційним матеріалом та теоретичними відомостями до практичної роботи.

2. Ознайомитися з бібліотекою Matplotlib (***https://matplotlib.org/stable/)***

3. Оформити звіт, в якому дати відповідь на одне теоретичне питання на Ваш вибір.

Теоретичні питання:

1. Назвіть основні елементи управління (віджети) бібліотеки Tkinter.
2. Для чого використовується віджет Canvas () ?
3. Який метод віджету Tk() дозволяє задати заголовок вікна?
4. Яким чином можна відобразити графік функції?
5. За допомогою якого параметру можна змінити колір графіка?
6. Як додати легенду до графіка?
7. Як побудувати декілька кривих на одному графіку?

КРИТЕРІЇ ОЦІНЮВАННЯ РОБОТИ НА ПРАКТИЧНОМУ ЗАНЯТТІ

Максимальна кількість балів за роботу на практичному занятті – **5 балів.** Оцінка за роботу на практичному занятті оголошується наприкінці заняття і може бути оскаржена відразу ж.

Бали	*Критерії оцінювання*
5	Робота оформлена відповідно до поставлених вимог. Здобувач(ка) дає повну відповідь на поставлені викладачем питання; володіє узагальненими знаннями з предмету; уміє використовувати їх у різних ситуаціях, в тому числі вільно змінює відповідь на зміну вхідних умов; схильний(а) до критичного мислення, аналізу та прогнозування явищ і процесів; здатен(на) робити обґрунтовані висновки та узагальнення. Завантажив(ла) оформлений відповідно до вимог звіт в Moodle згідно з семестровим графіком.
4	Робота оформлена відповідно до поставлених вимог. Здобувач(ка) правильно виконав(ла) завдання, але допустив(ла) несуттєві неточності та помилки. Завантажив(ла) оформлений відповідно до вимог звіт в Moodle згідно з семестровим графіком. Під час захисту роботи продемонстрував(ла) володіння на достатньому рівні фаховою термінологією, відповів(ла) на більшість запитань викладача, пояснив(ла) переважну більшість наведених формул та розрахунків, їх складові та призначення.
3	Робота в цілому оформлена відповідно до поставлених вимог. Здобувач(ка) завдання виконав(ла) у повному обсязі, але допустив(ла) суттєві помилки, які призвели до викривлення результату, не зміг(ла) пояснити окремі етапи роботи. Завантажив(ла) оформлений відповідно до вимог звіт в Moodle згідно з семестровим графіком. Під час захисту роботи продемонстрував(ла) задовільне володіння фаховою термінологією, відповів(ла) на меншу частину запитань викладача, пояснив(ла) частину наведених формул та розрахунків.
2	Робота частково оформлена у відповідності до поставлених вимог. Здобувач(ка) виконав(ла) завдання у повному обсязі з грубими помилками, які спотворили результат, пояснити та обґрунтувати хід своїх думок не зміг(ла). Завантажив(ла) звітні матеріали в Moodle пізніше терміну вказаного у семестровому графіку. Продемонстрував(ла) незадовільне володіння фаховою термінологією, не відповів(ла) на запитання викладача, не в змозі пояснити наведені формули та розрахунки.
1	Не дотримано вимог щодо оформлення практичної роботи. Завдання виконав(ла) фрагментарно, не довів(ла) до логічного завершення, пояснити та обґрунтувати хід своїх думок не зміг(ла). Завантажив(ла) звітні матеріали в Moodle пізніше терміну вказаного у семестровому графіку.
0	Здобувач(ка) не виконав(ла) практичну роботу та не завантажив(ла) її в Moodle.

РЕКОМЕНДОВАНІ ДЖЕРЕЛА

Базові

1. Креневич А.П. Python у прикладах і задачах. Частина 2. Об’єктноорієнтоване програмування. Навчальний посібник – К.: ВПЦ "Київський Університет", 2019. – 152 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. – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 Маттес Е. Пришвидшений курс Python. – Л.: Видавництво Старого Лева, 2021. – 600 с.
3. Щербаков, О. В. Основи об'єктно-орієнтованого програмування [Електронний ресурс] : навч. посіб. / О. В. Щербаков, Ю. Е. Парфьонов, В. М. Федорченко ; Харківський національний економічний університет ім. С. Кузнеця. - Електрон. текстові дан. (2,13 МБ). - Харків : ХНЕУ ім. С. Кузнеця, 2019. - 236 с. : іл. - Загол. з титул. екрану. - Бібліогр.: с. 231-232.- Режим доступу: http://www.repository.hneu.edu.ua/handle/123456789/23847
4. Методичні вказівки до виконання лабораторних робіт з навчальної дисципліни «Програмування. Частина 2. Програмування мовою Python» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація та комп’ютерно-інтегровані технології» спеціальностей 151 «Автоматизація та комп’ютерно- інтегровані технології», 141 «Електроенергетика, електротехніка та електромеханіка» денної та заочної форм навчання [Електронне видання] / Присяжнюк О. В. – Рівне : НУВГП, 2020. – 165 с.

Додаткові

5. Мартін Р. Чистий кодер. – Х.: ФАБУЛА, 2023. – 256 с.
6. Васильєв О.М. Програмування мовою Python. – Л.: Bohdan Books, 2022. – 504 с.
7. Беррі П. Head First. Python. – Х.: ФАБУЛА, 2021. – 624 с.
8. Optimization problems and algorithms: Udemy: веб-сайт. URL: https://www.udemy.com/course/optimisation/ (дата звернення: 06.12.2024).

Web-ресурси

9 Патерни проектування [Електронний ресурс]. – Режим доступа : https://refactoring.guru/uk/design-patterns

ДОДАТОК А

Приклад титульного аркуша

ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
Кафедра автоматизації, електро- та робототехнічних систем

Практична робота ___

з навчальної дисципліни
«СУЧАСНІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ»

Назва роботи_____

Варіант №_

Здобувача групи ___
Прізвище Ім’я По батькові

Викладач:
к.т.н., доцент
С.В. Малигіна

Запоріжжя, 20ХХ

Навчально-методичне видання

Малигіна Світлана Валеріївна
Бережна Олена Валеріївна

СУЧАСНІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

методичні рекомендації
до виконання практичних робіт

Самостійне електронне мережеве видання

Публікується в авторський редакції