**Звіт про виконання практичних завдань до лекцій з курсу Технології програмування на мові Python**

Звіт до Теми №1

Функції та змінні

Під час виконання практичного завдання до Теми №1 було надано варіанти рішення до наступних задач:

**Перетворення рядка**

Необхідно рядок, що має вигляд "abcdefg123" перетворити наступним чином "321gfedcba", вважаючи сталою довжину рядку в 10 символів.

Хід виконання завдання:

В ПЗ пишем код:

Крок 1: Створюємо функцію 'revers', яка містить рядок 's' зі значенням "abcdefg123"

Крок 2: Перевертаємо рядок 's' за допомогою зрізу '[::-1]' і зберігаємо результат у змінну 'result'. (Перший порожній елемент означає "почни з першого символу" (або з кінця, бо ми йдемо у зворотному напрямку). Другий порожній елемент означає "йди до останнього символу" та -1 вказує на те, що ми рухаємося у зворотному порядку.

Крок 3: Функція повертає перевернутий рядок, який зберігається в 'result'

Текст програми:

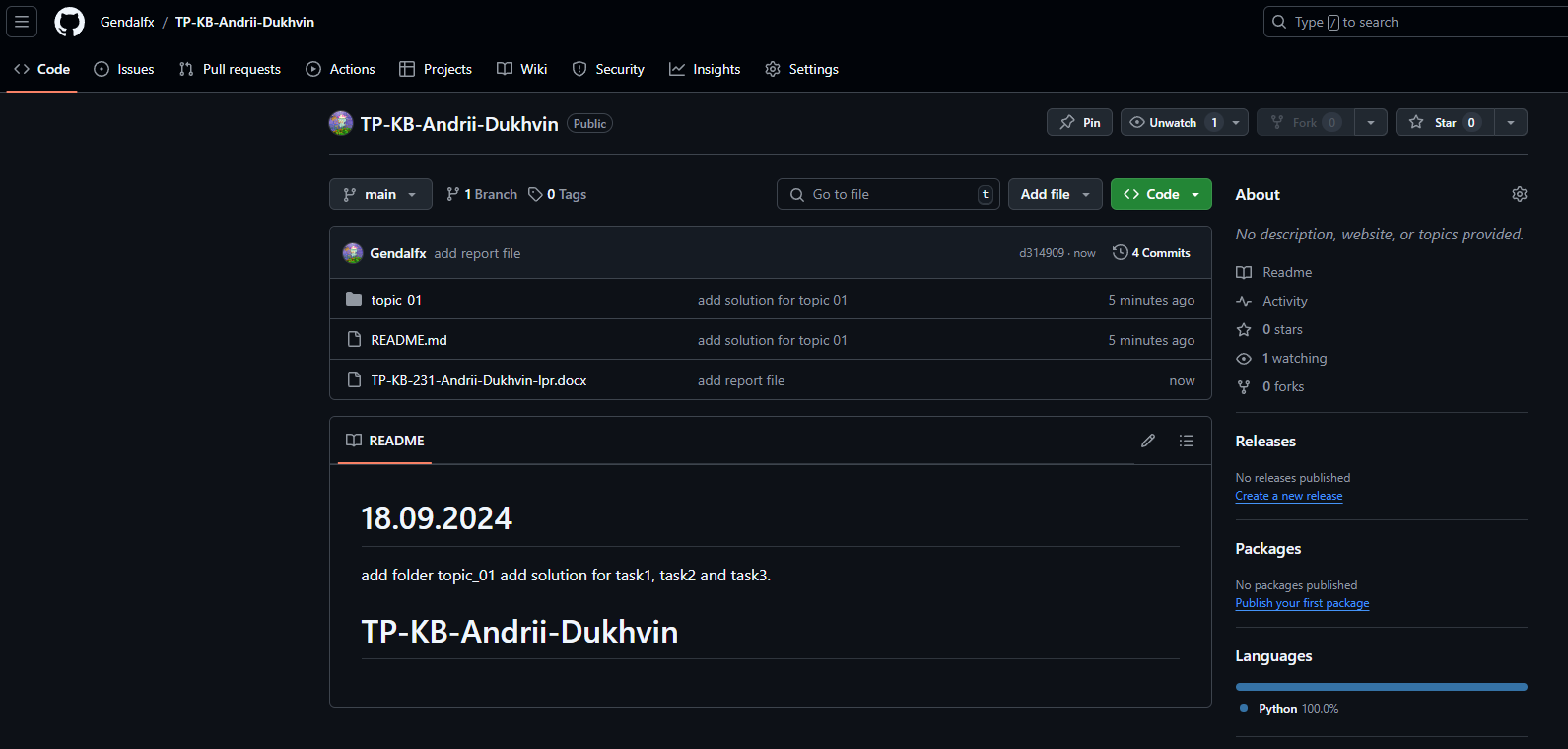
|  |
| --- |
| def revers():      s ="abcdefg123"      result = s[::-1]      return result  print (revers()) |

При компіляції він виводить:



Посилання на github: https://github.com/Gendalfx/TP-KB-Andrii-Dukhvin

Знімок екрану з посилання на github:



**Виконати тестування функцій, що працюють з рядками: strip(), capitalize(), title(), upper(), lower().**

Текст програми:

def test\_string\_functions():

    s = "   hello world! welcome to python.   "

    # strip() - Видаляє пробіли з обох боків рядка

    stripped = s.strip()

    print(f"strip():'{stripped}'")

   # capitalize() - Робить першу літеру великою, а інші маленькими

    capitalized = stripped.capitalize()

    print(f"capitalize():'{capitalized}'")

    # title() - Робить першу літеру кожного слова великою

    titled = s.title()

    print(f"title():'{titled}'")

    # upper() - Перетворює всі символи в рядку на великі

    uppercased = s.upper()

    print(f"upper():'{uppercased}'")

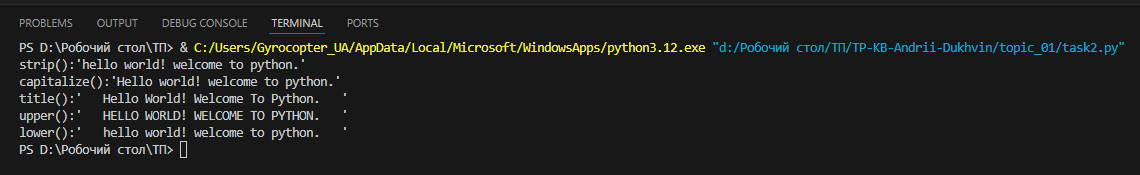
    # lower() - Перетворює всі символи в рядку на маленькі

    lowercased = s.lower()

    print(f"lower():'{lowercased}'")

test\_string\_functions()

При компіляції він виводить:



**Написати функцію пошуку дискримінанту квадратного рівняння.**

Формула дискримінанту: D=b^2−4ac

Текст програми:

def find\_discriminant(a, b, c):

    # Обчислюємо дискримінант

    discriminant = b\*\*2 - 4\*a\*c

    return discriminant

# Приклад використання

a = float(input("Введіть коефіцієнт a: "))

b = float(input("Введіть коефіцієнт b: "))

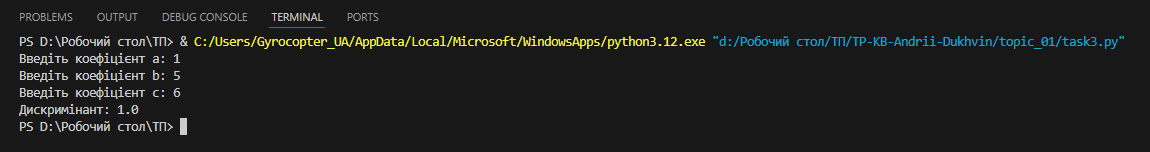
c = float(input("Введіть коефіцієнт c: "))

# Виклик функції та виведення результату

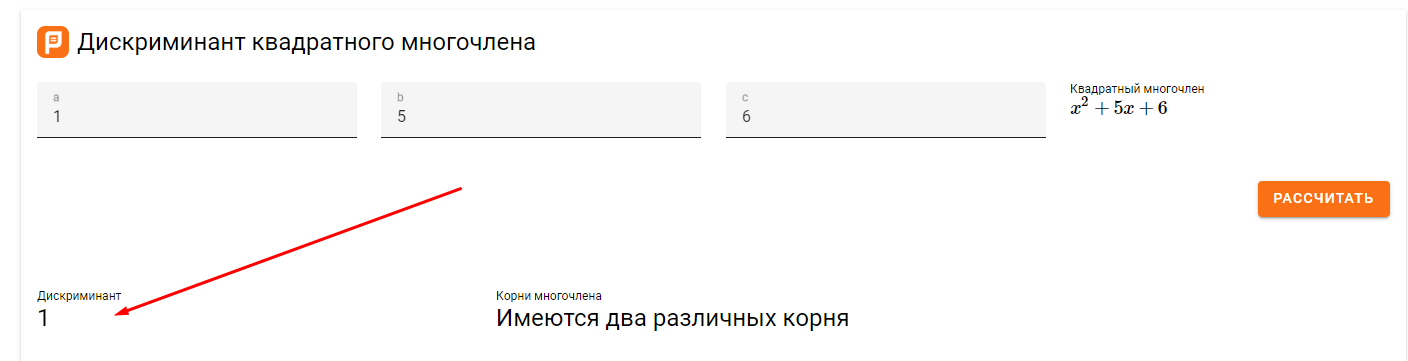
D = find\_discriminant(a, b, c)

print(f"Дискримінант: {D}")

При компіляції він виводить:



Перевірка:



Звіт до Теми №2

Умовний перехід

Під час виконання практичного завдання до Теми №2 було надано варіанти рішення до наступних задач:

**Завдання 1**

Написати функцію пошуку коренів квадратного рівняння використовуючи функцію розрахунку дискримінанту з попередньої теми та умовні переходи.

Хід роботи

**Визначення задачі**:

Програма повинна розв’язувати квадратні рівняння у формі ax^2+bx+c=0.

**Формули**:

* Обчислення дискримінанта: D=b2−4acD.
* Визначення коренів на основі значення дискримінанта:
  + Два дійсних корені, якщо D>0.
  + Один дійсний корінь, якщо D=0.
  + Два комплексних корені, якщо D<0.

**Написання функцій**:

* **Функція для обчислення дискримінанта**: приймає aaa, bbb, ccc і повертає дискримінант.
* **Функція для знаходження коренів**: використовує значення дискримінанта для обчислення коренів.

**Основна програма**:

* Запитує у користувача значення коефіцієнтів a, b, c.
* Викликає функцію для знаходження коренів і виводить результати.

Текст програми:

import math

def find\_discriminant(a, b, c):

    # Обчислюємо дискримінант

    discriminant = b\*\*2 - 4\*a\*c

    return discriminant

def find\_roots(a, b, c):

    # Знаходимо дискримінант

    D = find\_discriminant(a, b, c)

    if D > 0:

        # Два різних дійсних корені

        x1 = (-b + math.sqrt(D)) / (2\*a)

        x2 = (-b - math.sqrt(D)) / (2\*a)

        return x1, x2

    elif D == 0:

        # Один дійсний корінь

        x = -b / (2\*a)

        return x,

    else:

        # Комплексні корені

        real\_part = -b / (2\*a)

        imaginary\_part = math.sqrt(-D) / (2\*a)

        x1 = complex(real\_part, imaginary\_part)

        x2 = complex(real\_part, -imaginary\_part)

        return x1, x2

# Приклад використання

a = float(input("Введіть коефіцієнт a: "))

b = float(input("Введіть коефіцієнт b: "))

c = float(input("Введіть коефіцієнт c: "))

# Виклик функції та виведення результату

roots = find\_roots(a, b, c)

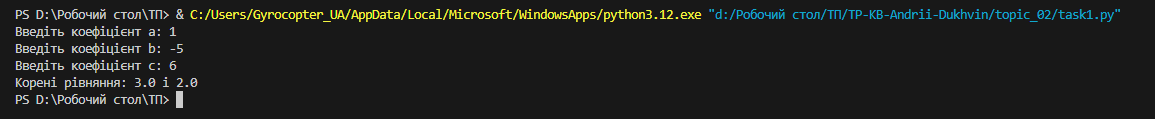
if len(roots) == 1:

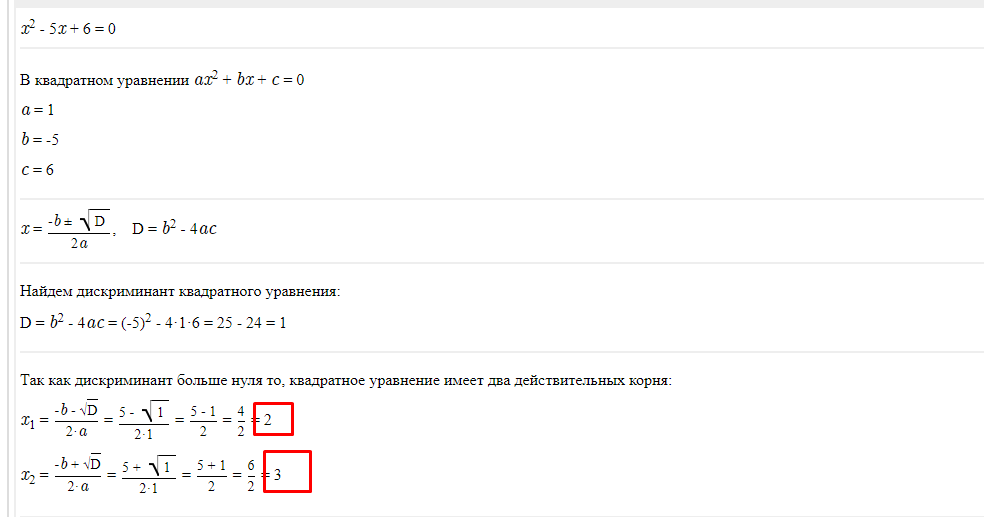
    print(f"Корінь рівняння: {roots[0]}")

else:

    print(f"Корені рівняння: {roots[0]} і {roots[1]}")

Перевірка:





**Завдання 2**

Написати програму калькулятор використовуючи **if else** конструкцію. Кожна операція має бути виконана в окремій функції.

Хід роботи

**Визначення задачі**:

Створити простий калькулятор, який виконує основні арифметичні операції (додавання, віднімання, множення, ділення).

**Структура програми**:

* Реалізувати функції для кожної арифметичної операції: add, subtract, multiply, divide.
* Функція divide повинна обробляти випадок ділення на нуль.

**Написання функцій**:

* **Функція add**: приймає два аргументи і повертає їхню суму.
* **Функція subtract**: приймає два аргументи і повертає результат віднімання другого з першого.
* **Функція multiply**: приймає два аргументи і повертає їхній добуток.
* **Функція divide**: перевіряє, чи другий аргумент не дорівнює нулю, і повертає результат ділення або повідомлення про помилку.

**Основна програма**:

* Запитує у користувача ввести два числа та вибрати операцію.
* Використовує умовні конструкції if-elif-else, щоб викликати відповідну функцію на основі вибраної операції.
* Виводить результат виконаної операції або повідомлення про помилку для невідомої операції.

Текст програми:

# Функції для операцій

def add(a, b):

    return a + b

def subtract(a, b):

    return a - b

def multiply(a, b):

    return a \* b

def divide(a, b):

    if b != 0:

        return a / b

    return "Ділення на нуль!"

# Основна програма

def calculator():

    num1 = float(input("Введіть перше число: "))

    num2 = float(input("Введіть друге число: "))

    operation = input("Оберіть операцію (+, -, \*, /): ")

    if operation == '+':

        print("Результат:", add(num1, num2))

    elif operation == '-':

        print("Результат:", subtract(num1, num2))

    elif operation == '\*':

        print("Результат:", multiply(num1, num2))

    elif operation == '/':

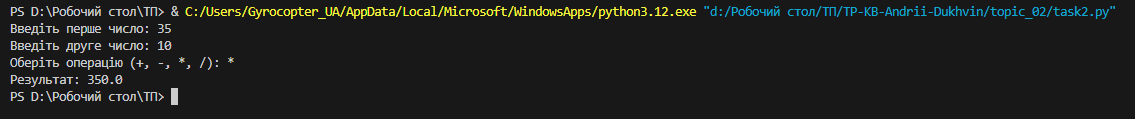
        print("Результат:", divide(num1, num2))

    else:

        print("Невідома операція")

calculator()

Перевірка:





**Завдання 3**

Написати програму калькулятор використовуючи **match** конструкцію. Кожна операція має бути виконана в окремій функції.

Хід роботи

**Визначення задачі**: Розробити калькулятор, який виконує основні арифметичні операції (додавання, віднімання, множення, ділення) за допомогою конструкції match.

**Структура програми**:

* Реалізувати функції для кожної арифметичної операції: add, subtract, multiply, divide.
* У функції divide обробити випадок ділення на нуль, щоб уникнути помилок.

**Написання функцій**:

* **Функція add**: приймає два аргументи і повертає їхню суму.
* **Функція subtract**: приймає два аргументи і повертає результат віднімання другого з першого.
* **Функція multiply**: приймає два аргументи і повертає їхній добуток.
* **Функція divide**: перевіряє, чи другий аргумент не дорівнює нулю, і повертає результат ділення або повідомлення про помилку.

**Основна програма**:

* Запитує у користувача ввести два числа та вибрати арифметичну операцію.
* Використовує конструкцію match для вибору відповідної функції на основі вибраної операції.
* Виводить результат виконаної операції або повідомлення про помилку для невідомої операції.

Текст програми:

# Функції для операцій

def add(a, b):

    return a + b

def subtract(a, b):

    return a - b

def multiply(a, b):

    return a \* b

def divide(a, b):

    if b != 0:

        return a / b

    return "Помилка: ділення на нуль!"

# Основна програма

def calculator():

    num1 = float(input("Введіть перше число: "))

    num2 = float(input("Введіть друге число: "))

    operation = input("Оберіть операцію (+, -, \*, /): ")

    match operation:

        case '+':

            print("Результат:", add(num1, num2))

        case '-':

            print("Результат:", subtract(num1, num2))

        case '\*':

            print("Результат:", multiply(num1, num2))

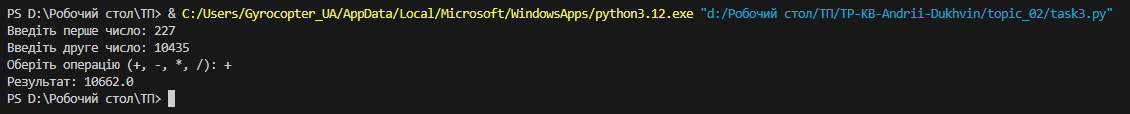
        case '/':

            print("Результат:", divide(num1, num2))

        case \_:

            print("Невідома операція")

calculator()

Перевірка:  




Звіт до Теми №3

Цикли

Під час виконання практичного завдання до Теми №1 було надано варіанти рішення до наступних задач:

1. Написати програму калькулятор з постійними запитами на введення нових даних та операцій. За основу взяти програму калькулятор з попередньої теми. Реалізувати механізм завершення програми після отримання відповідної команди.
2. Написати програму тестування функцій списків таких як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy()
3. Написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items()
4. Маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список.

Хід роботи

Завдання 1:

Написати програму калькулятор з постійними запитами на введення нових даних та операцій. За основу взяти програму калькулятор з попередньої теми. Реалізувати механізм завершення програми після отримання відповідної команди.

1. Створюємо безкінечний цикл, який продовжує працювати, поки користувач не введе команду для завершення програми.

**Запит на введення чисел**:

* Запитуємо у користувача перше число (num1) або можливість вийти “q”.
* Запитуємо у користувача друге число (num2) або можливість вийти “q”.

**Запит на вибір операції**:

* Запитуємо у користувача, яку математичну операцію він хоче виконати: +, -, \*, /.
* Додаємо можливість ввести команду для виходу "q".

**Запит на нові дані**:

* Після виконання операції запитуємо у користувача, чи бажає він виконати ще одну операцію.
* Якщо так, цикл повторюється, і програма знову запитує числа та операцію.

**Завершення програми**:

* Якщо користувач ввів команду для виходу, програма виводить прощальне повідомлення і завершує свою роботу.

Текст програми:

# Функції для операцій

def add(a, b):

    return a + b

def subtract(a, b):

    return a - b

def multiply(a, b):

    return a \* b

def divide(a, b):

    if b != 0:

        return a / b

    return "Помилка: ділення на нуль!"

# Основна програма

def calculator():

    while True:

        # Запит на введення числа або команди для завершення

        user\_input = input("Введіть перше число: або 'q': ")

        if user\_input.lower() == 'q':

            print("Програма завершена.")

            break

        try:

            num1 = float(user\_input)

        except ValueError:

            print("Введіть перше число: або 'q': ")

            continue

        # Запит на введення другого числа

        user\_input = input("Введіть друге число: або 'q': ")

        if user\_input.lower() == 'q':

            print("Програма завершена.")

            break

        try:

            num2 = float(user\_input)

        except ValueError:

            print("Введіть перше число: або 'q': ")

            continue

        # Запит на вибір операції

        operation = input("Оберіть операцію (+, -, \*, /): ")

        if operation.lower() == 'q':

            print("Програма завершена.")

            break

        match operation:

            case '+':

                print("Результат:", add(num1, num2))

            case '-':

                print("Результат:", subtract(num1, num2))

            case '\*':

                print("Результат:", multiply(num1, num2))

            case '/':

                print("Результат:", divide(num1, num2))

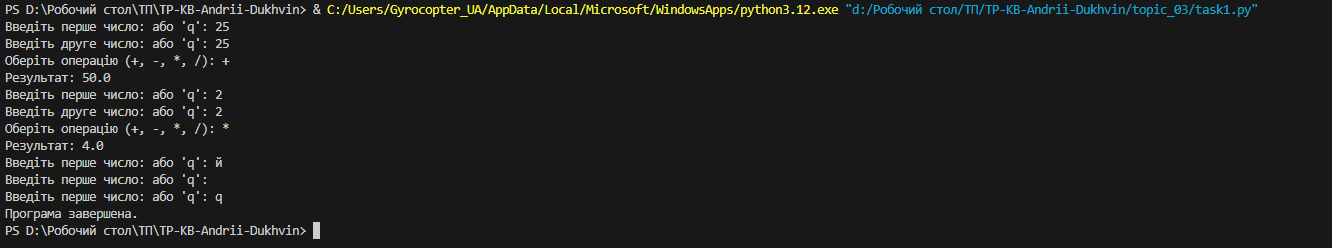
            case \_:

                print("Невідома операція")

# Виклик програми

calculator()

При компіляції виводить:



Завдання 2

Написати програму тестування функцій списків таких як: extend(), append(), insert(id, val), remove(val), clear(), sort(), reverse(), copy().

**Опис програми:**

1. **Початковий список**: Створюється початковий список my\_list з кількома значеннями.
2. **Тестування функцій**:
   * append(value): Додає значення в кінець списку.
   * extend(iterable): Додає всі елементи ітерованого об'єкта до списку.
   * insert(index, value): Вставляє значення на вказану позицію в списку.
   * remove(value): Видаляє перше входження значення зі списку.
   * clear(): Очищає весь список.
   * sort(): Сортує список за зростанням.
   * reverse(): Реверсує порядок елементів у списку.
   * copy(): Створює поверхневу копію списку.

Текст програми:

def test\_list\_functions():

    # Початковий список

    my\_list = [1, 2, 3, 4]

    print("Початковий список:", my\_list)

    # 1. append()

    my\_list.append(5)

    print("Після append(5):", my\_list)

    # 2. extend()

    my\_list.extend([6, 7])

    print("Після extend([6, 7]):", my\_list)

    # 3. insert(index, value)

    my\_list.insert(0, 0)  # Вставка 0 на початок списку

    print("Після insert(0, 0):", my\_list)

    # 4. remove(value)

    my\_list.remove(3)  # Видалення значення 3

    print("Після remove(3):", my\_list)

    # 5. clear()

    my\_list.clear()

    print("Після clear():", my\_list)

    # Повернемо список для подальшого тестування

    my\_list = [3, 1, 4, 2]

    # 6. sort()

    my\_list.sort()

    print("Після sort():", my\_list)

    # 7. reverse()

    my\_list.reverse()

    print("Після reverse():", my\_list)

    # 8. copy()

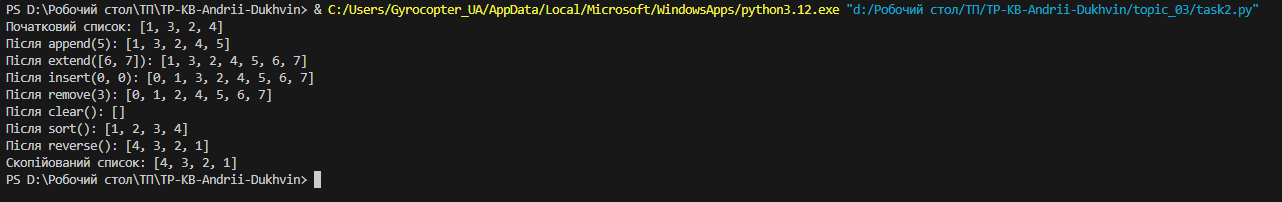
    copied\_list = my\_list.copy()

    print("Скопійований список:", copied\_list)

# Виклик функції тестування

test\_list\_functions()

Після компіляції:



**Завдання 3**

Написати програму тестування функцій словників таких як: update(), del(), clear(), keys(), values(), items()

**Опис програми:**

1. **Початковий словник**: Створюється початковий словник my\_dict з кількома парами ключ-значення.
2. **Тестування функцій**:
   * update(other\_dict): Оновлює значення ключів з іншого словника або додає нові ключі.
   * del: Видаляє пару ключ-значення зі словника за вказаним ключем.
   * clear(): Очищає весь словник, видаляючи всі пари ключ-значення.
   * keys(): Повертає об'єкт dict\_keys, що містить усі ключі словника.
   * values(): Повертає об'єкт dict\_values, що містить усі значення словника.
   * items(): Повертає об'єкт dict\_items, що містить усі пари (ключ, значення) словника.

Текст програми:

def test\_dict\_functions():

    # Початковий словник

    my\_dict = {

        "name": "Andrii",

        "age": 25,

        "city": "Chernihiv"

    }

    print("Початковий словник:", my\_dict)

    # 1. update()

    my\_dict.update({

        "age": 26, "country": "Ukraine"

    })  # Оновлюємо вік та додаємо країну

    print("Після update:", my\_dict)

    # 2. del

    del my\_dict["city"]  # Видаляємо ключ "city"

    print("Після del: ", my\_dict)

    # 3. clear()

    my\_dict.clear()  # Очищаємо словник

    print("Після clear: ", my\_dict)

    print("Повернемо словник для подальшого тестування")

    # Повернемо словник для подальшого тестування

    my\_dict = {

        "name": "Andrii",

        "age": 25,

        "city": "Chernihiv"

    }

    # 4. keys()

    keys = my\_dict.keys()  # Отримуємо ключі словника

    print("Ключі словника: ", list(keys))

    # 5. values()

    values = my\_dict.values()  # Отримуємо значення словника

    print("Значення словника: ", list(values))

    # 6. items()

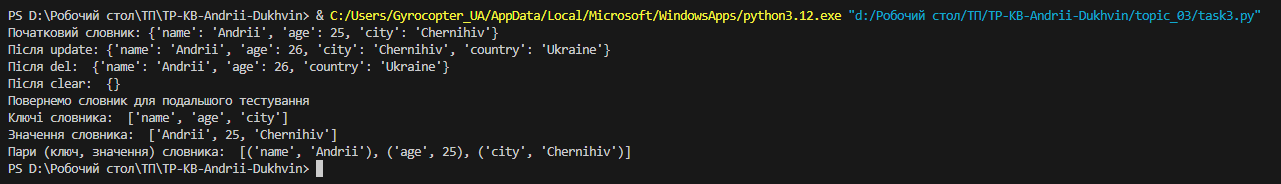
    items = my\_dict.items()  # Отримуємо пари (ключ, значення)

    print("Пари (ключ, значення) словника: ", list(items))

# Виклик функції тестування

test\_dict\_functions()

Після компіляції:



**Завдання 4**

Маючи відсортований список, написати функцію пошуку позиції для вставки нового елементу в список.

Щоб знайти позицію для вставки нового елемента в відсортований список, можна використовувати алгоритм бінарного пошуку.

**Опис алгоритму:**

1. **Ініціалізація**:
   * low = 0: початковий індекс (ліва межа) списку.
   * high = len(sorted\_list): початковий індекс (права межа) списку, що дорівнює кількості елементів у списку.
2. **Цикл while**:
   * Цикл виконується, поки low менше high. Це означає, що в межах діапазону low до high все ще залишаються елементи для розгляду.
3. **Обчислення середнього індексу**:
   * mid = (low + high) // 2: обчислює середній індекс списку. // — це оператор цілочисельного ділення, який ділить націло (без залишку).
4. **Порівняння**:
   * if sorted\_list[mid] < new\_element: перевіряє, чи елемент у середньому індексі менший за новий елемент.
     + Якщо так, то low = mid + 1: це означає, що новий елемент повинен бути у верхній половині списку (всі елементи до mid вже менші).
   * else: high = mid: якщо елемент у середньому індексі не менший, то новий елемент повинен бути у нижній половині (включаючи mid).
5. **Повернення значення**:
   * return low: коли цикл закінчується, low вказує на позицію, куди новий елемент можна вставити, щоб зберегти порядок у списку. Ця позиція — це місце, де всі елементи перед нею менші, а всі елементи після неї — більші або рівні.

Текст програми:

def find\_insert\_position(sorted\_list, new\_element):

    low = 0

    high = len(sorted\_list)

    while low < high:

        mid = (low + high) // 2

        if sorted\_list[mid] < new\_element:

            low = mid + 1

        else:

            high = mid

    return low

# Введення відсортованого списку

sorted\_list = [1, 2, 3, 4]

# Введення нового елемента

new\_element = int(input("Введіть новий елемент для вставки: "))

# Знаходження позиції для вставки

position = find\_insert\_position(sorted\_list, new\_element)

# Вивід результату

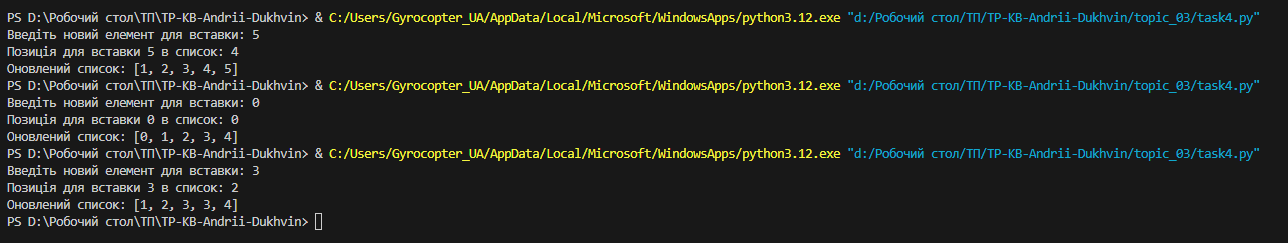
print(f"Позиція для вставки {new\_element} в список: {position}")

# Вставка нового елементу у список і виведення оновленого списку

sorted\_list.insert(position, new\_element)

print(f"Оновлений список: {sorted\_list}")

Після компіляції:



Звіт до Теми № 4

Виняткові ситуації

Під час виконання практичного завдання до Теми №1 було надано варіанти рішення до наступних задач:

**Попередні умови**: реалізована програма калькулятор, що використовує метод нескінченного введення даних для обробки. Всі дії (додавання, віднімання, множення, ділення) реалізовані як окремі функції та використовуються у відповідних місцях.

1. Розширити програму калькулятор функцією запитів даних для виконання операцій від користувача, що обробляє виняткові ситуації.
2. Розширити функцію ділення обробкою виняткової ситуації ділення но нуль
3. Ознайомитись зі списком виняткових ситуацій за посиланням <https://docs.python.org/3/library/exceptions.html>

Хід роботи

1. **Розробка функцій для операцій:**

Спочатку були створені окремі функції для основних математичних операцій: додавання (add), віднімання (subtract), множення (multiply) та ділення (divide).

Функції add, subtract, multiply реалізують базові операції, приймаючи два аргументи, a та b, і повертаючи відповідний результат.

Функція divide має додаткову обробку помилок. Вона перевіряє, чи є хоча б одне з чисел рівним нулю, і в такому разі підкидає помилку ZeroDivisionError. Якщо все гаразд, вона повертає результат ділення.

1. **Основна програма:**

Створена основна програма calculator, яка організовує всю логіку взаємодії з користувачем:

**Цикл вводу чисел:** Програма запитує у користувача введення двох чисел. Для кожного з чисел використовується блок try-except для обробки можливих помилок, таких як введення нечислових значень.

Якщо користувач вводить q, програма завершується.

Якщо введене значення не є числом, програма повторює запит.

**Вибір операції:** Після введення двох чисел, програма запитує, яку операцію користувач хоче виконати (+, -, \*, /). Знову ж таки, програма обробляє введення через цикли та перевірки:

Якщо користувач вводить неправильну операцію (не одну з чотирьох), програма повторює запит.

Якщо користувач хоче завершити програму, він може ввести q.

**Виконання операції:** В залежності від вибору користувача виконується відповідна операція:

Для кожної операції, функція викликається з переданими аргументами. Якщо операція — ділення, викликається функція divide, і якщо ділення на нуль, програма виводить повідомлення про помилку.

Після виконання операції, результат виводиться на екран, і цикл продовжується знову.

1. **Обробка помилок:**

Для кожної з функцій (особливо для divide), обробка помилок є важливою частиною програми. Вона забезпечує коректну роботу програми у разі введення неправильних або нечислових значень, а також при діленні на нуль.

Вся логіка пов'язана з обробкою винятків реалізована за допомогою конструкцій try-except, що дозволяє програмі не аварійно завершуватися, а просто виводити корисне повідомлення та продовжувати виконання.

1. **Завершення програми:**

Програма дозволяє користувачу завершити роботу в будь-який момент, ввівши q на етапах введення числа або вибору операції. Це забезпечується перевіркою if user\_input.lower() == 'q' в циклах вводу та при виборі операції.

Текст програми:

# Функції для операцій

def add(a, b):

    return a + b

def subtract(a, b):

    return a - b

def multiply(a, b):

    return a \* b

def divide(a, b):

    try:

        if a == 0 or b == 0:

            raise ZeroDivisionError("Помилка: ділення на нуль!")

        return a / b

    except ZeroDivisionError as e:

        print(e)  # Виводимо повідомлення про помилку

# Основна програма

def calculator():

    while True:

        # Запит на введення першого числа або команди для завершення

        try:

            user\_input = input("Введіть перше число або 'q': ")

            if user\_input.lower() == 'q':

                print("Програма завершена.")

                break  # Завершення програми при введенні 'q'

            num1 = float(user\_input)

        except ValueError:

            print("Помилка: Введено нечислове значення. Спробуйте ще раз.")

            continue  # Продовжуємо запит

        # Запит на введення другого числа

        while True:  # Цикл для другого числа

            try:

                user\_input = input("Введіть друге число або 'q': ")

                if user\_input.lower() == 'q':

                    print("Програма завершена.")

                    return  # Завершення програми при введенні 'q'

                num2 = float(user\_input)

                break  # Виходимо з циклу при коректному введенні

            except ValueError:

                print("Помилка: Введено нечислове значення. Спробуйте ще раз.")

                continue  # Повторюємо запит

        # Запит на вибір операції

        while True:

            operation = input("Оберіть операцію (+, -, \*, /): ")

            if operation.lower() == 'q':

                print("Програма завершена.")

                return  # Завершення програми при введенні 'q'

            # Обробка операцій

            if operation == '+':

                print("Результат:", add(num1, num2))

                break

            elif operation == '-':

                print("Результат:", subtract(num1, num2))

                break

            elif operation == '\*':

                print("Результат:", multiply(num1, num2))

                break

            elif operation == '/':

                result = divide(num1, num2)

                if result is not None:  # Якщо не було помилки

                    print("Результат:", result)

                    break  # Завершуємо цикл після виконання операції

            else:

                print("Невідома операція. Спробуйте ще раз.")

                continue  # Якщо операція неправильна, запитуємо ще раз

# Виклик програми

calculator()

Перевірка:

