**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:

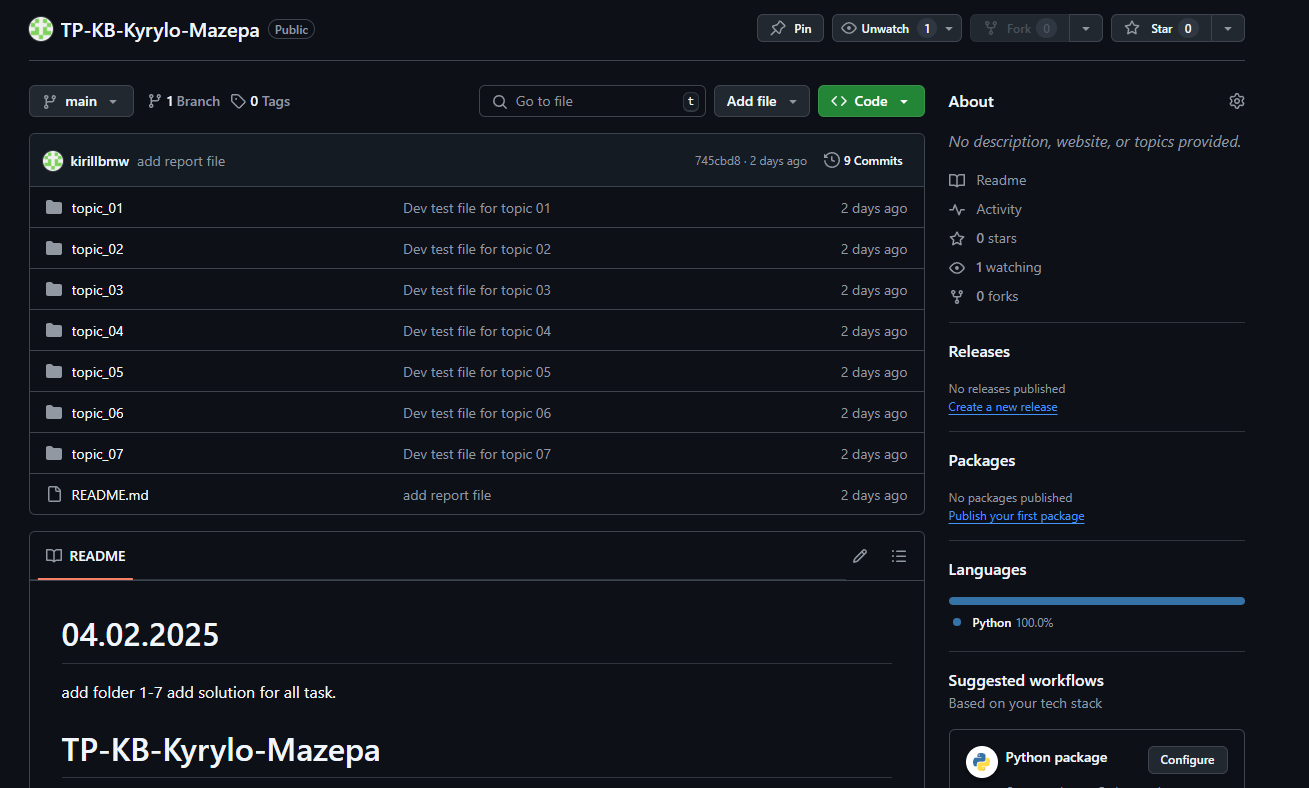
|  |
| --- |
| def revers():      s ="abcdefg123"      result = s[::-1]      return result  print (revers()) |

При компіляції він виводить:



Посилання на github: https://github.com/kirillbmw/TP-KB-Kyrylo-Mazepa

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.   "

    #Видаляє пробіли з обох боків рядка

    stripped = s.strip()

    print(f"strip():'{stripped}'")

   #Робить першу літеру великою, а інші маленькими

    capitalized = stripped.capitalize()

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

    #Робить першу літеру кожного слова великою

    titled = s.title()

    print(f"title():'{titled}'")

    #Всі символи в рядку на великі

    uppercased = s.upper()

    print(f"upper():'{uppercased}'")

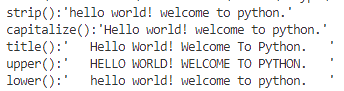
    #Всі символи в рядку на маленькі

    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: "))

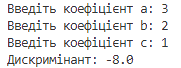
c = float(input("Введіть коефіцієнт c: "))

# Виклик функції та виведення результату

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

print(f"Дискримінант: {D}")

При компіляції він виводить:



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):

    discriminant = b\*\*2 - 4\*a\*c

    return discriminant

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

    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(abs(D)) / (2\*a)

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

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

        return x1, x2

# Введення даних

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)

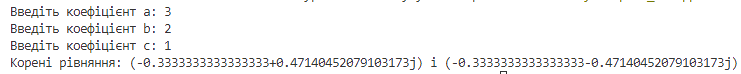
if len(roots) == 1:

    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 == '/':

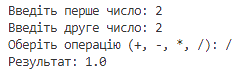
        print("Результат:", divide(num1, num2))

    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))

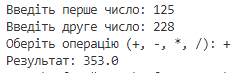
        case '/':

            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:

# Операції калькулятора

def calculate(a, b, operation):

    if operation == '+':

        return a + b

    elif operation == '-':

        return a - b

    elif operation == '\*':

        return a \* b

    elif operation == '/':

        return a / b if b != 0 else "Помилка: ділення на нуль!"

    return "Невідома операція!"

# Основна функція

def run\_calculator():

    while True:

        # Введення першого числа або вихід

        num1\_input = input("Введіть перше число або 'q' для виходу: ").lower()

        if num1\_input == 'q':

            print("Програма завершена.")

            break

        try:

            num1 = float(num1\_input)

        except ValueError:

            print("Помилка: введіть коректне число.")

            continue

        # Введення другого числа або вихід

        num2\_input = input("Введіть друге число або 'q' для виходу: ").lower()

        if num2\_input == 'q':

            print("Програма завершена.")

            break

        try:

            num2 = float(num2\_input)

        except ValueError:

            print("Помилка: введіть коректне число.")

            continue

        # Введення операції

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

        if operation.lower() == 'q':

            print("Програма завершена.")

            break

        # Результат обчислення

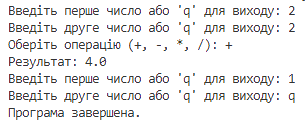
        result = calculate(num1, num2, operation)

        print("Результат:", result)

# Запуск програми

run\_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, 3, 2, 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()

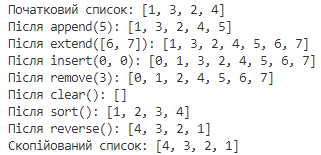
    copied\_list = my\_list.copy()

    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": "Kyrylo",

        "age": 18,

        "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": "Kyrylo",

        "age": 18,

        "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()

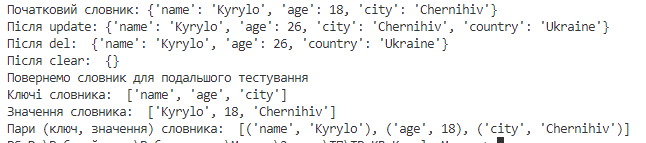
    items = my\_dict.items()  # Отримуємо пари (ключ, значення)

    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

# Відсортований список

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):

    if b == 0:

        print("Помилка: ділення на нуль!")

        return None

    return a / b

# Основна функція калькулятора

def calculator():

    while True:

        # Введення першого числа

        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("Помилка: введіть коректне число.")

            continue

        # 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("Помилка: введіть коректне число.")

            continue

        # Вибір операції

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

        if operation.lower() == 'q':

            print("Програма завершена.")

            break

        # Виконання операції через match-case

        match operation:

            case '+':

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

            case '-':

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

            case '\*':

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

            case '/':

                result = divide(num1, num2)

                if result is not None:

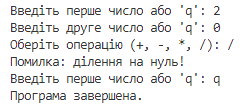
                    print("Результат:", result)

            case \_:

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

calculator()

Перевірка:



Звіт до Теми №5

Бібліотеки

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

1. Гра з комп’ютером: камінь, ножиці, папір. Програма виконує запит від користувача на введення одного із значень ["stone", "scissor", "paper"]. Наступним кроком, використовуючи модуль random, програма у випадковому порядку вибирає одне із значень ["stone", "scissor", "paper"]. В залежності від умови, що камінь перемагає ножиці, ножиці перемагають папір, а папір перемагає камінь визначити переможця.
2. Програма конвертування іноземної валюти в українську гривню. Для отримання актуальних курсів валют необхідно використовувати API НБУ та модуль, що надає можливість виконувати запити до сторонніх сервісів requests. Достатня умова роботи – можливість конвертації для трьох іноземних валют EUR, USD, PLN. Користувачу надається можливість введення кількості та типу валюти, результат роботи програми – конвертоване значення в українських гривнях.
3. Використання модулів для програми калькулятор. Функції додавання, віднімання, множення та ділення перенести в файл functions.py. Функції запиту на введення даних для операцій та самих операцій перемістити в файл operations.py. Програму калькулятор реалізувати в файлі calc.py, до якого підключають файл functions.py та operations.py.

Хід роботи

**Завдання 1**

* **Імпортуємо необхідну бібліотеку:**

Спочатку ми імпортуємо бібліотеку random, яка дозволяє комп'ютеру випадковим чином вибирати один з варіантів (камінь, ножиці чи папір).

* **Створення списку можливих варіантів:**

Створюємо список choices, який містить три елементи: "stone" (камінь), "scissor" (ножиці) та "paper" (папір). Цей список використовуватимемо для вибору варіанту комп'ютером.

* **Отримання вибору користувача:**

Запитуємо у користувача вибір за допомогою функції input(), вказуючи, що він має вибрати між "stone", "scissor" або "paper". Щоб уникнути проблем з регістром, перетворюємо введене значення на малу літеру за допомогою методу .lower().

* **Перевірка правильності введення:**

Перевіряємо, чи ввів користувач правильний варіант (міститься в списку choices). Якщо введено неправильне значення, виводимо повідомлення про помилку.

* **Вибір комп'ютером варіанту:**

Якщо введення користувача коректне, комп'ютер вибирає свій варіант випадковим чином, використовуючи random.choice(choices).

* **Виведення вибору комп'ютера:**

Після того, як комп'ютер вибрав варіант, виводимо його вибір на екран.

* **Визначення переможця:**

Порівнюємо вибір користувача з вибором комп'ютера. Існують три можливі результати:

1. Якщо вибори однакові, оголошуємо нічию.
2. Якщо вибір користувача виграє проти вибору комп'ютера (камінь б'є ножиці, ножиці б'ють папір, папір б'є камінь), оголошуємо перемогу користувача.
3. В іншому випадку комп'ютер виграє.

* **Виведення результату:**

В залежності від результату, виводимо відповідне повідомлення про перемогу, поразку або нічию.

Код програми:

import random

choices = ["stone", "scissor", "paper"]

user\_choice = input("Виберіть (stone, scissor, paper): ").lower()

# Перевірка на правильність введення

if user\_choice not in choices:

    print("Невірний вибір! Будь ласка, виберіть 'stone', 'scissor' або 'paper'")

else:

    # Випадковий вибір комп'ютера

    computer\_choice = random.choice(choices)

    print(f"Комп'ютер вибрав: {computer\_choice}")

    # Визначення переможця

    if user\_choice == computer\_choice:

        print("Нічия!")

    elif (user\_choice == "stone" and computer\_choice == "scissor") or (user\_choice == "scissor" and computer\_choice == "paper") or (user\_choice == "paper" and computer\_choice == "stone"):

        print("Ви виграли!")

    else:

        print("Комп'ютер виграв!")

Перевірка:



**Завдання 2**

1. **Імпортуємо бібліотеку requests:** Спочатку імпортуємо бібліотеку requests, яка дозволяє робити HTTP-запити для отримання даних з API НБУ.
2. **Визначаємо URL для отримання курсів валют:** Оголошуємо змінну NBU\_API\_URL, яка містить URL для доступу до актуальних курсів валют з сайту Національного банку України (НБУ). Це API надає дані у форматі JSON.
3. **Створюємо функцію get\_exchange\_rates:** Функція get\_exchange\_rates надсилає GET-запит до API НБУ за допомогою requests.get(). Отриманий результат конвертується в формат JSON за допомогою методу .json(), після чого повертається список курсів валют.
4. **Створюємо функцію для конвертації валют convert\_currency:** Функція convert\_currency приймає три параметри:
   * amount — кількість валюти для конвертації,
   * currency — код валюти, яку треба конвертувати,
   * rates — список всіх курсів валют.

Функція ітерує по всіх курсах валют і перевіряє, чи є в списку валюта з вказаним кодом. Якщо знаходить відповідний курс, повертає результат множення кількості валюти на курс. Якщо валюта не знайдена, повертається None.

1. **Головна функція main:** У функції main викликаємо get\_exchange\_rates, щоб отримати актуальні курси валют. Далі створюємо словник currencies, який містить коди валют і їхні назви українською мовою для зручності відображення.
2. **Запит користувача:** Виводимо список доступних валют (EUR, USD, PLN) і запитуємо користувача про вибір валюти за кодом (наприклад, EUR для євро). Потім запитуємо, скільки одиниць цієї валюти користувач хоче конвертувати.
3. **Конвертація валюти:** Викликаємо функцію convert\_currency для виконання конвертації і отримання результату.
4. **Виведення результату:** Після конвертації результат виводиться у форматі: "Кількість валюти Код валюти = Результат в UAH". Форматуємо результат до двох знаків після коми для зручності.
5. **Перевірка, чи виконується скрипт безпосередньо:** В кінці файлу викликається функція main() для запуску програми.

**Алгоритм роботи:**

1. Отримуємо актуальні курси валют з API НБУ.
2. Запитуємо користувача про валюту та кількість.
3. Конвертуємо вказану кількість валюти у гривні за поточним курсом.
4. Виводимо результат конвертації.

Код програми:

import requests

NBU\_API\_URL = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json"

def get\_exchange\_rates():

    response = requests.get(NBU\_API\_URL)

    return response.json()

def convert\_currency(amount, currency, rates):

    for rate in rates: #Ітерація по списку курсів валют

        if rate['cc'] == currency:

            return amount \* rate['rate']

    return None

def main():

    rates = get\_exchange\_rates()

    currencies = {"EUR": "Євро", "USD": "Долар США", "PLN": "Польський злотий"}

    print("Доступні валюти для конвертації: EUR, USD, PLN")

    currency = input("Введіть код валюти (EUR, USD, PLN): ").upper()

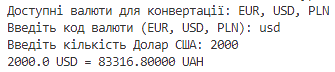
    amount = float(input(f"Введіть кількість {currencies[currency]}: "))

    result = convert\_currency(amount, currency, rates)

    print(f"{amount} {currency} = {result:.5f} UAH")

main()

Перевірка:



**Завдання 3**

**Структура програми:**

Програма калькулятора була організована за допомогою трьох окремих файлів:

* **functions.py**: містить математичні функції для операцій.
* **operations.py**: відповідає за обробку вводу користувача (введення чисел і вибір операції).
* **calc.py**: головний файл, який використовує функції з попередніх файлів для реалізації основного калькулятора.

**Перенесення функцій у модулі:**

Відповідно до завдання, я розділив код калькулятора на три частини, щоб кожен файл виконував свою окрему задачу.

Код програми:

**functions.py**:

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)

        return None

**operations.py**:

def get\_first\_number():

    while True:

        try:

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

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

                print("Програма завершена.")

                return None

            return float(user\_input)

        except ValueError:

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

def get\_second\_number():

    while True:

        try:

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

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

                print("Програма завершена.")

                return None

            return float(user\_input)

        except ValueError:

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

def get\_operation():

    while True:

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

        if operation.lower() == 'q':

            print("Програма завершена.")

            return None

        if operation in ('+', '-', '\*', '/'):

            return operation

        else:

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

**calc.py**:

from functions import add, subtract, multiply, divide

from operations import get\_first\_number, get\_second\_number, get\_operation

def calculator():

    while True:

        num1 = get\_first\_number()

        if num1 is None:

            break

        num2 = get\_second\_number()

        if num2 is None:

            break

        # Вибір операції

        while True:

            operation = get\_operation()

            if operation is None:

                return

            # Обробка операцій

            match operation:

                case '+':

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

                    break

                case '-':

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

                    break

                case '\*':

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

                    break

                case '/':

                    result = divide(num1, num2)

                    if result is not None:

                        print("Результат:", result)

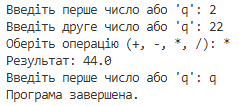
                        break

                    else:

                        continue  # Запитуємо операцію знову

calculator()

Перевірка:



Звіт до Теми №6

Робота з файлами

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

Попередні умови: Реалізована програма калькулятор в файлі calc.py, до якого підключають файл functions.py та operations.py. Інструкції до оформлення вмісту файлів розміщенні в завданні 3 до теми 5.

1. Розробити механізм логування всіх дій, що виконує програма. Забезпечити зберігання інформації про введені данні, виконану операцію та результат виконання операції над даними.
2. Маючи не відсортований список, елементами якого є словники з двома параметрами (ім’я та оцінка) виконати сортування списку, використовуючи стандартну функцію sorted(). Другим параметром для функції sorted() має бути lambda функція, що повертає ім’я або оцінку із елемента словника.

**Завдання 1**

**Підготовка структури проекту**:

* Створити директорію проекту, наприклад, topic\_06/task\_1.
* Всередині calcmodules створити такі файли:
  + calc.py (основний файл програми).
  + functions.py (функції для математичних операцій).
  + operations.py (функції для отримання введених даних та операцій).
  + log.py (функція для логування).
* Створити директорію для логів: topic\_06/task\_1/log.
* Створити порожній файл для логів: topic\_06/task\_1/log/log.txt.

**Створення функції для логування у log.py**:

Текст програми:

from datetime import datetime

def makeLog(First\_num, operation, Second\_num, result):

    current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    with open("topic\_06/task\_1/log/log.txt", "a") as f:

        f.write(f"{current\_time} - First\_num={First\_num} operation='{operation}' Second\_num={Second\_num} result={result}\n")

**Реалізація основної логіки калькулятора у calc.py**:

Текст програми:

from log import makeLog

from functions import add, subtract, multiply, divide

from operations import get\_first\_number, get\_second\_number, get\_operation

def calculator():

    while True:

        num1 = get\_first\_number()

        if num1 is None:

            break

        num2 = get\_second\_number()

        if num2 is None:

            break

        while True:

            operation = get\_operation()

            if operation is None:

                return

            if operation not in ('+', '-', '\*', '/'):

                error\_message = f"Unknown operation: {operation}. Please try again."

                print(error\_message)

                makeLog(num1, operation, num2, error\_message)  # Логування помилки

                continue  # Запитуємо операцію знову

            # Обробка операцій

            match operation:

                case '+':

                    result = add(num1, num2)

                    print("Result:", result)

                    makeLog(num1, operation, num2, result)

                    break

                case '-':

                    result = subtract(num1, num2)

                    print("Result:", result)

                    makeLog(num1, operation, num2, result)

                    break

                case '\*':

                    result = multiply(num1, num2)

                    print("Result:", result)

                    makeLog(num1, operation, num2, result)

                    break

                case '/':

                    result = divide(num1, num2)

                    if result is not None:

                        print("Result:", result)

                        makeLog(num1, operation, num2, result)

                        break

                    else:

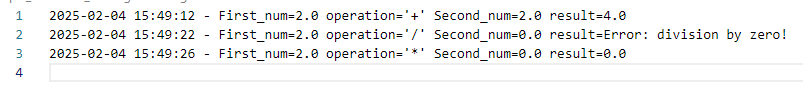
                        makeLog(num1, operation, num2, "Error: division by zero!")

                        continue  # Запитуємо операцію знову

calculator()

**Результати**:

* Програма правильно виконує всі математичні операції.
* Всі дії успішно логуються в файл log.txt, де зберігаються введені числа, операції та результати.



**Завдання 2**

**Підготовка списку студентів:** Для виконання завдання спочатку створюється список, що містить інформацію про студентів у вигляді словників. Кожен словник має два ключі: "name" (ім'я студента) та "score" (оцінка студента). Ось як виглядає цей список:

students = [

    {"name": "Eva", "score": 18},

    {"name": "Grace", "score": 14},

    {"name": "Hannah", "score": 92},

    {"name": "Ian", "score": 4},

    {"name": "Mia", "score": 7},

    {"name": "Nina", "score": 79},

    {"name": "Frank", "score": 19},

]

**Сортування за іменем (в алфавітному порядку):** Для сортування списку студентів за іменами використовується вбудована функція sorted(). Параметр key вказує, що сортувати список потрібно за значенням ключа "name" в кожному словнику.

* Використовуємо lambda-функцію для отримання імені студента: lambda student: student["name"].
* Функція sorted() відсортовує список студентів за іменами в алфавітному порядку.

Код для цього виглядає так:

# Сортування за ім'ям

sorted\_by\_name = sorted(students, key=lambda student: student["name"])

**Сортування за балом (по зростанню оцінки):** Для сортування студентів за оцінками використовуємо таку ж функцію sorted(), але цього разу параметр key вказує, що потрібно сортувати за значенням ключа "score".

* Використовуємо lambda-функцію для отримання оцінки студента: lambda student: student["score"].
* Сортування за оцінкою відбувається по зростанню, тобто від найменшої до найбільшої оцінки.

Код для цього виглядає так:

# Сортування за балом

sorted\_by\_score = sorted(students, key=lambda student: student["score"])

**Виведення результатів сортування:** Після виконання сортування, результати виводяться на екран за допомогою функції print().

print("Сортування за ім'ям в алфавітному порядку: ", sorted\_by\_name)

print("Сортування за найвищим балом: ", sorted\_by\_score)

Код програми:

students = [

    {"name": "Eva", "score": 18},

    {"name": "Grace", "score": 14},

    {"name": "Hannah", "score": 92},

    {"name": "Ian", "score": 4},

    {"name": "Mia", "score": 7},

    {"name": "Nina", "score": 79},

    {"name": "Frank", "score": 19},

]

# Сортування за ім'ям

sorted\_by\_name = sorted(students, key=lambda student: student["name"])

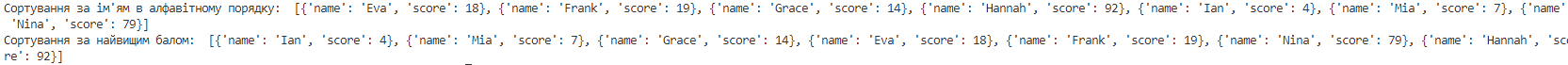
# Сортування за балом

sorted\_by\_score = sorted(students, key=lambda student: student["score"])

print("Сортування за ім'ям в алфавітному порядку: ", sorted\_by\_name)

print("Сортування за найвищим балом: ", sorted\_by\_score)

Результат:



Звіт до Теми №7

Об'єктно-орієнтоване програмування

Під час виконання практичного завдання до Теми №7 було надано варіанти рішення до наступних задач:

1. Ознайомитись з документацією що описує можливості використання класів у мові Python <https://docs.python.org/3/tutorial/classes.html>
2. Ознайомитись з існуючими за замовченням методами класу по типу \_\_init\_\_(self) \_\_str\_\_(self)\_\_ та надати приклади використання.
3. Розробити клас **Student** атрибутами якого э два параметра **name** та **age**. Створити список елементами якого є об'єкти класу **Student**. Написати цикл який виводить на екран елементи списку у відсортованому порядку. Для сортування використати стандартну функцію **sorted**. Функція **sorted** має використовувати **lambda** функцію для визначення ключа сортування.
4. Використовуючи принципи ООП переписати програму Калькулятор. Завдання має бути виконано використовуючи модульний підхід.

Хід роботи

1. Ознайомитись з існуючими за замовченням методами класу по типу \_\_init\_\_(self) \_\_str\_\_(self)\_\_ та надати приклади використання.

В Python існують деякі спеціальні методи, які дозволяють реалізовувати певні поведінкові особливості класів. Два з таких методів — \_\_init\_\_ та \_\_str\_\_. Давайте розглянемо кожен із них детальніше і наведемо приклади використання.

**Метод \_\_init\_\_**

Метод \_\_init\_\_ є конструктором класу і викликається автоматично під час створення нового екземпляра класу. Він використовується для ініціалізації атрибутів класу.

**Метод \_\_str\_\_**

Метод \_\_str\_\_ визначає поведінку функції print і методу str() при виклику для екземпляра класу. Він має повертати рядок, який буде використовуватися як текстове представлення об'єкта.

Код програми:

class Person:

    def \_\_init\_\_(self, name, age):        #Він використовується для ініціалізації атрибутів класу

        self.name = name

        self.age = age

    def \_\_str\_\_(self):                   #Він має повертати рядок, який буде використовуватися як текстове представлення об'єкта

        return f"Person(name={self.name}, age={self.age})"

# Створюємо новий екземпляр класу Person

person1 = Person("Alice", 30)

person2 = Person("Bob", 25)

print(person1.name)  # Виведе name

print(person1.age)   # Виведе age

print(person2)

Вивід:



1. Розробити клас **Student** атрибутами якого э два параметра **name** та **age**. Створити список елементами якого є об'єкти класу **Student**. Написати цикл який виводить на екран елементи списку у відсортованому порядку. Для сортування використати стандартну функцію **sorted**. Функція **sorted** має використовувати **lambda** функцію для визначення ключа сортування.

Щоб розробити клас Student з атрибутами name та age, а також створити список, елементами якого є об'єкти цього класу, і написати цикл, що виводить на екран елементи списку у відсортованому порядку потрібно використовувати для сортування функцію sorted з лямбда-функцією для визначення ключа сортування.

Кроки:

1. Створимо клас Student.
2. Створимо список об'єктів цього класу.
3. Використаємо функцію sorted з лямбда-функцією для сортування списку.
4. Виведемо відсортовані об'єкти на екран.

Код програми:

class Student:

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

    def \_\_str\_\_(self):

        return f"Student(name={self.name}, age={self.age})"

# Створюємо список об'єктів класу Student

students = [

    Student("Alice", 21),

    Student("Bob", 19),

    Student("Charlie", 22),

    Student("David", 20)

]

# Сортуємо список за ім'ям (name)

sorted\_by\_name = sorted(students, key=lambda student: student.name)

# Сортуємо список за віком (age)

sorted\_by\_age = sorted(students, key=lambda student: student.age)

# Виводимо відсортовані елементи списку за ім'ям

print("Sorted by name:")

for student in sorted\_by\_name:

    print(student)

# Виводимо відсортовані елементи списку за віком

print("\nSorted by age:")

for student in sorted\_by\_age:

    print(student)

**Пояснення коду**

1. **Клас**Student:
   * Метод \_\_init\_\_ ініціалізує об'єкти класу з атрибутами name та age.
   * Метод \_\_str\_\_ забезпечує зручне текстове представлення об'єкта класу, яке використовується при виклику функції print.
2. **Список**students:
   * Створюємо список, елементами якого є об'єкти класу Student.
3. **Сортування списку**:
   * Використовуємо функцію sorted для сортування списку за атрибутом age.
   * Лямбда-функція lambda student: student.age визначає, що сортування буде проводитись за віком студента.
4. **Виведення відсортованих об'єктів**:
   * Проходимо по відсортованому списку і виводимо кожен об'єкт за допомогою функції print, яка викликає метод \_\_str\_\_.
5. Використовуючи принципи ООП переписати програму Калькулятор. Завдання має бути виконано використовуючи модульний підхід.

Щоб переписати програму калькулятора з використанням принципів об'єктно-орієнтованого програмування (ООП) і модульного підходу, ми створимо окремі класи для кожного функціонального компонента. Кожен модуль буде відповідати за свій аспект роботи калькулятора: виконання операцій, отримання введення, логування та сам процес калькуляції.

Структура проекту

calculator.py: основний модуль, який запускає калькулятор.

functions.py: модуль для математичних операцій.

log.py: модуль для логування операцій.

operations.py: модуль для отримання введення від користувача.

calculator\_class.py: модуль з класом Calculator, який об'єднує всі компоненти.

Код програми:

**calculator.py**

from calculator\_class import Calculator

if \_\_name\_\_ == "\_\_main\_\_":

    calc = Calculator()

    calc.run()

**functions.py**

class Functions:

    def add(self, a, b):

        return a + b

    def subtract(self, a, b):

        return a - b

    def multiply(self, a, b):

        return a \* b

    def divide(self, a, b):

        if b == 0:

            print("Error: Division by zero!")

            return None

        return a / b

**log.py**

from datetime import datetime

class Logger:

    def make\_log(self, first\_num, operation, second\_num, result):

        current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        with open("topic\_07/task\_4/log.txt", "a") as f:

            f.write(f"{current\_time} - First\_num={first\_num} operation='{operation}' Second\_num={second\_num} result={result}\n")

**operations.py**

class Operations:

    def get\_first\_number(self):

        while True:

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

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

                print("Програму завершено.")

                return None

            try:

                return float(user\_input)

            except ValueError:

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

    def get\_second\_number(self):

        while True:

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

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

                print("Програму завершено.")

                return None

            try:

                return float(user\_input)

            except ValueError:

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

    def get\_operation(self):

        while True:

            operation = input("Оберіть операцію (+, -, \*, /) або 'q' для виходу: ")

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

                print("Програму завершено.")

                return None

            if operation in ('+', '-', '\*', '/'):

                return operation

            print(f"Невідома операція: {operation}. Спробуйте ще раз.")

**calculator\_class.py**

from log import Logger

from functions import Functions

from operations import Operations

class Calculator:

    def \_\_init\_\_(self):

        self.functions = Functions()

        self.operations = Operations()

        self.logger = Logger()

    def run(self):

        while True:

            num1 = self.operations.get\_first\_number()

            if num1 is None:

                break

            num2 = self.operations.get\_second\_number()

            if num2 is None:

                break

            while True:

                operation = self.operations.get\_operation()

                if operation is None:

                    return

                if operation not in ('+', '-', '\*', '/'):

                    error\_message = f"Невідома операція: {operation}. Спробуйте ще раз."

                    print(error\_message)

                    self.logger.make\_log(num1, operation, num2, error\_message)

                    continue

                match operation:

                    case '+':

                        result = self.functions.add(num1, num2)

                    case '-':

                        result = self.functions.subtract(num1, num2)

                    case '\*':

                        result = self.functions.multiply(num1, num2)

                    case '/':

                        result = self.functions.divide(num1, num2)

                        if result is None:

                            self.logger.make\_log(num1, operation, num2, "Error: division by zero!")

                            continue

                print("Result:", result)

                self.logger.make\_log(num1, operation, num2, result)

                break

**Пояснення**

* **calculator.py**: Запускає калькулятор, створюючи об'єкт класу Calculator і викликаючи його метод run.
* **functions.py**: Вміщує клас Functions з методами для виконання основних математичних операцій.
* **log.py**: Вміщує клас Logger з методом для логування операцій в файл.
* **operations.py**: Вміщує клас Operations з методами для отримання введення від користувача.
* **calculator\_class.py**: Вміщує клас Calculator, який об'єднує всі компоненти і керує процесом калькуляції.

**ООП в цій програмі**

**Інкапсуляція:** Кожен клас відповідає за свою частину логіки, наприклад, обчислення, логування, отримання вводу. Це дозволяє зберігати код чистим і модульним.

**Статичні методи:** Для функцій, які не потребують доступу до стану об'єкта (наприклад, математичні операції), використовуються статичні методи, що дозволяє зручно викликати їх без створення екземплярів класу.

Клас Calculator є головним об'єктом, який взаємодіє з усіма іншими класами, надаючи користувачеві інтерактивний інтерфейс.

Цей підхід дозволяє легко підтримувати, розширювати та модифікувати програму в майбутньому.