# **МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА” ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ** **Кафедра ІСМ**

# **Звіт**

# **до лабораторної роботи №2**

# **З дисципліни “Спеціальні мови програмування”**

**Виконав:  
ст.гр.ІТ-31  
Шельвах Максим  
  
Прийняв:  
Щербак С.С.**

# **Львів — 2023**

**Тема роботи:** Основи побудови об’єктно-орієнтованих додатків на Python.

**Мета роботи:** Розробка консольного калькулятора в об’єктно орієнтованому стилі з використанням класів.

## План роботи

Завдання 1: Створення класу Calculator

Створіть клас Calculator, який буде служити основою для додатка калькулятора.

Завдання 2: Ініціалізація калькулятора

Реалізуйте метод \_\_init\_\_ у класі Calculator для ініціалізації необхідних атрибутів або змінних.

Завдання 3: Введення користувача

Перемістіть функціональність введення користувача в метод у межах класу Calculator. Метод повинен приймати введення для двох чисел і оператора.

Завдання 4: Перевірка оператора

Реалізуйте метод у класі Calculator, щоб перевірити, чи введений оператор є дійсним (тобто одним із +, -, \*, /). Відобразіть повідомлення про помилку, якщо він не є дійсним.

Завдання 5: Обчислення

Створіть метод у класі Calculator, який виконує обчислення на основі введення користувача (наприклад, додавання, віднімання, множення, ділення).

Завдання 6: Обробка помилок

Реалізуйте обробку помилок у межах класу Calculator для обробки ділення на нуль або інших потенційних помилок. Відобразіть відповідні повідомлення про помилку.

Завдання 7: Повторення обчислень

Додайте метод до класу Calculator, щоб запитати користувача, чи він хоче виконати ще одне обчислення. Якщо так, дозвольте йому ввести нові числа і оператор. Якщо ні, вийдіть з програми.

Завдання 8: Десяткові числа

Модифікуйте клас Calculator для обробки десяткових чисел (плаваюча кома) для більш точних обчислень.

Завдання 9: Додаткові операції

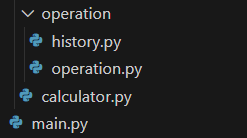
Розширте клас Calculator, щоб підтримувати додаткові операції, такі як піднесення до степеня (^), квадратний корінь (√) та залишок від ділення (%).

Завдання 10: Інтерфейс, зрозумілий для користувача

Покращте інтерфейс користувача у межах класу Calculator, надавши чіткі запити, повідомлення та форматування виводу для зручності читання.

Хід роботи

**Структура файлів:**

****

**Реалізований код:**

calculator.py

from calculator.operation.operation import (

Addition,

Subtraction,

Multiplication,

Division,

Exponentiation,

SquareRoot,

Modulus,

)

from calculator.operation.history import History

# Головний клас калькулятора

class Calculator:

def \_\_init\_\_(self):

self.decimal\_places = 2

self.memory = 0

self.history = History()

# Відображення історії обчислень

def display\_history(self):

self.history.display()

# Зміна кількості десяткових розрядів

def change\_decimal\_places(self):

self.decimal\_places = int(input("Введіть кількість десяткових розрядів: "))

# Головний метод для виконання калькулятора

def run(self):

while True:

try:

self.display\_history()

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

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

operator = input("Введіть оператор (+, -, \*, /, ^, √, %, M - для збереження в пам'ять, D - для зміни десяткових розрядів): ")

if operator in ('+', '-', '\*', '/', '^', '√', '%'):

# Створення об'єкту відповідно до вибраної операції

operation = self.create\_operation(num1, num2, operator)

result = operation.execute()

elif operator == "M":

self.memory = 0

print(f"Значення {self.memory} збережено в пам'яті.")

elif operator == "D":

self.change\_decimal\_places()

continue

else:

print("Помилка! Введіть дійсний оператор (+, -, \*, /, ^, √, %, M, D).")

continue

formatted\_result = round(result, self.decimal\_places)

print(f"Результат обчислення: {formatted\_result}")

self.history.add\_entry(f"{num1} {operator} {num2}", formatted\_result)

except ValueError:

print("Помилка! Введено недійсне число.")

except ZeroDivisionError as e:

print(f"Помилка: {e}")

repeat = input("Виконати ще одне обчислення? (Так/Ні): ").lower()

if repeat != "так":

break

self.display\_history()

# Метод для створення об'єкту операції залежно від вибору користувача

def create\_operation(self, num1, num2, operator):

operations = {

'+': Addition(num1, num2),

'-': Subtraction(num1, num2),

'\*': Multiplication(num1, num2),

'/': Division(num1, num2),

'^': Exponentiation(num1, num2),

'√': SquareRoot(num1),

'%': Modulus(num1, num2),

}

return operations.get(operator, None)

main.py

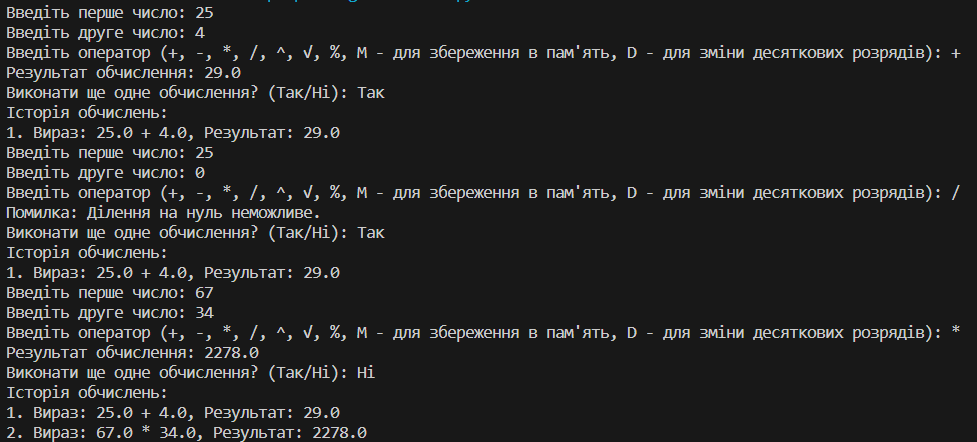
from calculator.calculator import Calculator

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

calculator = Calculator()

calculator.run()

Результат виконання програми:



**Висновок:** виконавши ці завдання, я перетворив калькулятор з минулої лабораторної роботи у об'єктно-орієнтований калькулятор, використовуючи класи в Python. Цей проект допоміг мені вивчити концепції об'єктно-орієнтованого програмування та організацію, зберігаючи функціональність і інтерфейс користувача калькулятора.