Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Лабораторна роботи №2

з дисципліни «Спеціалізовані мови програмування»

на тему

«Основи побудови об’єктно-орієнтованих додатків на Python»

Виконав:

Ложовський В. А.

Перевірив:

Щербак С. С.

Львів 2024

**Мета:** 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.

**Реалізація:**

**Файл AppSettings.py:**  
import lab1.constants.constants as const

class AppSettings:

    def \_\_init\_\_(self):

        self.decimal\_places = const.DECIMALS

        self.console\_color = "white"

    def change\_decimals(self, decimals):

        self.decimal\_places = decimals

    def change\_color(self, color):

        self.console\_color = color

**Папка functions, файл calcFunctions.py:**import math

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:

        return a / b

    except ZeroDivisionError:

        return "Error: Division by zero"

def power(a, b):

    return a \*\* b

def sqrt(a):

    return math.sqrt(a)

def modulus(a, b):

    return a % b

**Папка interface, файл userInterface.py:**

def get\_user\_input(prompt):

return input(prompt)

def display\_output(message):

print(message)

**Папка functions, файл Calculator.py:**

from lab2.functions.calcFunctions import AddOperation, SubtractOperation, MultiplyOperation, DivideOperation, PowerOperation, SqrtOperation, ModulusOperation

from lab2.init import memory, app\_settings

import lab2.constants.constants as const

class Calculator:

    def \_\_init\_\_(self):

        self.history = []

        self.result = None

    def log\_to\_file(self, expression, result):

        try:

            with open("lab2/logs/calculator\_history.txt", "a") as logFile:

                logFile.write(f"{expression} = {result}\n")

        except Exception as e:

            print(f"An error occurred while writing to the file: {e}")

    def get\_input(self):

        return input("Enter an expression (e.g., '5 + 3' or 'exit' to quit): ")

    def check\_operator(self, operator):

        if operator in const.VALID\_OPERATORS:

            return True

        else:

            print("Invalid operator!")

            return False

    def get\_operation(self, operator):

        operations = {

            '+': AddOperation(),

            '-': SubtractOperation(),

            '\*': MultiplyOperation(),

            '/': DivideOperation(),

            '^': PowerOperation(),

            '√': SqrtOperation(),

            '%': ModulusOperation(),

        }

        return operations.get(operator)

    def calculate(self, num1, num2, operator):

        operation = self.get\_operation(operator)

        if operation:

            return operation.calculate(num1, num2)

        else:

            print("Invalid operation.")

            return None

    def parse\_expression(self, expression):

        try:

            expression = expression.replace(' ', '')

            if '√' in expression:

                num = float(expression[1:])

                return num, None, '√'

            for operator in const.VALID\_OPERATORS:

                if operator in expression:

                    num1, num2 = expression.split(operator)

                    return float(num1), float(num2), operator

        except ValueError:

            raise ValueError("Invalid expression format")

    def display\_history(self):

        if self.history:

            print("\n--- Calculation History ---")

            for record in self.history:

                print(record)

        else:

            print("No history available.")

    def perform\_calculation(self):

        while True:

            user\_input = self.get\_input()

            if user\_input.lower() == "history":

                self.display\_history()

                continue

            elif user\_input.lower() == "exit":

                break

            try:

                num1, num2, operator = self.parse\_expression(user\_input)

                if not self.check\_operator(operator):

                    continue

                result = self.calculate(num1, num2, operator)

                if result is not None:

                    expression = f"{num1} {operator} {num2}" if operator != '√' else f"{operator}({num1})"

                    formatted\_result = f"{expression} = {round(result, app\_settings.decimal\_places)}"

                    self.history.append(formatted\_result)

                    self.log\_to\_file(expression, formatted\_result)

                    print(formatted\_result)

            except ValueError:

                print("Error: Invalid input.")

            except ZeroDivisionError as zde:

                print(zde)

            choice = input("Do you want to perform another calculation? (y/n): ")

            if choice.lower() != 'y':

                break

**Файл main.py:**

from lab1.functions.calcFunctions import add, subtract, multiply, divide, power, sqrt, modulus

from lab1.init import memory, app\_settings

import lab1.constants.constants as const

history = []

def logToFile(expression, result):

    with open("lab1/logs/calculator\_history.txt", "a") as logFile:

        logFile.write(f"{expression} = {result}\n")

def displayHistory():

    if history:

        print("\n--- Calculation History ---")

        for record in history:

            print(record)

    else:

        print("No history available.")

def parse\_expression(expression):

    try:

        expression = expression.replace(' ', '')

        if '√' in expression:

            num = float(expression[1:])

            return num, None, '√'

        for operator in const.VALID\_OPERATORS:

            if operator in expression:

                num1, num2 = expression.split(operator)

                return float(num1), float(num2), operator

    except ValueError:

        raise ValueError("Invalid expression format")

def main():

    while True:

        print("\nTo view calculation history, type 'history'")

        userInput = input("Enter an expression (e.g., '5 + 3' or 'exit' to quit): ")

        if userInput.lower() == "history":

            displayHistory()

            continue

        elif userInput.lower() == "exit":

            break

        try:

            num1, num2, operator = parse\_expression(userInput)

            if operator not in const.VALID\_OPERATORS:

                print("Invalid operator!")

                continue

            if operator == '+':

                result = add(num1, num2)

            elif operator == '-':

                result = subtract(num1, num2)

            elif operator == '\*':

                result = multiply(num1, num2)

            elif operator == '/':

                result = divide(num1, num2)

            elif operator == '^':

                result = power(num1, num2)

            elif operator == '√':

                result = sqrt(num1)

            elif operator == '%':

                result = modulus(num1, num2)

            expression = f"{num1} {operator} {num2}" if operator != '√' else f"{operator}({num1})"

            result = f"{expression} = {round(result, app\_settings.decimal\_places)}"

            history.append(result)

            logToFile(expression, result)

            print(result)

        except ValueError:

            print("Error: Invalid input.")

        choice = input("Do you want to perform another calculation? (y/n): ")

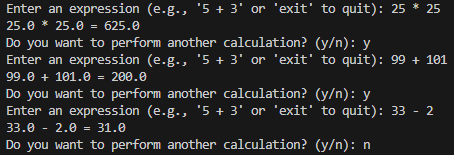
        if choice.lower() != 'y':

            break

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Результат виконання:**

****

**Висновок:** на цій лабораторній роботі я перетворив консольний калькулятор у об'єктно-орієнтований калькулятор, використовуючи класи в Python.