Міністерство освіти і науки України

Національний університет «Львівська політехніка»

******

Звіт про виконання лабораторної роботи №2  
з дисципліни  
“Спеціалізовані мови програмування”  
на тему  
“Основи побудови об’єктно-орієнтованих додатків на Python ”

Виконала:  
студентка групи РІ-31,  
Пятницька Вікторія

Прийняв:

Щербак С.С.

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

**Хід 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.

**Код програми:**

***./bll/classes/calculator.py****:*

from bll.functions import (

addition, substraction, multiplication, division, power, square\_root, modulus,

memory\_store, memory\_recall, memory\_clear, memory\_add,

add\_to\_history, get\_history, clear\_history

)

from app\_settings import decimal\_places

class Calculator:

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

self.operations = {

'+': addition,

'-': substraction,

'\*': multiplication,

'/': division,

'^': power,

'sqrt': square\_root,

'%': modulus

}

def get\_operator(self):

while True:

try:

operator = input("Enter operator (+, -, \*, /, ^, sqrt, %): ").strip()

if operator in self.operations:

return operator

print("Invalid operator! Please enter a valid operator.")

except Exception as e:

print(f"Error: {e}")

def get\_operand(self, prompt):

while True:

try:

operand = input(prompt).strip()

if operand.upper() == 'MR':

value = memory\_recall()

if value is None:

print("Memory is empty. Please enter a valid operand.")

continue

return value

return float(operand)

except ValueError:

print("Invalid input! Please enter a float number or 'MR'.")

except Exception as e:

print(f"Error: {e}")

def input(self):

try:

operator = self.get\_operator()

first\_operand = self.get\_operand("Enter the first operand or MR (Memory recall): ")

if operator == 'sqrt':

second\_operand = None

else:

second\_operand = self.get\_operand("Enter the second operand or MR (Memory recall): ")

return first\_operand, second\_operand, operator

except Exception as e:

print(f"Error: {e}")

return None, None, None

def calculate(self, first\_operand, second\_operand, operator):

if not isinstance(first\_operand, (int,float)) or (second\_operand is not None and not isinstance(second\_operand, (int,float))):

raise TypeError("All operands must be float numbers.")

if operator == '/' and second\_operand == 0:

raise ZeroDivisionError("Division by zero is not allowed.")

try:

operation = self.operations[operator]

result = operation(first\_operand, second\_operand) if operator != 'sqrt' else operation(first\_operand)

return round(result, decimal\_places) if result is not None else None

except Exception as e:

print(f"Error: {e}")

return None

def perform\_memory\_operation(self, result):

try:

memory\_choice = input('Do you want to perform any memory operations ("MS" to store, "M+" to add, "MC" to clear)? Enter any other character if no : ').strip().upper()

if memory\_choice == 'MS':

memory\_store(result)

elif memory\_choice == 'M+':

memory\_add(result)

elif memory\_choice == 'MC':

memory\_clear()

except Exception as e:

print(f"Error: {e}")

def show\_history(self):

try:

show\_history = input("Do you want to see the history of operations? (Enter 'yes' or any other character if no): ").strip().lower()

if show\_history == 'yes':

for record in get\_history():

print(record)

else:

print("History display canceled.")

except Exception as e:

print(f"Error: {e}")

def run(self):

while True:

first\_operand, second\_operand, operator = self.input()

if operator is None:

print("Invalid input. Please try again.")

continue

result = None

try:

result = self.calculate(first\_operand, second\_operand, operator)

except ZeroDivisionError as e:

print(e)

continue

except TypeError as e:

print(e)

continue

except Exception as e:

print(f"An unexpected error occurred: {e}")

continue

if result is not None:

print(f"Result: {result}")

add\_to\_history(f"{first\_operand} {operator} {second\_operand if second\_operand is not None else ''} = {result}")

self.perform\_memory\_operation(result)

self.show\_history()

repeat = input("Do you want to perform another calculation? (Enter 'yes' or any other character if no): ").strip().lower()

if repeat != 'yes':

break

***./bll/functions.py***

import math

def check\_floats(\*operands):

for operand in operands:

if not isinstance(operand, float):

raise ValueError("All operands must be float numbers.")

# operation functions

def addition(first\_operand, second\_operand):

check\_floats(first\_operand, second\_operand)

return first\_operand + second\_operand

def substraction(first\_operand, second\_operand):

check\_floats(first\_operand, second\_operand)

return first\_operand - second\_operand

def multiplication(first\_operand, second\_operand):

check\_floats(first\_operand, second\_operand)

return first\_operand \* second\_operand

def division(first\_operand, second\_operand):

check\_floats(first\_operand, second\_operand)

if second\_operand == 0:

raise ZeroDivisionError("You cannot divide by zero")

return first\_operand / second\_operand

def power(first\_operand, second\_operand):

check\_floats(first\_operand, second\_operand)

return pow(first\_operand, second\_operand)

def square\_root(operand):

check\_floats(operand)

if operand >= 0:

return math.sqrt(operand)

else:

return ("It is impossible to take the root of a negative number.")

def modulus(first\_operand, second\_operand):

check\_floats(first\_operand, second\_operand)

return first\_operand % second\_operand

# memory functions

memory = None

def memory\_store(value):

global memory

memory = value

def memory\_recall():

return memory

def memory\_clear():

global memory

memory = None

def memory\_add(value):

global memory

if memory is None:

memory = value

else:

memory += value

# history functions

history = []

def add\_to\_history(record):

history.append(record)

def get\_history():

return history

def clear\_history():

history.clear()

**Висновки:** внаслідок виконання лабораторної роботи я створила консольну версію калькулятора в об’єктно орієнтованому стилі з використанням класів.

**GitHub:** <https://github.com/viksi01/Calculator/tree/lab2>