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:

Бабич О. Ю.

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:**

**Папка AppSettings, файл AppSettings.py:**  
decimal\_places = 2

**Папка classes, файл calculator.py:**

from GlobalVariables.GlobalVariables import memory, history

from functions.functions import get\_operator, calculate

from tools.tools import save\_to\_memory, get\_from\_memory, show\_history

from AppSettings.AppSettings import decimal\_places

class Calculator:

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

self.memory = memory

self.history = history

self.decimal\_places = decimal\_places

def get\_numbers(self):

choice = input("Використати значення з пам'яті? (так/ні): ").lower()

if choice not in ['так', 'ні']:

print("Помилка: введіть так або ні.")

return self.get\_numbers()

if choice == 'так':

num1 = get\_from\_memory()

if num1 is None:

return self.get\_numbers()

print(f"Використано значення з пам'яті: {num1}")

else:

try:

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

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

try:

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

return num1, num2

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

def run(self):

while True:

num1, num2 = self.get\_numbers()

operator = get\_operator()

result = calculate(num1, num2, operator)

if result is not None:

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

print(f"Результат: {result}")

self.history.append(f"{num1} {operator} {num2} = {result}")

self.ask\_save\_to\_memory(result)

if not self.ask\_continue():

break

if self.ask\_view\_history():

show\_history()

if self.ask\_customize():

self.customize()

def ask\_save\_to\_memory(self, result):

save\_choice = input("Зберегти результат у пам'ять? (так/ні): ").lower()

if save\_choice == 'так':

save\_to\_memory(result)

def ask\_continue(self):

return input("Чи хочете виконати ще одне обчислення? (так/ні): ").lower() == 'так'

def ask\_view\_history(self):

return input("Бажаєте переглянути історію? (так/ні): ").lower() == 'так'

def ask\_customize(self):

return input("Бажаєте налаштувати калькулятор? (так/ні): ").lower() == 'так'

def customize(self):

try:

new\_decimal\_places = int(input("Введіть кількість десяткових розрядів для результатів (за замовчуванням 2): "))

self.decimal\_places = new\_decimal\_places

print(f"Кількість десяткових розрядів змінена на: {self.decimal\_places}")

self.history.append(f"Користувач змінив кількість десяткових знаків на: {self.decimal\_places}")

except ValueError:

print("Помилка: потрібно ввести ціле число.")

self.history.append("Помилка при налаштуванні десяткових розрядів.")

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

def validate\_input(prompt, validation\_fn, error\_message):

while True:

user\_input = input(prompt)

if validation\_fn(user\_input):

return user\_input

print(error\_message)

def get\_operator():

return validate\_input("Введіть оператор (+, -, \*, /, %, ^, √): ",

lambda op: op in ['+', '-', '\*', '/', '%', '^', '√'],

"Помилка: некоректний оператор.")

def calculate(num1, num2, operator):

try:

operations = {

'+': lambda: num1 + num2,

'-': lambda: num1 - num2,

'\*': lambda: num1 \* num2,

'/': lambda: num1 / num2 if num2 != 0 else None,

'%': lambda: num1 % num2,

'^': lambda: num1 \*\* num2,

'√': lambda: math.sqrt(num1) if num1 >= 0 else None,

}

return operations.get(operator, lambda: None)()

except (ZeroDivisionError, ValueError) as e:

print(f"Помилка: {e}")

return None

**Папка GlobalVariables, файл GlobalVariables.py:**

memory = None

history = []

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

def get\_user\_input(prompt):

return input(prompt)

def display\_output(message):

print(message)

**Папка logs, файл logs.py:**

def log\_error(error\_message):

with open("error\_log.txt", "a") as log\_file:

log\_file.write(f"Помилка: {error\_message}\n")

def log\_calculation(expression, result):

with open("calculation\_log.txt", "a") as log\_file:

log\_file.write(f"Обчислення: {expression} = {result}\n")

**Папка tools, файл tools.py:**

from GlobalVariables.GlobalVariables import memory, history

def save\_to\_memory(result):

global memory

memory = result

print(f"Збережено у пам'яті: {memory}")

def get\_from\_memory():

if memory is None:

print("Помилка: пам'ять порожня.")

return None

return memory

def show\_history():

if history:

for i, entry in enumerate(history, 1):

print(f"{i}: {entry}")

else:

print("Історія порожня.")

**Файл main.py:**

import sys

import os

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

parent\_dir = os.path.abspath(os.path.join(current\_dir, '..'))

sys.path.append(parent\_dir)

from classes.calculator import Calculator

def main():

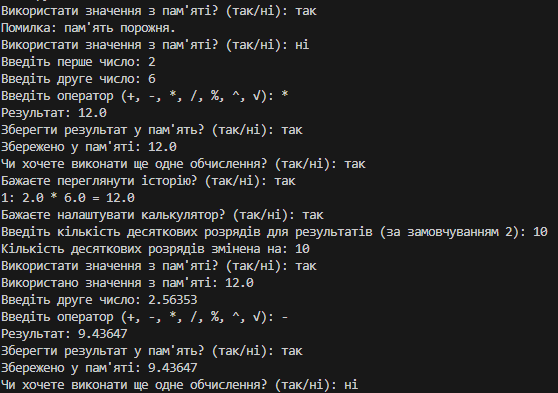
calculator = Calculator()

calculator.run()

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

main()

**Результат виконання:**

****

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