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

​*Спеціалізовані мови програмування*

на тему:

“**Основи побудови об’єктно-орієнтованих додатків на Python** ”

Виконав студент РІ-31

**Гуледза Тарас**

Прийняв: Щербак С.С.

Львів – 2024

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

**Завдання лабораторної 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 програми:

from Lib import functions  
from configs import global\_value  
  
  
class Calculator:  
 def \_\_init\_\_(self):  
 *"""Initialize the Calculator with memory value from global settings."""* self.memory\_value = global\_value.memory\_value  
  
 def get\_input(self):  
 *"""  
 Get user input for operands and operator.  
 This function handles input validation for operands and operators.  
 """* while True:  
 try:  
 *# Get the first operand, or recall it from memory* user\_input = input('Input first operand (or MR for memory recall): ').upper()  
 if user\_input == 'MR':  
 first\_operand = self.memory\_value  
 print(f"Recalled from memory: {first\_operand}")  
 else:  
 first\_operand = float(user\_input)  
  
 *# Get and validate the operator* operator = input("Input operator (+, -, \*, /, ^, %, sq): ").strip()  
 if not self.validate\_operator(operator):  
 print("Invalid operator. Try again.")  
 continue  
  
 *# Get the second operand, or recall it from memory* while True:  
 user\_input = input('Input second operand (or MR for memory recall): ').upper()  
 if user\_input == 'MR':  
 second\_operand = self.memory\_value  
 print(f"Recalled from memory: {second\_operand}")  
 else:  
 second\_operand = float(user\_input)  
  
 *# If division operator, check for zero in the second operand* if operator == '/' and second\_operand == 0:  
 print("Error: Division by zero is not allowed. Please enter a non-zero second operand.")  
 else:  
 break *# Exit the loop if the second operand is valid* return first\_operand, second\_operand, operator  
  
 except ValueError:  
 print("Invalid number format. Try again.")  
  
 def validate\_operator(self, operator):  
 *"""  
 Validate if the operator is in the list of supported operators.  
 """* valid\_operators = ['+', '-', '\*', '/', '^', '%', 'sq']  
 return operator in valid\_operators  
  
 def calculate(self, first\_operand, second\_operand, operator):  
 *"""  
 Perform the calculation based on the operator and operands.  
 This function handles errors like division by zero and invalid square roots.  
 """* try:  
 result = functions.choose\_operator(first\_operand, second\_operand, operator)  
 *# Log the history of the calculation* functions.log\_history(first\_operand, operator, second\_operand, round(result, global\_value.round\_number))  
 return round(result, global\_value.round\_number)  
  
 except (ZeroDivisionError, ValueError) as e:  
 print(e)  
 return None  
  
 def run(self):  
 *"""  
 Main loop for running the calculator.  
 This handles memory operations and allows the user to perform calculations repeatedly.  
 """* while True:  
 first\_operand, second\_operand, operator = self.get\_input()  
 result = self.calculate(first\_operand, second\_operand, operator)  
 if result is not None:  
 print(f"Result: {result}")  
  
 choice\_memory = input(  
 'Would you like to store result in memory (MS), add to memory (M+), clear memory (MC), or skip? '  
 ).upper()  
  
 match choice\_memory:  
 case 'MS':  
 self.memory\_value = result  
 print(f"Stored {result} in memory.")  
 case 'M+':  
 self.memory\_value += result  
 print(f"Added {result} to memory. New memory value: {self.memory\_value}.")  
 case 'MC':  
 self.memory\_value = 0  
 print("Memory cleared.")  
  
 if input("Do you want to view history? (yes/no): ").strip().lower() == 'yes':  
 print(functions.show\_history())  
  
 if input('Do you want to make another calculation? (yes/no): ').lower() != 'yes':  
 break

**Висновок**

Виконавши ці завдання, я перетворив консольний калькулятор 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.