МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Лабораторна робота №2

з дисципліни

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

на тему

Основи побудови об'єктно-орієнтованих додатків на Python

Виконав:

ст. гр. РІ-21сп

Павлусь В.П

Прийняв:

Шербак С.С.

Львів-2024

**Мета лабораторної роботи:** 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.

**Виконання роботи**

**Вміст файлу classes.py:**

import math

from abc import ABC, abstractmethod

from variables import memory, history, decimal\_point

class Memory:

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

self.\_\_memory = memory

def set\_memory(self, value):

self.\_\_memory = value

def get\_memory(self):

return self.\_\_memory

def clear\_memory(self):

self.\_\_memory = 0

class History:

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

self.\_\_history = history

def add\_to\_history(self, expression, result):

self.\_\_history.append(f"{expression} = {result}")

def show\_history(self):

if self.\_\_history:

print("History of calculations:")

for entry in self.\_\_history:

print(entry)

else:

print("History is empty.")

class Set:

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

self.\_\_decimal\_point = decimal\_point

def set\_decimal\_point(self):

point = input("Enter the number of symbols after the decimal point: ")

if point.isalpha():

raise TypeError("Please, enter a positive integer!")

elif int(point) >= 0:

self.\_\_decimal\_point = int(point)

print(f"Number of symbols after decimal point - {self.\_\_decimal\_point}.")

else:

raise ValueError("Please, try again. Number of symbols must not be negated!")

def get\_decimal\_point(self):

return self.\_\_decimal\_point

class BaseCalculator(ABC):

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

self.memory = Memory()

self.history = History()

self.set = Set()

def get\_input(self):

try:

num1\_input = input("Enter first number or 'm' for use value from memory: ")

num1 = self.memory.get\_memory() if num1\_input.lower() == 'm' else float(num1\_input)

operator = self.get\_operator()

num2 = None

if operator != '√':

while True:

try:

num2\_input = input("Enter second number or 'm' for use value from memory: ")

num2 = self.memory.get\_memory() if num2\_input.lower() == 'm' else float(num2\_input)

break

except ValueError:

print("Incorrect input! Please, try again!")

return num1, operator, num2

except ValueError:

print("Incorrect input! Please, try again!")

return self.get\_input()

def check\_operator(self, operator):

return operator in ['+', '-', '\*', '/', '^', '√', '%']

def get\_operator(self):

while True:

operator = input("Choose one of the operators (+, -, \*, /, ^, √, %): ")

if self.check\_operator(operator):

return operator

else:

print("Incorrect operator! Please, try again!")

@abstractmethod

def perform\_operators(self, num1, operator, num2):

pass

class Calculator(BaseCalculator):

def perform\_operators(self, num1, operator, num2):

decimal\_point = self.set.get\_decimal\_point()

match operator:

case '+':

return round(num1 + num2, decimal\_point)

case '-':

return round(num1 - num2, decimal\_point)

case '\*':

return round(num1 \* num2, decimal\_point)

case '/':

if num2 == 0:

raise ZeroDivisionError("You cant't divide by zero!")

return round(num1 / num2, decimal\_point)

case '^':

return round(math.pow(num1, num2), decimal\_point)

case '√':

if num1 < 0:

raise ValueError("You can't get a square root from a negative number!")

return round(math.sqrt(num1), decimal\_point)

case '%':

if num2 == 0:

raise ZeroDivisionError("You can't get a percentage from zero!")

return round(num1 % num2, decimal\_point)

case \_:

raise ValueError("Incorrect operator! Please, try again!")

def calculate(self):

while True:

num1, operator, num2 = self.get\_input()

result = self.perform\_operators(num1, operator, num2)

if result is not None:

print(f"Result of calculations: {result}")

self.history.add\_to\_history(f"{num1} {operator} {num2 if num2 is not None else ''}", result)

if input("Do you want to save result? (y/n)").lower() == 'y':

self.memory.set\_memory(result)

print(f"Result {result} was saved.")

if input("Do you want to do new calculation? (y/n)").lower() == 'n':

break

**Вміст файлу interface.py:**

from classes import Calculator

def main():

try:

calc = Calculator()

except TypeError as e:

print(e)

exit(1)

while True:

print("\nCalculator")

print("1. Perform calculations.")

print("2. Show history.")

print("3. Get result from memory.")

print("4. Clean memory.")

print("5. Change the number of symbols after the decimal point.")

print("6. Exit.")

choice = input("Select option: ")

match choice:

case '1':

try:

calc.calculate()

except ZeroDivisionError as e:

print(e)

except ValueError as e:

print(e)

case '2':

calc.history.show\_history()

case '3':

print(f"Saved value: {calc.memory.get\_memory()}")

case '4':

calc.memory.clear\_memory()

print("Saved value cleared.")

case '5':

try:

calc.set.set\_decimal\_point()

except ValueError as e:

print(e)

except TypeError as e:

print(e)

case '6':

print("Exit from calculator.")

break

case \_:

print("Wrong choice! Please try again!")

**Вміст файлу variables.py:**

memory = 0

history = []

decimal\_point = 2

**Вміст файлу runner.py:**

from interface import main

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

main()

Результатом виконання є калькулятор, дію якого зображено на рис. 1.

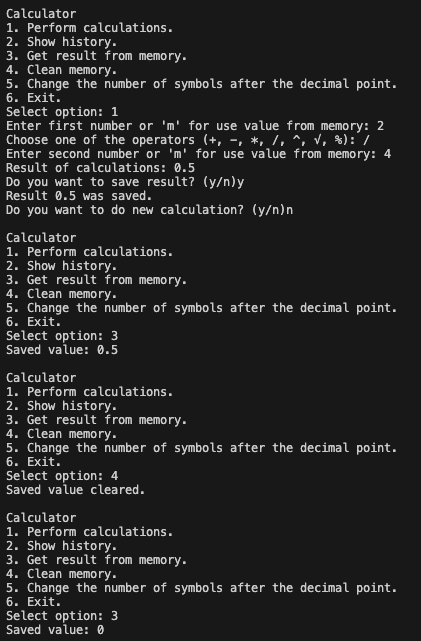


Рис. 1 – Результат роботи калькулятора

**Висновок:** У ході виконання ЛР я перетворив консольний калькулятор у об'єктно-орієнтований калькулятор, використовуючи класи в Python. Калькулятор виконує обчислення, зберігає результат у пам’яті та має зручний інтерфейс.