Міністерство освіти і науки України
Національний університет «Львівська політехніка»

# лабораторної роботи №2
з дисципліни «Спеціалізовані мови програмування»
на тему «Основи побудови об’єктно-орієнтованих додатків на Python»

Виконав:
Кулявець В. Р.
Перевірив:
Щербак С.С.

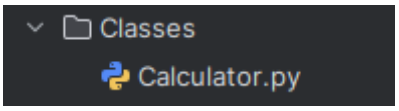
Львів 2024

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

Git: https://github.com/Pivinter/-.git

Завдання 1: Створення класу Calculator

Створіть клас Calculator, який буде служити основою для додатка калькулятора.




Мал.1.Клас Calculator.

Завдання 2: Ініціалізація калькулятора

Реалізуйте метод __init__ у класі Calculator для ініціалізації необхідних атрибутів або змінних.

```
class Calculator:
    def __init__(self):
```

Мал.2.метод __init__ .

Завдання 3: Введення користувача

Перемістіть функціональність введення користувача в метод у межах класу Calculator. Метод повинен приймати введення для двох чисел і оператора.

```
def set_decimal_places():
    try:
        decimal_places = int(input("Enter the number of decimal places to display (0-10): "))
        if 0 <= decimal_places <= 10:
            settings["decimal_places"] = decimal_places
            print(f"Decimal places set to {decimal_places}.")
        else:
            print("Please enter a number between 0 and 10.")
    except ValueError:
        print("Invalid input. Please enter a number.")
```

Мал.3. Функціональність.

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.

```python
def add(self, x, y):  1 usage (1 dynamic)
    return x + y

def subtract(self, x, y):  1 usage (1 dynamic)
    return x - y

def multiply(self, x, y):  1 usage (1 dynamic)
    return x * y

def divide(self, x, y):  1 usage (1 dynamic)
    if y == 0:
        return "Error: Division by zero."
    return x / y

def exponentiate(self, x, y):  1 usage (1 dynamic)
    return x ** y

def sqrt(self, x):  1 usage (1 dynamic)
    if x < 0:
        return "Error: Cannot take square root of a negative number."
    return math.sqrt(x)

def remainder(self, x, y):  1 usage (1 dynamic)
    if y == 0:
        return "Error: Division by zero."
    return x % y
```

Мал.4. Перевірка оператора.

Завдання 5: Обчислення

Створіть метод у класі Calculator, який виконує обчислення на основі введення користувача (наприклад, додавання, віднімання, множення, ділення).

```python
    operator = input("✍  Enter operator (+, -, *, /, ^, %): ")
```

Мал.5. бчислення на основі введення користувача.

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.

```
def divide(self, x, y):  1 usage (1 dynamic)
    if y == 0:
        return "Error: Division by zero."
    return x / y

def exponentiate(self, x, y):  1 usage (1 dynamic)
    return x ** y

def sqrt(self, x):  1 usage (1 dynamic)
    if x < 0:
        return "Error: Cannot take square root of a negative number."
    return math.sqrt(x)

def remainder(self, x, y):  1 usage (1 dynamic)
    if y == 0:
        return "Error: Division by zero."
    return x % y
```

Мал.6. Обробка помилок.

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.

```
    elif choice == 'Q':
        print("👋 Exiting the calculator. Goodbye!")
        return False
    else:
        print("❌ Invalid input. Please enter a valid choice.")
    return True
```

Мал.7. Повторення обчислень.

Завдання 8: Десяткові числа

Модифікуйте клас Calculator для обробки десяткових чисел (плаваюча кома) для більш точних обчислень.

```
num1 = float(input("🔢 Enter the first number: "))
#operator = input("✍️  Enter operator (+, -, *, /, ^
num2 = float(input("🔢 Enter the second number: "))
```

Мал.8. Десяткові числа.

Завдання 9: Додаткові операції

Розширте клас Calculator, щоб підтримувати додаткові операції, такі як піднесення до степеня (^), квадратний корінь (√) та залишок від ділення (%).

```python
def exponentiate(self, x, y):  1 usage (1 dynamic)
    return x ** y

def sqrt(self, x):  1 usage (1 dynamic)
    if x < 0:
        return "Error: Cannot take square root of a negative number."
    return math.sqrt(x)

def remainder(self, x, y):  1 usage (1 dynamic)
    if y == 0:
        return "Error: Division by zero."
    return x % y
```

Мал.9. Десяткові числа.

Завдання 10: Інтерфейс, зрозумілий для користувача

Покращте інтерфейс користувача у межах класу Calculator, надавши чіткі запити, повідомлення та форматування виводу для зручності читання.

```
🧮  Welcome to the Friendly Calculator!
Please select an operation:
1. Add (+)
2. Subtract (-)
3. Multiply (*)
4. Divide (/)
5. Exponentiation (x^y)
6. Square Root (√x)
7. Remainder (x % y)
M+. Save to Memory
MR. Recall from Memory
MC. Clear Memory
H. View History
CH. Clear History
S1. Set Decimal Places
S2. Toggle Auto Memory Save
S3. Toggle Auto Memory Clear
Q. Quit
🔽 Enter your choice:
```

Мал.10.Інтерфейс, зрозумілий для користувача.

Висновок: Виконавши ці завдання, я перетворив консольний калькулятор у об'єктно-орієнтований калькулятор, використовуючи класи в Python. Цей проект допоможе вам вивчити концепції об'єктно-орієнтованого програмування та організацію, зберігаючи функціональність і інтерфейс користувача калькулятора.