МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №6

з дисципліни

«Спеціалізовані мови приграмування»

на тему

«Розробка та Unit тестування Python додатку»

Виконав:

ст. гр. РІ-32

Сергій БИХНЮК

Прийняв:

Сергій Щербак

Львів – 2024

**Мета роботи:**

Cтворення юніт-тестів для додатка-калькулятора на основі класів.

**Завдання:**

**Завдання 1**: Тестування Додавання

Напишіть юніт-тест, щоб перевірити, що операція додавання в вашому додатку-калькуляторі працює правильно. Надайте тестові випадки як для позитивних, так і для негативних чисел.

**Завдання 2:** Тестування Віднімання

Створіть юніт-тести для переконання, що операція віднімання працює правильно. Тестуйте різні сценарії, включаючи випадки з від'ємними результатами.

**Завдання 3:** Тестування Множення

Напишіть юніт-тести, щоб перевірити правильність операції множення в вашому калькуляторі. Включіть випадки з нулем, позитивними та від'ємними числами.

**Завдання 4:** Тестування Ділення

Розробіть юніт-тести для підтвердження точності операції ділення. Тести повинні охоплювати ситуації, пов'язані з діленням на нуль та різними числовими значеннями.

**Завдання 5:** Тестування Обробки Помилок

Створіть юніт-тести, щоб перевірити, як ваш додаток-калькулятор обробляє помилки. Включіть тести для ділення на нуль та інших потенційних сценаріїв помилок. Переконайтеся, що додаток відображає відповідні повідомлення про помилки.

**Код програми:**

import unittest

from unittest.mock import patch

from lab1.calculator import Calculator # Імпортуйте клас Calculator

class TestCalculator(unittest.TestCase):

def setUp(self):

# Ініціалізація об'єкта Calculator перед кожним тестом

self.calc = Calculator()

@patch('builtins.input', side\_effect=['+', '5', '3'])

def test\_addition(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

result = self.calc.calculate()

self.assertEqual(result, 8.0)

@patch('builtins.input', side\_effect=['-', '10', '4'])

def test\_subtraction(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

result = self.calc.calculate()

self.assertEqual(result, 6.0)

@patch('builtins.input', side\_effect=['\*', '7', '6'])

def test\_multiplication(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

result = self.calc.calculate()

self.assertEqual(result, 42.0)

@patch('builtins.input', side\_effect=['/', '8', '2'])

def test\_division(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

result = self.calc.calculate()

self.assertEqual(result, 4.0)

@patch('builtins.input', side\_effect=['^', '3', '2'])

def test\_exponentiation(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

result = self.calc.calculate()

self.assertEqual(result, 9.0)

@patch('builtins.input', side\_effect=['√', '16'])

def test\_square\_root(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

result = self.calc.calculate()

self.assertEqual(result, 4.0)

@patch('builtins.input', side\_effect=['%', '10', '3'])

def test\_modulus(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

result = self.calc.calculate()

self.assertEqual(result, 1.0)

@patch('builtins.input', side\_effect=['/', '5', '0'])

def test\_division\_by\_zero(self, mock\_input):

self.calc.get\_operator()

self.calc.get\_numbers()

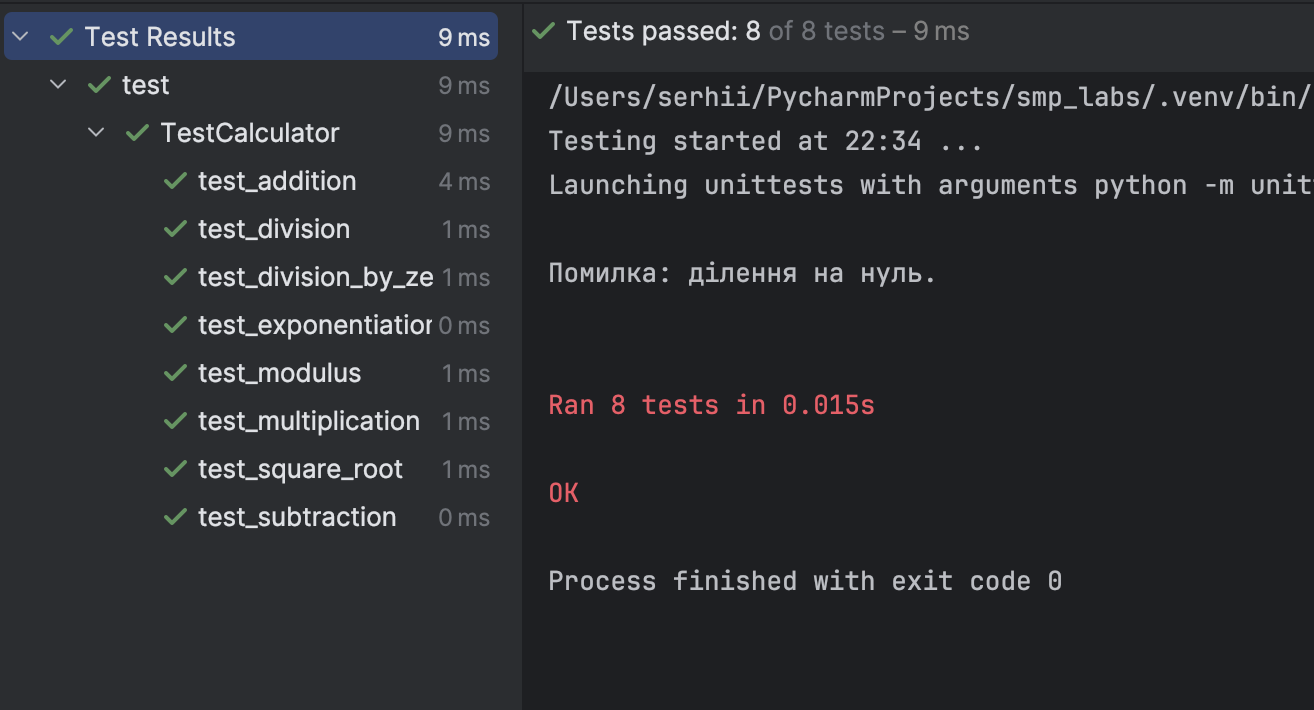
result = self.calc.calculate()

self.assertIsNone(result) # Перевірка, що при діленні на нуль результат - None

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Виконання:**

****

**Висновок:**

Виконавши ці завдання, у мене є набір юніт-тестів, які перевіряють правильність основних арифметичних операцій у мойому додатку-калькуляторі. Ці тести допоможуть виявити та виправити будь-які проблеми або помилки, які можуть виникнути під час розробки чи обслуговування мого додатку, забезпечуючи його надійність і точність