МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Лабораторна робота №6

з дисципліни

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

на тему

Розробка та Unit тестування Python додатку

Виконав:

ст. гр. РІ-21сп

Шептицький Р.М

Прийняв:

Шербак С.С.

Львів-2024

**Мета лабораторної 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.**

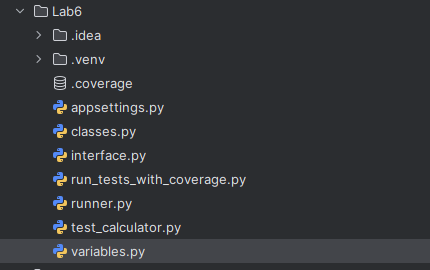
Розробіть юніт-тести для підтвердження точності операції ділення. Тести повинні охоплювати ситуації, пов'язані з діленням на нуль та різними числовими значеннями.

**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.

**Виконання роботи**

**Структура файлів:**



**Вміст файлу test\_calculator.py:**

import os  
import sys  
import unittest  
from unittest.mock import patch  
from classes import Calculator  
  
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../lab2")))  
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../Shared")))  
  
class TestCalculator(unittest.TestCase):  
 def setUp(self):  
 self.calc = Calculator()  
  
 # Тест перевіряє правильне введення чисел і оператора  
 @patch('builtins.input', side\_effect=['5', '+', '3'])  
 def test\_get\_input\_valid(self, mock\_input):  
 num1, operator, num2 = self.calc.get\_input()  
 self.assertEqual(num1, 5.0)  
 self.assertEqual(operator, '+')  
 self.assertEqual(num2, 3.0)  
  
 # Тест перевіряє використання пам'яті та оператора квадратного кореня  
 @patch('builtins.input', side\_effect=['m', '√'])  
 def test\_get\_input\_memory\_sqrt(self, mock\_input):  
 self.calc.memory.set\_memory(9.0)  
 num1, operator, num2 = self.calc.get\_input()  
 self.assertEqual(num1, 9.0)  
 self.assertEqual(operator, '√')  
 self.assertIsNone(num2)  
  
 # Тест перевіряє обробку неправильного оператора та повторне запитання оператора  
 @patch('builtins.input', side\_effect=['$', '+'])  
 def test\_get\_operator\_invalid\_then\_valid(self, mock\_input):  
 operator = self.calc.get\_operator()  
 self.assertEqual(operator, '+')  
  
 # Тест перевіряє правильність розпізнавання коректних операторів  
 def test\_check\_operator\_valid(self):  
 self.assertTrue(self.calc.check\_operator('+'))  
 self.assertTrue(self.calc.check\_operator('-'))  
 self.assertTrue(self.calc.check\_operator('\*'))  
 self.assertTrue(self.calc.check\_operator('/'))  
 self.assertTrue(self.calc.check\_operator('^'))  
 self.assertTrue(self.calc.check\_operator('√'))  
 self.assertTrue(self.calc.check\_operator('%'))  
  
 # Тест перевіряє розпізнавання некоректних операторів  
 def test\_check\_operator\_invalid(self):  
 self.assertFalse(self.calc.check\_operator('&'))  
 self.assertFalse(self.calc.check\_operator('!'))  
 self.assertFalse(self.calc.check\_operator('invalid'))  
 self.assertFalse(self.calc.check\_operator('1'))  
  
 # Тест перевіряє встановлення та отримання значення з пам'яті  
 def test\_memory\_set\_get(self):  
 self.calc.memory.set\_memory(10)  
 self.assertEqual(self.calc.memory.get\_memory(), 10)  
  
 # Тест перевіряє очищення пам'яті  
 def test\_memory\_clear(self):  
 self.calc.memory.set\_memory(10)  
 self.calc.memory.clear\_memory()  
 self.assertEqual(self.calc.memory.get\_memory(), 0)  
  
 # Тест перевіряє додавання записів в історію та перевірку наявності запису  
 def test\_history\_add\_and\_show(self):  
 self.calc.history.add\_to\_history("1 + 1", 2)  
 self.assertIn("1 + 1 = 2", self.calc.history.\_History\_\_history)  
  
 # Тест перевіряє встановлення кількості знаків після коми  
 def test\_set\_decimal\_places(self):  
 with patch('builtins.input', side\_effect=['2']):  
 self.calc.settings.set\_decimal\_places()  
 self.assertEqual(self.calc.settings.get\_decimal\_places(), 2)  
  
 # Тест перевіряє обробку неправильного введення для кількості знаків після коми  
 def test\_set\_decimal\_places\_invalid(self):  
 with patch('builtins.input', side\_effect=['-1']):  
 with self.assertRaises(ValueError):  
 self.calc.settings.set\_decimal\_places()  
  
 # Тести для базових операцій: додавання, віднімання, множення, ділення  
 def test\_addition(self):  
 self.assertEqual(self.calc.perform\_operators(1, '+', 1), 2)  
 self.assertEqual(self.calc.perform\_operators(-1, '+', -1), -2)  
 self.assertEqual(self.calc.perform\_operators(-1, '+', 1), 0)  
  
 def test\_subtraction(self):  
 self.assertEqual(self.calc.perform\_operators(1, '-', 1), 0)  
 self.assertEqual(self.calc.perform\_operators(-1, '-', -1), 0)  
 self.assertEqual(self.calc.perform\_operators(1, '-', 2), -1)  
  
 def test\_multiplication(self):  
 self.assertEqual(self.calc.perform\_operators(2, '\*', 3), 6)  
 self.assertEqual(self.calc.perform\_operators(0, '\*', 5), 0)  
 self.assertEqual(self.calc.perform\_operators(-2, '\*', 3), -6)  
  
 def test\_division(self):  
 self.assertEqual(self.calc.perform\_operators(6, '/', 2), 3)  
 with self.assertRaises(ZeroDivisionError):  
 self.calc.perform\_operators(1, '/', 0)  
  
 # Тест для операції квадратного кореня  
 def test\_square\_root(self):  
 self.assertEqual(self.calc.perform\_operators(4, '√', None), 2)  
 with self.assertRaises(ValueError):  
 self.calc.perform\_operators(-4, '√', None)  
  
 # Тест для операції остачі від ділення (modulo)  
 def test\_modulo(self):  
 self.assertEqual(self.calc.perform\_operators(5, '%', 3), 2)  
 with self.assertRaises(ZeroDivisionError):  
 self.calc.perform\_operators(1, '%', 0)  
  
 # Тести для операції піднесення до степеня (exponentiation)  
 def test\_exponentiation(self):  
 self.assertEqual(self.calc.perform\_operators(2, '^', 3), 8)  
 self.assertEqual(self.calc.perform\_operators(5, '^', 0), 1)  
 self.assertEqual(self.calc.perform\_operators(-2, '^', 3), -8)  
 self.assertEqual(self.calc.perform\_operators(2, '^', -1), 0.5)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

Результат виконання: а саме результат запущених юніт тестів на рисунку 1

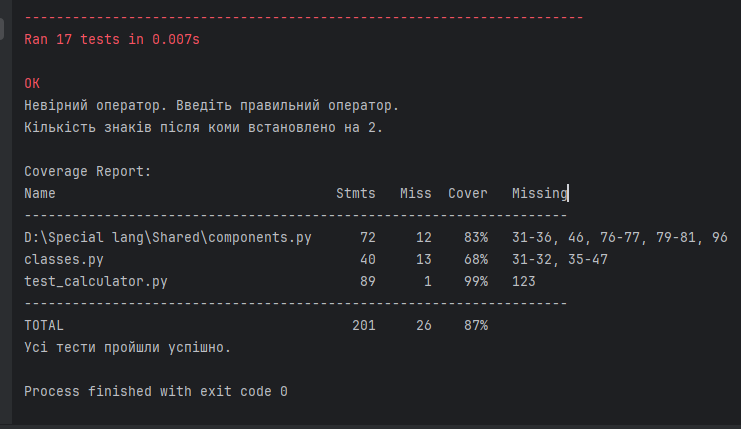


Рисунок 1 – Результат тестування

**Висновок:** У ході виконання ЛР я створив набір юніт-тестів, які перевіряють правильність основних арифметичних операцій у додатку-калькуляторі. Ці тести допомагають виявити та виправити будь-які проблеми або помилки, які можуть виникнути під час розробки чи обслуговування додатку, забезпечуючи його надійність і точність.