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. ІТ-21сп

Олександр КОЗАК

Прийняв

доцент каф. ІСМ:

Сергій ЩЕРБАК

|  |  |
| --- | --- |
| **Балів** | **Дата** |
|  |  |

Львів-2023

**Мета**: Cтворення юніт-тестів для додатка-калькулятора на основі класів.

**Хід роботи:**

**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.

**unittests\_for\_calculator.py**

import unittest

from unittest.mock import patch

from lab2.calculator\_engine import Calculator\_engine

class CalculatorUnittest(unittest.TestCase):

def setUp(self):

# Initialization of the calculator object before each test

self.calculator = Calculator\_engine()

# Test cases for addition

def test\_positive\_numbers\_addition(self):

result = self.calculator.calculateValue(3, 7, '+')

self.assertEqual(result, 10, "Addition of positive numbers is incorrect")

def test\_negative\_numbers\_addition(self):

result = self.calculator.calculateValue(-5, -3, '+')

self.assertEqual(result, -8, "Addition of negative numbers is incorrect")

def test\_mixed\_numbers\_addition(self):

result = self.calculator.calculateValue(8, -4, '+')

self.assertEqual(result, 4, "Addition of mixed numbers is incorrect")

def test\_zero\_addition(self):

result = self.calculator.calculateValue(0, 12, '+')

self.assertEqual(result, 12, "Addition with zero is incorrect")

# Test cases for subtraction

def test\_positive\_numbers\_subtraction(self):

result = self.calculator.calculateValue(10, 4, '-')

self.assertEqual(result, 6, "Subtraction of positive numbers is incorrect")

def test\_negative\_numbers\_subtraction(self):

result = self.calculator.calculateValue(-5, -3, '-')

self.assertEqual(result, -2, "Subtraction of negative numbers is incorrect")

def test\_mixed\_numbers\_subtraction(self):

result = self.calculator.calculateValue(8, -4, '-')

self.assertEqual(result, 12, "Subtraction of mixed numbers is incorrect")

def test\_zero\_subtraction(self):

result = self.calculator.calculateValue(0, 7, '-')

self.assertEqual(result, -7, "Subtraction with zero is incorrect")

# Test cases for multiplication

def test\_positive\_numbers\_multiplication(self):

result = self.calculator.calculateValue(5, 3, '\*')

self.assertEqual(result, 15, "Multiplication of positive numbers is incorrect")

def test\_negative\_numbers\_multiplication(self):

result = self.calculator.calculateValue(-4, -2, '\*')

self.assertEqual(result, 8, "Multiplication of negative numbers is incorrect")

def test\_mixed\_numbers\_multiplication(self):

result = self.calculator.calculateValue(6, -2, '\*')

self.assertEqual(result, -12, "Multiplication of mixed numbers is incorrect")

def test\_zero\_multiplication(self):

result = self.calculator.calculateValue(8, 0, '\*')

self.assertEqual(result, 0, "Multiplication with zero is incorrect")

# Test cases for division

def test\_positive\_numbers\_division(self):

result = self.calculator.calculateValue(10, 2, '/')

self.assertEqual(result, 5, "Division of positive numbers is incorrect")

def test\_negative\_numbers\_division(self):

result = self.calculator.calculateValue(-6, -2, '/')

self.assertEqual(result, 3, "Division of negative numbers is incorrect")

def test\_mixed\_numbers\_division(self):

result = self.calculator.calculateValue(8, -2, '/')

self.assertEqual(result, -4, "Division of mixed numbers is incorrect")

def test\_zero\_division(self):

result = self.calculator.calculateValue(0, 7, '/')

self.assertEqual(result, 0, "Division involving zero should return 0")

# Test cases for getOperator method

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

def test\_getOperator\_valid\_operators(self, mock\_input):

# The calls to getOperator should succeed with valid operators ('/', '√')

for expected\_result in ('/', '√'):

result = self.calculator.getOperator("Enter an operator: ")

self.assertEqual(result, expected\_result)

@patch('builtins.input', side\_effect=['+', '-', 'invalid', '\*', '^'])

def test\_getOperator\_invalid\_then\_valid\_operators(self, mock\_input):

# The first call to getOperator should fail with an invalid operator ('invalid')

with self.assertRaises(ValueError):

self.calculator.getOperator("Enter an operator: ")

# The next calls to getOperator should succeed with valid operators ('+', '\*', '^')

for expected\_result in ('+', '\*', '^'):

result = self.calculator.getOperator("Enter an operator: ")

self.assertEqual(result, expected\_result)

# Test case for getParamValue method

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

def test\_getParamValue\_invalid\_input\_then\_valid\_input(self, mock\_input):

result = self.calculator.getParamValue("Enter a number: ")

self.assertEqual(result, 5.0)

**main.py**

import runner

import unittest

class main:

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

run\_unittest = runner.Unittest\_calculator\_runner()

run\_unittest.run\_unittest()

**runner.py**

from unittests\_for\_calculator import CalculatorUnittest

import unittest

class Unittest\_calculator\_runner:

def run\_unittest (self):

# Load unit tests from CalculatorTest

test\_suite = unittest.TestLoader().loadTestsFromTestCase(CalculatorUnittest)

# Create a test runner with output to a file

with open('test\_results.txt', 'w') as f:

runner = unittest.TextTestRunner(stream=f, verbosity=2)

runner.run(test\_suite)

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

*Рис. 1 Результат виконання юніттесту*

**Висновок:** Виконавши ці завдання, у мене виник набір юніт-тестів, які перевіряють правильність основних арифметичних операцій у моєму 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.