

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Звіт

до лабораторної роботи №6

Розробка та Unit тестування Python додатку

Виконала:

ст. гр. РІ-32
Довгошия А.А

Балів	Дата

Прийняв:

асис. каф. ІСМ

Щербак С.С

Львів — 2024

Мета: Створення 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 Помилки

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.

Lab6_calculator_test.py

```
import unittest
```

```
import os # Remove if not needed
```

```
import coverage # Remove if not needed
```

```
import math
```

```
from src.classes.lab2.operations import Operations
```

```
class TestCalculatorOperations(unittest.TestCase):
```

```
    """
```

```
    Test case for verifying the correct functionality of various calculator  
operations
```

```
    like addition, subtraction, multiplication, etc.
```

```
    """
```

```
    def setUp(self):
```

```
        """
```

```
        Sets up the necessary resources for each test case.
```

```
        """
```

```
        self.operations = Operations()
```

```
    def test_addition(self):
```

```
        """
```

```
        Test the addition operation of the calculator with different sets of numbers.
```

```
        """
```

```
        self.assertEqual(self.operations.calculate(3, 2, "+"), 5)
```

```
        self.assertEqual(self.operations.calculate(-3, -2, "+"), -5)
```

```
        self.assertEqual(self.operations.calculate(3, -2, "+"), 1)
```

```

def test_subtraction(self):
    """
    Test the subtraction operation of the calculator with different sets of
    numbers.
    """
    self.assertEqual(self.operations.calculate(3, 2, "-"), 1)
    self.assertEqual(self.operations.calculate(-3, -2, "-"), -1)
    self.assertEqual(self.operations.calculate(3, -2, "-"), 5)
    self.assertEqual(self.operations.calculate(-3, 2, "-"), -5)

def test_multiplication(self):
    """
    Test the multiplication operation of the calculator with different sets of
    numbers.
    """
    self.assertEqual(self.operations.calculate(3, 2, "*"), 6)
    self.assertEqual(self.operations.calculate(-3, -2, "*"), 6)
    self.assertEqual(self.operations.calculate(3, -2, "*"), -6)
    self.assertEqual(self.operations.calculate(3, 0, "*"), 0)

def test_division(self):
    """
    Test the division operation of the calculator with different sets of numbers.
    """
    self.assertEqual(self.operations.calculate(6, 2, "/"), 3)
    self.assertEqual(self.operations.calculate(-6, -2, "/"), 3)
    self.assertEqual(self.operations.calculate(6, -2, "/"), -3)
    self.assertEqual(self.operations.calculate(0, 3, "/"), 0)

```

```
self.assertEqual(self.operations.calculate(3, 0, "/"), "division by zero cannot  
be done")
```

```
def test_sqrt_negative(self):
```

```
    """
```

```
    Test the square root operation with a negative number.
```

```
    """
```

```
self.assertEqual(self.operations.calculate(-9, None, "√"), "number cannot be  
negative")
```

```
self.assertEqual(self.operations.calculate(9, None, "√"), 3)
```

```
def test_square_root(self):
```

```
    """
```

```
    Test the square root operation with positive and zero numbers.
```

```
    """
```

```
self.assertEqual(self.operations.calculate(9, None, "√"), 3)
```

```
self.assertEqual(self.operations.calculate(0, None, "√"), 0)
```

```
self.assertEqual(self.operations.calculate(2, None, "√"), math.sqrt(2))
```

```
self.assertEqual(self.operations.calculate(-2, None, "√"), "number cannot be  
negative")
```

```
def test_exponentiation(self):
```

```
    """
```

```
    Test the exponentiation operation with different sets of numbers.
```

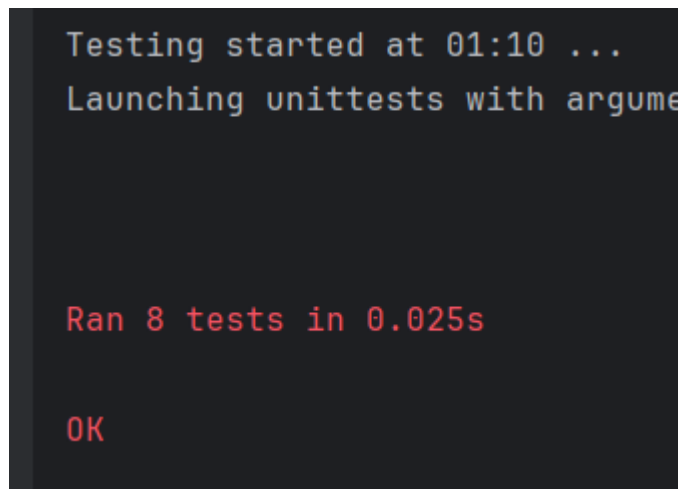
```
    """
```

```
self.assertEqual(self.operations.calculate(2, 3, "^"), 8)
```

```
self.assertEqual(self.operations.calculate(5, 0, "^"), 1)
```

```
self.assertEqual(self.operations.calculate(-2, 2, "^"), 4)
```

```
def test_modulo(self):  
    """  
    Test the modulo operation with different sets of numbers.  
    """  
    self.assertEqual(self.operations.calculate(10, 3, "%"), 1)  
    self.assertEqual(self.operations.calculate(10, 5, "%"), 0)  
    self.assertEqual(self.operations.calculate(-10, 3, "%"), 2)  
  
if __name__ == "__main__":  
    unittest.main()
```



```
Testing started at 01:10 ...  
Launching unittests with arguments  
  
Ran 8 tests in 0.025s  
  
OK
```

Рис.1. Результат програми.

Висновок: на цій лабораторній роботі я вивчила створення юніт-тестів для додатка-калькулятора на основі класів