МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №6

з курсу

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

Виконав студент

групи ІТ-21сп

**Гузар А.І.**

Прийняв

**Щербак С.С.**

Львів - 2023

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

**План 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.Код виконаних завдань представлено нижче.

Клас calculator\_test:

import unittest

from Data.lab\_2.BaseCalculator import BaseCalculator

class TestCalculator(unittest.TestCase):

def setUp(self):

# Initialize the calculator with some default values

self.calculator = BaseCalculator(num\_1=10, num\_2=5, operator='+', custom\_round=2)

# Test getting num\_1 property

def test\_get\_num\_1(self):

result = self.calculator.num\_1

self.assertEqual(result, 10)

# Test getting num\_2 property

def test\_get\_num\_2(self):

result = self.calculator.num\_2

self.assertEqual(result, 5)

# Test getting operator property

def test\_get\_operator(self):

result = self.calculator.operator

self.assertEqual(result, '+')

# Test getting custom\_round property

def test\_get\_custom\_round(self):

result = self.calculator.custom\_round

self.assertEqual(result, 2)

# Test valid number

def test\_valid\_number(self):

result = self.calculator.\_number\_validate('10')

self.assertEqual(result, 10.0)

# Test invalid number

def test\_invalid\_number(self):

with self.assertRaises(ValueError) as context:

self.calculator.\_number\_validate('abc')

self.assertEqual(str(context.exception),

"Parameter 'abc' is not number")

# Test valid operator

def test\_valid\_operator(self):

result = self.calculator.\_operator\_validate('+')

self.assertEqual(result, '+')

# Test invalid operator

def test\_invalid\_operator(self):

with self.assertRaises(TypeError) as context:

self.calculator.\_operator\_validate('$')

self.assertEqual(str(context.exception),

"Operator is not in ('+', '-', '\*', '/', '^', '√', '%')!")

# Test addition

def test\_addition(self):

self.calculator.operator = '+'

self.calculator.calc()

self.assertEqual(self.calculator.\_result, 15)

# Test subtraction

def test\_subtraction(self):

self.calculator.operator = '-'

self.calculator.calc()

self.assertEqual(self.calculator.\_result, 5)

# Test multiplication

def test\_multiplication(self):

self.calculator.operator = '\*'

self.calculator.calc()

self.assertEqual(self.calculator.\_result, 50)

# Test division with a non-zero divisor

def test\_division(self):

self.calculator.operator = '/'

self.calculator.calc()

self.assertEqual(self.calculator.\_result, 2)

# Test division by zero

def test\_division\_by\_zero(self):

self.calculator.operator = '/'

self.calculator.num\_2 = 0

with self.assertRaises(ZeroDivisionError):

self.calculator.calc()

# Test exponentiation

def test\_exponentiation(self):

self.calculator.operator = '^'

self.calculator.calc()

self.assertEqual(self.calculator.\_result, 100000)

# Test square root with a non-negative number

def test\_square\_root(self):

self.calculator.operator = '√'

self.calculator.calc()

self.assertEqual(self.calculator.\_result, 3.16)

# Test square root with a negative number

def test\_square\_root\_negative(self):

self.calculator.operator = '√'

self.calculator.num\_1 = -10

with self.assertRaises(ArithmeticError):

self.calculator.calc()

# Test remainder

def test\_remainder(self):

self.calculator.operator = '%'

self.calculator.calc()

self.assertEqual(self.calculator.\_result, 0)

# Test remainder with division by zero

def test\_remainder\_division\_by\_zero(self):

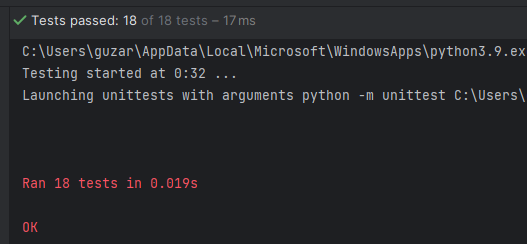
self.calculator.operator = '%'

self.calculator.num\_2 = 0

with self.assertRaises(ZeroDivisionError):

self.calculator.calc()

На рисунку 1 зображено результат виконання програми.



*Рис.1 Робота юніт тестів для калькулятора.*

**Висновок:** під час виконання лабораторної роботи я навчився розробляти ASCII ART генератор для візуалізації текстових даних без використання зовнішніх бібліотек.