# **МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

# **Кафедра ІСМ**

****

**Звіт**

# **до лабораторної роботи №6**

# **З дисципліни “Спеціалізовані мови програмування”**

**Виконав:**

**ст. гр. ІТ-32**

**Пазюк Сергій**

**Прийняв:**

**Щербак С.С.**

# **Львів — 2023**

**Тема роботи:**  Розробка та Unit тестування Python додатку.

**Мета роботи:** 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.

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

**Хід роботи**

**Реалізований код:**

import unittest

import sys

import os

# Додаємо шлях до вашої папки "Спеціалізовані мови програмування" для пошуку файлу lab2.py

sys.path.append(os.path.join(os.path.dirname(file), '..', '..', '..', 'Спеціалізовані мови програмування'))

from lab2.lab2 import Calculator

class TestCalculator(unittest.TestCase):

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

calculator = Calculator()

result = calculator.calculate(2, '+', 3)

self.assertEqual(result, 5, "Adding positive numbers does not work correctly!", "Додавання позитивних чисел працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(-2, '+', -3)

self.assertEqual(result, -5, "Adding negative numbers does not work properly!", "Додавання негативних чисел працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(5, '-', 3)

self.assertEqual(result, 2, "Subtraction of positive numbers does not work correctly!", "Віднімання позитивних чисел працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(-2, '-', -3)

self.assertEqual(result, 1, "Subtraction of negative numbers does not work properly!", "Віднімання негативних чисел працює неправильно")

def test\_multiplication\_with\_zero(self):

calculator = Calculator()

result = calculator.calculate(5, '\*', 0)

self.assertEqual(result, 0, "Multiplication by zero does not work properly!", "Множення на нуль працює неправильно")

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

calculator = Calculator()

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

self.assertEqual(result, 15, "Multiplication of positive numbers does not work correctly!", "Множення позитивних чисел працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(-5, '\*', -3)

self.assertEqual(result, 15, "Multiplication of negative numbers does not work properly", "Множення негативних чисел працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(5, '/', 0)

self.assertEqual(result, "Error: Cannot divide by zero!", "Division by zero does not work properly", "Помилка: Не можна ділити на нуль!", "Ділення на нуль працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(15, '/', 3)

self.assertEqual(result, 5, "Division of positive numbers does not work correctly", "Ділення позитивних чисел працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(-15, '/', -3)

self.assertEqual(result, 5, "Division of negative numbers does not work properly", "Ділення негативних чисел працює неправильно")

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

calculator = Calculator()

result = calculator.calculate(10, '/', 0)

self.assertEqual(result, "Error: Cannot divide by zero!", "Повідомлення про помилку ділення на нуль неправильне")

def test\_invalid\_operator\_error\_message(self):

calculator = Calculator()

result = calculator.calculate(5, '$', 3)

self.assertEqual(result, "Error: Invalid operator!", "Повідомлення про помилку невірного оператора неправильне")

def test\_invalid\_number\_format\_error\_message(self):

calculator = Calculator()

result = calculator.calculate('abc', '+', 3)

self.assertEqual(result, "Error: Invalid number format!", "Повідомлення про помилку невірного формату числа неправильне")

def test\_negative\_number\_square\_root\_error\_message(self):

calculator = Calculator()

result = calculator.calculate(-25, '√', 0)

self.assertEqual(result, "Error: Negative number under the square root!", "Повідомлення про помилку витягування кореня з від'ємного числа неправильне")

if name == 'main':

unittest.main()

**Результат виконання программи**:

**Висновки:** Виконуючи ці завдання, я створив Генератор ASCII-арту без використання зовнішніх бібліотек.