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

Костецький І.С.

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:**

A screenshot of a computer

Description automatically generated

**Вміст файлу run\_tests.py:**

import os

import sys

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(file), "..", "..")))

import coverage

import unittest

def main():

# Start coverage

cov = coverage.Coverage()

cov.start()

# Discover and run tests

loader = unittest.TestLoader()

suite = loader.discover(os.path.dirname(file), pattern='test\_\*.py')

result = unittest.TextTestRunner(verbosity=2).run(suite)

# Stop coverage and save report

cov.stop()

# Якщо тестів не було виконано

if not result.testsRun:

print("Увага: Жодних тестів не було виконано.")

else:

cov.save()

cov.report()

cov.html\_report(directory='coverage\_html\_report')

if name == 'main':

main()

**Вміст файлу test\_calculator.py:**

import os

import sys

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(file), "..")))

import unittest

from unittest.mock import patch

from Shared.Classes.calculatorSettings import CalculatorSettings

from Shared.Classes.inputHandler import InputHandler

from Shared.Classes.historyManager import HistoryManager

from Shared.Classes.memory import Memory

from Shared.Classes.Calculator import Calculator

class TestCalculator(unittest.TestCase):

def setUp(self):

self.calculator = Calculator()

def test\_addition(self):

self.assertEqual(self.calculator.perform\_operation(5, '+', 3), 8)

def test\_subtraction(self):

self.assertEqual(self.calculator.perform\_operation(5, '-', 3), 2)

def test\_multiplication(self):

self.assertEqual(self.calculator.perform\_operation(5, '\*', 3), 15)

def test\_division(self):

self.assertEqual(self.calculator.perform\_operation(6, '/', 3), 2)

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

with self.assertRaises(ZeroDivisionError):

self.calculator.perform\_operation(6, '/', 0)

def test\_exponentiation(self):

self.assertEqual(self.calculator.perform\_operation(2, '^', 3), 8)

def test\_square\_root(self):

self.assertEqual(self.calculator.perform\_operation(9, '√', None), 3)

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

with self.assertRaises(ValueError):

self.calculator.perform\_operation(-9, '√', None)

def test\_modulo(self):

self.assertEqual(self.calculator.perform\_operation(10, '%', 3), 1)

def test\_invalid\_operator(self):

with self.assertRaises(ValueError):

self.calculator.perform\_operation(5, '&', 3)

class TestInputHandler(unittest.TestCase):

def setUp(self):

self.input\_handler = InputHandler()

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

def test\_get\_operator\_valid(self, mock\_input):

self.assertEqual(self.input\_handler.get\_operator(), '+')

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

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

self.assertEqual(self.input\_handler.get\_operator(), '+')

@patch('builtins.input', side\_effect=['5', '+', '3'])

def test\_get\_input(self, mock\_input):

with patch.object(self.input\_handler.memory, 'get\_memory', return\_value=10):

self.assertEqual(self.input\_handler.get\_input(), (5.0, '+', 3.0))

def test\_check\_operator(self):

self.assertTrue(self.input\_handler.check\_operator('+'))

self.assertFalse(self.input\_handler.check\_operator('&'))

class TestHistoryManager(unittest.TestCase):

def setUp(self):

self.history\_manager = HistoryManager()

self.history\_manager.\_history\_file\_path = 'Data/test\_history.txt'

def tearDown(self):

try:

os.remove(self.history\_manager.\_history\_file\_path)

except FileNotFoundError:

pass

def test\_save\_to\_history(self):

self.history\_manager.save\_to\_history("5 + 3 = 8")

with open(self.history\_manager.\_history\_file\_path, 'r') as file:

lines = file.readlines()

self.assertEqual(lines[-1].strip(), "5 + 3 = 8")

def test\_display\_history\_empty(self):

with self.assertRaises(FileNotFoundError):

self.history\_manager.display\_history()

class TestCalculatorSettings(unittest.TestCase):

def setUp(self):

self.settings = CalculatorSettings()

def test\_get\_decimals(self):

self.settings.set\_decimals(2)

self.assertEqual(self.settings.get\_decimals(), 2)

@patch('builtins.input', side\_effect=['2'])

def test\_change\_settings\_valid(self, mock\_input):

self.settings.change\_settings()

self.assertEqual(self.settings.get\_decimals(), 2)

@patch('builtins.input', side\_effect=['-1'])

def test\_change\_settings\_invalid\_negative(self, mock\_input):

with self.assertRaises(ValueError):

self.settings.change\_settings()

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

def test\_change\_settings\_invalid\_non\_numeric(self, mock\_input):

with self.assertRaises(TypeError):

self.settings.change\_settings()

class TestMemory(unittest.TestCase):

def setUp(self):

self.memory = Memory()

def test\_set\_memory(self):

self.memory.set\_memory(10)

self.assertEqual(self.memory.get\_memory(), 10)

def test\_get\_memory\_default(self):

self.assertEqual(self.memory.get\_memory(), 0)

if name == 'main':

unittest.main()

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

A screenshot of a computer program

Description automatically generated

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.