Міністерство освіти і науки України

Національний університет «Львівська політехніка»

******

Звіт про виконання лабораторної роботи №6  
з дисципліни  
“Спеціалізовані мови програмування”  
на тему  
“Розробка та Unit тестування Python додатку ”

Виконала:  
студентка групи РІ-31,  
Пятницька Вікторія

Прийняв:

Щербак С.С.

**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.

**Код програми:**

***TestAddition.py****:*

import random

import string

import unittest

import sys

import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from bll.classes.calculator.calculator import Calculator

from app\_settings import decimal\_places

class AdditionUnitTests(unittest.TestCase):

def setUp(self):

self.operator = '+'

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

test\_first\_operand = random.uniform(1, 100)

test\_second\_operand = random.uniform(1, 100)

expected = test\_first\_operand + test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

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

test\_first\_operand = random.uniform(-100, -1)

test\_second\_operand = random.uniform(-100, -1)

expected = test\_first\_operand + test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

def test\_addition\_wrong\_input(self):

test\_first\_operand = ''.join(random.choice(string.ascii\_letters) for \_ in range(10))

test\_second\_operand = ''.join(random.choice(string.ascii\_letters) for \_ in range(10))

calc = Calculator()

with self.assertRaises(TypeError):

calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

***TestDivision.py****:*

import random

import string

import unittest

import sys

import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from bll.classes.calculator.calculator import Calculator

from app\_settings import decimal\_places

class DivisionUnitTests(unittest.TestCase):

def setUp(self):

self.operator = '/'

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

test\_first\_operand = random.uniform(1, 100)

test\_second\_operand = random.uniform(1, 100)

expected = test\_first\_operand / test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

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

test\_first\_operand = random.uniform(-100, -1)

test\_second\_operand = random.uniform(-100, -1)

expected = test\_first\_operand / test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

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

test\_first\_operand = random.uniform(1, 100)

test\_second\_operand = 0

calc = Calculator()

with self.assertRaises(ZeroDivisionError):

calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

***TestMultiplication.py****:*

import random

import unittest

import sys

import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from bll.classes.calculator.calculator import Calculator

from app\_settings import decimal\_places

class MultiplicationUnitTests(unittest.TestCase):

def setUp(self):

self.operator = '\*'

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

test\_first\_operand = random.uniform(1, 100)

test\_second\_operand = random.uniform(1, 100)

expected = test\_first\_operand \* test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

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

test\_first\_operand = random.uniform(-100, -1)

test\_second\_operand = random.uniform(-100, -1)

expected = test\_first\_operand \* test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

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

test\_first\_operand = random.uniform(1, 100)

test\_second\_operand = 0.0

expected = test\_first\_operand \* test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

***TestSubstraction.py****:*

import random

import unittest

import sys

import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

from bll.classes.calculator.calculator import Calculator

from app\_settings import decimal\_places

class SubstractionUnitTests(unittest.TestCase):

def setUp(self):

self.operator = '-'

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

test\_first\_operand = random.uniform(1, 100)

test\_second\_operand = random.uniform(1, 100)

expected = test\_first\_operand - test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

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

test\_first\_operand = random.uniform(-100, -1)

test\_second\_operand = random.uniform(-100, -1)

expected = test\_first\_operand - test\_second\_operand

calc = Calculator()

result = calc.calculate(test\_first\_operand, test\_second\_operand, self.operator)

self.assertAlmostEqual(expected, result, places = decimal\_places)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Висновки:** внаслідок виконання лабораторної роботи я створила юніт-тести для додатка-калькулятора на основі класів.

**GitHub:**[***https://github.com/viksi01/Calculator/tree/lab6***](https://github.com/viksi01/Calculator/tree/lab6)