МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторної роботи № 5
«Розробка та Unit тестування Python додатку»
з дисципліни "Спеціалізовані мови програмування”

Виконала:
ст. гр. ІТ-32,
Троцько О. М.
Прийняв:
Щербак С. С.

ЛЬВІВ – 2023

Мета: створення 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 програми:

```
# test_addition.py
"""
Module: test_math_operations

This module contains test cases for the count_sum function in the math_operations module.
"""
```

```
import unittest

from classes.lab1.math_operations.math_operations import count_sum


class AdditionTestCase(unittest.TestCase):
    """
    Test case for the count_sum function in the math_operations module.
    """

    def test_addition_positive_numbers(self):
        """
        Test addition of two positive numbers.
        """
        result = count_sum(5, 7)
        self.assertEqual(result, 12, "Expected 5 + 7 to equal 12")

    def test_addition_negative_numbers(self):
        """
        Test addition of two negative numbers.
        """
        result = count_sum(-7, -5)
        self.assertEqual(result, -12, "Expected -7 + (-5) to equal -12")

    def test_addition_mixed_numbers(self):
        """
        Test addition of a positive and a negative number.
        """
        result = count_sum(22, -10)
        self.assertEqual(result, 12, "Expected 22 + (-10) to equal 12")

    def test_addition_int_and_zero(self):
        """
        Test addition of an integer and zero.
        """
        result = count_sum(1, 0)
        self.assertEqual(result, 1, "Expected 1 + 0 to equal 1")

    def test_addition_positive_floats(self):
        """
```

```
        Test addition of two positive floats.
        """
        result = count_sum(3.2, 2.8)
        self.assertAlmostEqual(result, 6.0, places=1, msg="Expected 3.2 + 2.8 to be
approximately 6.0")

    def test_addition_negative_floats(self):
        """
        Test addition of two negative floats.
        """
        result = count_sum(-1.5, -2.5)
         self.assertAlmostEqual(result, -4.0, places=1, msg="Expected -1.5 + (-2.5)
to be approximately -4.0")

    def test_addition_mixed_floats(self):
        """
        Test addition of a positive float and a negative float.
        """
        result = count_sum(4.3, -1.7)
        self.assertAlmostEqual(result, 2.6, places=1, msg="Expected 4.3 + (-1.7) to
be approximately 2.6")

    def test_addition_float_and_zero(self):
        """
        Test addition of a float and zero.
        """
        result = count_sum(0.1, 0)
        self.assertEqual(result, 0.1, "Expected 0.1 + 0 to equal 0.1")

    def test_addition_int_and_float(self):
        """
        Test addition of an integer and a float.
        """
        result = count_sum(3, 0.3)
         self.assertAlmostEqual(result, 3.3, places=1, msg="Expected 3 + 0.3 to be
approximately 3.3")

    def test_addition_float_and_int(self):
        """
        Test addition of a float and an integer.
```

```
        """
        result = count_sum(0.4, 4)
         self.assertAlmostEqual(result, 4.4, places=1, msg="Expected 0.4 + 4 to be approximately 4.4")
```

 # test_division.py

```
"""
Module: test_math_operations

This module contains test cases for the count_quotient function in the math_operations module.
"""
import unittest

from classes.lab1.math_operations.math_operations import count_quotient

class DivisionTestCase(unittest.TestCase):
    """
    Test case for the count_quotient function in the math_operations module.
    """

    def test_division_positive_numbers(self):
        """
        Test division of two positive numbers.
        """
        result = count_quotient(40, 2)
        self.assertEqual(result, 20, "Expected 40 / 2 to equal 20")

    def test_division_negative_numbers(self):
        """
        Test division of two negative numbers.
        """
        result = count_quotient(-100, -5)
        self.assertEqual(result, 20, "Expected -100 / -5 to equal 20")

    def test_division_mixed_numbers(self):
        """
        Test division of a positive number and a negative number.
        """
        result = count_quotient(60, -3)
```

```
        self.assertEqual(result, -20, "Expected 60 / -3 to equal -20")

    def test_division_int_by_zero(self):
        """
        Test division of an integer by zero.
        """
        with self.assertRaises(ZeroDivisionError):
            count_quotient(20, 0)

    def test_division_positive_floats(self):
        """
        Test division of two positive floats.
        """
        result = count_quotient(10.0, 3.0)
          self.assertAlmostEqual(result, 3.3333333, places=5, msg="Expected 10.0 /
3.0 to be approximately 3.3333333")

    def test_division_negative_floats(self):
        """
        Test division of two negative floats.
        """
        result = count_quotient(-8.5, -2.5)
         self.assertAlmostEqual(result, 3.4, places=1, msg="Expected -8.5 / -2.5 to
be approximately 3.4")

    def test_division_mixed_floats(self):
        """
        Test division of a positive float and a negative float.
        """
        result = count_quotient(15.6, -3.2)
         self.assertAlmostEqual(result, -4.875, places=3, msg="Expected 15.6 / -3.2
to be approximately -4.875")

    def test_division_float_by_zero(self):
        """
        Test division of a float by zero.
        """
        with self.assertRaises(ZeroDivisionError):
            count_quotient(20.0, 0)
```

```
    def test_division_int_and_float(self):
        """
        Test division of an integer and a float.
        """
        result = count_quotient(3, 0.5)
         self.assertAlmostEqual(result, 6.0, places=1, msg="Expected 3 / 0.5 to be approximately 6.0")

    def test_division_float_and_int(self):
        """
        Test division of a float and an integer.
        """
        result = count_quotient(2.5, 5)
         self.assertAlmostEqual(result, 0.5, places=1, msg="Expected 2.5 / 5 to be approximately 2.5")
```

 # test_multiplication.py

```
"""
Module: test_math_operations

This module contains test cases for the count_product function in the math_operations module.
"""
import unittest

from classes.lab1.math_operations.math_operations import count_product

class MultiplicationTestCase(unittest.TestCase):
    """
    Test case for the count_product function in the math_operations module.
    """

    def test_multiplication_positive_numbers(self):
        """
        Test multiplication of two positive numbers.
        """
        result = count_product(5, 2)
        self.assertEqual(result, 10, "Expected 5 * 2 to equal 10")

    def test_multiplication_negative_numbers(self):
```

```
        """
        Test multiplication of two negative numbers.
        """
        result = count_product(-5, -2)
        self.assertEqual(result, 10, "Expected -5 * (-2) to equal 10")

    def test_multiplication_mixed_numbers(self):
        """
        Test multiplication of a positive and a negative number.
        """
        result = count_product(5, -2)
        self.assertEqual(result, -10, "Expected 5 * (-2) to equal -10")

    def test_multiplication_int_and_zero(self):
        """
        Test multiplication of an integer and zero.
        """
        result = count_product(10, 0)
        self.assertEqual(result, 0, "Expected 10 * 0 to equal 0")

    def test_multiplication_positive_floats(self):
        """
        Test multiplication of two positive floats.
        """
        result = count_product(2.5, 3.5)
         self.assertAlmostEqual(result, 8.75, places=2, msg="Expected 2.5 * 3.5 to
be approximately 8.75")

    def test_multiplication_negative_floats(self):
        """
        Test multiplication of two negative floats.
        """
        result = count_product(-2.0, -1.5)
         self.assertAlmostEqual(result, 3.0, places=1, msg="Expected -2.0 * -1.5 to
be approximately 3.0")

    def test_multiplication_mixed_floats(self):
        """
        Test multiplication of a positive float and a negative float.
        """
```

```
        result = count_product(3.0, -2.5)
        self.assertAlmostEqual(result, -7.5, places=1, msg="Expected 3.0 * -2.5 to be approximately -7.5")

    def test_multiplication_float_and_zero(self):
        """
        Test multiplication of a float and zero.
        """
        result = count_product(10.0, 0)
        self.assertEqual(result, 0, "Expected 10.0 * 0 to equal 0")

    def test_multiplication_int_and_float(self):
        """
        Test multiplication of an integer and a float.
        """
        result = count_product(-9, -0.2)
        self.assertAlmostEqual(result, 1.8, places=1, msg="Expected -9 * (-0.2) to be approximately 1.8")

    def test_multiplication_float_and_int(self):
        """
        Test multiplication of a float and an integer.
        """
        result = count_product(-1.25, 4)
        self.assertAlmostEqual(result, -5.0, places=1, msg="Expected -1.25 * 4 to be approximately 5.0")
```

 # test_substraction.py

```
"""
Module: test_math_operations

This module contains test cases for the count_difference function in the math_operations module.
"""
import unittest

from classes.lab1.math_operations.math_operations import count_difference

class SubtractionTestCase(unittest.TestCase):
    """
```

```
    Test case for the count_difference function in the math_operations module.
    """

    def test_subtraction_positive_numbers(self):
        """
        Test subtraction of two positive numbers.
        """
        result = count_difference(20, 5)
        self.assertEqual(result, 15, "Expected 20 - 5 to equal 15")

    def test_subtraction_negative_numbers(self):
        """
        Test subtraction of two negative numbers.
        """
        result = count_difference(-5, -20)
        self.assertEqual(result, 15, "Expected -5 - (-20) to equal 15")

    def test_subtraction_mixed_numbers(self):
        """
        Test subtraction of a positive and a negative number.
        """
        result = count_difference(12, -3)
        self.assertEqual(result, 15, "Expected 12 - (-3) to equal 15")

    def test_subtraction_int_and_zero(self):
        """
        Test subtraction of an integer and zero.
        """
        result = count_difference(15, 0)
        self.assertEqual(result, 15, "Expected 15 - 0 to equal 15")

    def test_subtraction_positive_floats(self):
        """
        Test subtraction of two positive floats.
        """
        result = count_difference(5.0, 2.5)
        self.assertAlmostEqual(result, 2.5, places=2, msg="Expected 5.0 - 2.5 to be
approximately 2.5")

    def test_subtraction_negative_floats(self):
```

```
        """
        Test subtraction of two negative floats.
        """
        result = count_difference(-8.5, -3.5)
         self.assertAlmostEqual(result, -5.0, places=2, msg="Expected -8.5 - (-3.5) to be approximately -5.0")

    def test_subtraction_mixed_floats(self):
        """
        Test subtraction of a positive float and a negative float.
        """
        result = count_difference(15.6, -7.2)
         self.assertAlmostEqual(result, 22.8, places=2, msg="Expected 15.6 - (-7.2) to be approximately 22.8")

    def test_subtraction_float_and_zero(self):
        """
        Test subtraction of a float and zero.
        """
        result = count_difference(-15, 0)
        self.assertEqual(result, -15, "Expected -15 - 0 to equal -15")

    def test_subtraction_int_and_float(self):
        """
        Test subtraction of an integer and a float.
        """
        result = count_difference(6, -0.7)
         self.assertAlmostEqual(result, 6.7, places=1, msg="Expected 6 - (-0.7) to be approximately 6.7")

    def test_subtraction_float_and_int(self):
        """
        Test subtraction of a float and an integer.
        """
        result = count_difference(8.53, 5)
        self.assertAlmostEqual(result, 3.53, places=2, msg="Expected 8.53 - 5 to be approximately 3.53")

# test_exceptions.py
"""
```

```
Module: test_math_operations

This module contains test cases for the count_difference function in the
math_operations module.
"""
import unittest

from classes.lab1.math_operations.math_operations import count_difference

class SubtractionTestCase(unittest.TestCase):
    """
    Test case for the count_difference function in the math_operations module.
    """

    def test_subtraction_positive_numbers(self):
        """
        Test subtraction of two positive numbers.
        """
        result = count_difference(20, 5)
        self.assertEqual(result, 15, "Expected 20 - 5 to equal 15")

    def test_subtraction_negative_numbers(self):
        """
        Test subtraction of two negative numbers.
        """
        result = count_difference(-5, -20)
        self.assertEqual(result, 15, "Expected -5 - (-20) to equal 15")

    def test_subtraction_mixed_numbers(self):
        """
        Test subtraction of a positive and a negative number.
        """
        result = count_difference(12, -3)
        self.assertEqual(result, 15, "Expected 12 - (-3) to equal 15")

    def test_subtraction_int_and_zero(self):
        """
        Test subtraction of an integer and zero.
        """
        result = count_difference(15, 0)
```

```
        self.assertEqual(result, 15, "Expected 15 - 0 to equal 15")

    def test_subtraction_positive_floats(self):
        """
        Test subtraction of two positive floats.
        """
        result = count_difference(5.0, 2.5)
        self.assertAlmostEqual(result, 2.5, places=2, msg="Expected 5.0 - 2.5 to be
approximately 2.5")

    def test_subtraction_negative_floats(self):
        """
        Test subtraction of two negative floats.
        """
        result = count_difference(-8.5, -3.5)
         self.assertAlmostEqual(result, -5.0, places=2, msg="Expected -8.5 - (-3.5)
to be approximately -5.0")

    def test_subtraction_mixed_floats(self):
        """
        Test subtraction of a positive float and a negative float.
        """
        result = count_difference(15.6, -7.2)
         self.assertAlmostEqual(result, 22.8, places=2, msg="Expected 15.6 - (-7.2)
to be approximately 22.8")

    def test_subtraction_float_and_zero(self):
        """
        Test subtraction of a float and zero.
        """
        result = count_difference(-15, 0)
        self.assertEqual(result, -15, "Expected -15 - 0 to equal -15")

    def test_subtraction_int_and_float(self):
        """
        Test subtraction of an integer and a float.
        """
        result = count_difference(6, -0.7)
         self.assertAlmostEqual(result, 6.7, places=1, msg="Expected 6 - (-0.7) to
be approximately 6.7")
```

```
    def test_subtraction_float_and_int(self):
        """
        Test subtraction of a float and an integer.
        """
        result = count_difference(8.53, 5)
        self.assertAlmostEqual(result, 3.53, places=2, msg="Expected 8.53 - 5 to be
approximately 3.53")
```

 # test_menu.py

```
"""
Module: test_menu

This module provides a test menu for running unit tests in Lab 6.
"""
import unittest
from UI.menu import Menu
from UI.menu_item import Item
from shared.settings import get_lab_settings

settings = get_lab_settings("lab6")
TEST_DIR = settings["test_dir"]

class TestMenu:
    """
    A class representing the test menu.

    Attributes:
    - test_dir (str): The directory containing the unit test files.
    """

    def __init__(self):
        """
        Initializes a TestMenu object.

        Parameters:
        - None

        Returns:
        - None
```

```
        """
        self.test_dir = TEST_DIR

    def menu(self):
        """
        Displays a test menu and runs the selected option.
        """
        menu = Menu("\nTest Menu(Lab 6)")
        menu.add_item(Item("1", "Run Tests", self.run_tests))
        menu.add_item(Item("0", "Exit"))
        menu.run()

    def run_tests(self):
        """
        Run the unit tests for the project.

        Returns:
            None
        """
        loader = unittest.TestLoader()
        suite = loader.discover(start_dir=self.test_dir, pattern="test_*.py")
        runner = unittest.TextTestRunner(verbosity=2)
        result = runner.run(suite)

        if result.wasSuccessful():
            print("All tests passed.")
        else:
            print("Some tests failed.")
```

 # runner.py

```
"""
Module: run_tests_menu
Module provides a simple script to run the Tests Menu for Lab 6.
"""
from classes.lab6.tests.test_menu import TestMenu
def run():
    """
    Initializes and runs the Tests Menu.
    """
```

```
tests_menu = TestMenu()
tests_menu.menu()
```

GitHub Repository: https://github.com/trolchiha/SPL-labs.git

Висновок: під час виконання лабораторної роботи створила набір юніт-тестів, які перевіряють правильність основних арифметичних операцій у додатку-калькуляторі.