Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»
(МТУСИ)

Кафедра «Сетевых Информационных Технологий и Сервисов»
дисциплина «Рефакторинг баз данных и приложений»

Лабораторная работа № 2

«Рефакторинг приложений с изучением модульного тестирования»

Выполнил: студент группы БПИ2303
Березdовец Илья Сергеевич

Ссылка на GitHub: https://github.com/iberezaa

Цели лабораторной работы:
1. Изучить основные принципы и задачи модульного тестирования.
2. Научиться разрабатывать модульные тесты для проверки функциональности отдельных
частей приложения.
3. Применить модульное тестирование для контроля качества кода в процессе рефакторинга.
4. Освоить использование инструментов для написания и запуска модульных тестов.

Ход работы:
Написал плохой код для реализации калькулятора.

```python
class Calculator:

    def add(self, a, b):
        if type(a) == str or type(b) == str:
            return "Error"
        return a + b

    def subtract(self, a, b):
        if type(a) == str or type(b) == str:
            return "Error"
        return a - b

    def multiply(self, a, b):
        if type(a) == str or type(b) == str:
            return "Error"
        result = 0
        for i in range(abs(b)):
            result += a
        if b < 0:
            result = -result
        return result

    def divide(self, a, b):
        if b == 0:
            return "Can't divide by zero"
        if type(a) == str or type(b) == str:
            return "Error"
        return a / b

    def power(self, a, b):
        result = 1
        for _ in range(b):
            result *= a
        return result

    def sqrt(self, a):
        if a < 0:
            return "Error"
        x = a
        for _ in range(10):
            x = 0.5 * (x + a / x)
        return x

    def factorial(self, n):
        if n < 0:
            return "Error"
        result = 1
        while n > 0:
            result *= n
            n -= 1
        return result
```

Рис.1 (Код до рефакторинга)

2) Провел рефакторинг кода:

```python
class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

    def multiply(self, a, b):
        return a * b

    def divide(self, a, b):
        if b == 0:
            raise ValueError("Division by zero is not allowed.")
        return a / b

    def power(self, a, b):
        return a ** b

    def sqrt(self, a):
        if a < 0:
            raise ValueError("Square root of negative number is not allowed.")
        return a ** 0.5

    def factorial(self, n):
        if n < 0:
            raise ValueError("Factorial of negative number is not defined.")
        if n == 0:
            return 1
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result
```

Рис. 2 (Код после рефакторинга)

Преимущества кода после рефакторинга:

1) Использует корректные исключения (raise ValueError) для критических ошибок (например, деление на ноль, вычисление корня из отрицательного числа)
2) Все функции реализованы максимально лаконично и читаемо
3) Использует встроенные операторы (*, /, **) для стандартных математических операций, что делает код быстрее и проще.
4) Прямые математические операции работают быстрее, поскольку используют оптимизированные механизмы Python.
5) Реализует факториал корректно с явным учетом n == 0

Вывод: Код после рефакторинга стал более чище, быстрее и проще для чтения.

3) Написал модульные тесты:

```python
import pytest
from calculator import Calculator

@pytest.fixture
def calc():
    return Calculator()

def test_add(calc):
    assert calc.add(3, 5) == 8
    assert calc.add(-1, 1) == 0

def test_subtract(calc):
    assert calc.subtract(10, 5) == 5
    assert calc.subtract(0, 0) == 0

def test_multiply(calc):
    assert calc.multiply(4, 3) == 12
    assert calc.multiply(-1, -1) == 1

def test_divide(calc):
    assert calc.divide(10, 2) == 5
    with pytest.raises(ValueError):
        calc.divide(5, 0)

def test_power(calc):
    assert calc.power(2, 3) == 8
    assert calc.power(5, 0) == 1

def test_sqrt(calc):
    assert calc.sqrt(9) == 3
    with pytest.raises(ValueError):
        calc.sqrt(-4)

def test_factorial(calc):
    assert calc.factorial(5) == 120
    assert calc.factorial(0) == 1
    with pytest.raises(ValueError):
        calc.factorial(-3)

```

Рис. 3 (Модульные тесты)

4) Тесты прошли успешно.

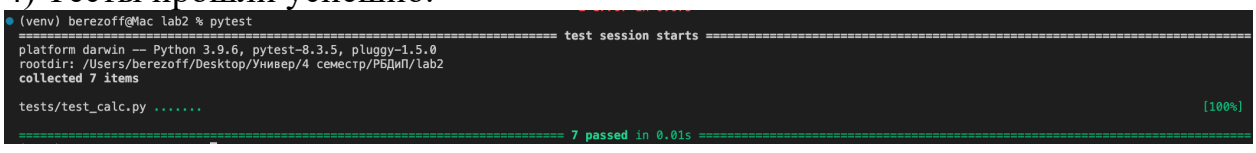

Рис. 4 (Успешно пройденные тесты)

Описание тестов:

1) Все основные методы класса Calculator проверяются: add(), subtract(), multiply(), divide(), power(), sqrt(), factorial().
2) Проверка на различные сценарии поведения программы: Проверяются стандартные случаи с корректными данными.
3) Каждый тест проверяет только одну конкретную функцию класса Calculator.
4) Код полностью совместим с pytest, что позволяет быстро и удобно запускать тесты.

Вывод:

Написан проект калькулятора. Произведен рефакторинг кода. Написаны модульный тесты. Тесты прошли успешно.