

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  
Берездовец 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.

```
1 class Calculator:
2
3     def add(self, a, b):
4         if type(a) == str or type(b) == str:
5             return "Error"
6         return a + b
7
8     def subtract(self, a, b):
9         if type(a) == str or type(b) == str:
10            return "Error"
11        return a - b
12
13    def multiply(self, a, b):
14        if type(a) == str or type(b) == str:
15            return "Error"
16        result = 0
17        for i in range(abs(b)):
18            result += a
19        if b < 0:
20            result = -result
21        return result
22
23    def divide(self, a, b):
24        if b == 0:
25            return "Can't divide by zero"
26        if type(a) == str or type(b) == str:
27            return "Error"
28        return a / b
29
30    def power(self, a, b):
31        result = 1
32        for _ in range(b):
33            result *= a
34        return result
35
36    def sqrt(self, a):
37        if a < 0:
38            return "Error"
39        x = a
40        for _ in range(10):
41            x = 0.5 * (x + a / x)
42        return x
43
44    def factorial(self, n):
45        if n < 0:
46            return "Error"
47        result = 1
48        while n > 0:
49            result *= n
50            n -= 1
51        return result
```

Рис.1 (Код до рефакторинга)

2) Провел рефакторинг кода:

```
1 class Calculator:
2     def add(self, a, b):
3         return a + b
4
5     def subtract(self, a, b):
6         return a - b
7
8     def multiply(self, a, b):
9         return a * b
10
11    def divide(self, a, b):
12        if b == 0:
13            raise ValueError("Division by zero is not allowed.")
14        return a / b
15
16    def power(self, a, b):
17        return a ** b
18
19    def sqrt(self, a):
20        if a < 0:
21            raise ValueError("Square root of negative number is not allowed.")
22        return a ** 0.5
23
24    def factorial(self, n):
25        if n < 0:
26            raise ValueError("Factorial of negative number is not defined.")
27        if n == 0:
28            return 1
29        result = 1
30        for i in range(1, n + 1):
31            result *= i
32        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:

```
1  import pytest
2  from calculator import Calculator
3
4  @pytest.fixture
5  def calc():
6      return Calculator()
7
8  def test_add(calc):
9      assert calc.add(3, 5) == 8
10     assert calc.add(-1, 1) == 0
11
12  def test_subtract(calc):
13     assert calc.subtract(10, 5) == 5
14     assert calc.subtract(0, 0) == 0
15
16  def test_multiply(calc):
17     assert calc.multiply(4, 3) == 12
18     assert calc.multiply(-1, -1) == 1
19
20  def test_divide(calc):
21     assert calc.divide(10, 2) == 5
22     with pytest.raises(ValueError):
23         calc.divide(5, 0)
24
25  def test_power(calc):
26     assert calc.power(2, 3) == 8
27     assert calc.power(5, 0) == 1
28
29  def test_sqrt(calc):
30     assert calc.sqrt(9) == 3
31     with pytest.raises(ValueError):
32         calc.sqrt(-4)
33
34  def test_factorial(calc):
35     assert calc.factorial(5) == 120
36     assert calc.factorial(0) == 1
37     with pytest.raises(ValueError):
38         calc.factorial(-3)
39
```

Рис. 3 (Модульные тесты)

#### 4) Тесты прошли успешно.

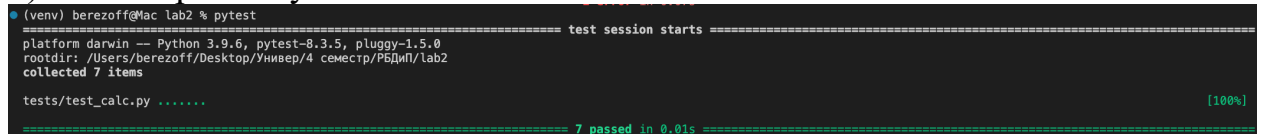
A screenshot of a terminal window with a dark background. The prompt is '(venv) berezoff@Mac lab2 %'. The command 'pytest' has been executed. The output shows the test session starting, the platform as 'darwin', Python version '3.9.6', pytest version '8.3.5', and pluggy version '1.5.0'. It indicates the root directory and that 7 items were collected. A progress bar shows 'tests/test\_calc.py' with a green progress indicator reaching 100%. The final summary line states '7 passed in 0.01s'.

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.