**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

***Факультет Информационных технологий***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**Лабораторная работа № 6**

**Дисциплина: Тестирование программного обеспечения**

**Тема: Написание модульных тестов (Unit Tests)**

**Выполнил: студент группы 221-371**

**Евдокимов Артемий Русланович**

**(Фамилия И.О )**

**Дата, подпись \_\_21.09.2025\_\_ *\_\_\_\_\_\_\_\_\_\_\_***

**(Дата) (Подпись)**

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**(Фамилия И.О., степень, звание)**

**Дата, подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_***

**(Дата) (Подпись)**

**Замечания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва**

**2025**

**Цель работы**

Понять принципы unit-тестирования и научиться тестировать изолированные функции и методы с использованием фреймворка pytest.

**Объект тестирования**

Класс Calculator с методами:

* add - сложение двух чисел
* divide - деление двух чисел
* is\_prime\_number - проверка, является ли число простым

**Реализация**

**Код калькулятора (calculator.py)**

class Calculator:

"""Класс калькулятора с базовыми математическими операциями"""

def add(self, a, b):

"""Сложение двух чисел"""

return a + b

def divide(self, a, b):

"""Деление двух чисел"""

if b == 0:

raise ZeroDivisionError("Деление на ноль невозможно!")

return a / b

def is\_prime\_number(self, n):

"""Проверка, является ли число простым"""

if not isinstance(n, int) or n <= 1:

return False

if n == 2:

return True

if n % 2 == 0:

return False

#Проверяем нечетные делители до квадратного корня из n

for i in range(3, int(n\*\*0.5) + 1, 2):

if n % i == 0:

return False

return True

**Код тестов (test\_calculator.py)**

import pytest

import sys

from calculator import Calculator

class TestCalculator:

"""Тесты для класса Calculator"""

@pytest.fixture(autouse=True)

def setup(self):

"""Фикстура для инициализации калькулятора перед каждым тестом"""

self.calc = Calculator()

yield

def \_print\_result(self, message, status="INFO"):

"""Вывод результата в stderr"""

print(f"\n{status}: {message}", file=sys.stderr)

sys.stderr.flush()

*#Тесты для метода add*

@pytest.mark.parametrize("a,b,expected", [

(2, 3, 5),

(-1, 1, 0),

(0, 0, 0),

(-5, -3, -8),

(1.5, 2.5, 4.0),

])

def test\_add(self, a, b, expected):

"""Тестирование метода сложения с различными параметрами"""

result = self.calc.add(a, b)

assert result == expected

self.\_print\_result(f"add({a}, {b}) = {result} (ожидалось: {expected})", "SUCCESS")

#Тесты для метода divide

@pytest.mark.parametrize("a,b,expected", [

(10, 2, 5),

(9, 3, 3),

(5, 2, 2.5),

(-10, 2, -5),

(0, 5, 0),

])

def test\_divide(self, a, b, expected):

"""Тестирование метода деления с различными параметрами"""

result = self.calc.divide(a, b)

assert result == expected

self.\_print\_result(f"divide({a}, {b}) = {result} (ожидалось: {expected})", "SUCCESS")

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

"""Тестирование обработки деления на ноль"""

with pytest.raises(ZeroDivisionError) as exc\_info:

self.calc.divide(5, 0)

assert "Деление на ноль невозможно" in str(exc\_info.value)

self.\_print\_result("divide(5, 0) вызвал ожидаемое исключение ZeroDivisionError", "SUCCESS")

#Тесты для метода is\_prime\_number

@pytest.mark.parametrize("n,expected", [

(2, True),

(3, True),

(5, True),

(7, True),

(11, True),

(13, True),

(17, True),

(19, True),

(4, False),

(6, False),

(8, False),

(9, False),

(10, False),

(12, False),

(25, False),

(1, False),

(0, False),

(-5, False),

])

def test\_is\_prime\_number(self, n, expected):

"""Тестирование проверки простых чисел с различными параметрами"""

result = self.calc.is\_prime\_number(n)

assert result == expected

status = "простым" if expected else "НЕ простым"

self.\_print\_result(f"Число {n} является {status}", "SUCCESS")

if \_\_name\_\_ == "\_\_main\_\_":

pytest.main([\_\_file\_\_, "-v", "-s"])

**Особенности реализации**

**1. Использование pytest.fixture**

Фикстура setup с параметром autouse=True автоматически инициализирует экземпляр калькулятора перед каждым тестом, обеспечивая изолированность тестов.

**2. Параметризация тестов**

Применена техника параметризации для тестирования методов с различными наборами входных данных:

* Метод add тестируется с 5 различными наборами параметров
* Метод divide тестируется с 5 различными наборами параметров
* Метод is\_prime\_number тестируется с 18 различными наборами параметров

**3. Проверка исключений**

Для метода divide реализована проверка возникновения исключения ZeroDivisionError при делении на ноль с проверкой текста сообщения об ошибке.

**4. Вывод результатов**

Реализован метод \_print\_result для вывода подробной информации о результатах тестирования в stderr.

**Результаты тестирования**

**Успешно выполненные тесты:**

1. **Метод add** - тестирование сложения с различными параметрами
   * Позитивные числа: (2, 3) → 5
   * Числа с разными знаками: (-1, 1) → 0
   * Нули: (0, 0) → 0
   * Отрицательные числа: (-5, -3) → -8
   * Дробные числа: (1.5, 2.5) → 4.0
2. **Метод divide** - тестирование деления с различными параметрами
   * Целочисленное деление: (10, 2) → 5
   * Деление с остатком: (5, 2) → 2.5
   * Деление отрицательных чисел: (-10, 2) → -5
   * Деление нуля: (0, 5) → 0
3. **Метод divide** - тестирование обработки исключений
   * Проверка возникновения ZeroDivisionError при делении на ноль
   * Проверка текста сообщения об ошибке: "Деление на ноль невозможно"
4. **Метод is\_prime\_number** - тестирование проверки простых чисел
   * Простые числа: 2, 3, 5, 7, 11, 13, 17, 19 (возвращает True)
   * Составные числа: 4, 6, 8, 9, 10, 12, 25 (возвращает False)
   * Граничные случаи: 1, 0, -5 (возвращает False)

**Статистика тестирования:**

* Всего тестов: 3 тестовых функции с 28 параметризованными случаями
* Успешно пройдено: 28
* Провалено: 0
* Пропущено: 0

**Выводы**

В ходе лабораторной работы были успешно выполнены все поставленные задачи:

1. Реализованы модульные тесты для всех методов класса Calculator с использованием фреймворка pytest
2. Применена техника параметризации для тестирования с разными наборами данных
3. Для метода divide реализована проверка возникновения исключения при делении на ноль
4. Обеспечено полное покрытие тестами всех методов класса

Основные преимущества реализованного подхода:

* Использование параметризации для тестирования множества сценариев
* Четкое разделение тестовых случаев на позитивные и негативные
* Проверка возникновения исключений в соответствующих ситуациях
* Подробное логирование результатов тестирования
* Простота добавления новых тестовых случаев

Разработанные модульные тесты обеспечивают надежную проверку корректности работы всех методов класса Calculator и могут быть легко расширены для покрытия дополнительных сценариев.