МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №6

Специальность ПО11

Выполнил
П. А. Захарчук
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
25.04.2025 г.

Брест 2025

Цель работы: освоить приемы тестирования кода на примере использования пакета pytest.

Задание 1: Написание тестов для мини-библиотеки покупок (shopping.py)

1. Создайте файл test_cart.py. Реализуйте следующие тесты:
   - Проверка добавления товара: после add_item("Apple", 10.0) в корзине должен быть один элемент.
   - Проверка выброса ошибки при отрицательной цене.
   - Проверка вычисления общей стоимости (total()).
2. Протестируйте метод apply_discount с разными значениями скидки:
   - 0% - цена остаётся прежней
   - 50% - цена уменьшается вдвое
   - 100% - цена становится ноль
   - < 0% и > 100% - должно выбрасываться исключение

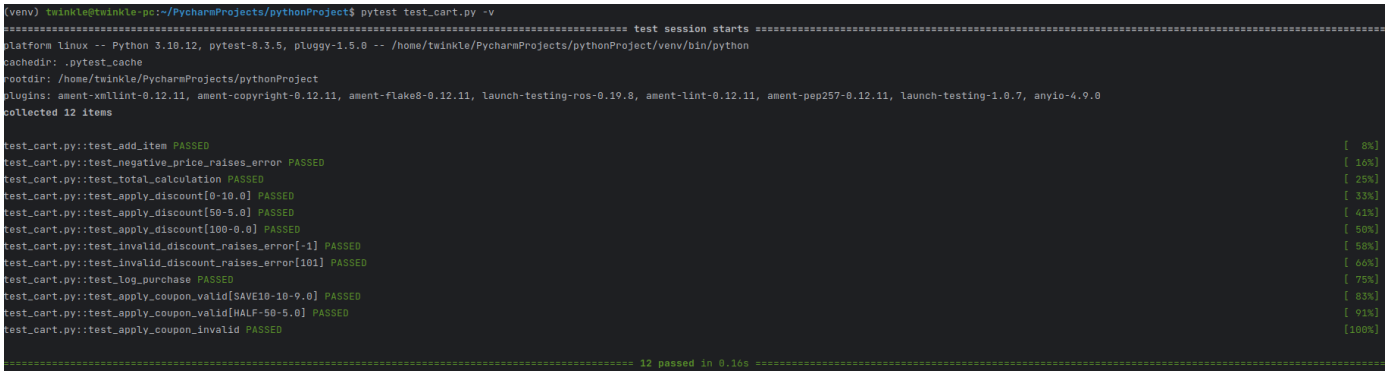
   Используйте @pytest.mark.parametrize
3. Создайте фикстуру empty_cart, которая возвращает пустой экземпляр Cart
   @pytest.fixture
   def empty_cart():
   return Cart()
   Используйте эту фикстуру в тестах, где нужно создать новую корзину.
4. Допустим, у нас есть функция, которая логирует покупку в удалённую систему:
   import requests
   def log_purchase(item):
   requests.post("https://example.com/log", json=item)
   - Замокайте requests.post, чтобы не было реального HTTP-запроса
   - Убедитесь, что он вызывается с корректными данными
5. Добавьте поддержку купонов:
   def apply_coupon(cart, coupon_code):
   coupons = {"SAVE10": 10, "HALF": 50}
   if coupon_code in coupons:
   cart.apply_discount(coupons[coupon_code])
   else:
   raise ValueError("Invalid coupon")
   - Напишите тесты на apply_coupon
   - Замокайте словарь coupons с помощью monkeypatch или patch.dict

Код программы:

```
    import pytest
import requests
from unittest.mock import patch
from shopping import Cart, apply_coupon

@pytest.fixture
def empty_cart():
    return Cart()

def test_add_item(empty_cart):
    empty_cart.add_item("Apple", 10.0)
    assert len(empty_cart.items) == 1
    assert empty_cart.items[0]["name"] == "Apple"
    assert empty_cart.items[0]["price"] == 10.0

def test_negative_price_raises_error(empty_cart):
    with pytest.raises(ValueError, match="Price cannot be negative"):
```

```
        empty_cart.add_item("Apple", -10.0)

def test_total_calculation(empty_cart):
    empty_cart.add_item("Apple", 10.0)
    empty_cart.add_item("Banana", 20.0)
    assert empty_cart.total() == 30.0

@pytest.mark.parametrize("discount, expected_total", [
    (0, 10.0),
    (50, 5.0),
    (100, 0.0),
])
def test_apply_discount(empty_cart, discount, expected_total):
    empty_cart.add_item("Apple", 10.0)
    empty_cart.apply_discount(discount)
    assert empty_cart.total() == expected_total

@pytest.mark.parametrize("invalid_discount", [-1, 101])
def test_invalid_discount_raises_error(empty_cart, invalid_discount):
    empty_cart.add_item("Apple", 10.0)
    with pytest.raises(ValueError, match="Discount must be between 0 and 100"):
        empty_cart.apply_discount(invalid_discount)

@patch("requests.post")
def test_log_purchase(mock_post, empty_cart):
    item = {"name": "Apple", "price": 10.0}
    empty_cart.add_item("Apple", 10.0)
    empty_cart.log_purchase(item)
    mock_post.assert_called_once_with("https://example.com/log", json=item)

@pytest.mark.parametrize("coupon_code, discount, expected_total", [
    ("SAVE10", 10, 9.0),
    ("HALF", 50, 5.0),
])
def test_apply_coupon_valid(empty_cart, coupon_code, discount, expected_total, monkeypatch):
    coupons = {"SAVE10": 10, "HALF": 50}
    monkeypatch.setattr("shopping.coupons", coupons)
    empty_cart.add_item("Apple", 10.0)
    apply_coupon(empty_cart, coupon_code)
    assert empty_cart.total() == expected_total

def test_apply_coupon_invalid(empty_cart, monkeypatch):
    coupons = {"SAVE10": 10, "HALF": 50}
    monkeypatch.setattr("shopping.coupons", coupons)
    empty_cart.add_item("Apple", 10.0)
    with pytest.raises(ValueError, match="Invalid coupon"):
        apply_coupon(empty_cart, "INVALID")
```

Рисунки с результатами работы программы:



Задание 2:

Напишите тесты к реализованным функциям из лабораторной работы No1. Проверьте тривиальные и граничные случае, а также варианты, когда может возникнуть исключительная ситуация. Если при реализации не использовались отдельные функции, необходимо провести рефакторинг кода.

Код программы:

Lab1_1.py

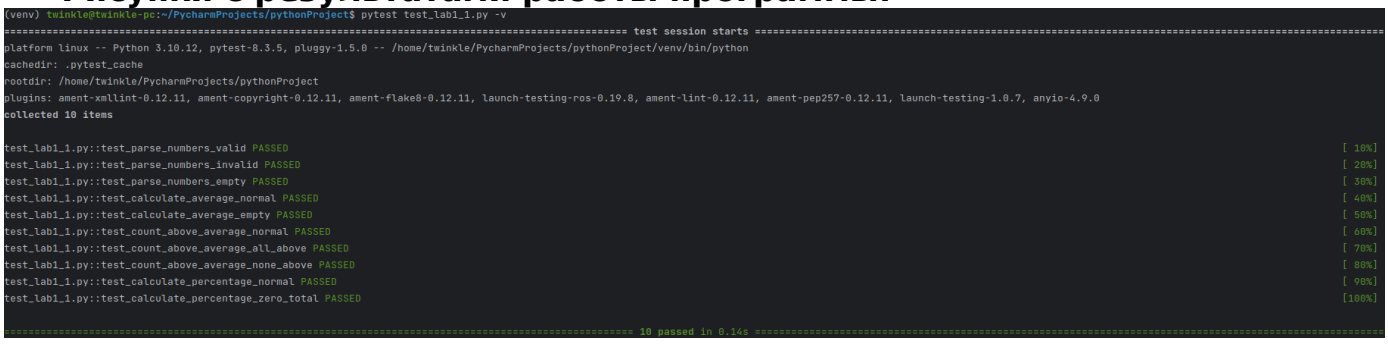
```
    import pytest
from lab1_1 import parse_numbers, calculate_average, count_above_average, calculate_percentage

def test_parse_numbers_valid():
    """Тестирование корректного ввода."""
    assert parse_numbers("1 2 3") == [1, 2, 3]
    assert parse_numbers("0") == [0]
    assert parse_numbers("-1 0 1") == [-1, 0, 1]

def test_parse_numbers_invalid():
    """Тестирование некорректного ввода."""
    with pytest.raises(ValueError, match="Все элементы должны быть целыми числами"):
        parse_numbers("1 2 a")
    with pytest.raises(ValueError, match="Все элементы должны быть целыми числами"):
        parse_numbers("1.5 2 3")

def test_parse_numbers_empty():
    """Тестирование пустого ввода."""
    with pytest.raises(ValueError, match="Все элементы должны быть целыми числами"):
        parse_numbers("")

def test_calculate_average_normal():
    """Тестирование среднего значения для обычных случаев."""
    assert calculate_average([1, 2, 3]) == 2.0
    assert calculate_average([10]) == 10.0
    assert calculate_average([-1, 0, 1]) == 0.0

def test_calculate_average_empty():
    """Тестирование пустого списка."""
    with pytest.raises(ValueError, match="Список чисел не может быть пустым"):
        calculate_average([])

def test_count_above_average_normal():
    """Тестирование подсчёта чисел, больших среднего."""
    numbers = [1, 2, 3, 4]
    average = 2.5
    assert count_above_average(numbers, average) == 2  # 3 и 4 больше 2.5
    assert count_above_average([1], 1.0) == 0  # Нет чисел больше 1.0
    assert count_above_average([1, 1, 1], 1.0) == 0  # Нет чисел больше 1.0

def test_count_above_average_all_above():
    """Тестирование, когда все числа больше среднего."""
    assert count_above_average([2, 3, 4], 1.0) == 3

def test_count_above_average_none_above():
    """Тестирование, когда нет чисел больше среднего."""
    assert count_above_average([1, 2, 3], 5.0) == 0

def test_calculate_percentage_normal():
    """Тестирование вычисления процента."""
    assert calculate_percentage(2, 4) == 50.0
    assert calculate_percentage(0, 5) == 0.0
```

```
    assert calculate_percentage(1, 1) == 100.0

def test_calculate_percentage_zero_total():
    """Тестирование деления на ноль."""
    with pytest.raises(ValueError, match="Общее количество не может быть нулевым"):
        calculate_percentage(1, 0)
```

Lab1_2.py

```
    import pytest
from lab1_2 import generate_pascal_triangle

def test_generate_pascal_triangle_zero_rows():
    """Тестирование для нуля строк."""
    assert generate_pascal_triangle(0) == []

def test_generate_pascal_triangle_one_row():
    """Тестирование для одной строки."""
    assert generate_pascal_triangle(1) == [[1]]

def test_generate_pascal_triangle_two_rows():
    """Тестирование для двух строк."""
    assert generate_pascal_triangle(2) == [[1], [1, 1]]

def test_generate_pascal_triangle_three_rows():
    """Тестирование для трёх строк."""
    assert generate_pascal_triangle(3) == [[1], [1, 1], [1, 2, 1]]

def test_generate_pascal_triangle_five_rows():
    """Тестирование для пяти строк."""
    assert generate_pascal_triangle(5) == [
        [1],
        [1, 1],
        [1, 2, 1],
        [1, 3, 3, 1],
        [1, 4, 6, 4, 1]
    ]

def test_generate_pascal_triangle_negative_rows():
    """Тестирование отрицательного количества строк."""
    with pytest.raises(ValueError, match="Количество строк не может быть отрицательным"):
        generate_pascal_triangle(-1)
```

Рисунки с результатами работы программы:



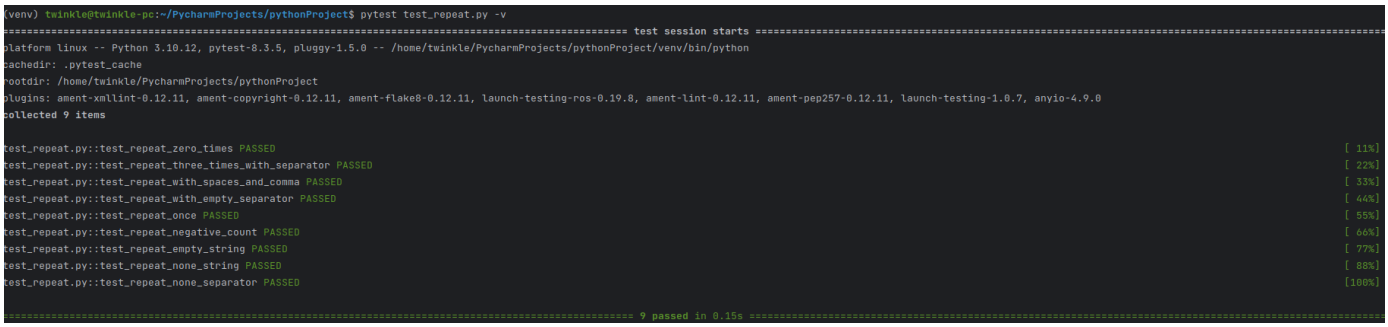
```
(venv) twinkle@twinkle-pc:~/PycharmProjects/pythonProject$ pytest test_lab1_2.py -v
============================================================================================ test session starts ============================================================================================
platform linux -- Python 3.10.12, pytest-8.3.5, pluggy-1.5.0 -- /home/twinkle/PycharmProjects/pythonProject/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/twinkle/PycharmProjects/pythonProject
plugins: ament-xmllint-0.12.11, ament-copyright-0.12.11, ament-flake8-0.12.11, launch-testing-ros-0.19.8, ament-lint-0.12.11, ament-pep257-0.12.11, launch-testing-1.0.7, anyio-4.9.0
collected 6 items

test_lab1_2.py::test_generate_pascal_triangle_zero_rows PASSED                                                                                                                                    [ 16%]
test_lab1_2.py::test_generate_pascal_triangle_one_row PASSED                                                                                                                                      [ 33%]
test_lab1_2.py::test_generate_pascal_triangle_two_rows PASSED                                                                                                                                     [ 50%]
test_lab1_2.py::test_generate_pascal_triangle_three_rows PASSED                                                                                                                                   [ 66%]
test_lab1_2.py::test_generate_pascal_triangle_five_rows PASSED                                                                                                                                    [ 83%]
test_lab1_2.py::test_generate_pascal_triangle_negative_rows PASSED                                                                                                                                [100%]

============================================================================================ 6 passed in 0.14s ============================================================================================
```

Задание 3:
Написать тесты к методу, а затем реализовать сам метод по заданной спецификации.

Код программы:

```
    import pytest
from repeat import repeat

def test_repeat_zero_times():
    """Тестирование повторения 0 раз."""
    assert repeat("e", "|", 0) == ""

def test_repeat_three_times_with_separator():
    """Тестирование повторения 3 раза с разделителем."""
    assert repeat("e", "|", 3) == "e|e|e"

def test_repeat_with_spaces_and_comma():
    """Тестирование строки с пробелами и запятой как разделителем."""
    assert repeat(" ABC ", ",", 2) == " ABC , ABC "

def test_repeat_with_empty_separator():
    """Тестирование с пустым разделителем."""
    assert repeat(" DBE ", "", 2) == " DBE  DBE "

def test_repeat_once():
    """Тестирование повторения один раз."""
    assert repeat(" DBE ", ":", 1) == " DBE "

def test_repeat_negative_count():
    """Тестирование отрицательного количества повторений."""
    with pytest.raises(ValueError, match="Количество повторений не может быть отрицательным"):
        repeat("e", "|", -2)

def test_repeat_empty_string():
    """Тестирование пустой строки с разделителем."""
    assert repeat("", ":", 3) == "::"

def test_repeat_none_string():
    """Тестирование, когда строка равна None."""
    with pytest.raises(TypeError, match="Строка должна быть типа str, а не NoneType"):
        repeat(None, "a", 1)

def test_repeat_none_separator():
    """Тестирование, когда разделитель равен None."""
    with pytest.raises(TypeError, match="Разделитель должен быть типа str, а не NoneType"):
        repeat("a", None, 2)
```

Рисунки с результатами работы программы:



Вывод: освоил приемы тестирования кода на примере использования пакета pytest.