МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №6
По дисциплине “**Современные платформы программирования**”

Выполнил:
студент группы ПО-11
Сымоник И.А.
Проверил:
Козик И. Д.

Брест 2025

Цель: освоить приемы тестирования кода на примере использования пакета pytest

Вариант 6

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

Код:

```
import requests

class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, name, price):
        if price < 0:
            raise ValueError("Price cannot be negative")
        self.items.append({"name": name, "price": price})

    def total(self):
        return sum(item["price"] for item in self.items)

    def apply_discount(self, discount_percent):
        if not 0 <= discount_percent <= 100:
            raise ValueError("Invalid discount percent")
        for item in self.items:
            item["price"] *= 1 - discount_percent / 100
```

```
    def log_purchase(self, item):
        requests.post("https://example.com/log", json=item)

    def apply_coupon(self, coupon_code):
        coupons = {"SAVE10": 10, "HALF": 50}
        if coupon_code in coupons:
            self.apply_discount(coupons[coupon_code])
        else:
            raise ValueError("Invalid coupon")

    def remove_item(self, item_name: str) -> None:
        """Удаляет товар из корзины по имени"""
        if not item_name:
            raise ValueError("Имя товара не может быть пустым")

        indexes = [i for i, item in enumerate(self.items) if item["name"] == item_name]

        if not indexes:
            raise ValueError(f"Товар {item_name} не найден в корзине")

        del self.items[indexes[0]]

from unittest.mock import patch
import pytest
from shopping import ShoppingCart

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

@pytest.fixture
def filled_cart():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0)
    cart.add_item("banana", 0.5)
    return cart

def test_add_item(empty_cart):
    """Тест добавления товара в корзину"""
    empty_cart.add_item("apple", 1.0)
    assert len(empty_cart.items) == 1
    assert empty_cart.items[0]["name"] == "apple"
    assert empty_cart.items[0]["price"] == 1.0

def test_add_item_with_quantity(empty_cart):
    """Тест добавления товара с указанием количества"""
    empty_cart.add_item("apple", 1.0)
    empty_cart.add_item("apple", 1.0)
    assert len(empty_cart.items) == 2
    assert empty_cart.items[0]["name"] == "apple"
    assert empty_cart.items[0]["price"] == 1.0
    assert empty_cart.items[1]["name"] == "apple"
    assert empty_cart.items[1]["price"] == 1.0

def test_add_existing_item(filled_cart):
    """Тест добавления существующего товара"""
    filled_cart.add_item("apple", 1.0)
    assert len(filled_cart.items) == 3
    assert filled_cart.items[0]["name"] == "apple"
    assert filled_cart.items[0]["price"] == 1.0

def test_remove_item(filled_cart):
    """Тест удаления товара из корзины"""
    filled_cart.remove_item("apple")
    assert len(filled_cart.items) == 1
    assert filled_cart.items[0]["name"] == "banana"

def test_remove_nonexistent_item(filled_cart):
```

```
    """Тест удаления несуществующего товара"""
    with pytest.raises(ValueError):
        filled_cart.remove_item("orange")

def test_remove_item_with_empty_name(filled_cart):
    """Тест удаления товара с пустым именем"""
    with pytest.raises(ValueError):
        filled_cart.remove_item("")

def test_get_total(filled_cart):
    """Тест расчета общей стоимости"""
    assert filled_cart.total() == 1.5

def test_apply_discount(filled_cart):
    """Тест применения скидки"""
    filled_cart.apply_discount(10)
    assert filled_cart.total() == pytest.approx(1.35)

def test_apply_invalid_discount(filled_cart):
    """Тест применения недопустимой скидки"""
    with pytest.raises(ValueError):
        filled_cart.apply_discount(110)

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

def test_apply_coupon_valid(empty_cart):
    """Тест применения валидного купона"""
    empty_cart.add_item("Item", 100.0)
    empty_cart.apply_coupon("SAVE10")
    assert empty_cart.total() == pytest.approx(90.0)

def test_apply_coupon_invalid(empty_cart):
    """Тест применения невалидного купона"""
    empty_cart.add_item("Item", 100.0)
    with pytest.raises(ValueError, match="Invalid coupon"):
        empty_cart.apply_coupon("INVALID")
```

```
PS C:\rep\spp\lab6\task1> pytest test_cart.py -v
============================= test session starts ==============================
platform win32 -- Python 3.13.1, pytest-8.3.5, pluggy-1.5.0 -- c:\rep\spp\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\rep\spp\lab6\task1
plugins: anyio-4.9.0
collected 12 items

test_cart.py::test_add_item PASSED                                       [  8%]
test_cart.py::test_add_item_with_quantity PASSED                         [ 16%]
test_cart.py::test_add_existing_item PASSED                              [ 25%]
test_cart.py::test_remove_item PASSED                                    [ 33%]
test_cart.py::test_remove_nonexistent_item PASSED                        [ 41%]
test_cart.py::test_remove_item_with_empty_name PASSED                    [ 50%]
test_cart.py::test_get_total PASSED                                      [ 58%]
test_cart.py::test_apply_discount PASSED                                 [ 66%]
test_cart.py::test_apply_invalid_discount PASSED                         [ 75%]
test_cart.py::test_log_purchase PASSED                                   [ 83%]
test_cart.py::test_apply_coupon_valid PASSED                             [ 91%]
test_cart.py::test_apply_coupon_invalid PASSED                           [100%]
```

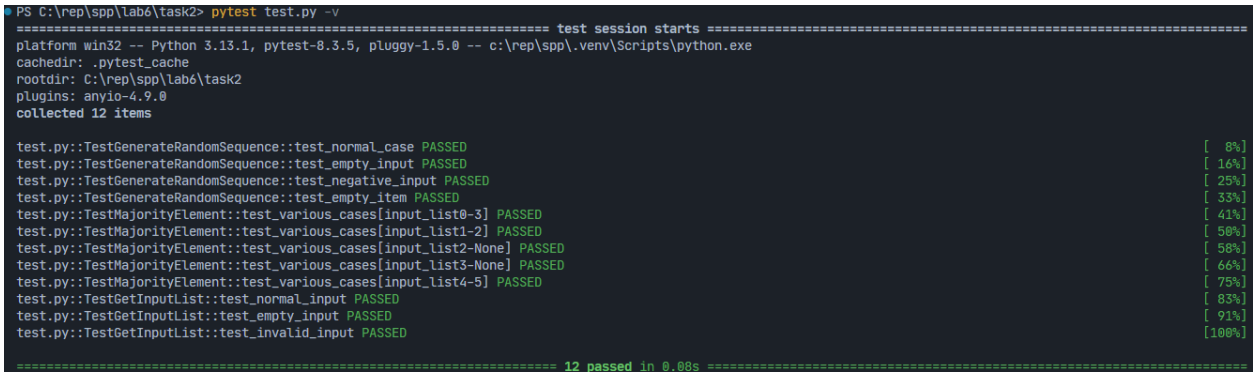
Задание 2

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.

```
import random

def generate_random_sequence(count):
    if count <= 0:
        raise IndexError("Количество элементов должно быть положительным числом")

    arr = []
```

```
    for i in range(count):
        item = input(f"Введите элемент {i+1}: ")
        if not item.strip():
            raise ValueError("Элемент не может быть пустым")
        arr.append(item)

    random.shuffle(arr)
    return ' '.join(arr)

def majority_element(nums):
    if not nums:
        return None

    candidate = None
    count = 0

    for num in nums:
        if count == 0:
            candidate = num
        count += (1 if num == candidate else -1)

    return candidate if nums.count(candidate) > len(nums) // 2 else None

def get_input_list():
    user_input = input("Введите числа через пробел: ").strip()
    if not user_input:
        raise ValueError("Ввод не может быть пустым")

    try:
        return list(map(int, user_input.split()))
    except ValueError:
        raise ValueError("Все элементы должны быть числами")


import pytest
from main import generate_random_sequence, majority_element, get_input_list
from unittest.mock import patch, MagicMock

class TestGenerateRandomSequence:
    def test_normal_case(self, monkeypatch):
        inputs = ["a", "b", "c"]
        monkeypatch.setattr('builtins.input', lambda _: inputs.pop(0))

        result = generate_random_sequence(3)
        assert sorted(result.split()) == ["a", "b", "c"]

    def test_empty_input(self):
        with pytest.raises(IndexError):
            generate_random_sequence(0)

    def test_negative_input(self):
        with pytest.raises(IndexError):
            generate_random_sequence(-1)

    def test_empty_item(self, monkeypatch):
        monkeypatch.setattr('builtins.input', lambda _: "")
        with pytest.raises(ValueError):
            generate_random_sequence(1)

class TestMajorityElement:
    @pytest.mark.parametrize("input_list, expected", [
        ([3, 2, 3], 3),
        ([2, 2, 1, 1, 1, 2, 2], 2),
        ([1, 1, 2, 2], None),
        ([], None),
        ([5], 5),
    ])
    def test_various_cases(self, input_list, expected):
        assert majority_element(input_list) == expected
```

```
class TestGetInputList:
    def test_normal_input(self, monkeypatch):
        monkeypatch.setattr('builtins.input', lambda _: "1 2 3")
        assert get_input_list() == [1, 2, 3]

    def test_empty_input(self, monkeypatch):
        monkeypatch.setattr('builtins.input', lambda _: "")
        with pytest.raises(ValueError):
            get_input_list()

    def test_invalid_input(self, monkeypatch):
        monkeypatch.setattr('builtins.input', lambda _: "1 a 3")
        with pytest.raises(ValueError):
            get_input_list()
```



Задание 3

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

Напишите метод String common(String str1, String str2) сравнивающий две строки и возвращающий наибольшую общую часть.

Спецификация метода:

common (None , None) = TypeError
common ("", "") = ""
common ("", " abc ") = ""
common (" abc ", "") = ""
common (" abc ", "abc") = " abc "
common ("ab", " abxyz ") = "ab"
common (" abcde ", " abxyz ") = "ab"
common (" abcde ", " xyz ") = ""
common (" deabc ", " abcdeabcd ") = " deabc "
common (" dfabcegt ", " rtoefabceiq ") = " fabce "

```
class TypeError(Exception):
    pass

def common(str1, str2):
    if str1 is None or str2 is None:
        raise TypeError("Both strings must not be None")

    max_substring = ""
    len1, len2 = len(str1), len(str2)

    for i in range(len1):
        for j in range(i + 1, len1 + 1):
            substring = str1[i:j]
            if substring in str2 and len(substring) > len(max_substring):
```

```
                max_substring = substring

    return max_substring
import pytest
from main import common, TypeError

def test_both_none():
    with pytest.raises(TypeError):
        common(None, None)

def test_both_empty():
    assert common("", "") == ""

def test_first_empty():
    assert common("", "abc") == ""

def test_second_empty():
    assert common("abc ", "") == ""

def test_one_substring_of_another():
    assert common("abc", "abc") == "abc"

def test_common_prefix():
    assert common("ab", "abxyz") == "ab"

def test_common_prefix_2():
    assert common("abcde", "abxyz") == "ab"

def test_no_common():
    assert common("abcde", "xyz") == ""

def test_common_substring():
    assert common("deabc", "abcdeabcd") == "deabc"

def test_common_middle():
    assert common("dfabcegt", "rtoefabceiq") == "fabce"
```

```
PS C:\rep\spp\lab6\task3> pytest test.py -v
============================= test session starts ==============================
platform win32 -- Python 3.13.1, pytest-8.3.5, pluggy-1.5.0 -- c:\rep\spp\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\rep\spp\lab6\task3
plugins: anyio-4.9.0
collected 10 items

test.py::test_both_none PASSED                                           [ 10%]
test.py::test_both_empty PASSED                                          [ 20%]
test.py::test_first_empty PASSED                                         [ 30%]
test.py::test_second_empty PASSED                                        [ 40%]
test.py::test_one_substring_of_another PASSED                            [ 50%]
test.py::test_common_prefix PASSED                                       [ 60%]
test.py::test_common_prefix_2 PASSED                                     [ 70%]
test.py::test_no_common PASSED                                           [ 80%]
test.py::test_common_substring PASSED                                    [ 90%]
test.py::test_common_middle PASSED                                       [100%]

============================== 10 passed in 0.05s ==============================
```

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