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")

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

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()

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.**

**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"

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

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