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
С.В. Зайченко
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
12.04.2025 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 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 calculate_total(self):
        return sum(item["price"] for item in self.items)

    def apply_discount(self, discount_percent):
        if discount_percent < 0 or discount_percent > 100:
            raise ValueError("Discount must be between 0 and 100 percent")
```

```
        for item in self.items:
            item["price"] *= (1 - discount_percent / 100)

    def log_purchase(self, item):
        response = requests.post("https://example.com/purchase", json=item)
        if response.status_code == 200:
            print("Покупка успешно зарегистрирована!")
        else:
            print("Ошибка при регистрации покупки!")

    def apply_coupon(self, coupon_code):
        coupons = {"DISCOUNT20": 20, "HALFOFF": 50}
        if coupon_code in coupons:
            self.apply_discount(coupons[coupon_code])
        else:
            raise ValueError("Неверный код купона")

    def remove_item(self, name):
        if not name:
            raise ValueError("Item name cannot be empty")
        for item in self.items:
            if item["name"] == name:
                self.items.remove(item)
                print(f"Товар '{name}' удалён из корзины")
                return
        raise ValueError(f"Товар '{name}' не найден в корзине")
```

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 os
import sys
import pytest
from lab1_1 import find_the_max

current_dir = os.path.dirname(os.path.abspath(__file__))
lab1_dir = os.path.abspath(os.path.join(current_dir, "..", "..", "1", "src"))
sys.path.append(lab1_dir)


@pytest.mark.parametrize(
    "numbers,expected",
    [
        ([1, 2, 3], {1: 1, 2: 1}),
        ([2, 2, 3], {1: 0, 2: 1}),
        ([-2, 0, 2], {0: 2, 2: 2}),
        ([0, 0, 0], {1: 0, 2: 0}),
        ([5, 3, 4], {1: 1, 2: 1}),
        ([1, 1, 1, 1], {1: 0, 2: 0, 3: 0}),
        ([10 ** 6, 10 ** 6 + 1], {1: 1}),
        ([-5, 1], {0: 4}),
```

```
        ([1, -5], {0: 4}),
        ([-1, -5], {0: 4}),
        ([-3, -1, 5], {0: 2, 2: 4})
    ]
)
def test_valid_numeric_sequences(numbers, expected):
    assert find_the_max(numbers) == expected


@pytest.mark.parametrize(
    "invalid_input_for_type_error",
    [
        [1, "a"],
        ["a", 1],
        ["a", "b"],
        ["(", ")"],
        [1, 2, "a"],
        ["1", "2", {}],
        [1, None],
        [None, 1],
        [None, None],
    ]
)
def test_inputs_causing_type_error(invalid_input_for_type_error):
    with pytest.raises(TypeError):
        find_the_max(invalid_input_for_type_error)


@pytest.mark.parametrize(
    "single_element_input",
    [
        ["a"],
        ["()"],
        [None],
        [{}],
        [[1, 2]],
    ]
)
def test_single_element_various_types_returns_empty_dict(single_element_input):
    assert find_the_max(single_element_input) == {}


def test_empty_list_boundary():
    assert find_the_max([]) == {}


def test_single_numeric_element_list_boundary():
    assert find_the_max([1]) == {}
    assert find_the_max([-100]) == {}
    assert find_the_max([0]) == {}


@pytest.mark.parametrize(
    "two_elements_input, expected_two_elements",
    [
        ([5, 10], {1: 5}),
        ([10, 5], {1: 5}),
        ([-10, 5], {0: 5}),
        ([5, -10], {0: 5}),
```

```
        ([-5, -10], {0: 5}),
        ([-10, -5], {0: 5}),
        ([0, 0], {1: 0}),
        ([5, 5], {1: 0}),
        ([-5, -5], {1: 0}),
    ]
)
def test_two_elements_list_boundary(two_elements_input, expected_two_elements):
    assert find_the_max(two_elements_input) == expected_two_elements
```

Lab1_2.py

```
    import os
import sys
import pytest
from lab1_2 import isPalindrome

current_dir = os.path.dirname(os.path.abspath(__file__))
lab1_dir = os.path.abspath(os.path.join(current_dir, "..", "..", "1", "src"))
sys.path.append(lab1_dir)


@pytest.mark.parametrize("s, expected", [
    ("121", True),
    ("12321", True),
    ("1", True),
    ("0", True),
    ("11", True),
    ("999", True),
    ("101", True),
    ("1221", True),
    ("10", False),
    ("123", False),
    ("100", False),
    ("12345", False),
])
def test_isPalindrome_numbers_as_strings(s, expected):
    assert isPalindrome(s) == expected


@pytest.mark.parametrize("s, expected", [
    ("", True),
    ("a", True),
    ("aa", True),
    ("aba", True),
])
def test_isPalindrome_letter_strings(s, expected):
    assert isPalindrome(s) == expected


@pytest.mark.parametrize("s, expected", [
    ("a1a", True),
    ("1a1", True),
    (" ", True),
    ("  ", True),
    ("!@!", True),
    ("a!b!a", True),
    ("a!b!!b!a", True),
])
def test_isPalindrome_mixed_strings(s, expected):
    assert isPalindrome(s) == expected
```

```
def test_isPalindrome_empty_string():
   assert isPalindrome("") is True


def test_isPalindrome_single_character():
   assert isPalindrome("x") is True
   assert isPalindrome("7") is True
   assert isPalindrome("[") is True
```

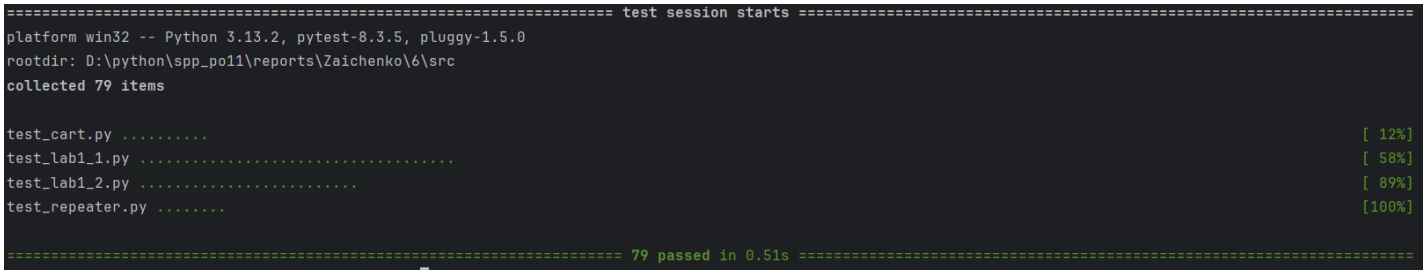
Задание 3:

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

Код программы:

```
    import pytest
from repeater import repeat_string


def test_spec_e_0():
   assert repeat_string("e", 0) == ""


def test_spec_e_3():
   assert repeat_string("e", 3) == " eee "


def test_spec_abc_2():
   assert repeat_string(" ABC ", 2) == " ABCABC "


def test_spec_e_negative():
   with pytest.raises(ValueError):
      repeat_string("e", -2)


def test_spec_none_1():
   with pytest.raises(TypeError):
      repeat_string(None, 1)


def test_empty_pattern_positive_repeat():
   assert repeat_string("", 5) == "  "


def test_simple_pattern_once():
   assert repeat_string("abc", 1) == " abc "


def test_pattern_with_internal_spaces():
   assert repeat_string("a b", 2) == " a ba b "
```

Рисунки с результатами работы программы:



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