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
Лесько М.И.
студент группы ПО11

Проверил
А. А. Крощенко ст.
преп. кафедры ИИТ,
11.04.2025 г.

Брест 2025

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

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

Код программы:

shopping.py:

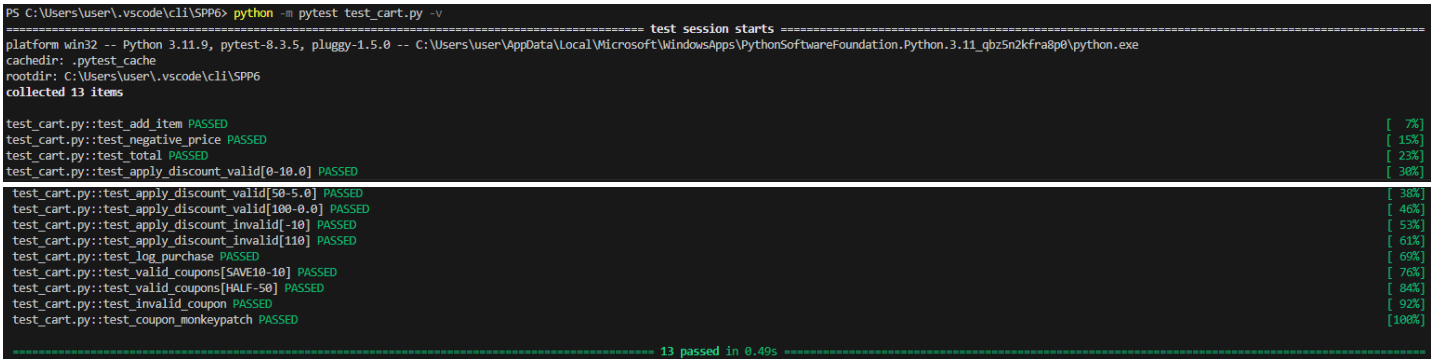
```
import requests
class Cart:
    def __init__(self):
        self.items = []

    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, percentage):
        if percentage < 0 or percentage > 100:
            raise ValueError("Discount percentage must be between 0 and 100")
        total = self.total()
        return total * (1 - percentage / 100)

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

# Move coupons to module level
COUPONS = {"SAVE10": 10, "HALF": 50}

def apply_coupon(cart, coupon_code):
    if coupon_code in COUPONS:
        cart.apply_discount(COUPONS[coupon_code])
    else:
        raise ValueError("Invalid coupon")
```

test_cart.py:

```
import pytest
from unittest.mock import patch, MagicMock
from shopping import Cart, log_purchase, apply_coupon, COUPONS

@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(empty_cart):
    with pytest.raises(ValueError, match="Price cannot be negative"):
        empty_cart.add_item("Apple", -10.0)

def test_total(empty_cart):
    empty_cart.add_item("Apple", 10.0)
    empty_cart.add_item("Banana", 5.0)
    assert empty_cart.total() == 15.0

@pytest.mark.parametrize("discount,expected", [
    (0, 10.0),
    (50, 5.0),
    (100, 0.0),
```

```
])
def test_apply_discount_valid(empty_cart, discount, expected):
    empty_cart.add_item("Apple", 10.0)
    assert empty_cart.apply_discount(discount) == expected

@pytest.mark.parametrize("invalid_discount", [-10, 110])
def test_apply_discount_invalid(empty_cart, invalid_discount):
    empty_cart.add_item("Apple", 10.0)
    with pytest.raises(ValueError, match="Discount percentage must be between 0 and 100"):
        empty_cart.apply_discount(invalid_discount)

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

@pytest.mark.parametrize("coupon_code,expected_discount", [
    ("SAVE10", 10),
    ("HALF", 50),
])
def test_valid_coupons(empty_cart, coupon_code, expected_discount):
    empty_cart.add_item("Apple", 100.0)
    apply_coupon(empty_cart, coupon_code)
    assert empty_cart.apply_discount(expected_discount) == 100.0 * (1 - expected_discount / 100)

def test_invalid_coupon(empty_cart):
    with pytest.raises(ValueError, match="Invalid coupon"):
        apply_coupon(empty_cart, "INVALID")

@patch('shopping.COUPONS', {"NEW10": 10})
def test_coupon_monkeypatch(empty_cart):
    empty_cart.add_item("Apple", 100.0)
    apply_coupon(empty_cart, "NEW10")
    assert empty_cart.apply_discount(10) == 90.0
```

Рисунок с результатом работы программы:



Задание 2

Напишите тесты к реализованным функциям из лабораторной работы №1. Проверьте тривиальные и граничные случае, а также варианты, когда может возникнуть исключительная ситуация. Если при реализации не использовались отдельные функции, необходимо провести рефакторинг кода.

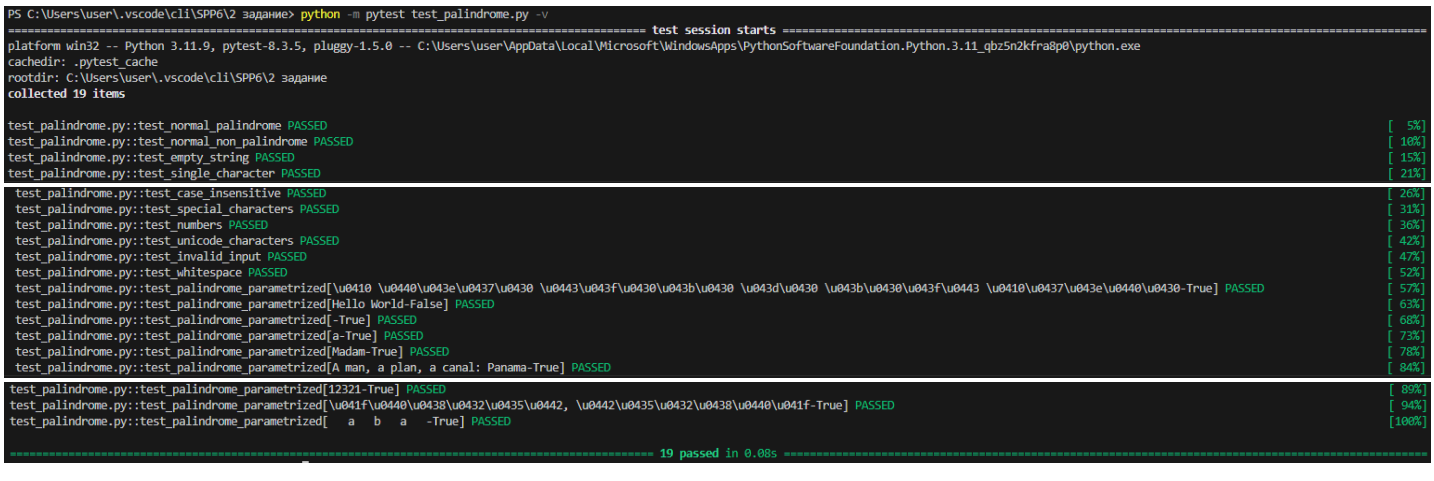
Код программы:

test_palindrome.py:

```
import pytest
from SPP1_2 import is_palindrome

def test_normal_palindrome():
    """Тест обычного палиндрома"""
    assert is_palindrome("А роза упала на лапу Азора") is True

def test_normal_non_palindrome():
```

```
    """Тест обычного не-палиндрома"""
    assert is_palindrome("Hello World") is False

def test_empty_string():
    """Тест пустой строки"""
    assert is_palindrome("") is True

def test_single_character():
    """Тест строки из одного символа"""
    assert is_palindrome("a") is True

def test_case_insensitive():
    """Тест на регистронезависимость"""
    assert is_palindrome("Madam") is True

def test_special_characters():
    """Тест со специальными символами"""
    assert is_palindrome("A man, a plan, a canal: Panama") is True

def test_numbers():
    """Тест с числами"""
    assert is_palindrome("12321") is True

def test_unicode_characters():
    """Тест с Unicode символами"""
    assert is_palindrome("Привет, тевирП") is True

def test_invalid_input():
    """Тест с неверным типом входных данных"""
    with pytest.raises(TypeError, match="Входные данные должны быть строкой"):
        is_palindrome(123)

def test_whitespace():
    """Тест с пробелами"""
    assert is_palindrome("   a   b   a   ") is True

@pytest.mark.parametrize("input_str,expected", [
    ("А роза упала на лапу Азора", True),
    ("Hello World", False),
    ("", True),
    ("a", True),
    ("Madam", True),
    ("A man, a plan, a canal: Panama", True),
    ("12321", True),
    ("Привет, тевирП", True),
    ("   a   b   a   ", True),
])
def test_palindrome_parametrized(input_str, expected):
    """Параметризованный тест для различных случаев"""
    assert is_palindrome(input_str) is expected
```

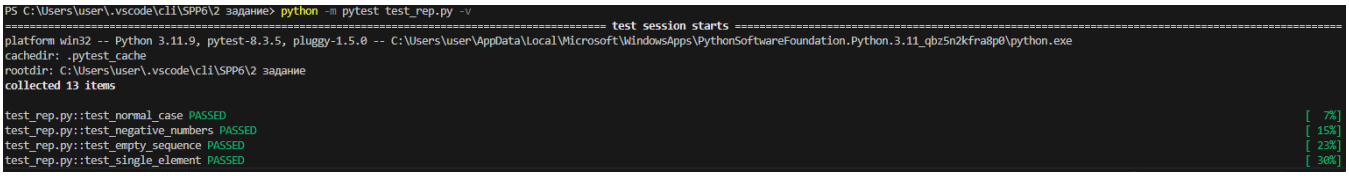
Рисунок с результатом работы программы:

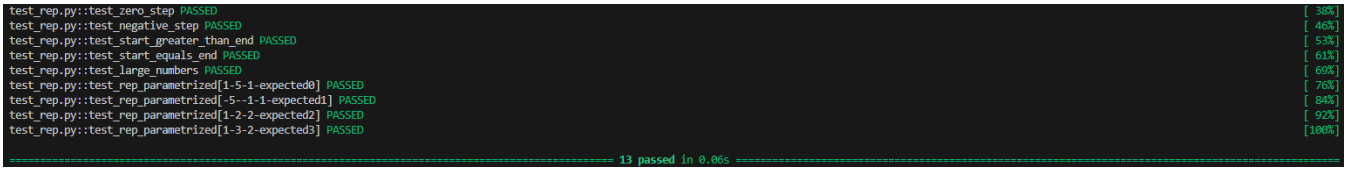


test_rep.py:

```
import pytest
from SPP1_1 import rep

def test_normal_case():
    """Тест нормального случая с положительными числами"""
    assert rep(1, 5, 1) == [1, 2, 3, 4]

def test_negative_numbers():
    """Тест с отрицательными числами"""
    assert rep(-5, -1, 1) == [-5, -4, -3, -2]

def test_empty_sequence():
    """Тест случая, когда последовательность пуста"""
    assert rep(1, 2, 2) == [1]

def test_single_element():
    """Тест случая с одним элементом"""
    assert rep(1, 3, 2) == [1]

def test_zero_step():
    """Тест случая с нулевым шагом"""
    with pytest.raises(ValueError, match="шаг не может быть равен нулю"):
        rep(1, 5, 0)

def test_negative_step():
    """Тест случая с отрицательным шагом"""
    with pytest.raises(ValueError, match="шаг должен быть положительным"):
        rep(1, 5, -1)

def test_start_greater_than_end():
    """Тест случая, когда начало больше конца"""
    with pytest.raises(ValueError, match="start должен быть меньше end"):
        rep(5, 1, 1)

def test_start_equals_end():
    """Тест случая, когда начало равно концу"""
    with pytest.raises(ValueError, match="start должен быть меньше end"):
        rep(5, 5, 1)

def test_large_numbers():
    """Тест с большими числами"""
    assert rep(1000000, 1000005, 1) == [1000000, 1000001, 1000002, 1000003, 1000004]

@pytest.mark.parametrize("start,end,step,expected", [
    (1, 5, 1, [1, 2, 3, 4]),
    (-5, -1, 1, [-5, -4, -3, -2]),
    (1, 2, 2, [1]),
    (1, 3, 2, [1]),
])
def test_rep_parametrized(start, end, step, expected):
    """Параметризованный тест для различных случаев"""
    assert rep(start, end, step) == expected
```

Рисунок с результатом работы программы:





Задание 3.

3) Напишите метод String keep(String str, String pattern) который оставляет в первой строке все символы, которые присутствуют во второй. Спецификация метода:

keep (None , None) = TypeError

keep (None, *) = None

keep ("", *) = ""

keep (* , None) = ""

keep (* , "") = ""

keep (" hello ", "hl") = " hll "

keep (" hello ", "le") = " ell "

Код программы:

keep.py:

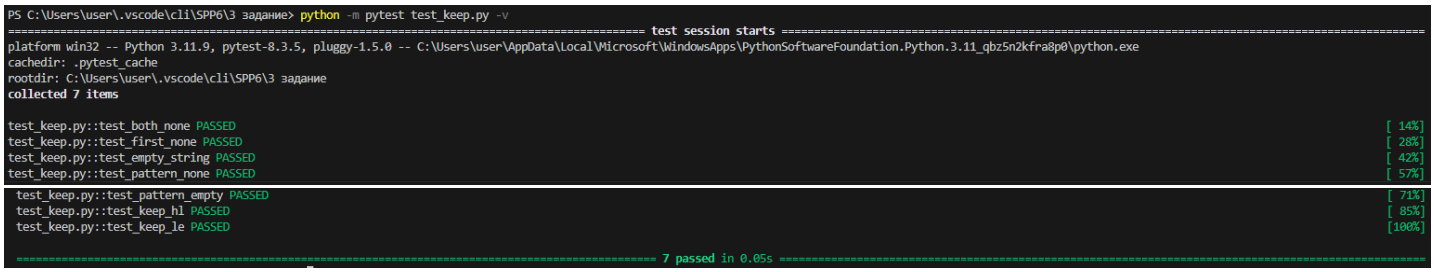
```
def keep(str, pattern):
    if str is None and pattern is None:
        raise TypeError("Both arguments cannot be None")
    if str is None:
        return None
    if pattern is None:
        return ""

    if not str or not pattern:
        return ""

    keep_chars = set(pattern)

    result = []
    for char in str:
        if char in keep_chars or char.isspace():
            result.append(char)

    return "".join(result)
```

test_keep.py:

```
import pytest
from keep import keep

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

def test_first_none():
    assert keep(None, "abc") is None

def test_empty_string():
    assert keep("", "abc") == ""

def test_pattern_none():
    assert keep("abc", None) == ""

def test_pattern_empty():
```

```
    assert keep("abc", "") == ""

def test_keep_hl():
    assert keep(" hello ", "hl") == " hll "

def test_keep_le():
    assert keep(" hello ", "le") == " ell "
```

Рисунок с результатом работы программы:



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