

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №6
Специальность ПО-11

Выполнил:
А. А. Билялова
студент группы ПО11

Проверил:
А. А. Крощенко,
ст. преп. кафедры ИИТ,
16.04.2025

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)
```

- Замокните `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`
- Замокните словарь `coupons` с помощью `monkeypatch` или `patch.dict`

Код программы:

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

```

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

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

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

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

@pytest.mark.parametrize("discount, expected_total", [
    (0, 100.0),
    (50, 50.0),
    (100, 0.0),
])
def test_apply_discount_valid(empty_cart, discount, expected_total):
    cart = empty_cart
    cart.add_item("Item", 100.0)
    cart.apply_discount(discount)
    assert cart.total() == expected_total

@pytest.mark.parametrize("invalid_discount", [
    -10, 110
])
def test_apply_discount_invalid(empty_cart, invalid_discount):
    cart = empty_cart
    cart.add_item("Item", 100.0)
    with pytest.raises(ValueError, match="Discount must be between 0 and 100"):
        cart.apply_discount(invalid_discount)

def test_log_purchase():
    mock_response = Mock()
    mock_response.status_code = 200

    with patch('requests.post', return_value=mock_response) as mock_post:
        log_purchase({"item": "Apple", "price": 10.0})

        mock_post.assert_called_once_with(
            "https://example.com/log",
            json={"item": "Apple", "price": 10.0}
        )

def test_apply_coupon_valid(empty_cart, monkeypatch):
    cart = empty_cart
    cart.add_item("Item", 100.0)

    monkeypatch.setattr("shopping.coupons", {"SAVE10": 10, "HALF": 50})

    apply_coupon(cart, "SAVE10")
    assert cart.total() == 90.0

    apply_coupon(cart, "HALF")
    assert cart.total() == 45.0

def test_apply_coupon_invalid(empty_cart, monkeypatch):

```

```

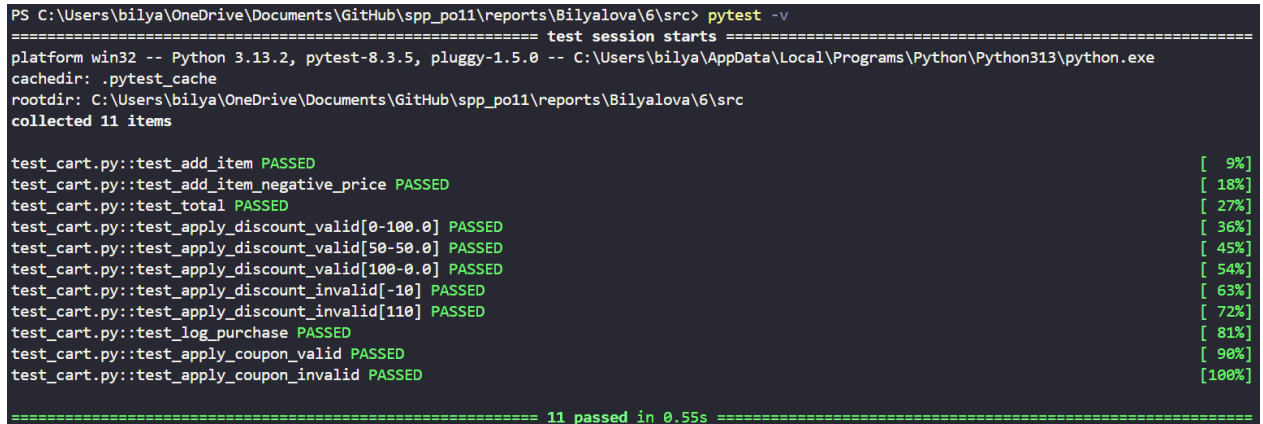
cart = empty_cart
cart.add_item("Item", 100.0)

monkeypatch.setattr("shopping.coupons", {"SAVE10": 10, "HALF": 50})

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

```

Рисунки с результатами работы программы



```

PS C:\Users\bilya\OneDrive\Documents\GitHub\spp_po11\reports\Bilyalova\6\src> pytest -v
===== test session starts =====
platform win32 -- Python 3.13.2, pytest-8.3.5, pluggy-1.5.0 -- C:\Users\bilya\AppData\Local\Programs\Python\Python313\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\bilya\OneDrive\Documents\GitHub\spp_po11\reports\Bilyalova\6\src
collected 11 items

test_cart.py::test_add_item PASSED [ 9%]
test_cart.py::test_add_item_negative_price PASSED [ 18%]
test_cart.py::test_total PASSED [ 27%]
test_cart.py::test_apply_discount_valid[0-100.0] PASSED [ 36%]
test_cart.py::test_apply_discount_valid[50-50.0] PASSED [ 45%]
test_cart.py::test_apply_discount_valid[100-0.0] PASSED [ 54%]
test_cart.py::test_apply_discount_invalid[-10] PASSED [ 63%]
test_cart.py::test_apply_discount_invalid[110] PASSED [ 72%]
test_cart.py::test_log_purchase PASSED [ 81%]
test_cart.py::test_apply_coupon_valid PASSED [ 90%]
test_cart.py::test_apply_coupon_invalid PASSED [100%]

===== 11 passed in 0.55s =====

```

Задание 2. Напишите тесты к реализованным функциям из лабораторной работы No1. Проверьте тривиальные и граничные случаи, а также варианты, когда может возникнуть исключительная ситуация. Если при реализации не использовались отдельные функции, необходимо провести рефакторинг кода.

Код программы:

```

import pytest
import sys
import os
from pathlib import Path

lab1_path = str(Path(__file__).parent.parent.parent / "1" / "src")
sys.path.insert(0, lab1_path)

from main import find_modes
from main2 import find_first_occurrence

class TestFindModes:
    def test_empty_sequence(self):
        assert find_modes([]) is None

    def test_no_mode(self):
        assert find_modes([1, 2, 3]) is None

    def test_single_mode(self):
        assert find_modes([1, 2, 2, 3]) == [2]

    def test_multiple_modes(self):
        result = find_modes([1, 1, 2, 2, 3])
        assert sorted(result) == [1, 2]

    def test_all_elements_same(self):
        assert find_modes([5, 5, 5]) == [5]

    def test_negative_numbers(self):
        assert find_modes([-1, -1, -2]) == [-1]

    def test_single_element(self):
        assert find_modes([7]) is None

    def test_invalid_input(self):
        with pytest.raises(TypeError):

```

```

        find_modes([1, 2, {}])

class TestFindFirstOccurrence:
    def test_normal_case(self):
        assert find_first_occurrence("hello world", "world") == 6

    def test_not_found(self):
        assert find_first_occurrence("hello", "world") == -1

    def test_empty_needle(self):
        assert find_first_occurrence("hello", "") == 0

    def test_empty_haystack(self):
        assert find_first_occurrence("", "hello") == -1

    def test_both_empty(self):
        assert find_first_occurrence("", "") == 0

    def test_multiple_occurrences(self):
        assert find_first_occurrence("ababab", "ab") == 0

    def test_case_sensitive(self):
        assert find_first_occurrence("Hello", "hello") == -1

    def test_unicode(self):
        assert find_first_occurrence("привет мир", "мир") == 7

    def test_none_input(self):
        assert find_first_occurrence(None, "test") == -1
        assert find_first_occurrence("test", None) == -1
        assert find_first_occurrence(None, None) == -1

    def test_non_string_input(self):
        assert find_first_occurrence(123, "23") == -1
        assert find_first_occurrence("hello", 123) == -1

def test_main_input_handling(monkeypatch):
    inputs = iter(["3", "1", "2", "2"])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    from main import main
    main()

```

Рисунки с результатами работы программы

```

collected 19 items

test_lab1.py::TestFindModes::test_empty_sequence PASSED [ 5%]
test_lab1.py::TestFindModes::test_no_mode PASSED [ 10%]
test_lab1.py::TestFindModes::test_single_mode PASSED [ 15%]
test_lab1.py::TestFindModes::test_multiple_modes PASSED [ 21%]
test_lab1.py::TestFindModes::test_all_elements_same PASSED [ 26%]
test_lab1.py::TestFindModes::test_negative_numbers PASSED [ 31%]
test_lab1.py::TestFindModes::test_single_element PASSED [ 36%]
test_lab1.py::TestFindModes::test_invalid_input PASSED [ 42%]
test_lab1.py::TestFindFirstOccurrence::test_normal_case PASSED [ 47%]
test_lab1.py::TestFindFirstOccurrence::test_not_found PASSED [ 52%]
test_lab1.py::TestFindFirstOccurrence::test_empty_needle PASSED [ 57%]
test_lab1.py::TestFindFirstOccurrence::test_empty_haystack PASSED [ 63%]
test_lab1.py::TestFindFirstOccurrence::test_both_empty PASSED [ 68%]
test_lab1.py::TestFindFirstOccurrence::test_multiple_occurrences PASSED [ 73%]
test_lab1.py::TestFindFirstOccurrence::test_case_sensitive PASSED [ 78%]
test_lab1.py::TestFindFirstOccurrence::test_unicode PASSED [ 84%]
test_lab1.py::TestFindFirstOccurrence::test_none_input PASSED [ 89%]
test_lab1.py::TestFindFirstOccurrence::test_non_string_input PASSED [ 94%]
test_lab1.py::test_main_input_handling PASSED [100%]

===== 19 passed in 0.14s =====

```

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

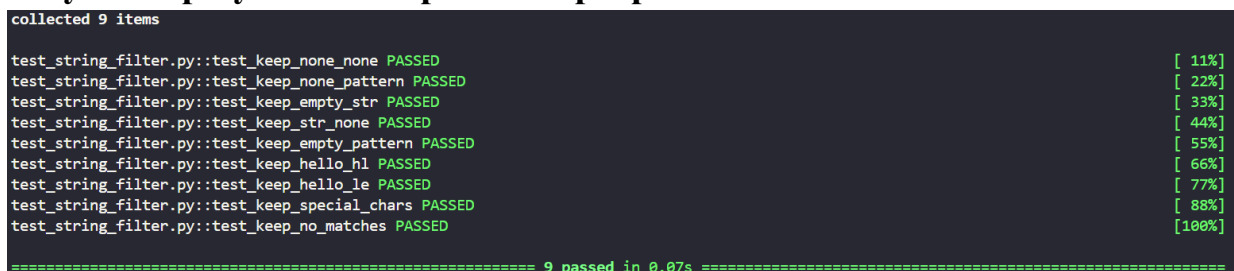
Тесты:

```
import pytest
from string_utils import keep
def test_keep_none_none():
    with pytest.raises(TypeError):
        keep(None, None)
def test_keep_none_pattern():
    assert keep(None, "abc") is None
def test_keep_empty_str():
    assert keep("", "abc") == ""
def test_keep_str_none():
    assert keep("hello", None) == ""
def test_keep_empty_pattern():
    assert keep("hello", "") == ""
def test_keep_hello_hl():
    assert keep(" hello ", "hl ") == " hll "
def test_keep_hello_le():
    assert keep(" hello ", "le ") == " ell "
def test_keep_special_chars():
    assert keep("a1!b2@c3#", "123!@#") == "1!2@3#"
def test_keep_no_matches():
    assert keep("hello", "xyz") == ""
```

Реализация метода:

```
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 str_ == "" or pattern is None or pattern == "":
        return ""
    keep_chars = set(pattern)
    return ''.join(char for char in str_ if char in keep_chars)
```

Рисунки с результатами работы программы



```
collected 9 items

test_string_filter.py::test_keep_none_none PASSED [ 11%]
test_string_filter.py::test_keep_none_pattern PASSED [ 22%]
test_string_filter.py::test_keep_empty_str PASSED [ 33%]
test_string_filter.py::test_keep_str_none PASSED [ 44%]
test_string_filter.py::test_keep_empty_pattern PASSED [ 55%]
test_string_filter.py::test_keep_hello_hl PASSED [ 66%]
test_string_filter.py::test_keep_hello_le PASSED [ 77%]
test_string_filter.py::test_keep_special_chars PASSED [ 88%]
test_string_filter.py::test_keep_no_matches PASSED [100%]

===== 9 passed in 0.07s =====
```

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