

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра интеллектуальных информационных технологий

Лабораторная работа №6
**По дисциплине «Современные платформы
программирования»**

Выполнил
Студент 3 курса
Группы ПО-11
Хведорец В.С.
Проверил:
Козик И.Д.

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:

```
from unittest.mock import patch
```

```
import pytest
```

```
import requests
```

```
from shopping import Cart
```

```
@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_add_item_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_total", [(0, 100.0), (50, 50.0), (100, 0.0)])
def test_apply_discount_valid(empty_cart, discount, expected_total):
    empty_cart.add_item("Item", 100.0)
    empty_cart.apply_discount(discount)
    assert empty_cart.total() == expected_total

@pytest.mark.parametrize("invalid_discount", [-10, 110])
def test_apply_discount_invalid(empty_cart, invalid_discount):
    empty_cart.add_item("Item", 100.0)
    with pytest.raises(ValueError, match="Invalid discount percent"):
        empty_cart.apply_discount(invalid_discount)

@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):
    empty_cart.add_item("Item", 100.0)
    empty_cart.apply_coupon("SAVE10")
    assert empty_cart.total() == 90.0

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

```

Рисунки с результатами работы программы:

```

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 1.67s =====

```

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:

Lab1_1.py

```
import os
import sys

import pytest

# Добавляем путь к директории с исходными файлами в sys.path
current_dir = os.path.dirname(os.path.abspath(__file__))
lab1_dir = os.path.join(os.path.dirname(os.path.dirname(current_dir)), "Lab1", "src")
sys.path.append(lab1_dir)

from Lab1_1 import calculate_negative_squares_sum
from Lab1_2 import is_valid

@pytest.mark.parametrize(
    "numbers,expected",
    [
        ([], 0),
        ([1, 2, 3], 0),
        ([-1, -2, -3], 14),
        ([-1, 0, 1], 1),
        ([-5, 2, -3, 1], 34),
    ],
)

def test_calculate_negative_squares_sum(numbers, expected):
    """Тест на корректное вычисление суммы квадратов отрицательных чисел"""
    assert calculate_negative_squares_sum(numbers) == expected

def test_calculate_negative_squares_sum_with_large_numbers():
    """Тест на работу с большими числами"""
    numbers = [-1000, 1000]
    assert calculate_negative_squares_sum(numbers) == 1_000_000

def test_calculate_negative_squares_sum_with_zero():
    """Тест на работу с нулями"""
    numbers = [0, 0, 0]
    assert calculate_negative_squares_sum(numbers) == 0

def test_calculate_negative_squares_sum_with_single_negative():
    """Тест на один отрицательный элемент"""
    numbers = [-5]
    assert calculate_negative_squares_sum(numbers) == 25

def test_calculate_negative_squares_sum_with_single_positive():
    """Тест на один положительный элемент"""
    numbers = [5]
    assert calculate_negative_squares_sum(numbers) == 0

@pytest.mark.parametrize(
    "brackets,expected",
    [
```

```

    ("", True), # Пустая строка
    ("()", True), # Простые круглые скобки
    ("[]", True), # Простые квадратные скобки
    ("{}", True), # Простые фигурные скобки
    ("{{{}}}", True), # Вложенные скобки
    ("([)]", False), # Неправильно вложенные скобки
    ("(", False), # Незакрытые скобки
    (")", False), # Незакрытые скобки с начала
    ("[]{}]", True), # Последовательные правильные скобки
    ("{{{()}}}", True), # Сложные вложенные скобки
],
)
def test_is_valid_brackets(brackets, expected):
    assert is_valid(brackets) == expected

@pytest.mark.parametrize(
    "invalid_input",
    [
        "a", # Буква
        "1", # Цифра
        "()", # Пробел
        "[a]", # Буква внутри скобок
    ],
)
def test_is_valid_with_invalid_chars(invalid_input):
    """Тест на наличие недопустимых символов"""
    assert not is_valid(invalid_input)

def test_is_valid_with_long_sequence():
    """Тест на длинную последовательность скобок"""
    brackets = "(" * 1000 + ")" * 1000
    assert is_valid(brackets)

def test_is_valid_with_mixed_long_sequence():
    """Тест на длинную смешанную последовательность скобок"""
    brackets = "({{" * 100 + "})" * 100
    assert is_valid(brackets)

```

Lab1_2.py

```

import os
import sys

import pytest

# Добавляем путь к директории с исходными файлами в sys.path
current_dir = os.path.dirname(os.path.abspath(__file__))
lab1_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(current_dir))), "Lab1", "src")
sys.path.append(lab1_dir)

from Lab1_2 import is_valid
# Тесты для проверки корректных последовательностей скобок
@pytest.mark.parametrize(
    "brackets, expected",
    [
        ("", True), # Пустая строка
        ("()", True), # Простые круглые скобки

```

```

    ("[]", True), # Простые квадратные скобки
    ("{}", True), # Простые фигурные скобки
    ("{}", True), # Вложенные скобки
    ("[]{}", True), # Последовательные правильные скобки
    ("({})", True), # Сложные вложенные скобки
    ("[]{}", True), # Несколько пар скобок
    ("((()))", True), # Много вложенных круглых скобок
    ("({{}})", True), # Смешанные вложенные скобки
],
)
def test_valid_brackets_sequences(brackets, expected):
    """Тест на корректные последовательности скобок"""
    assert is_valid(brackets) == expected

# Тесты для проверки некорректных последовательностей скобок
@pytest.mark.parametrize(
    "brackets,expected",
    [
        ("()", False), # Неправильно вложенные скобки
        ("(", False), # Незакрытые скобки
        (")", False), # Незакрытые скобки с начала
        ("{}", False), # Неправильное закрытие
        ("()()", False), # Закрывающая скобка в начале
        ("{}()", False), # Неправильное вложение
        ("({)", False), # Незакрытые скобки
        ("({})", False), # Неправильное закрытие
    ],
)
def test_invalid_brackets_sequences(brackets, expected):
    """Тест на некорректные последовательности скобок"""
    assert is_valid(brackets) == expected

# Тесты на граничные случаи
def test_empty_string():
    """Тест на пустую строку"""
    assert is_valid("") == True

def test_single_opening_bracket():
    """Тест на одну открывающую скобку"""
    assert is_valid("(") == False

def test_single_closing_bracket():
    """Тест на одну закрывающую скобку"""
    assert is_valid(")") == False

# Тесты на длинные последовательности
def test_long_sequence():
    """Тест на длинную последовательность скобок"""
    brackets = "(" * 1000 + ")" * 1000
    assert is_valid(brackets) == True

def test_mixed_long_sequence():
    """Тест на длинную смешанную последовательность скобок"""
    brackets = "({{" * 100 + "})" * 100
    assert is_valid(brackets) == True

```

```

# Тесты на недопустимые символы
@pytest.mark.parametrize(
    "invalid_input",
    [
        "a", # Буква
        "1", # Цифра
        "(", # Пробел
        "([a]", # Буква внутри скобок
        "()[]{}a", # Буква в конце
        "a()[]{}", # Буква в начале
        "()a[]{}", # Буква в середине
    ],
)
def test_invalid_characters(invalid_input):
    """Тест на наличие недопустимых символов"""
    assert is_valid(invalid_input) == False

# Тест на производительность
def test_performance():
    """Тест на производительность с очень длинной последовательностью"""
    brackets = "({[" * 10000 + "})]" * 10000
    assert is_valid(brackets) == True

```

Рисунки с результатами работы программы:

```

test_lab1_1.py::test_is_valid_brackets[({[]})-True] PASSED [ 72%]
test_lab1_1.py::test_is_valid_brackets[{[()]})-True] PASSED [ 76%]
test_lab1_1.py::test_is_valid_with_invalid_chars[a] PASSED [ 80%]
test_lab1_1.py::test_is_valid_with_invalid_chars[1] PASSED [ 84%]
test_lab1_1.py::test_is_valid_with_invalid_chars[( )] PASSED [ 88%]
test_lab1_1.py::test_is_valid_with_invalid_chars[([a])] PASSED [ 92%]
test_lab1_1.py::test_is_valid_with_long_sequence PASSED [ 96%]
test_lab1_1.py::test_is_valid_with_mixed_long_sequence PASSED [100%]

===== 25 passed in 0.13s =====

```

```

test_lab1_2.py::test_invalid_characters[a] PASSED [ 77%]
test_lab1_2.py::test_invalid_characters[1] PASSED [ 80%]
test_lab1_2.py::test_invalid_characters[( )] PASSED [ 83%]
test_lab1_2.py::test_invalid_characters[([a])] PASSED [ 87%]
test_lab1_2.py::test_invalid_characters[()[]{}a] PASSED [ 90%]
test_lab1_2.py::test_invalid_characters[a()[]{}] PASSED [ 93%]
test_lab1_2.py::test_invalid_characters[()a[]{}] PASSED [ 96%]
test_lab1_2.py::test_performance PASSED [100%]

===== 31 passed in 0.16s =====

```

Задание 3:

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

Код программы:

```

from unittest.mock import patch

import pytest
import requests
from shopping import Cart
@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_add_item_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_total", [(0, 100.0), (50, 50.0), (100, 0.0)])
def test_apply_discount_valid(empty_cart, discount, expected_total):
    empty_cart.add_item("Item", 100.0)
    empty_cart.apply_discount(discount)
    assert empty_cart.total() == expected_total

@pytest.mark.parametrize("invalid_discount", [-10, 110])
def test_apply_discount_invalid(empty_cart, invalid_discount):
    empty_cart.add_item("Item", 100.0)
    with pytest.raises(ValueError, match="Invalid discount percent"):
        empty_cart.apply_discount(invalid_discount)

@pytest.mark.parametrize("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):
    empty_cart.add_item("Item", 100.0)
    empty_cart.apply_coupon("SAVE10")
    assert empty_cart.total() == 90.0

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

```

Рисунки с результатами работы программы:

```

test_string.py::test_loose_any_empty PASSED [ 41%]
test_string.py::test_loose_hello_hl PASSED [ 50%]
test_string.py::test_loose_hello_le PASSED [ 58%]
test_string.py::test_loose_same_string PASSED [ 66%]
test_string.py::test_loose_no_common_chars PASSED [ 75%]
test_string.py::test_loose_duplicate_chars PASSED [ 83%]
test_string.py::test_loose_whitespace PASSED [ 91%]
test_string.py::test_loose_special_chars PASSED [100%]
===== 12 passed in 0.08s =====

```

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