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

Е. А. Германович

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:**

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

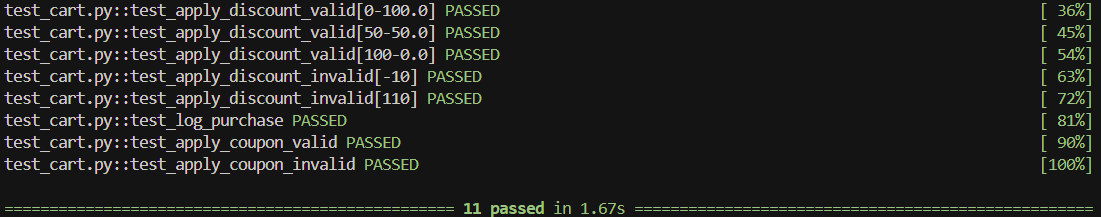
def test\_apply\_coupon\_invalid(empty\_cart):

empty\_cart.add\_item("Item", 100.0)

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

empty\_cart.apply\_coupon("INVALID")

**Рисунки с результатами работы программы:**

****

**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

# 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\_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

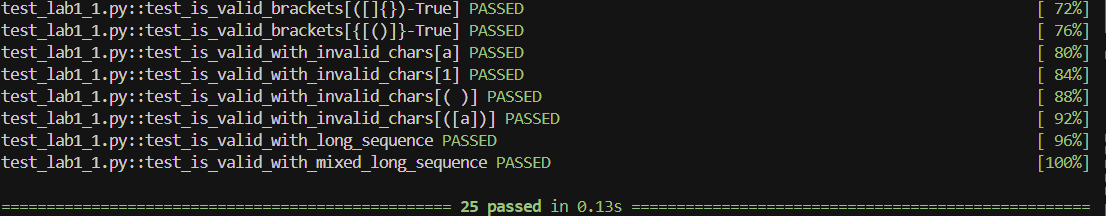
def test\_performance():

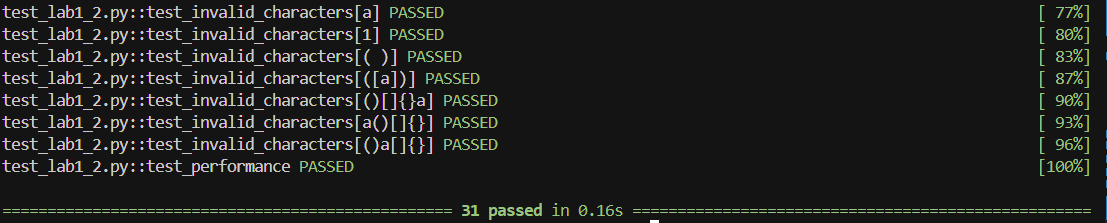
"""Тест на производительность с очень длинной последовательностью"""

brackets = "([{" \* 10000 + "}])" \* 10000

assert is\_valid(brackets) == True

**Рисунки с результатами работы программы:**

****

****

**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)

@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

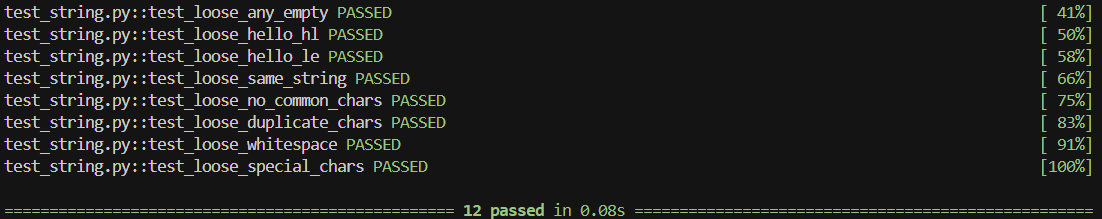
def test\_apply\_coupon\_invalid(empty\_cart):

empty\_cart.add\_item("Item", 100.0)

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

empty\_cart.apply\_coupon("INVALID")

**Рисунки с результатами работы программы:**

****

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