Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6
По дисциплине: “Современные платформы программирования”
Тема: “Тестирование кода с использованием пакета pytest”

Выполнил:
Студент 3 курса
Группы ПО-11
Янущик Д.Д.
Проверил:
Козик И.Д.

Брест 2025

Цель: Освоить приемы тестирования кода на примере использования пакета pytest.

Вариант 6

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
```

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

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.

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

6) Напишите метод String common(String str1, String str2) сравнивающий две строки и возвращающий наибольшую общую часть.
Спецификация метода:
common (None , None) = TypeError
common ("", "") = ""
common ("", " abc ") = ""
common (" abc ", "") = ""
common (" abc ", "abc") = " abc "
common ("ab", " abxyz ") = "ab"
common (" abcde ", " abxyz ") = "ab"
common (" abcde ", " xyz ") = ""
common (" deabc ", " abcdeabcd ") = " deabc "
common (" dfabcegt ", " rtoefabceiq ") = " fabce "

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, discount_percentage):
        if discount_percentage < 0 or discount_percentage > 100:
            raise ValueError("Discount must be between 0 and 100")
        for item in self.items:
            item["price"] *= (100 - discount_percentage) / 100

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

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:

```
@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", 20.0)
    assert empty_cart.total() == 30.0

@pytest.mark.parametrize("discount,expected_total", [
    (0, 30.0),
    (50, 15.0),
    (100, 0.0)
])
def test_apply_discount(empty_cart, discount, expected_total):
    empty_cart.add_item("Apple", 10.0)
    empty_cart.add_item("Banana", 20.0)
    empty_cart.apply_discount(discount)
    assert empty_cart.total() == pytest.approx(expected_total)

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

@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)

def test_apply_coupon_valid(empty_cart):
    empty_cart.add_item("Apple", 100.0)
    apply_coupon(empty_cart, "SAVE10")
    assert empty_cart.total() == 90.0
    apply_coupon(empty_cart, "HALF")
    assert empty_cart.total() == 45.0

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

def test_apply_coupon_mocked_coupons(empty_cart, monkeypatch):
    coupons = {"TEST": 20}
    monkeypatch.setattr("shopping.coupons", coupons)
    empty_cart.add_item("Apple", 100.0)
    apply_coupon(empty_cart, "TEST")
    assert empty_cart.total() == 80.0
```

find_outlier.py:

```
def find_outlier(sequence):
    mean = sum(sequence) / len(sequence)
```

```
    max_diff = 0
    outlier = sequence[0]

    for num in sequence:
        diff = abs(num - mean)
        if diff > max_diff:
            max_diff = diff
            outlier = num

    return outlier
```

is_palindrome.py:

```
def is_palindrome(x):
    return str(x) == str(x)[::-1]
```

test_functions.py:

```
import pytest
from find_outlier import find_outlier
from is_palindrome import is_palindrome

def test_find_outlier():
    assert find_outlier([1, 2, 3, 4, 5]) in [1, 5]
    assert find_outlier([10, 20, 30, 40, 100]) == 100
    assert find_outlier([-5, -4, -3, -2, -10]) == -10

def test_find_outlier_empty():
    with pytest.raises(ZeroDivisionError):
        find_outlier([])

@pytest.mark.parametrize("num,expected", [
    (121, True),
    (123, False),
    (0, True),
    (-121, False),
])
def test_is_palindrome(num, expected):
    assert is_palindrome(num) == expected
```

string_utils.py:

```
def common(str1, str2):
    if str1 is None or str2 is None:
        raise TypeError("Input strings cannot be None")

    if not str1 or not str2:
        return ""

    m = [[0] * (len(str2) + 1) for _ in range(len(str1) + 1)]
    max_length = 0
    end_pos = 0

    for i in range(1, len(str1) + 1):
        for j in range(1, len(str2) + 1):
            if str1[i - 1] == str2[j - 1]:
                m[i][j] = m[i - 1][j - 1] + 1
                if m[i][j] > max_length:
                    max_length = m[i][j]
                    end_pos = i
            else:
                m[i][j] = 0
    if max_length == 0:
        return ""
    return str1[end_pos - max_length:end_pos]
```

test_string_utils.py:

```
import pytest
from string_utils import common

@pytest.mark.parametrize("str1,str2,expected", [
    ("", "", ""),
    ("", " abc ", ""),
    (" abc ", "", ""),
    ("abc", "abc", "abc"),
    ("ab", "abxyz", "ab"),
    ("abcde", "abxyz", "ab"),
    ("abcde", "xyz", ""),
    ("deabc", "abcdeabcd", "deabc"),
    ("dfabcegt", "rtoefabceiq", "fabce"),
])
def test_common(str1, str2, expected):
    assert common(str1, str2) == expected

def test_common_none_input():
    with pytest.raises(TypeError):
        common(None, None)
    with pytest.raises(TypeError):
        common("abc", None)
    with pytest.raises(TypeError):
        common(None, "abc")
```

```
(.venv) PS C:\Users\kreot\PycharmProjects\SPP_L6> pytest
============================= test session starts =============================
platform win32 -- Python 3.13.1, pytest-8.3.5, pluggy-1.5.0
rootdir: C:\Users\kreot\PycharmProjects\SPP_L6
plugins: mock-3.14.0
collected 27 items

test_cart.py ...........
test_functions.py ......
test_string_utils.py ..........

============================== 27 passed in 0.76s =
```

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