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
С. С. Жватель
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

Выполнение:

Код программы:

```
"""Module for testing the shopping cart functionality."""

from unittest.mock import patch

import pytest

from .shopping import Cart, apply_coupon, log_purchase



@pytest.fixture
def cart():
    """Create and return an empty Cart instance.

    Returns:
        Cart: An empty shopping cart.
```

```
    """
    return Cart()


def test_add_item_adds_one_item(test_cart):
    """Test that adding an item increases the cart's item count and stores details correctly."""
    test_cart.add_item("Apple", 10.0)
    assert len(test_cart.items) == 1
    assert test_cart.items[0] == {"name": "Apple", "price": 10.0}


def test_add_item_negative_price_raises_error(test_cart):
    """Test that adding an item with a negative price raises a ValueError."""
    with pytest.raises(ValueError, match="Price cannot be negative"):
        test_cart.add_item("Apple", -10.0)


def test_total_calculation(test_cart):
    """Test that the total price is calculated correctly for multiple items."""
    test_cart.add_item("Apple", 10.0)
    test_cart.add_item("Banana", 20.0)
    assert test_cart.total() == 30.0


@pytest.mark.parametrize(
    "discount, expected_total",
    [
        (0, 10.0),  # 0% discount
        (50, 5.0),  # 50% discount
        (100, 0.0),  # 100% discount
    ],
)
def test_apply_discount(test_cart, discount, expected_total):
    """Test that applying a discount correctly adjusts the total price.

    Args:
        test_cart (Cart): The cart fixture.
        discount (float): The discount percentage to apply.
        expected_total (float): The expected total price after discount.
    """
    test_cart.add_item("Apple", 10.0)
    test_cart.apply_discount(discount)
    assert test_cart.total() == expected_total


@pytest.mark.parametrize("invalid_discount", [-1, 101])
def test_apply_discount_invalid_raises_error(test_cart, invalid_discount):
    """Test that applying an invalid discount raises a ValueError.
```

```
    Args:
        test_cart (Cart): The cart fixture.
        invalid_discount (float): The invalid discount value to test.
    """
    with pytest.raises(ValueError, match="Discount must be between 0 and 100"):
        test_cart.apply_discount(invalid_discount)


@patch("requests.post")
def test_log_purchase(mock_post, test_cart):
    """Test that log_purchase sends the correct item data to the server.

    Args:
        mock_post: Mocked requests.post function.
        test_cart (Cart): The cart fixture.
    """
    item = {"name": "Apple", "price": 10.0}
    test_cart.add_item("Apple", 10.0)
    log_purchase(item)
    mock_post.assert_called_once_with("https://example.com/log", json=item, timeout=5)


def test_apply_coupon_valid(test_cart):
    """Test that applying valid coupon codes correctly adjusts the total price."""
    test_cart.add_item("Apple", 10.0)
    apply_coupon(test_cart, "SAVE10")
    assert test_cart.total() == 9.0  # 10% discount

    apply_coupon(test_cart, "HALF")
    assert test_cart.total() == 5.0  # 50% discount


def test_apply_coupon_invalid_raises_error(test_cart):
    """Test that applying an invalid coupon code raises a ValueError."""
    with pytest.raises(ValueError, match="Invalid coupon"):
        apply_coupon(test_cart, "INVALID")


def test_apply_coupon_another_valid(test_cart):
    """Test that applying the HALF coupon correctly adjusts the total price."""
    test_cart.add_item("Apple", 10.0)
    apply_coupon(test_cart, "HALF")
    assert test_cart.total() == 5.0  # 50% discount
```

Рисунки с результатами работы программы:

```
reports/task1/test_cart.py::test_add_item_adds_one_item PASSED                                                    [  8%]
reports/task1/test_cart.py::test_add_item_negative_price_raises_error PASSED                                      [ 16%]
reports/task1/test_cart.py::test_total_calculation PASSED                                                         [ 25%]
reports/task1/test_cart.py::test_apply_discount[0-10.0] PASSED                                                    [ 33%]
reports/task1/test_cart.py::test_apply_discount[50-5.0] PASSED                                                    [ 41%]
reports/task1/test_cart.py::test_apply_discount[100-0.0] PASSED                                                   [ 50%]
reports/task1/test_cart.py::test_apply_discount_invalid_raises_error[-1] PASSED                                   [ 58%]
reports/task1/test_cart.py::test_apply_discount_invalid_raises_error[101] PASSED                                  [ 66%]
reports/task1/test_cart.py::test_log_purchase PASSED                                                              [ 75%]
reports/task1/test_cart.py::test_apply_coupon_valid PASSED                                                        [ 83%]
reports/task1/test_cart.py::test_apply_coupon_invalid_raises_error PASSED                                         [ 91%]
reports/task1/test_cart.py::test_apply_coupon_another_valid PASSED                                                [100%]
```

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.

Выполнение:

Код программы:

test_count_digits_distribution.py:

```
"""Module for testing the count_digits_distribution function."""

from .task1 import count_digits_distribution


def test_empty_sequence():
    """Test that an empty sequence returns an empty dictionary."""
    assert not count_digits_distribution([])  # Simplified: empty dict is falsy


def test_single_number():
    """Test that a single number returns the correct digit count."""
    assert count_digits_distribution([123]) == {3: 1}


def test_multiple_numbers():
    """Test that multiple numbers return the correct digit distribution."""
    assert count_digits_distribution([1, 12, 123, 1234]) == {1: 1, 2: 1, 3: 1, 4: 1}


def test_negative_numbers():
    """Test that negative numbers are handled correctly."""
    assert count_digits_distribution([-12, -123]) == {2: 1, 3: 1}


def test_zero():
    """Test that zero is counted as having one digit."""
    assert count_digits_distribution([0]) == {1: 1}


def test_large_number():
    """Test that a large number returns the correct digit count."""
    assert count_digits_distribution([1000000]) == {7: 1}
```

```
    """Module for calculating the Hamming weight of positive integers."""


    def hamming_weight(n):
        """Calculate the Hamming weight (number of 1s in binary) of a number.

        Args:
            n (int): A positive integer.

        Returns:
            int: The count of 1s in the binary representation of n.
        """
        return bin(n).count("1")


    print("Введите количество тестов:")
    t = int(input())

    print(f"Введите {t} положительных целых чисел (каждое на новой строке):")
    for _ in range(t):
        input_n = int(input())
        print(f"Input: n = {input_n}, Output: {hamming_weight(input_n)}")
```

test_hamming_weight.py:

```
"""Module for testing the hamming_weight function."""

import pytest

from .task2 import hamming_weight


def test_zero():
    """Test that the Hamming weight of zero is zero."""
    assert hamming_weight(0) == 0


def test_positive_number():
    """Test that the Hamming weight of a positive number (11) is correct."""
    assert hamming_weight(11) == 3  # 11 = 1011 in binary


def test_power_of_two():
    """Test that the Hamming weight of a power of two (16) is correct."""
    assert hamming_weight(16) == 1  # 16 = 10000 in binary


def test_large_number():
    """Test that the Hamming weight of a large number (2^32 - 1) is correct."""
    assert hamming_weight(4294967295) == 32  # 2^32 - 1, all bits 1
```

```
def test_negative_number_raises_error():
    """Test that a negative number raises a ValueError."""
    with pytest.raises(ValueError):
        hamming_weight(-1)
```

Рисунки с результатами работы программы:

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

Реализуйте метод int hamingDistance(String str1, String str2), определяющий расстояние Хэминга для двух строк. Дистанция Хэминга – это число позиций, в которых соответствующие символы двух слов одинаковой длины различны.

Спецификация метода:

hammingDistance (None, None) = TypeError

hammingDistance (None, *) = -1

hammingDistance (* , None) = -1

hammingDistance ("abc", " abcd ") = ValueError

hammingDistance ("","") = 0

hammingDistance (" father ", " father ") = 0

hammingDistance ("pip", " pop ") = 1

hammingDistance (" abcd ", " abab ") = 2

hammingDistance (" hello ", " hallo ") = 1

hammingDistance (" abcd ", " efgi ") = 4

Выполнение:

Код программы:

```
"""Module for testing the hamming_distance function."""

import pytest
from .hamming_distance import hamming_distance


def test_both_none_raises_type_error():
    """Test that passing None for both arguments raises a TypeError."""
    with pytest.raises(TypeError):
        hamming_distance(None, None)


def test_first_none_returns_minus_one():
    """Test that passing None as the first argument returns -1."""
```

```
    assert hamming_distance(None, "abc") == -1


def test_second_none_returns_minus_one():
    """Test that passing None as the second argument returns -1."""
    assert hamming_distance("abc", None) == -1


def test_different_lengths_raises_value_error():
    """Test that strings of different lengths raise a ValueError."""
    with pytest.raises(ValueError):
        hamming_distance("abc", "abcd")


def test_empty_strings():
    """Test that empty strings return a Hamming distance of 0."""
    assert hamming_distance("", "") == 0


def test_identical_strings():
    """Test that identical strings return a Hamming distance of 0."""
    assert hamming_distance("father", "father") == 0


def test_one_difference():
    """Test that strings with one difference return a Hamming distance of 1."""
    assert hamming_distance("pip", "pop") == 1


def test_two_differences():
    """Test that strings with two differences return a Hamming distance of 2."""
    assert hamming_distance("abcd", "abab") == 2


def test_one_difference_hello():
    """Test that 'hello' and 'hallo' return a Hamming distance of 1."""
    assert hamming_distance("hello", "hallo") == 1


def test_all_different():
    """Test that completely different strings return the correct Hamming distance."""
    assert hamming_distance("abcd", "efgi") == 4
```

Рисунки с результатами работы программы:

Вывод: закрепил базовые знания Python, а также научились создавать тесты при использовании пакета pytest при решении практических задач.