

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №6

Специальность ПО

Выполнил

В. С. Юрашевич,

студент группы ПО-11

Проверил

И. Д. Козик

Брест 2025

Цель работы: освоить приемы тестирования кода на примере использования пакета pytest.

Ход работы

Задание 1: Написание тестов для мини-библиотеки покупок (shopping.py)

Код программы

shopping.py

```
import requests

coupons = {
    "SAVE10": 10,
    "HALF": 50
}

class Cart:
    def __init__(self):
        self.items = []
        self.discount = 0

    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):
        total = sum(item["price"] for item in self.items)
        return total * (1 - self.discount / 100)

    def apply_discount(self, percent):
        if percent < 0 or percent > 100:
            raise ValueError("Invalid discount")
        self.discount = percent

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

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

```
import pytest

from shopping import Cart, apply_coupon, log_purchase, coupons
from unittest.mock import patch

@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):
        empty_cart.add_item("Banana", -5.0)

def test_total_calculation(empty_cart):
    empty_cart.add_item("A", 10.0)
    empty_cart.add_item("B", 15.0)
    assert empty_cart.total() == 25.0

@pytest.mark.parametrize("discount,expected_total", [
    (0, 100.0),
    (50, 50.0),
    (100, 0.0),
])

def test_apply_discount_valid(discount, expected_total):
    cart = Cart()
    cart.add_item("Item", 100.0)
    cart.apply_discount(discount)
```

```

        assert cart.total() == expected_total

@pytest.mark.parametrize("discount", [-10, 110])
def test_apply_discount_invalid(discount):

    cart = Cart()

    cart.add_item("Item", 100.0)

    with pytest.raises(ValueError):

        cart.apply_discount(discount)

@patch("shopping.requests.post")
def test_log_purchase_called(mock_post):

    item = {"name": "TestItem", "price": 25.0}

    log_purchase(item)

    mock_post.assert_called_once_with("https://example.com/log", json=item)

def test_apply_coupon_valid(monkeypatch):

    cart = Cart()

    cart.add_item("Item", 100.0)

    monkeypatch.setitem(coupons, "SAVE10", 10)

    apply_coupon(cart, "SAVE10")

    assert cart.total() == 90.0

def test_apply_coupon_invalid():

    cart = Cart()

    cart.add_item("Item", 100.0)

    with pytest.raises(ValueError):

        apply_coupon(cart, "INVALIDCODE")

```

Рисунок с результатами работы



```

test_cart.py::test_add_item PASSED [ 9%]
test_cart.py::test_add_item_negative_price PASSED [ 18%]
test_cart.py::test_total_calculation 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_called PASSED [ 81%]
test_cart.py::test_apply_coupon_valid PASSED [ 90%]
test_cart.py::test_apply_coupon_invalid PASSED [100%]

```

Задание 2: Напишите тесты к реализованным функциям из лабораторной работы №1. 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.py

```
def countDigits(_number):
    return len(str(abs(_number)))

def calculateDigitDistribution(_numbers):
    digitCounts = dict()

    for number in _numbers:
        digitCounts[countDigits(number)] = 0

    for number in _numbers:
        digitCounts[countDigits(number)] += 1

    return digitCounts

def determineHammingWeight(_number):
    if(_number < 0):
        return 0

    counter = 0

    while(_number):
        _number &= _number - 1

        counter += 1

    return counter
```

test_lab1.py

```
import pytest

from lab1 import countDigits, calculateDigitDistribution, determineHammingWeight

@pytest.mark.parametrize("number,expected", [
    (0, 1),
    (5, 1),
    (42, 2),
    (123, 3),
    (-7, 1),
```

```

        (-999, 3),
    ])

def test_countDigits(number, expected):
    assert countDigits(number) == expected

def test_calculateDigitDistribution_normal():
    numbers = [1, 22, 333, 4444, 5, 123]
    result = calculateDigitDistribution(numbers)
    assert result == {1: 2, 2: 1, 3: 2, 4: 1}

def test_calculateDigitDistribution_all_same():
    numbers = [111, 222, 333]
    assert calculateDigitDistribution(numbers) == {3: 3}

def test_calculateDigitDistribution_empty():
    assert calculateDigitDistribution([]) == {}

@pytest.mark.parametrize("number,expected", [
    (0, 0),
    (1, 1),
    (3, 2),
    (8, 1),
    (15, 4),
    (255, 8),
])

def test_determineHammingWeight_positive(number, expected):
    assert determineHammingWeight(number) == expected

def test_determineHammingWeight_negative():
    assert determineHammingWeight(-10) == 0

```

Рисунок с результатами работы

```

test_lab1.py::test_countDigits[0-1] PASSED [ 6%]
test_lab1.py::test_countDigits[5-1] PASSED [ 12%]
test_lab1.py::test_countDigits[42-2] PASSED [ 18%]
test_lab1.py::test_countDigits[123-3] PASSED [ 25%]
test_lab1.py::test_countDigits[-7-1] PASSED [ 31%]
test_lab1.py::test_countDigits[-999-3] PASSED [ 37%]
test_lab1.py::test_calculateDigitDistribution_normal PASSED [ 43%]
test_lab1.py::test_calculateDigitDistribution_all_same PASSED [ 50%]
test_lab1.py::test_calculateDigitDistribution_empty PASSED [ 56%]
test_lab1.py::test_determineHammingWeight_positive[0-0] PASSED [ 62%]
test_lab1.py::test_determineHammingWeight_positive[1-1] PASSED [ 68%]
test_lab1.py::test_determineHammingWeight_positive[3-2] PASSED [ 75%]
test_lab1.py::test_determineHammingWeight_positive[8-1] PASSED [ 81%]
test_lab1.py::test_determineHammingWeight_positive[15-4] PASSED [ 87%]
test_lab1.py::test_determineHammingWeight_positive[255-8] PASSED [ 93%]
test_lab1.py::test_determineHammingWeight_negative PASSED [100%]
===== 16 passed in 0.08s =====

```

Задание 3: Написать тесты к методу, а затем реализовать сам метод по заданной спецификации. Реализуйте и протестируйте метод `indexOfDifference(String str1, String str2)`, который сравнивает две строки и возвращает индекс той позиции, в которой они различаются. Например, `indexOfDifference("i am a machine" , "i am a robot")` должно вернуть 7.

Код программы

index_of_difference.py

```

def indexOfDifference(str1, str2):

    if str1 is None or str2 is None:

        raise TypeError("Input strings must not be None")

    str1 = str1.lstrip()

    str2 = str2.lstrip()

    if str1 == str2:

        return -1

    max_len = max(len(str1), len(str2))

    for i in range(max_len):

        c1 = str1[i] if i < len(str1) else None

        c2 = str2[i] if i < len(str2) else None

        if c1 != c2:

            return i

    return -1

```

test_index_of_difference.py

```

import pytest

from index_of_difference import indexOfDifference

@pytest.mark.parametrize("str1, str2, expected", [

```

```

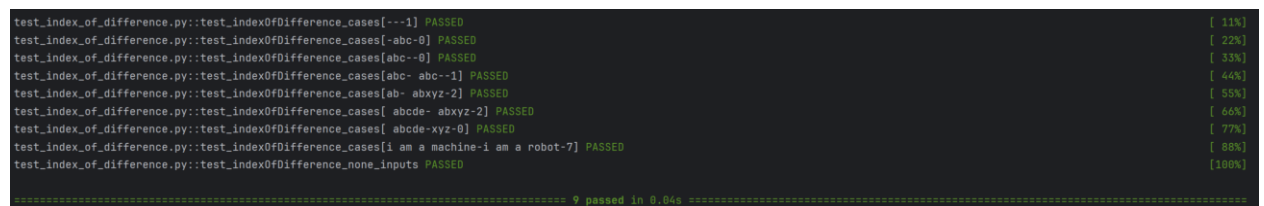
        ("", "", -1),
        ("", "abc", 0),
        ("abc", "", 0),
        ("abc", " abc", -1),
        ("ab", " abxyz", 2),
        (" abcde", " abxyz", 2),
        (" abcde", "xyz", 0),
        ("i am a machine", "i am a robot", 7),
    ])

def test_indexOfDifference_cases(str1, str2, expected):
    assert indexOfDifference(str1, str2) == expected

def test_indexOfDifference_none_inputs():
    with pytest.raises(TypeError):
        indexOfDifference(None, None)

```

Рисунок с результатами работы



```

test_index_of_difference.py::test_indexOfDifference_cases[---1] PASSED [ 11%]
test_index_of_difference.py::test_indexOfDifference_cases[-abc-0] PASSED [ 22%]
test_index_of_difference.py::test_indexOfDifference_cases[abc--0] PASSED [ 33%]
test_index_of_difference.py::test_indexOfDifference_cases[abc- abc--1] PASSED [ 44%]
test_index_of_difference.py::test_indexOfDifference_cases[ab- abxyz-2] PASSED [ 55%]
test_index_of_difference.py::test_indexOfDifference_cases[ abcde- abxyz-2] PASSED [ 66%]
test_index_of_difference.py::test_indexOfDifference_cases[ abcde-xyz-0] PASSED [ 77%]
test_index_of_difference.py::test_indexOfDifference_cases[i am a machine-i am a robot-7] PASSED [ 88%]
test_index_of_difference.py::test_indexOfDifference_none_inputs PASSED [100%]

===== 9 passed in 0.04s =====

```

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