

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №6

Специальность ПО11(о)

Выполнил
И. А. Головач,
студент группы ПО11

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«26» апрель 2025 г.

Вариант 5

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:

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):
        if discount < 0 or discount > 100:
            raise ValueError("Discount must be between 0 and 100")
        total = self.total()
        self.items.append({"name": "Discount", "price": -total * discount / 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:

```
import pytest
from shopping import Cart, log_purchase, apply_coupon
from unittest.mock import patch, MagicMock

# Фикстура для пустой корзины
@pytest.fixture
def empty_cart():
    return Cart()

# Тест добавления товара
def test_add_item(empty_cart):
    cart = empty_cart
    cart.add_item("Apple", 10.0)
```

```

assert len(cart.items) == 1
assert cart.items[0]["name"] == "Apple"
assert cart.items[0]["price"] == 10.0

# Тест на отрицательную цену
def test_negative_price(empty_cart):
    cart = empty_cart
    with pytest.raises(ValueError, match="Price cannot be negative"):
        cart.add_item("Apple", -10.0)

# Тест вычисления общей стоимости
def test_total(empty_cart):
    cart = empty_cart
    cart.add_item("Apple", 10.0)
    cart.add_item("Banana", 20.0)
    assert cart.total() == 30.0

# Параметризованный тест для apply_discount
@pytest.mark.parametrize("discount, expected_total", [
    (0, 100.0),
    (50, 50.0),
    (100, 0.0),
])
def test_apply_discount(empty_cart, discount, expected_total):
    cart = empty_cart
    cart.add_item("Item", 100.0)
    cart.apply_discount(discount)
    assert cart.total() == expected_total

# Тест на недопустимые значения скидки
@pytest.mark.parametrize("invalid_discount", [-10, 110])
def test_invalid_discount(empty_cart, invalid_discount):
    cart = empty_cart
    cart.add_item("Item", 100.0)
    with pytest.raises(ValueError, match="Discount must be between 0 and 100"):
        cart.apply_discount(invalid_discount)

# Тест для log_purchase с моком requests.post
@patch('requests.post')
def test_log_purchase(mock_post):
    test_item = {"name": "Apple", "price": 10.0}
    mock_response = MagicMock()
    mock_response.status_code = 200
    mock_post.return_value = mock_response

    log_purchase(test_item)

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

# Тесты для apply_coupon

```

```

def test_apply_coupon_valid(empty_cart):
    cart = empty_cart
    cart.add_item("Item", 100.0)
    apply_coupon(cart, "SAVE10")
    assert cart.total() == 90.0 # 10% скидка

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

# Тест с monkeypatch для мока словаря coupons
def test_apply_coupon_with_monkeypatch(empty_cart, monkeypatch):
    cart = empty_cart
    cart.add_item("Item", 100.0)

    # Мокаем словарь купонов
    monkeypatch.setattr("shopping.coupons", {"TEST50": 50})

    apply_coupon(cart, "TEST50")
    assert cart.total() == 50.0

```

Результаты работы программы:

```

Run Python tests in test_cart.py x
✓ Test Results 1 ms
✓ Tests passed: 12 of 12 tests - 1 ms

C:\Users\ilyag\AppData\Local\Programs\Python\Python313\python.exe "C:/Program Files/JetBrains/PyCharm 2024.3.3/plugins/py
Testing started at 12:38 ...
Launching pytest with arguments D:\labs_university\SSP\lab1\lab6\src\task_1\test_cart.py --no-header --no-summary -q in D

===== test session starts =====
collecting ... collected 12 items

test_cart.py::test_add_item PASSED [ 8%]
test_cart.py::test_negative_price PASSED [ 16%]
test_cart.py::test_total PASSED [ 25%]
test_cart.py::test_apply_discount[0-100.0] PASSED [ 33%]
test_cart.py::test_apply_discount[50-50.0] PASSED [ 41%]
test_cart.py::test_apply_discount[100-0.0] PASSED [ 50%]
test_cart.py::test_invalid_discount[-10] PASSED [ 58%]
test_cart.py::test_invalid_discount[110] PASSED [ 66%]
test_cart.py::test_log_purchase PASSED [ 75%]
test_cart.py::test_apply_coupon_valid PASSED [ 83%]
test_cart.py::test_apply_coupon_invalid PASSED [ 91%]
test_cart.py::test_apply_coupon_with_monkeypatch PASSED [100%]

===== 12 passed in 0.56s =====

Process finished with exit code 0

```

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:

```
def parse_input(user_input):
    if not user_input.strip(): # Проверяем, что строка не пустая
        raise ValueError("Входная строка не может быть пустой")
    try:
        return list(map(int, user_input.split()))
    except ValueError:
        raise ValueError("Все элементы должны быть целыми числами")

def calculate_range(numbers):
    if not numbers:
        raise ValueError("Список не может быть пустым")
    return max(numbers) - min(numbers)

def main():
    try:
        user_input = input("Введите массив целых чисел через пробел: ")
        numbers_list = parse_input(user_input)
        print("Введенный массив: ", numbers_list, "\n")
        print("Размах последовательности: ", calculate_range(numbers_list), "\n")
    except ValueError as e:
        print(f"Ошибка: {e}")

if __name__ == "__main__":
    main()
```

lab1_2.py:

```
def longest_common_prefix(strs):
    if not strs:
        return ""

    prefix = ""
    for chars in zip(*strs):
        if len(set(chars)) == 1:
            prefix += chars[0]
        else:
            break

    return prefix

def main():
    user_input = input("Введите строки через пробел: ")
    strings_list = list(map(str, user_input.split()))
    print(longest_common_prefix(strings_list))
```

```
if __name__ == "__main__":
    main()
```

test_lab1_1.py:

```
import pytest
from lab1_1 import parse_input, calculate_range

def test_parse_input_valid():
    assert parse_input("1 2 3") == [1, 2, 3]
    assert parse_input("-5 0 5") == [-5, 0, 5]
    assert parse_input("100") == [100]

def test_parse_input_invalid():
    with pytest.raises(ValueError, match="Все элементы должны быть целыми числами"):
        parse_input("1 2 abc")
    with pytest.raises(ValueError, match="Все элементы должны быть целыми числами"):
        parse_input("1.5 2 3")
    with pytest.raises(ValueError, match="Входная строка не может быть пустой"):
        parse_input("") # Теперь это вызывает исключение

def test_calculate_range():
    assert calculate_range([1, 2, 3]) == 2
    assert calculate_range([-5, 0, 5]) == 10
    assert calculate_range([100]) == 0
    assert calculate_range([5, 5, 5]) == 0

def test_calculate_range_empty():
    with pytest.raises(ValueError, match="Список не может быть пустым"):
        calculate_range([])
```

test_lab1_2.py:

```
import pytest
from lab1_2 import longest_common_prefix

def test_common_prefix():
    assert longest_common_prefix(["flower", "flow", "flight"]) == "fl"
    assert longest_common_prefix(["dog", "racecar", "car"]) == ""
    assert longest_common_prefix(["apple", "apple", "apple"]) == "apple"
    assert longest_common_prefix(["", "test", "test"]) == ""
    assert longest_common_prefix(["prefix", "preference", "preform"]) == "pref"

def test_edge_cases():
    assert longest_common_prefix([""]) == ""
    assert longest_common_prefix(["a"]) == "a"
    assert longest_common_prefix(["", ""]) == ""

def test_empty_input():
    assert longest_common_prefix([]) == ""

def test_different_cases():
```

```
assert longest_common_prefix(["Python", "python"]) == ""
assert longest_common_prefix(["Test", "TEst", "TEST"]) == "T"
```

Результаты работы программы:

The screenshot shows the PyCharm Run window for 'Python tests in task_2'. The 'Test Results' tab is active, showing a summary of 8 passed tests in 1 ms. The console output details the test session, including the collection of 8 items and the execution of each test case, all of which passed.

```
Run Python tests in task_2 x
Test Results 1 ms
Tests passed: 8 of 8 tests - 1 ms

C:\Users\ilyag\AppData\Local\Programs\Python\Python313\python.exe "C:/Program Files/JetBrains/PyCharm 2024.3.3/plugins/p
Testing started at 12:47 ...
Launching pytest with arguments D:\labs_university\SSP\lab1\lab6\src\task_2 --no-header --no-summary -q in D:\labs_unive

===== test session starts =====
collecting ... collected 8 items

test_lab1_1.py::test_parse_input_valid PASSED [ 12%]
test_lab1_1.py::test_parse_input_invalid PASSED [ 25%]
test_lab1_1.py::test_calculate_range PASSED [ 37%]
test_lab1_1.py::test_calculate_range_empty PASSED [ 50%]
test_lab1_2.py::test_common_prefix PASSED [ 62%]
test_lab1_2.py::test_edge_cases PASSED [ 75%]
test_lab1_2.py::test_empty_input PASSED [ 87%]
test_lab1_2.py::test_different_cases PASSED [100%]

===== 8 passed in 0.03s =====

Process finished with exit code 0
```

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

Реализуйте и протестируйте метод `int 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.

Спецификация метода:

`indexOfDifference (None, None) = TypeError`

`indexOfDifference ("", "") = -1`

`indexOfDifference ("", "abc ") = 0`

`indexOfDifference ("abc ", "") = 0`

`indexOfDifference ("abc ", " abc ") = -1`

`indexOfDifference ("ab", " abxyz ") = 2`

`indexOfDifference (" abcde ", " abxyz ") = 2`

`indexOfDifference (" abcde ", "xyz") = 0`

Код программы:

string_utils.py:

```
def indexOfDifference(str1, str2):  
    """  
        Сравнивает две строки и возвращает индекс позиции, в которой они различаются.  
  
        :param str1: Первая строка  
        :param str2: Вторая строка  
        :return: Индекс различия или -1, если строки полностью совпадают  
        :raises TypeError: Если оба аргумента равны None  
    """  
    # Проверка на None  
    if str1 is None and str2 is None:  
        raise TypeError("Оба аргумента не могут быть None")  
  
    # Обрезаем пробелы в начале и конце строк  
    str1 = str1.strip()  
    str2 = str2.strip()  
  
    # Определяем минимальную длину для итерации  
    min_length = min(len(str1), len(str2))  
  
    # Ищем первую позицию, где символы не совпадают  
    for i in range(min_length):  
        if str1[i] != str2[i]:  
            return i  
  
    # Если строки совпадают до конца одной из них  
    if len(str1) != len(str2):  
        return min_length  
  
    # Если строки полностью совпадают  
    return -1
```

test_string_utils.py:

```
import pytest  
from string_utils import indexOfDifference  
  
def test_indexOfDifference_both_none():  
    """Проверка на TypeError при обоих аргументах None"""  
    with pytest.raises(TypeError, match="Оба аргумента не могут быть None"):  
        indexOfDifference(None, None)  
  
def test_indexOfDifference_empty_strings():  
    """Сравнение двух пустых строк"""  
    assert indexOfDifference("", "") == -1  
  
def test_indexOfDifference_one_empty_string():  
    """Сравнение пустой строки с непустой"""  
    assert indexOfDifference("", "abc") == 0  
    assert indexOfDifference("abc", "") == 0  
  
def test_indexOfDifference_equal_strings():  
    """Сравнение одинаковых строк"""  
    assert indexOfDifference("abc", "abc") == -1
```

```
def test_indexOfDifference_different_strings():
    """Сравнение строк с разницей в середине"""
    assert indexOfDifference("i am a machine", "i am a robot") == 7
    assert indexOfDifference("ab", " abxyz ") == 2
    assert indexOfDifference(" abcde ", " abxyz ") == 2
    assert indexOfDifference(" abcde ", "xyz") == 0

def test_indexOfDifference_whitespace_handling():
    """Сравнение строк с пробелами"""
    assert indexOfDifference("abc ", " abc ") == -1

def test_indexOfDifference_different_lengths():
    """Сравнение строк разной длины"""
    assert indexOfDifference("short", "shorter") == 5
    assert indexOfDifference("longer", "long") == 4
```

Результаты работы программы:

The screenshot shows the PyCharm Run window for a Python test file. The interface includes a toolbar with icons for running, debugging, and other actions. Below the toolbar, a tab labeled 'Test Results' is active, showing a summary of the test execution. The main pane displays the output of the test runner, which includes the path to the Python interpreter, the start time, the arguments used to launch pytest, and a detailed list of the tests that were run and their results. All tests passed successfully.

```
Run Python tests in test_string_utils.py x
Test Results 0ms
Tests passed: 7 of 7 tests - 0ms
C:\Users\ilyag\AppData\Local\Programs\Python\Python313\python.exe "C:/Program Files/JetBrains/PyCharm 2024.3.3/plugins/py
Testing started at 12:54 ...
Launching pytest with arguments D:\labs_university\SSP\lab1\lab6\src\task_3\test_string_utils.py --no-header --no-summary

===== test session starts =====
collecting ... collected 7 items

test_string_utils.py::test_indexOfDifference_both_none PASSED [ 14%]
test_string_utils.py::test_indexOfDifference_empty_strings PASSED [ 28%]
test_string_utils.py::test_indexOfDifference_one_empty_string PASSED [ 42%]
test_string_utils.py::test_indexOfDifference_equal_strings PASSED [ 57%]
test_string_utils.py::test_indexOfDifference_different_strings PASSED [ 71%]
test_string_utils.py::test_indexOfDifference_whitespace_handling PASSED [ 85%]
test_string_utils.py::test_indexOfDifference_different_lengths PASSED [100%]

===== 7 passed in 0.02s =====

Process finished with exit code 0
```

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