МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №6

По дисциплине “**Современные платформы программирования**”

Выполнил:
студент группы ПО-11
Сильчук Д.А.
Проверил:
Козик И. Д.

Брест 2025

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

Вариант 16

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

1. Создайте файл test_cart.py. Реализуйте следующие тесты:
2. Протестируйте метод apply_discount с разными значениями скидки:
3. Создайте фикстуру empty_cart, которая возвращает пустой экземпляр Cart
4. Допустим, у нас есть функция, которая логирует покупку в удалённую систему:
5. Добавьте поддержку купонов:

Код программы:

Shopping.py

```
import requests

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

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

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, log_purchase, apply_coupon, 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_negative_price(empty_cart):
    with pytest.raises(ValueError):
        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", [
    (0, 30.0),
    (50, 15.0),
    (100, 0.0)
])
def test_apply_discount(empty_cart, discount, expected):
    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)

def test_invalid_discount(empty_cart):
    with pytest.raises(ValueError):
        empty_cart.apply_discount(-10)
    with pytest.raises(ValueError):
        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(empty_cart, monkeypatch):
    empty_cart.add_item("Apple", 100.0)
    apply_coupon(empty_cart, "SAVE10")
    assert empty_cart.total() == 90.0

    with pytest.raises(ValueError):
        apply_coupon(empty_cart, "INVALID")

    new_cart = Cart()
    new_cart.add_item("Apple", 100.0)
    monkeypatch.setitem(COUPONS, "NEW50", 50)
    apply_coupon(new_cart, "NEW50")
    assert new_cart.total() == 50.0
```

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:

Spp1.py

```
# -*- coding: utf-8 -*-

def calculate_stats(numbers):
    if not numbers:
        raise ValueError("The list of numbers must not be empty")

    stats = {
        "max": max(numbers),
        "min": min(numbers),
        "sum": sum(numbers),
        "product": 1
    }

    for num in numbers:
        stats["product"] *= num

    return stats

def main():
    input_str = input("Enter numbers: ")
```

```
    try:
        numbers = list(map(int, input_str.split()))
    except ValueError:
        print("Error: Please enter only numbers separated by spaces!")
        return

    if not numbers:
        print("Error: No numbers entered!")
        return

    stats = calculate_stats(numbers)

    print("\nResults:")
    print(f"Maximum value: {stats['max']}")
    print(f"Minimum value: {stats['min']}")
    print(f"Sum: {stats['sum']}")
    print(f"Product: {stats['product']}")

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

Spp1_2.py

```
# -*- coding: utf-8 -*-

def merge_sorted_arrays(nums1, m, nums2, n):
    i = m - 1
    j = n - 1
    k = m + n - 1

    while i >= 0 and j >= 0:
        if nums1[i] > nums2[j]:
            nums1[k] = nums1[i]
            i -= 1
        else:
            nums1[k] = nums2[j]
            j -= 1
        k -= 1

    while j >= 0:
        nums1[k] = nums2[j]
        j -= 1
        k -= 1

def main():
    try:
        m = int(input("m = "))
        n = int(input("n = "))

        while True:
            try:
                nums1 = list(map(int, input("nums1 = ").split()))
                if len(nums1) == m:
                    break
                print(f"Error: nums1 must contain exactly {m} numbers")
            except ValueError:
                print("Error: Please enter only numbers")

        while True:
            try:
                nums2 = list(map(int, input("nums2 = ").split()))
                if len(nums2) == n:
                    break
                print(f"Error: nums2 must contain exactly {n} numbers")
            except ValueError:
                print("Error: Please enter only numbers")

        nums1 += [0] * n

        print("\nOriginal nums1:", nums1[:m], "+ buffer:", nums1[m:])
```

```
        print("Original nums2:", nums2)

        merge_sorted_arrays(nums1, m, nums2, n)

        print("\nResult:", nums1)
    except ValueError:
        print("Error: Invalid input format")

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

Test_spp1.py

```
import pytest
from spp1 import calculate_stats

def test_calculate_stats():
    # Test with normal input
    numbers = [1, 2, 3, 4, 5]
    result = calculate_stats(numbers)
    assert result["max"] == 5
    assert result["min"] == 1
    assert result["sum"] == 15
    assert result["product"] == 120

    # Test with single number
    result = calculate_stats([10])
    assert result["max"] == 10
    assert result["min"] == 10
    assert result["sum"] == 10
    assert result["product"] == 10

    # Test with negative numbers
    result = calculate_stats([-1, -2, -3])
    assert result["max"] == -1
    assert result["min"] == -3
    assert result["sum"] == -6
    assert result["product"] == -6

def test_main_invalid_input(monkeypatch, capsys):
    # Test non-numeric input
    monkeypatch.setattr('builtins.input', lambda _: "a b c")
    from spp1 import main
    main()
    captured = capsys.readouterr()
    assert "Error: Please enter only numbers separated by spaces!" in captured.out

    # Test empty input
    monkeypatch.setattr('builtins.input', lambda _: "")
    main()
    captured = capsys.readouterr()
    assert "Error: No numbers entered!" in captured.out
```

Test_spp1_2.py

```
import pytest
from spp1_2 import merge_sorted_arrays, main

def test_merge_sorted_arrays():
    """Test merging of sorted arrays"""
    # Test normal case
    nums1 = [1, 3, 5, 0, 0, 0]
    nums2 = [2, 4, 6]
    merge_sorted_arrays(nums1, 3, nums2, 3)
    assert nums1 == [1, 2, 3, 4, 5, 6]

    # Test empty nums2
    nums1 = [1, 2, 3, 0, 0, 0]
    nums2 = []
    merge_sorted_arrays(nums1, 3, nums2, 0)
    assert nums1 == [1, 2, 3, 0, 0, 0]
```

```
    # Test all nums2 elements are larger
    nums1 = [1, 2, 3, 0, 0, 0]
    nums2 = [4, 5, 6]
    merge_sorted_arrays(nums1, 3, nums2, 3)
    assert nums1 == [1, 2, 3, 4, 5, 6]

def test_main_invalid_input(monkeypatch, capsys):
    """Test input validation in main()"""
    # Test case 1: Invalid nums1 length
    inputs = ["3", "2", "1 2", "1 2 3", "4 5"]  # Добавлены дополнительные входные данные
    input_gen = iter(inputs)
    monkeypatch.setattr('builtins.input', lambda _: next(input_gen))
    main()
    captured = capsys.readouterr()
    assert "Error: nums1 must contain exactly 3 numbers" in captured.out

    # Test case 2: Invalid nums2 length
    inputs = ["2", "3", "1 2", "3 4", "3 4 5"]  # Добавлены дополнительные входные данные
    input_gen = iter(inputs)
    monkeypatch.setattr('builtins.input', lambda _: next(input_gen))
    main()
    captured = capsys.readouterr()
    assert "Error: nums2 must contain exactly 3 numbers" in captured.out

    # Test case 3: Non-numeric input
    inputs = ["2", "2", "a b", "1 2", "3 4"]  # Добавлены дополнительные входные данные
    input_gen = iter(inputs)
    monkeypatch.setattr('builtins.input', lambda _: next(input_gen))
    main()
    captured = capsys.readouterr()
    assert "Error: Please enter only numbers" in captured.out
```

Задание 3

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

7) Напишите метод String substringBetween(String str, String open, String close) выделяющий подстроку относительно открывающей и закрывающей строки.

Спецификация метода:

substringBetween (None , None, None) = TypeError
substringBetween (None, * , *) = None
substringBetween (* , None, *) = None
substringBetween (* , * , None) = None
substringBetween ("", "", "") = ""
substringBetween ("", "", "]") = None
substringBetween ("", "[", "]") = None
substringBetween (" yabcz ", "", "") = ""
substringBetween (" yabcz ", "y", "z") = " abc"
substringBetween (" yabczyabcz ", "y", "z") = " abc "
substringBetween ("wx[b]yz", "[", "]") = "b"

Код программы:

`Substring.py`

```
def substringBetween(string, open_str, close_str):
    """
    Извлекает подстроку между open_str и close_str.
    Спецификация:
    - Если все аргументы None → TypeError
    - Любой аргумент None → None
    - Оба разделителя пустые → вся строка
    - Не найдены разделители → None
```

```
    - Найдены разделители → подстрока между ними
    - Для пробелов как разделителей → содержимое между первыми двумя пробелами, обрезанное от пробелов
    """
    if string is None or open_str is None or close_str is None:
        if string is None and open_str is None and close_str is None:
            raise TypeError("All arguments cannot be None")
        return None

    if open_str == "" and close_str == "":
        return string

    # Специальная обработка для пробелов как разделителей
    if open_str == " " and close_str == " ":
        stripped = string.strip()
        parts = stripped.split()
        return parts[0] if parts else None

    start_idx = string.find(open_str)
    if start_idx == -1:
        return None

    start_idx += len(open_str)
    end_idx = string.find(close_str, start_idx)
    if end_idx == -1:
        return None

    result = string[start_idx:end_idx]
    return result.strip() if open_str == " " and close_str == " " else result
```

test_substring.py

```
import pytest
from substring import substringBetween

@pytest.mark.parametrize("string,open_str,close_str,expected", [
    # Тесты с None
    (None, None, None, pytest.raises(TypeError)),
    (None, "*", "*", None),
    ("*", None, "*", None),
    ("*", "*", None, None),

    # Тесты с пустыми строками
    ("", "", "", ""),
    ("", "", "]", None),
    ("", "[", "]", None),

    # Тесты с пробелами
    (" yabcz ", "", "", " yabcz "),
    (" yabcz ", "y", "z", "abc"),
    (" yabczyabcz ", "y", "z", "abc"),

    # Обычные случаи
    ("wx[b]yz", "[", "]", "b"),
    ("no open", "[", "]", None),
    ("no close", "[", "]", None),
    ("multiple [a] [b]", "[", "]", "a"),

    # Тесты с пробелами как разделителями
    ("  abc  ", " ", " ", "abc"),
    ("  a  b  c  ", " ", " ", "a"),
    ("  word  ", " ", " ", "word"),
])
def test_substringBetween(string, open_str, close_str, expected):
    if isinstance(expected, str):
        result = substringBetween(string, open_str, close_str)
        assert result == expected, (
            f"For substringBetween('{string}', '{open_str}', '{close_str}')\n"
            f"Expected: '{expected}'\n"
            f"Got:      '{result}'"
        )
```

```
    elif expected is None:
        assert substringBetween(string, open_str, close_str) is None
    else:
        with expected:
            substringBetween(string, open_str, close_str)

def test_additional_cases():
    """Дополнительные тесты для проверки крайних случаев"""
    assert substringBetween("abc", "a", "c") == "b"
    assert substringBetween("  abc  ", " ", " ") == "abc"
    assert substringBetween("[[text]]", "[", "]") == "[text"
    assert substringBetween("hello world", "hello", "world") == " "
    assert substringBetween("a  b  c", "a", "c") == "  b  "
    assert substringBetween("  first  second  third  ", " ", " ") == "first"
```

Вывод: научился работать с Github API, приобрёл практические навыки написания программ для работы с REST API или GraphQL API.