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
Н.А. Антонюк
студент группы ПО11

Проверил
А. А. Крощенко ст.
преп. кафедры ИИТ,
26.04.2025 г.

Брест 2025

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

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

Код программы:

shopping.py:

```
import requests

class Cart:
    def __init__(self):
        self.item = []

    def add_item(self, name, price):
        if price < 0:
            raise ValueError("Price cannot be negative")
        self.item.append({"name": name, "price": price})

    def total(self):
        return sum(item["price"] for item in self.item)

    def apply_discount(self, discount_percent):
        if discount_percent < 0 or discount_percent > 100:
            raise ValueError("Discount must be between 0 and 100")
        total = self.total()
        return total * (1 - discount_percent / 100)

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

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

def apply_coupon(cart, coupon_code):
    if coupon_code in coupon:
        cart.apply_discount(coupon[coupon_code])
    else:
        raise ValueError("Invalid coupon")
```

test_cart.py:

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

@pytest.fixture(name="test_cart")
def empty_cart_fixture():
    """Fixture providing an empty cart for tests"""
    return Cart()

def test_add_item(test_cart):
    """Test adding item to cart"""
    test_cart.add_item("Apple", 10.0)
    assert len(test_cart.items) == 1
    assert test_cart.items[0]["name"] == "Apple"
    assert test_cart.items[0]["price"] == 10.0

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

def test_total(test_cart):
    """Test calculating total price"""
    test_cart.add_item("Apple", 10.0)
    test_cart.add_item("Banana", 5.0)
    assert test_cart.total() == 15.0
```

```
@pytest.mark.parametrize("discount,expected", [
    (0, 100.0),
    (50, 50.0),
    (100, 0.0),
])
def test_valid_discounts(test_cart, discount, expected):
    """Test valid discount values"""
    test_cart.add_item("Item", 100.0)
    assert test_cart.apply_discount(discount) == expected

@pytest.mark.parametrize("invalid_discount", [-10, 110])
def test_invalid_discounts(test_cart, invalid_discount):
    """Test invalid discount values"""
    test_cart.add_item("Item", 100.0)
    with pytest.raises(ValueError, match="Discount must be between 0 and 100"):
        test_cart.apply_discount(invalid_discount)

@patch('shopping.requests.post')
def test_log_purchase(mock_post):
    """Test logging purchase"""
    item = {"name": "Test", "price": 100}
    log_purchase(item)
    mock_post.assert_called_once_with("https://example.com/log", json=item)

def test_apply_coupon_valid(test_cart):
    """Test applying valid coupon"""
    test_cart.add_item("Item", 100.0)
    with patch.dict('shopping.coupons', {"TEST": 20}):
        apply_coupon(test_cart, "TEST")
        assert test_cart.apply_discount(20) == 80.0

def test_apply_coupon_invalid(test_cart):
    """Test applying invalid coupon"""
    with pytest.raises(ValueError, match="Invalid coupon"):
        apply_coupon(test_cart, "INVALID")
```

Рисунок с результатом работы программы:

```
PS C:\Users\Nikita\Documents\GitHub\spp_po11\reports\Antonyuk\6\src\Task1> pytest test_cart.py -v
========================================= test session starts =========================================
plugins: anyio-4.9.0
collected 11 items

test_cart.py::test_add_item PASSED                                                          [  9%]
test_cart.py::test_negative_price PASSED                                                    [ 18%]
test_cart.py::test_invalid_discounts[110] PASSED                                            [ 72%]
test_cart.py::test_log_purchase PASSED                                                      [ 81%]
test_cart.py::test_apply_coupon_valid PASSED                                                [ 90%]
test_cart.py::test_apply_coupon_invalid PASSED                                              [100%]

========================================== 11 passed in 0.34s ==========================================
PS C:\Users\Nikita\Documents\GitHub\spp_po11\reports\Antonyuk\6\src\Task1>
```

Задание 2

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

test_merge.py:

```
from SPP_TASK_2 import merge

def test_basic_merge():
    """Test basic merge functionality"""
```

```
    num1 = [1, 2, 3, 0, 0, 0]
    num2 = [2, 5, 6]
    merge(num1, 3, num2, 3)
    assert num1 == [1, 2, 2, 3, 5, 6]

def test_empty_num2():
    """Test when num2 is empty"""
    num1 = [1, 2, 3]
    num2 = []
    merge(num1, 3, num2, 0)
    assert num1 == [1, 2, 3]

def test_empty_num1():
    """Test when num1 has no elements (only zeros)"""
    num1 = [0, 0, 0]
    num2 = [1, 2, 3]
    merge(num1, 0, num2, 3)
    assert num1 == [1, 2, 3]

def test_all_same_number():
    """Test with all same numbers"""
    num1 = [1, 1, 1, 0, 0, 0]
    num2 = [1, 1, 1]
    merge(num1, 3, num2, 3)
    assert num1 == [1, 1, 1, 1, 1, 1]

def test_negative_numbers():
    """Test with negative numbers"""
    num1 = [-3, -2, -1, 0, 0, 0]
    num2 = [-2, 0, 2]
    merge(num1, 3, num2, 3)
    assert num1 == [-3, -2, -2, -1, 0, 2]

def test_different_sizes():
    """Test with different sizes"""
    num1 = [1, 2, 3, 4, 0, 0]
    num2 = [5, 6]
    merge(num1, 4, num2, 2)
    assert num1 == [1, 2, 3, 4, 5, 6]

def test_num2_larger_elements():
    """Test when all elements in num2 are larger"""
    num1 = [1, 2, 3, 0, 0, 0]
    num2 = [4, 5, 6]
    merge(num1, 3, num2, 3)
    assert num1 == [1, 2, 3, 4, 5, 6]

def test_num2_smaller_elements():
    """Test when all elements in num2 are smaller"""
    num1 = [4, 5, 6, 0, 0, 0]
    num2 = [1, 2, 3]
    merge(num1, 3, num2, 3)
    assert num1 == [1, 2, 3, 4, 5, 6]
```

Рисунок с результатом работы программы:

```
PS C:\Users\Nikita\Documents\GitHub\spp_po11\reports\Antonyuk\6\src> cd Task2; pytest test_merge.py -v
========================================= test session starts =========================================
test_merge.py::test_num2_larger_elements PASSED                                                [ 87%]
test_merge.py::test_num2_smaller_elements PASSED                                               [100%]

PS C:\Users\Nikita\Documents\GitHub\spp_po11\reports\Antonyuk\6\src\Task2>
```

test_sequence.py:

```
from SPP_TASK_1 import process_sequence
```

```
def test_normal_sequence():
    """Test with normal sequence of positive numbers"""
    result = process_sequence([1, 2, 3, 4, 5])
    assert result == (5, 1, 15, 120)

def test_negative_numbers():
    """Test with sequence containing negative numbers"""
    result = process_sequence([-1, -2, -3, -4, -5])
    assert result == (-1, -5, -15, -120)

def test_mixed_numbers():
    """Test with sequence containing both positive and negative numbers"""
    result = process_sequence([-1, 2, -3, 4, -5])
    assert result == (4, -5, -3, -120)

def test_single_element():
    """Test with sequence containing single element"""
    result = process_sequence([42])
    assert result == (42, 42, 42, 42)

def test_zero():
    """Test with sequence containing zero"""
    result = process_sequence([0, 1, 2, 3])
    assert result == (3, 0, 6, 0)

def test_empty_sequence():
    """Test with empty sequence"""
    result = process_sequence([])
    assert result == "The sequence is empty"

def test_same_numbers():
    """Test with sequence containing same numbers"""
    result = process_sequence([5, 5, 5, 5])
    assert result == (5, 5, 20, 625)
```

Рисунок с результатом работы программы:

```
Task2/test_sequence.py::test_normal_sequence PASSED                    [ 14%]
Task2/test_sequence.py::test_negative_numbers PASSED                   [ 28%]
Task2/test_sequence.py::test_mixed_numbers PASSED                      [ 42%]
Task2/test_sequence.py::test_single_element PASSED                     [ 57%]
Task2/test_sequence.py::test_zero PASSED                               [ 71%]
Task2/test_sequence.py::test_empty_sequence PASSED                     [ 85%]
Task2/test_sequence.py::test_same_numbers PASSED                       [100%]

========================================= 7 passed in 0.07s =========================================
PS C:\Users\Nikita\Documents\GitHub\spp_po11\reports\Antonyuk\6\src>
```

Задание 3.

2) Разработайте метод String repeat(String str, String separator, int repeat), который строит строку из указанного паттерна, повторённого заданное количество раз, вставляя строку-разделитель при каждом повторении.

Спецификация метода:

repeat ("e", "|", 0) = ""

repeat ("e", "|", 3) = "e|e|e"

repeat (" ABC ", ",", 2) = "ABC , ABC "

repeat (" DBE ", "", 2) = " DBEDBE "

repeat (" DBE ", ":", 1) = "DBE"
repeat ("e", -2) = ValueError
repeat ("", ":", 3) = "::"
repeat (None , "a", 1) = TypeError
repeat ("a", None , 2) = TypeErrorkeep (None , None) = TypeError
keep (None, *) = None
keep ("", *) = ""
keep (* , None) = ""
keep (* , "") = ""
keep (" hello ", "hl") = " hll "
keep (" hello ", "le") = " ell "

Код программы:

string_repeat.py:

```
def repeat(str_: str, separator: str, repeat_count: int) -> str:
    """
    Builds a string from the specified pattern, repeated the specified number of times,
    inserting a separator string at each repetition.

    Args:
        str_: The string to repeat
        separator: The string to insert between repetitions
        repeat_count: The number of times to repeat the string

    Returns:
        The resulting string after repetition

    Raises:
        ValueError: If repeat_count is negative
        TypeError: If str_ or separator is None
    """
    if str_ is None or separator is None:
        raise TypeError("String and separator cannot be None")

    if repeat_count < 0:
        raise ValueError("Repeat count cannot be negative")

    if repeat_count == 0:
        return ""

    # For empty string, handle it specially
    if not str_:
        return separator * (repeat_count - 1)

    # If only one repeat, strip spaces
    if repeat_count == 1:
        return str_.strip()

    # If separator is empty, just concatenate the string repeat_count times
    if not separator:
        return str_ * repeat_count

    # Repeat the string with separator (repeat_count - 1) times
    return (str_ + separator) * (repeat_count - 1) + str_
```

test_string_repeat.py:

```
import pytest
from string_repeat import repeat

def test_basic_repeat():
    """Test basic repeat functionality"""
    assert repeat("e", "|", 3) == "e|e|e"

def test_zero_repeat():
    """Test with zero repeat count"""
    assert repeat("e", "|", 0) == ""

def test_with_spaces():
    """Test with spaces in string"""
    assert repeat(" ABC ", ",", 2) == " ABC , ABC "

def test_empty_separator():
    """Test with empty separator"""
    assert repeat(" DBE ", "", 2) == " DBE  DBE "

def test_single_repeat():
    """Test with single repeat"""
    assert repeat(" DBE ", ":", 1) == "DBE"

def test_negative_repeat():
    """Test with negative repeat count"""
    with pytest.raises(ValueError):
        repeat("e", "|", -2)

def test_empty_string():
    """Test with empty string"""
    assert repeat("", ":", 3) == "::"

def test_none_string():
    """Test with None as string"""
    with pytest.raises(TypeError):
        repeat(None, "a", 1)

def test_none_separator():
    """Test with None as separator"""
    with pytest.raises(TypeError):
        repeat("a", None, 2)

def test_large_repeat():
    """Test with large repeat count"""
    result = repeat("a", "b", 1000)
    assert len(result) == 1999  # 1000 'a's and 999 'b's
    assert result.count("a") == 1000
    assert result.count("b") == 999
```

Рисунок с результатом работы программы:

```
Task3/test_string_repeat.py::test_negative_repeat PASSED                    [ 60%]
Task3/test_string_repeat.py::test_empty_string PASSED                       [ 70%]
Task3/test_string_repeat.py::test_none_string PASSED                        [ 80%]
Task3/test_string_repeat.py::test_none_separator PASSED                     [ 90%]
Task3/test_string_repeat.py::test_large_repeat PASSED                       [100%]

========================================= 10 passed in 0.10s =========================================
PS C:\Users\Nikita\Documents\GitHub\spp_po11\reports\Antonyuk\6\src>
```

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