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
Турабов А. В.
Проверил:
Козик И. Д.

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 ShoppingCart:
    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_percent):
        if not 0 <= discount_percent <= 100:
            raise ValueError("Invalid discount percent")
        for item in self.items:
            item["price"] *= 1 - discount_percent / 100

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

    def apply_coupon(self, coupon_code):
        coupons = {"SAVE10": 10, "HALF": 50}
        if coupon_code in coupons:
            self.apply_discount(coupons[coupon_code])
        else:
            raise ValueError("Invalid coupon")

    def remove_item(self, item_name: str) -> None:
        """Удаляет товар из корзины по имени"""
        if not item_name:
            raise ValueError("Имя товара не может быть пустым")

        indexes = [i for i, item in enumerate(self.items) if item["name"] == item_name]

        if not indexes:
            raise ValueError(f"Товар {item_name} не найден в корзине")

        del self.items[indexes[0]]
```

test_cart.py

```
from unittest.mock import patch
import pytest
from shopping import ShoppingCart

@pytest.fixture
def empty_cart():
    return ShoppingCart()

@pytest.fixture
def filled_cart():
    cart = ShoppingCart()
    cart.add_item("apple", 1.0)
    cart.add_item("banana", 0.5)
    return cart
```

```
def test_add_item(empty_cart):
    """Тест добавления товара в корзину"""
    empty_cart.add_item("apple", 1.0)
    assert len(empty_cart.items) == 1
    assert empty_cart.items[0]["name"] == "apple"
    assert empty_cart.items[0]["price"] == 1.0

def test_add_item_with_quantity(empty_cart):
    """Тест добавления товара с указанием количества"""
    empty_cart.add_item("apple", 1.0)
    empty_cart.add_item("apple", 1.0)
    assert len(empty_cart.items) == 2
    assert empty_cart.items[0]["name"] == "apple"
    assert empty_cart.items[0]["price"] == 1.0
    assert empty_cart.items[1]["name"] == "apple"
    assert empty_cart.items[1]["price"] == 1.0

def test_add_existing_item(filled_cart):
    """Тест добавления существующего товара"""
    filled_cart.add_item("apple", 1.0)
    assert len(filled_cart.items) == 3
    assert filled_cart.items[0]["name"] == "apple"
    assert filled_cart.items[0]["price"] == 1.0

def test_remove_item(filled_cart):
    """Тест удаления товара из корзины"""
    filled_cart.remove_item("apple")
    assert len(filled_cart.items) == 1
    assert filled_cart.items[0]["name"] == "banana"

def test_remove_nonexistent_item(filled_cart):
    """Тест удаления несуществующего товара"""
    with pytest.raises(ValueError):
        filled_cart.remove_item("orange")


def test_remove_item_with_empty_name(filled_cart):
    """Тест удаления товара с пустым именем"""
    with pytest.raises(ValueError):
        filled_cart.remove_item("")

def test_get_total(filled_cart):
    """Тест расчета общей стоимости"""
    assert filled_cart.total() == 1.5

def test_apply_discount(filled_cart):
    """Тест применения скидки"""
    filled_cart.apply_discount(10)
    assert filled_cart.total() == pytest.approx(1.35)

def test_apply_invalid_discount(filled_cart):
    """Тест применения недопустимой скидки"""
    with pytest.raises(ValueError):
        filled_cart.apply_discount(110)

@patch("requests.post")
def test_log_purchase(mock_post, empty_cart):
    item = {"name": "Apple", "price": 10.0}
    empty_cart.log_purchase(item)
    mock_post.assert_called_once_with("https://example.com/log", json=item)

def test_apply_coupon_valid(empty_cart):
    """Тест применения валидного купона"""
    empty_cart.add_item("Item", 100.0)
    empty_cart.apply_coupon("SAVE10")
    assert empty_cart.total() == pytest.approx(90.0)

def test_apply_coupon_invalid(empty_cart):
    """Тест применения невалидного купона"""
    empty_cart.add_item("Item", 100.0)
    with pytest.raises(ValueError, match="Invalid coupon"):
```

```
    empty_cart.apply_coupon("INVALID")
```

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.

median_program.py

```
import sys

def main():
  try:
    # Ввод данных
    count = int(input("Введите размер последовательности: "))
    if count <= 0:
      print("Ошибка: размер должен быть положительным!", file=sys.stderr)
      sys.exit(1)

    arr = []
    for i in range(count):
      num = int(input(f"Введите число {i+1}/{count}: "))
      if i > 0 and num <= arr[-1]:
        print("Ошибка: последовательность должна быть строго возрастающей!", file=sys.stderr)
        sys.exit(1)
      arr.append(num)

    # Вычисление медианы
    if count % 2 == 1:
      median = arr[count // 2]
    else:
      median = (arr[count//2 - 1] + arr[count//2]) / 2

    print(f"Медиана: {median}")

  except ValueError:
    print("Ошибка: введено не число!", file=sys.stderr)
    sys.exit(1)

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

test_median.py

```
import pytest
from io import StringIO
import sys
from unittest.mock import patch
import median_program

def run_median(inputs):
  """Запускает программу с заданными вводами и возвращает вывод"""
  with patch('sys.stdin', StringIO('\n'.join(inputs))):
    with patch('sys.stdout', new_callable=StringIO) as mock_stdout:
      try:
        median_program.main()
        return mock_stdout.getvalue().strip()
      except SystemExit:
```

```
            return None

def test_odd_length():
    inputs = ["3", "1", "2", "3"]
    output = run_median(inputs)
    assert "Медиана: 2" in output

def test_even_length():
    inputs = ["4", "1", "2", "3", "4"]
    output = run_median(inputs)
    assert "Медиана: 2.5" in output

def test_non_increasing():
    inputs = ["3", "1", "3", "2"]
    assert run_median(inputs) is None

def test_single_element():
    inputs = ["1", "5"]
    output = run_median(inputs)
    assert "Медиана: 5" in output

def test_empty_sequence():
    inputs = ["0"]
    assert run_median(inputs) is None

def test_non_numeric_input():
    inputs = ["2", "1", "abc"]
    assert run_median(inputs) is None
```

program2.py

```
def plus_one(digits):
    num = int(''.join(map(str, digits)))
    num += 1
    return [int(digit) for digit in str(num)]
```

test_plus_one.py

```
from program2 import plus_one

def test_plus_one_no_carry():
    assert plus_one([1, 2, 3]) == [1, 2, 4]

def test_plus_one_with_carry():
    assert plus_one([1, 2, 9]) == [1, 3, 0]

def test_plus_one_all_nines():
    assert plus_one([9, 9, 9]) == [1, 0, 0, 0]

def test_plus_one_single_digit():
    assert plus_one([0]) == [1]
    assert plus_one([9]) == [1, 0]
```

Задание 3

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

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"

substring.py

```
def substringBetween(str, open, close):
    # Проверка на TypeError (все аргументы None)
    if str is None and open is None and close is None:
        raise TypeError("All arguments are None")

    # Проверка на None в любом из аргументов
    if str is None or open is None or close is None:
        return None

    # Если все строки пустые
    if str == "" and open == "" and close == "":
        return ""

    # Если строка пустая, а open или close не пустые
    if str == "":
        return None

    # Поиск индексов открывающей и закрывающей подстрок
    start_idx = str.find(open)
    if start_idx == -1:
        return None

    start_idx += len(open)
    end_idx = str.find(close, start_idx)
    if end_idx == -1:
        return None

    return str[start_idx:end_idx]
```

test_substring.py

```
import pytest
from substring import substringBetween

def test_substring_between():
    # Тесты на TypeError при None во всех аргументах
    with pytest.raises(TypeError):
        substringBetween(None, None, None)
```

```
# Тесты на None при None в первом аргументе
assert substringBetween(None, "a", "b") is None

# Тесты на None при None во втором аргументе
assert substringBetween("abc", None, "b") is None

# Тесты на None при None в третьем аргументе
assert substringBetween("abc", "a", None) is None

# Тесты на пустые строки
assert substringBetween("", "", "") == ""
assert substringBetween("", "", "]") is None
assert substringBetween("", "[", "]") is None

# Тесты на подстроки
assert substringBetween(" yabcz ", "", "") == ""
assert substringBetween(" yabcz ", "y", "z") == "abc"
assert substringBetween(" yabczyabcz ", "y", "z") == "abc"
assert substringBetween("wx[b]yz", "[", "]") == "b"
```

Результатом выполнения всех программ было успешное прохождение всех тестов

Вывод: освоил приемы тестирования кода на примере использования пакета pytest