МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе No6

Специальность ПО11(о)

Выполнил

К. А. Головач, студент
группы ПО11

Проверил

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

Брест 2025

Вариант 6

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)

   - Замокайте 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
   - Замокайте словарь coupons с помощью monkeypatch или patch.dict

Код программы: shopping.py:

```
import requests
```

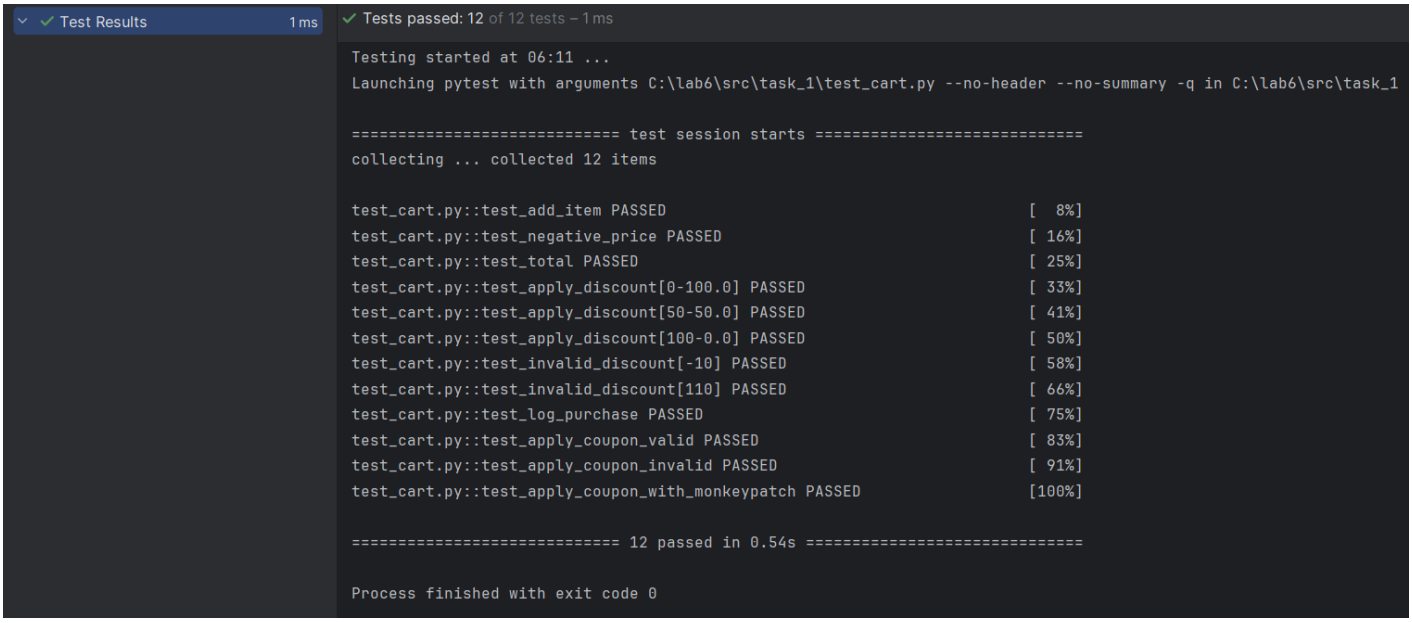
```
 class Cart:     def
__init__(self):
self.items = []
     def add_item(self, name,
price):         if price < 0:
            raise ValueError("Price cannot be negative")
self.items.append({"name": name, "price": price})
     def
total(self):
        return sum(item["price"] for item in self.items)
     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:

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

# Фикстура для пустой корзины с явным указанием имени для
использования в тестах
@pytest.fixture(name="cart")  # Теперь фикстура будет
доступна как 'cart' в тестах
def empty_cart_fixture():
    return Cart()

# Тест добавления товара
def test_add_item(cart):
    cart.add_item("Pear", 10.0)
    assert len(cart.items) == 1
    assert cart.items[0]["name"] == "Pear"
    assert cart.items[0]["price"] == 10.0

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

# Тест вычисления общей стоимости
def test_total(cart):
    cart.add_item("Pear", 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(cart, discount, expected_total):
    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(cart, invalid_discount):
    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": "Pear", "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(cart):
    cart.add_item("Item", 100.0)
    apply_coupon(cart, "SAVE10")
    assert cart.total() == 90.0  # 10% скидка

def test_apply_coupon_invalid(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(cart, monkeypatch):
    cart.add_item("Item", 100.0)
    monkeypatch.setattr("shopping.coupons", {"TEST50": 50})
    apply_coupon(cart, "TEST50")
    assert cart.total() == 50.0
```

Результаты работы программы:



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:

```
import random

def main():
    # Запрашиваем у пользователя количество чисел N
    try:
        N = int(input("Введите количество чисел (N): "))
        if N <= 0:
            print("Количество чисел должно быть положительным.")
            return
    except ValueError:
        print("Пожалуйста, введите целое число.")
        return

    # Создаем список для хранения чисел
    numbers = []

    # Запрашиваем у пользователя ввод каждого числа
    for i in range(N):
        while True:
            try:
                number = int(input(f"Введите число #{i + 1}: "))
                numbers.append(number)
                break
            except ValueError:
                print("Пожалуйста, введите корректное целое число.")

    # Перемешиваем числа в случайном порядке
    random.shuffle(numbers)
```

```
    # Выводим числа в случайном порядке
    print("Числа в случайном порядке:")
    print(*numbers, sep=", ")

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

lab2_2.py:

```
def majority_element(nums):
    # Инициализация переменных
    count = 0
    candidate = None

    # Первый проход: находим кандидата на роль элемента
большинства
    for num in nums:
        if count == 0:
            candidate = num
        count += (1 if num == candidate else -1)

    # Второй проход: проверяем, действительно ли candidate
является элементом большинства
    # (в данной задаче гарантируется, что элемент
большинства существует)
    return candidate


# Примеры использования
if __name__ == "__main__":
    # Тестовые случаи
    nums1 = [3, 2, 3]
    print(majority_element(nums1))  # Output: 3

    nums2 = [2, 2, 1, 1, 1, 2, 2]
    print(majority_element(nums2))  # Output: 2
```

Рефакторинг lab1_1.py

```
import random

def get_positive_integer(prompt):
  """Запрашивает у пользователя положительное целое число."""
  while True:
    try:
      value = int(input(prompt))
      if value > 0:
        return value
      print("Число должно быть положительным.")
    except ValueError:
      print("Пожалуйста, введите целое число.")

def get_numbers_from_user(count):
  """Запрашивает у пользователя ввод чисел."""
```

```
    numbers = []
    for i in range(count):
        while True:
            try:
                number = int(input(f"Введите число #{i + 1}: "))
                numbers.append(number)
                break
            except ValueError:
                print("Пожалуйста, введите корректное целое число.")
    return numbers

def shuffle_numbers(numbers):
    """Перемешивает числа в случайном порядке."""
    random.shuffle(numbers)
    return numbers

def main():
    # Запрашиваем у пользователя количество чисел N
    N = get_positive_integer("Введите количество чисел (N): ")

    # Запрашиваем у пользователя ввод каждого числа
    numbers = get_numbers_from_user(N)

    # Перемешиваем числа в случайном порядке
    shuffled_numbers = shuffle_numbers(numbers)

    # Выводим числа в случайном порядке
    print("Числа в случайном порядке:")
    print(*shuffled_numbers, sep=", ")

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

test_lab1_1.py:

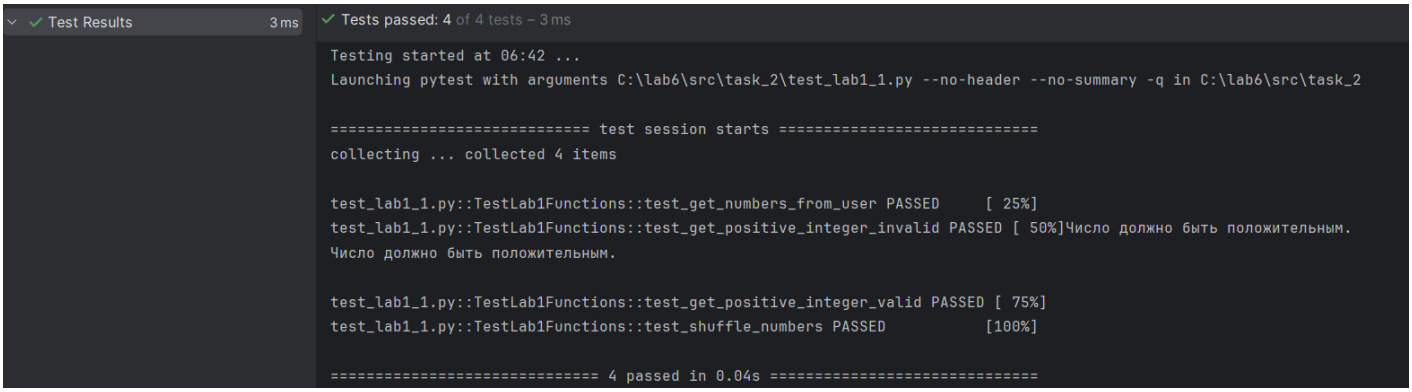
```
import unittest
from unittest.mock import patch  # Явно импортируем mock
from lab_1_1 import get_positive_integer, get_numbers_from_user, shuffle_numbers

class TestLab1Functions(unittest.TestCase):

    @patch('builtins.input', return_value="5")
    def test_get_positive_integer_valid(self, mock_input):
        """
        Тест на корректный ввод положительного числа.
        Проверяет, что функция возвращает ожидаемое значение при корректном вводе.
        """
        self.assertEqual(get_positive_integer("Введите число: "), 5)

    @patch('builtins.input', side_effect=["-3", "0", "5"])
    def test_get_positive_integer_invalid(self, mock_input):
        """
        Тест на некорректный ввод (отрицательное число или ноль).
        Проверяет, что функция продолжает запрашивать ввод до тех пор,
        пока не будет введено положительное число.
        """
```

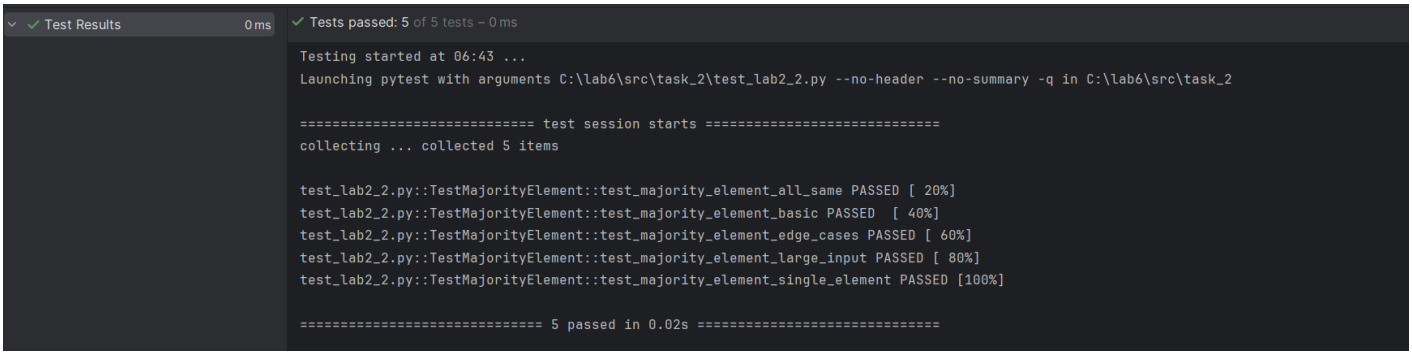
```
        self.assertEqual(get_positive_integer("Введите число: "), 5)

    @patch('builtins.input', side_effect=["1", "2", "3"])
    def test_get_numbers_from_user(self, mock_input):
        """
        Тест на корректный ввод списка чисел.
        Проверяет, что функция корректно обрабатывает последовательность ввода.
        """
        self.assertEqual(get_numbers_from_user(3), [1, 2, 3])

    def test_shuffle_numbers(self):
        """
        Тест на перемешивание чисел.
        Проверяет, что список после перемешивания содержит те же элементы,
        но в другом порядке.
        """
        numbers = [1, 2, 3, 4, 5]
        shuffled = shuffle_numbers(numbers.copy())
        self.assertNotEqual(numbers, shuffled)  # Проверяем, что порядок изменился
        self.assertCountEqual(numbers, shuffled)  # Проверяем, что элементы те же

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

test_lab2_2.py:

```
import unittest
from lab_2_2 import majority_element

class TestMajorityElement(unittest.TestCase):

    def test_majority_element_basic(self):
        """Тест на базовые случаи."""
        self.assertEqual(majority_element([3, 2, 3]), 3)
        self.assertEqual(majority_element([2, 2, 1, 1, 1, 2, 2]), 2)

    def test_majority_element_single_element(self):
        """Тест на массив с одним элементом."""
        self.assertEqual(majority_element([5]), 5)

    def test_majority_element_all_same(self):
        """Тест на массив, где все элементы одинаковые."""
        self.assertEqual(majority_element([7, 7, 7, 7, 7]), 7)

    def test_majority_element_edge_cases(self):
        """Тест на граничные случаи."""
        self.assertEqual(majority_element([1, 2, 1]), 1)
        self.assertEqual(majority_element([1, 1, 2, 2, 1]), 1)

    def test_majority_element_large_input(self):
        """Тест на большой массив."""
        nums = [1] * 5000 + [2] * 4999
        self.assertEqual(majority_element(nums), 1)

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

Результаты работы программы:





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

Напишите метод String common(String str1, String str2) сравнивающий две

строки и возвращающий наибольшую общую часть.

Спецификация метода:

common (None , None) = TypeError

common ("", "") = ""

common ("", " abc ") = ""

common (" abc ", "") = ""

common (" abc ", "abc") = " abc "

common ("ab", " abxyz ") = "ab"

common (" abcde ", " abxyz ") = "ab"

common (" abcde ", " xyz ") = ""

common (" deabc ", " abcdeabcd ") = " deabc "

common (" dfabcegt ", " rtoefabceiq ") = " fabce "

Код программы: common.py:

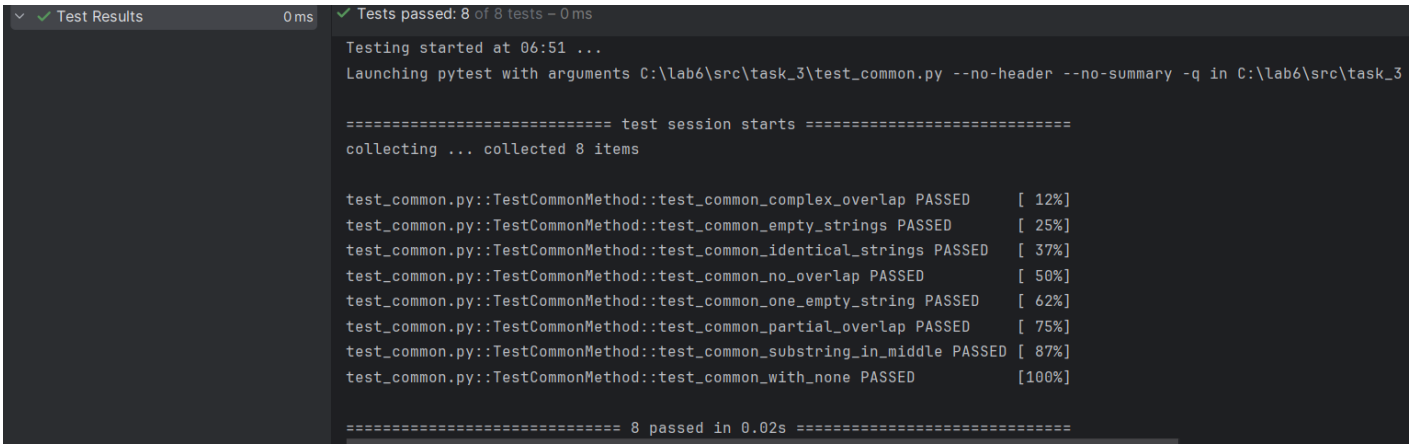
```
def common(str1, str2):
    # Проверка на None
    if str1 is None or str2 is None:
        raise TypeError("Arguments cannot be None")

    # Инициализация переменной для хранения наибольшей общей части
    longest_common = ""

    # Перебор всех возможных подстрок первой строки
    for i in range(len(str1)):
        for j in range(i + 1, len(str1) + 1):
            substring = str1[i:j]
            # Если подстрока найдена во второй строке и она длиннее текущей общей части
            if substring in str2 and len(substring) > len(longest_common):
                longest_common = substring

    return longest_common
```

test_common.py:

```
import unittest

from src.task_3.common import common


class TestCommonMethod(unittest.TestCase):

    def test_common_with_none(self):
        """Тест на передачу None в качестве аргументов."""
        with self.assertRaises(TypeError):
            common(None, None)

    def test_common_empty_strings(self):
        """Тест на пустые строки."""
        self.assertEqual(common("", ""), "")

    def test_common_one_empty_string(self):
        """Тест на одну пустую строку."""
        self.assertEqual(common("", "abc"), "")
        self.assertEqual(common("abc", ""), "")

    def test_common_identical_strings(self):
        """Тест на идентичные строки."""
        self.assertEqual(common("abc", "abc"), "abc")

    def test_common_partial_overlap(self):
        """Тест на частичное совпадение."""
        self.assertEqual(common("ab", "abxyz"), "ab")
        self.assertEqual(common("abcde", "abxyz"), "ab")

    def test_common_no_overlap(self):
        """Тест на отсутствие совпадений."""
        self.assertEqual(common("abcde", "xyz"), "")

    def test_common_substring_in_middle(self):
        """Тест на подстроку в середине."""
        self.assertEqual(common("deabc", "abcdeabcd"), "deabc")
```

```
    def test_common_complex_overlap(self):
        """Тест на сложное пересечение."""
        self.assertEqual(common("dfabcegt", "rtoefabceiq"),
"fabce")

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

Результаты работы программы:



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