Министерство образования Республики Беларусь

Министерство науки и высшего образования Российской Федерации

Межгосударственное образовательное учреждение высшего образования

Белорусско-Российский университет

Кафедра «Автоматизированные системы управления»

Отчёт

по лабораторной работе №1

«Разработка программ методом TDD»

по дисциплине «Интеграция программных модулей и компонент»

Выполнил: студент гр. ПМР-221

Прохоренко А. В.

Проверила: ст. преподаватель

Выговская Н.В.

Могилев, 2025

Цель: изучить методики гибкой (agile) разработки программного обеспечения и управления проектами на примере программирования на основе тестирования.

Задание 1: разработать библиотеку классов c покрытием Unit test для каждого метода приложения «Арифметический калькулятор». Обязательными методами являются: сложение, вычитание, умножение, деление.

Листинг кода задания 1:

Calculator.py:

import math

class Calculator:

def \_\_init\_\_(self):

pass

def add(self, x1, x2):

return x1 + x2

def multiply(self, x1, x2):

return x1 \* x2

def subtract(self, x1, x2):

return x1 - x2

def divide(self, x1, x2):

if x2 != 0:

return x1/x2

def power(self,x1,x2):

return x1\*\*x2

TestCalculator.py:

import unittest

from Calculator import Calculator

class TestCalculator(unittest.TestCase):

def setUp(self):

self.calculator = Calculator()

def test\_add(self):

self.assertEqual(self.calculator.add(4,7), 11)

def test\_subtract(self):

self.assertEqual(self.calculator.subtract(10,5), 5)

def test\_multiply(self):

self.assertEqual(self.calculator.multiply(3,7), 21)

def test\_divide(self):

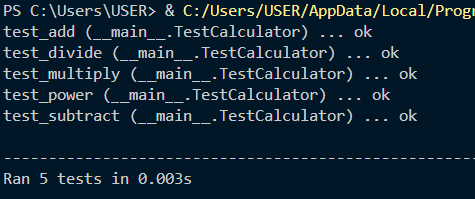
self.assertEqual(self.calculator.divide(10,2), 5)

def test\_power(self):

self.assertEqual(self.calculator.power(2,3), 8)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main(verbosity=2)Скриншот выполнения программы:



Задание 2: Применяя методику разработки на основе тестирования (testdrivendesign, сначала тесты), разработать библиотеку классов для следующего задания: Дан текст. В каждом слове каждого предложения для повторяющихся литер произвести следующую замену: повторные вхождения литер удалить, к первому вхождению литеры приписать число вхождений литеры в слово. Пример : ‘((aaabbccccddd)(eeefggghhkl)) преобразуется в ‘((a3b2 c4d3)(e3fg3 h2kl)) Листинг кода задания 2:

task.py:

*import* re

def *transform\_word*(word: str) -> str:

    """Преобразует одно слово, заменяя повторяющиеся буквы"""

    result = []

    counts = {}

*# Подсчет вхождений букв в слове*

*for* char in word:

*if* char not in counts:

            counts[char] = 0

            result.append(char)

        counts[char] += 1

*# Формируем окончательное слово с числами для повторяющихся букв*

*for* i in range(len(result)):

        char = result[i]

*if* counts[char] > 1:

            result[i] = f"{char}{counts[char]}"

            counts[char] = 0  *# После добавления числа больше не добавляем*

*return* ''.join(result)

def *transform\_text*(text: str) -> str:

    """Преобразует текст, применяя transform\_word для каждого слова в тексте"""

*# Функция для обработки фрагментов внутри скобок*

    def *process\_bracketed\_text*(match):

        inner\_text = match.group(1)

*# Разделяем текст внутри скобок на слова*

        words = inner\_text.split()

        transformed\_words = [transform\_word(word) *for* word in words]

*return* f"({' '.join(transformed\_words)})"

*# Обрабатываем все фрагменты внутри скобок*

    text = re.sub(r'\((.\*?)\)', process\_bracketed\_text, text)

*# Обрабатываем текст за пределами скобок и добавляем скобки вокруг каждого слова*

    words = text.split()

    transformed\_words = [transform\_word(word) *for* word in words]

*# Возвращаем текст с правильными скобками вокруг каждого слова*

*return* ' '.join([f"({word})" *for* word in transformed\_words])

taskTest.py: Для использования юнит-теста импортируем их из библиотеки unittest,так же импортируем нашу функцию из файла с реализацией задания. Создаем класс и в нем прописываем юнит-тесты.

import unittest  
*# test\_main.py*

*import* unittest

*from* task *import* transform\_text

class TestTransformText(unittest.TestCase):

    def *test\_single\_word*(self):

        self.assertEqual(transform\_text("aaabbccccddd"), "(a3b2c4d3)")

    def *test\_multiple\_words*(self):

        self.assertEqual(transform\_text("eeefggghhkl"), "(e3fg3h2kl)")

    def *test\_empty\_input*(self):

        self.assertEqual(transform\_text(""), "")

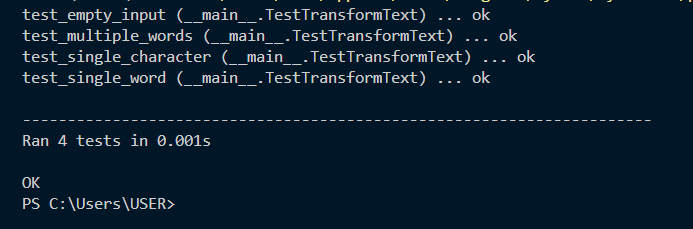
    def *test\_single\_character*(self):

        self.assertEqual(transform\_text("a"), "(a)")

*if* \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

Скриншот выполнения программы:



Вывод: в ходе лабораторной работы была изучена методология гибкой разработки программного обеспечения на примере подхода "программирование на основе тестирования" (Test-Driven Development, TDD). Был получен практический опыт применения цикла TDD: написание теста, его запуск, написание кода для прохождения теста и рефакторинг. Проведенная работа позволила оценить преимущества Agile-подходов в разработке ПО, такие как повышение качества кода, снижение количества ошибок и возможность быстрой адаптации к изменяющимся требованиям. Реализованный проект демонстрирует эффективность использования тестов как основы для разработки программных модулей.