**Московский государственный технический университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе № 4

«Шаблоны проектирования и модульное тестирование в Python.»

Выполнил: Проверил:

студент группы ИУ5-31Б преподаватель каф. ИУ5  
 Абуховский Иван Александрович Гапанюк Юрий Евгеньевич

Подпись и дата: Подпись и дата:

Москва, 2021 г.

Задание:

Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.

Вместо реализации паттерна Вы можете написать тесты для своей программы решения биквадратного уравнения. В этом случае, возможно, Вам потребуется доработать программу решения биквадратного уравнения, чтобы она была пригодна для модульного тестирования.

В модульных тестах необходимо применить следующие технологии:

TDD - фреймворк.

BDD - фреймворк.

Создание Mock-объектов.

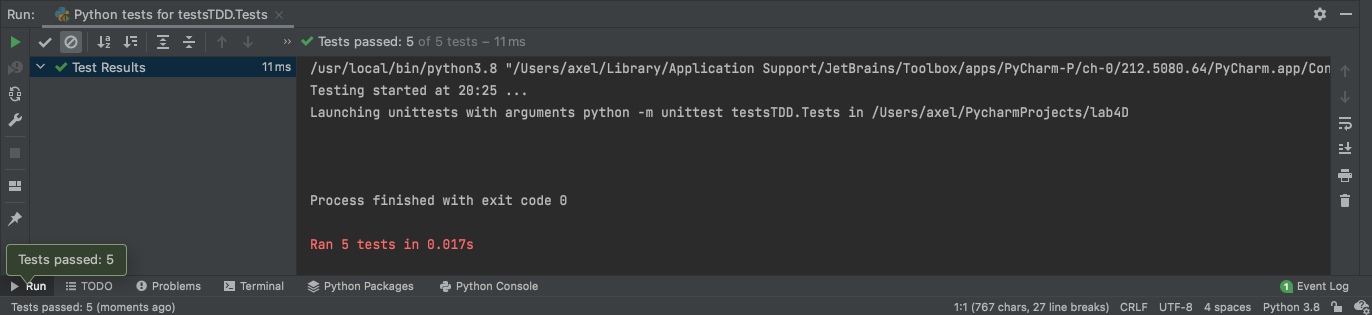
# Тексты программ

Файл main.py (модифицированная 1 лаб. работа)  
import sys  
import math  
  
  
def get\_coef(index, prompt):  
 *'''  
 Читаем коэффициент из командной строки или вводим с клавиатуры  
 Args:  
 index (int): Номер параметра в командной строке  
 prompt (str): Приглашение для ввода коэффицента  
 Returns:  
 float: Коэффициент квадратного уравнения  
 '''* try:  
 # Пробуем прочитать коэффициент из командной строки  
 coef\_str = sys.argv[index]  
 except:  
 # Вводим с клавиатуры  
 print(prompt)  
 coef\_str = input()  
  
 return coef\_str  
  
  
def checkingfloat(a, b, c):  
 *'''  
 Проверка на цифру  
 Args:  
 a (float): коэффициент А  
 b (float): коэффициент B  
 c (float): коэффициент C  
 Returns:  
 check(boolean): результат проверки  
 '''* check = True  
 try:  
 afloat = float(a)  
 bfloat = float(b)  
 cfloat = float(c)  
 except ValueError:  
 print('Ошибка!')  
 check = False  
 return check  
  
  
def get\_roots(a, b, c):  
 *'''  
 Получение корней  
 Args:  
 a (float): коэффициент А  
 b (float): коэффициент B  
 c (float): коэффициент C  
 Returns:  
 list[float]: Список корней  
 '''* result = []  
 D = b \* b - 4 \* a \* c  
 if D == 0.0:  
 doubleroot = -b / (2.0 \* a)  
 if (doubleroot > 0):  
 for i in range(2):  
 root = (-1) \*\* i \* math.sqrt(doubleroot)  
 result.append(root)  
 elif (doubleroot == 0):  
 root = 0;  
 result.append(root)  
 elif D > 0.0:  
 sqD = math.sqrt(D)  
 for i in range(2):  
 doubleroot = (-b + (-1) \*\* i \* sqD) / (2.0 \* a)  
 if (doubleroot > 0):  
 for j in range(2):  
 root = (-1) \*\* j \* math.sqrt(doubleroot)  
 result.append(root)  
 elif (doubleroot == 0):  
 root = 0;  
 result.append(root)  
  
 return result  
  
  
def main():  
 while True:  
 while True:  
 a = get\_coef(1, 'Введите коэффициент А:')  
 b = get\_coef(2, 'Введите коэффициент B:')  
 c = get\_coef(3, 'Введите коэффициент C:')  
 if (a == '0') and (b == '0') and ((c != '0') or (c == '0')):  
 print('Ошибка!')  
 else:  
 break  
 if (checkingfloat(a, b, c)):  
 break  
 a = float(a)  
 b = float(b)  
 c = float(c)  
 if (a == 0) and (c == 0):  
 print('Один корень: 0')  
 sys.exit()  
  
 # Вычисление корней  
 roots = get\_roots(a, b, c)  
 # Вывод корней  
 len\_roots = len(roots)  
 if len\_roots == 0:  
 print('Нет корней')  
 elif len\_roots == 1:  
 print('Один корень: {}'.format(roots[0]))  
 elif len\_roots == 2:  
 print('Два корня: {} и {}'.format(roots[0], roots[1]))  
 elif len\_roots == 3:  
 print('Три корня: {} и {} и {}'.format(roots[0], roots[1], roots[2]))  
 elif len\_roots == 4:  
 print('Четыре корня: {}, {} и {}, {}'.format(roots[0], roots[1], roots[2], roots[3]))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

(Скриншоты см. в первом отчёте)

Файл testTDD.py

import main  
import unittest  
from unittest import mock  
  
  
class Tests(unittest.TestCase):  
  
 def test\_chetire\_kornya(self):  
 roots = main.get\_roots(4, -5, 1)  
 self.assertEqual([1, -1, 0.5, -0.5], roots)  
  
 def test\_tri\_kornya(self):  
 roots = main.get\_roots(-1, 4, 0)  
 self.assertEqual([0, 2, -2], roots)  
  
 def test\_dva\_kornya(self):  
 roots = main.get\_roots(-2, 0, 10)  
 self.assertAlmostEqual(1.495, roots[0], 3)  
 self.assertAlmostEqual(-1.495, roots[1], 3)  
  
 def test\_nol\_korney(self):  
 roots = main.get\_roots(1, 2, 3)  
 self.assertEqual([], roots)  
  
 @mock.patch('main.get\_roots', return\_value=[322])  
 def test\_mock(self, get\_roots):  
 self.assertEqual(main.get\_roots(1, 2, 3), [322])



Файл steps.py

# -\*- coding: utf-8 -\*-  
from main import \*  
from behave import given, when, then  
  
  
@given(u'I have {a}\*x^4 + {b}\*x^2 + {c} = 0')  
def step\_impl(context, a: float, b: float, c: float):  
 context.a = float(a)  
 context.b = float(b)  
 context.c = float(c)  
  
  
@when(u'I solve this equation')  
def step\_impl(context):  
 context.roots = get\_roots(context.a, context.b, context.c)  
  
  
@then(u'I expect to get four roots: {x1}, {x2}, {x3}, {x4}')  
def step\_impl(context, x1: float, x2: float, x3: float, x4: float):  
 result = [float(x1), float(x2), float(x3), float(x4)]  
 assert context.roots == result

Файл bdd.feature

Feature: Four roots  
 Scenario: roots 4 -5 1  
 Given I have 4\*x^4 + -5\*x^2 + 1 = 0  
 When I solve this equation  
 Then I expect to get four roots: **1.0**, **-1.0**, **0.5**, **-0.5**

