**Министерство науки и высшего образования Российской Федерации** **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования** **«Московский государственный технический университет** **имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»**

**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:

студент группы ИУ5-33Б

Требуков Д.А.

Проверил:

Канев А.И.

2021 г.

**Задание:**

1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. 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.
3. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.

**Текст программы:**

Файл get\_roots.py

# -\*- coding: utf-8 -\*-  
import math  
  
def get\_roots(a, b, c):  
 *'''  
 Вычисление корней квадратного уравнения  
 Args:  
 a (float): коэффициент А  
 b (float): коэффициент B  
 c (float): коэффициент C  
 Returns:ƒ  
 list[float]: Список корней  
 '''* result = []  
 D = b \* b - 4 \* a \* c  
 if D == 0.0:  
 root1 = -b / (2.0 \* a)  
 if root1 > 0:  
 root1 = math.sqrt(root1)  
 root2 = -root1  
 result.append(root1)  
 result.append(root2)  
 if root1 == -0.0:  
 root1 = 0.0  
 result.append(root1)  
 elif D > 0.0:  
 sqD = math.sqrt(D)  
 root1 = (-b + sqD) / (2.0 \* a)  
 if root1 > 0:  
 root1 = math.sqrt(root1)  
 root2 = -root1  
 result.append(root1)  
 result.append(root2)  
 if root1 == -0.0:  
 root1 = 0.0  
 result.append(root1)  
 root3 = (-b - sqD) / (2.0 \* a)  
 if root3 > 0 and math.sqrt(root3) != -root1:  
 root3 = math.sqrt(root3)  
 root4 = -root3  
 result.append(root3)  
 result.append(root4)  
 if root3 == -0.0:  
 root3 = 0.0  
 result.append(root3)  
 return result

Файл main.py

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  
 coefStr = sys.argv[index]  
  
 if coefStr == '0' and index == 1:  
 raise  
  
 coef = float(coefStr)  
 return coef  
 except:  
 # Вводим с клавиатуры  
 while True:  
 print(prompt)  
 coefStr = input()  
 for i in range(len(coefStr)):  
 if not(coefStr[i].isdigit() or coefStr[i] == '-'):  
 break  
 else:  
 break  
 # Переводим строку в действительное число  
 coef = float(coefStr)  
 return coef  
  
  
def get\_roots(a, b, c):  
 *'''  
 Вычисление корней квадратного уравнения  
 Args:  
 a (float): коэффициент А  
 b (float): коэффициент B  
 c (float): коэффициент C  
 Returns:ƒ  
 list[float]: Список корней  
 '''* result = []  
 D = b \* b - 4 \* a \* c  
 if D == 0.0:  
 root1 = -b / (2.0 \* a)  
 if root1 > 0:  
 root1 = math.sqrt(root1)  
 root2 = -root1  
 result.append(root1)  
 result.append(root2)  
 if root1 == -0.0:  
 root1 = 0.0  
 result.append(root1)  
 elif D > 0.0:  
 sqD = math.sqrt(D)  
 root1 = (-b + sqD) / (2.0 \* a)  
 if root1 > 0:  
 root1 = math.sqrt(root1)  
 root2 = -root1  
 result.append(root1)  
 result.append(root2)  
 if root1 == -0.0:  
 root1 = 0.0  
 result.append(root1)  
 root3 = (-b - sqD) / (2.0 \* a)  
 if root3 > 0 and math.sqrt(root3) != -root1:  
 root3 = math.sqrt(root3)  
 root4 = -root3  
 result.append(root3)  
 result.append(root4)  
 if root3 == -0.0:  
 root3 = 0.0  
 result.append(root3)  
 return result  
  
  
def main():  
 print()  
 '''  
 Основная функция  
 '''  
 a = get\_coef(1, 'Введите коэффициент А:', )  
  
 while a == 0:  
 print('Коэфициент А не может быть равен 0')  
 a = get\_coef(1, 'Введите коэффициент А:')  
  
  
 b = get\_coef(2, 'Введите коэффициент B:')  
 c = get\_coef(3, 'Введите коэффициент C:')  
 # 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 {}'.format(roots[0], roots[1], roots[2]))  
 elif len\_roots == 4:  
 print('Четыре корня: {}, {}, {} и {}'.format(roots[0], roots[1], roots[2], roots[3]))  
  
  
# Если сценарий запущен из командной строки  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()  
  
# Пример запуска  
# qr.py 1 0 -4

Файл mock.py

from unittest import TestCase  
from unittest.mock import patch  
import main  
  
  
class TestRoots(TestCase):  
 @patch('main.get\_roots', return\_value=[])  
 def test\_no\_roots(self, get\_roots):  
 self.assertEqual(main.get\_roots(1, 2, 3), get\_roots(1, 2, 3))  
  
 @patch('main.get\_roots', return\_value=[0])  
 def test\_one\_root(self, get\_roots):  
 expected\_value = get\_roots(1, 1, 0)  
  
 cur\_value = main.get\_roots(1, 1, 0)  
 self.assertEqual(len(cur\_value), 1)  
 self.assertEqual(expected\_value[0] == cur\_value[0], True)  
  
 @patch('main.get\_roots', return\_value=[-2, 2])  
 def test\_two\_roots(self, get\_roots):  
 expected\_len = 2  
 expected\_value = get\_roots(1, -2, -8)  
  
 cur\_value = main.get\_roots(1, -2, -8)  
 self.assertEqual(len(cur\_value), expected\_len)  
 self.assertEqual(expected\_value[0] in cur\_value, True)  
 self.assertEqual(expected\_value[1] in cur\_value, True)  
  
 @patch('main.get\_roots', return\_value=[-1, 0, 1])  
 def test\_three\_roots(self, get\_roots):  
 expected\_len = 3  
 expected\_value = get\_roots(1, -1, 0)  
  
 cur\_value = main.get\_roots(1, -1, 0)  
 self.assertEqual(len(cur\_value), expected\_len)  
 self.assertEqual(expected\_value[0] in cur\_value, True)  
 self.assertEqual(expected\_value[1] in cur\_value, True)  
 self.assertEqual(expected\_value[2] in cur\_value, True)  
  
 @patch('main.get\_roots', return\_value=[-1, 1, 0.5, -0.5])  
 def test\_four\_roots(self, get\_roots):  
 expected\_len = 4  
 expected\_value = get\_roots(4, -5, 1)  
  
 cur\_value = main.get\_roots(4, -5, 1)  
 self.assertEqual(expected\_value[0] in cur\_value, True)  
 self.assertEqual(expected\_value[1] in cur\_value, True)  
 self.assertEqual(expected\_value[2] in cur\_value, True)  
 self.assertEqual(expected\_value[3] in cur\_value, True)

Файл tdd.py

from unittest import TestCase  
from unittest.mock import patch  
import main  
  
  
class TestRoots(TestCase):  
 @patch('main.get\_roots', return\_value=[])  
 def test\_no\_roots(self, get\_roots):  
 self.assertEqual(main.get\_roots(1, 2, 3), get\_roots(1, 2, 3))  
  
 @patch('main.get\_roots', return\_value=[0])  
 def test\_one\_root(self, get\_roots):  
 expected\_value = get\_roots(1, 1, 0)  
  
 cur\_value = main.get\_roots(1, 1, 0)  
 self.assertEqual(len(cur\_value), 1)  
 self.assertEqual(expected\_value[0] == cur\_value[0], True)  
  
 @patch('main.get\_roots', return\_value=[-2, 2])  
 def test\_two\_roots(self, get\_roots):  
 expected\_len = 2  
 expected\_value = get\_roots(1, -2, -8)  
  
 cur\_value = main.get\_roots(1, -2, -8)  
 self.assertEqual(len(cur\_value), expected\_len)  
 self.assertEqual(expected\_value[0] in cur\_value, True)  
 self.assertEqual(expected\_value[1] in cur\_value, True)  
  
 @patch('main.get\_roots', return\_value=[-1, 0, 1])  
 def test\_three\_roots(self, get\_roots):  
 expected\_len = 3  
 expected\_value = get\_roots(1, -1, 0)  
  
 cur\_value = main.get\_roots(1, -1, 0)  
 self.assertEqual(len(cur\_value), expected\_len)  
 self.assertEqual(expected\_value[0] in cur\_value, True)  
 self.assertEqual(expected\_value[1] in cur\_value, True)  
 self.assertEqual(expected\_value[2] in cur\_value, True)  
  
 @patch('main.get\_roots', return\_value=[-1, 1, 0.5, -0.5])  
 def test\_four\_roots(self, get\_roots):  
 expected\_len = 4  
 expected\_value = get\_roots(4, -5, 1)  
  
 cur\_value = main.get\_roots(4, -5, 1)  
 self.assertEqual(expected\_value[0] in cur\_value, True)  
 self.assertEqual(expected\_value[1] in cur\_value, True)  
 self.assertEqual(expected\_value[2] in cur\_value, True)  
 self.assertEqual(expected\_value[3] in cur\_value, True)

Папка feature, файл get\_roots.feature

Feature: get\_roots function  
  
 Scenario: test no roots  
 Given A = 1, B = 2, C = 3  
 When get\_roots run  
 Then roots array is empty  
  
 Scenario: test one root  
 Given A = 1, B = 1, C = 0  
 When get\_roots execute1  
 Then roots is [0]  
  
 Scenario: test two root  
 Given A = 1, B = -2, C = -8  
 When get\_roots execute2  
 Then roots are [-2, 2]  
  
 Scenario: test three root  
 Given A = 1, B = -1, C = 0  
 When get\_roots execute3  
 Then roots are [-1, 0, 1]  
  
 Scenario: test four root  
 Given A = 4, B = -5, C = 1  
 When get\_roots execute4  
 Then roots are [-1, 1, 0.5, -0,5]

Папка feature, папка steps,файл test\_no\_roots.py

from behave import \*  
from get\_roots import get\_roots  
  
@given('A = 1, B = 2, C = 3')  
def step\_impl(context):  
 context.A = 1  
 context.B = 2  
 context.C = 3  
 pass  
  
@when('get\_roots run')  
def step\_impl(context):  
 context.array\_len = len(get\_roots(context.A, context.B, context.C))  
 pass  
  
@then('roots array is empty')  
def step\_impl(context):  
 assert context.array\_len is 0

Папка feature, папка steps,файл test\_one\_roots.py

from behave import \*  
from get\_roots import get\_roots  
  
@given('A = 1, B = 1, C = 0')  
def step\_impl(context):  
 context.A = 1  
 context.B = 1  
 context.C = 0  
 pass  
  
@when('get\_roots execute1')  
def step\_impl(context):  
 context.roots = get\_roots(context.A, context.B, context.C)  
 pass  
  
@then('roots is [0]')  
def step\_impl(context):  
 assert len(context.roots) == 1  
 assert context.roots[0] == 0

Папка feature, папка steps,файл test\_two\_roots.py

from behave import \*  
from get\_roots import get\_roots  
  
@given('A = 1, B = -2, C = -8')  
def step\_impl(context):  
 context.A = 1  
 context.B = -2  
 context.C = -8  
 pass  
  
@when('get\_roots execute2')  
def step\_impl(context):  
 context.roots = get\_roots(context.A, context.B, context.C)  
 pass  
  
@then('roots are [-2, 2]')  
def step\_impl(context):  
 assert len(context.roots) == 2  
 assert -2 in context.roots  
 assert 2 in context.roots

Папка feature, папка steps,файл test\_three\_roots.py

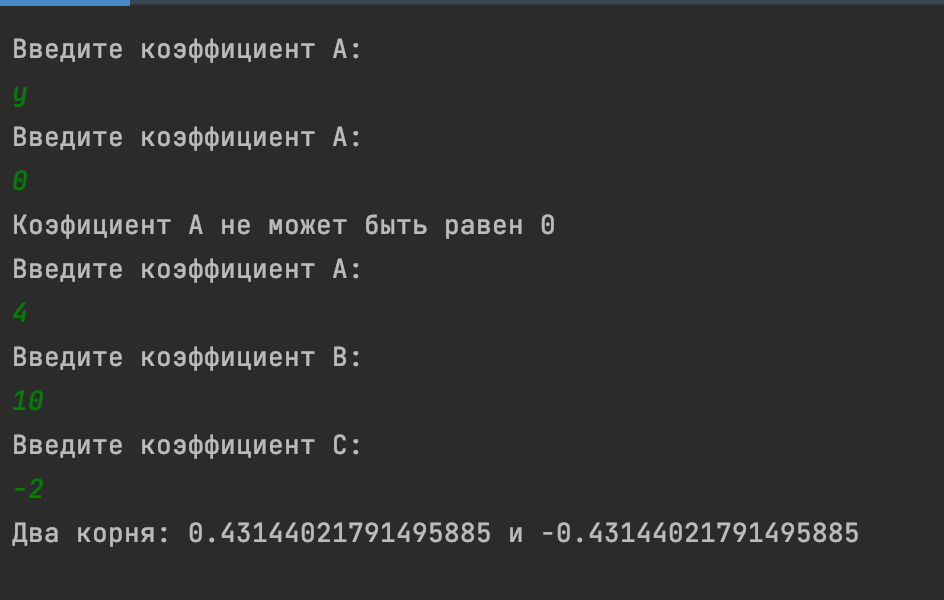
from behave import \*  
from get\_roots import get\_roots  
  
@given('A = 1, B = -1, C = 0')  
def step\_impl(context):  
 context.A = 1  
 context.B = -1  
 context.C = 0  
 pass  
  
@when('get\_roots execute3')  
def step\_impl(context):  
 context.roots = get\_roots(context.A, context.B, context.C)  
 pass  
  
@then('roots are [-1, 0, 1]')  
def step\_impl(context):  
 assert len(context.roots) == 3  
 assert -1 in context.roots  
 assert 0 in context.roots  
 assert 1 in context.roots

Папка feature, папка steps,файл test\_four\_roots.py

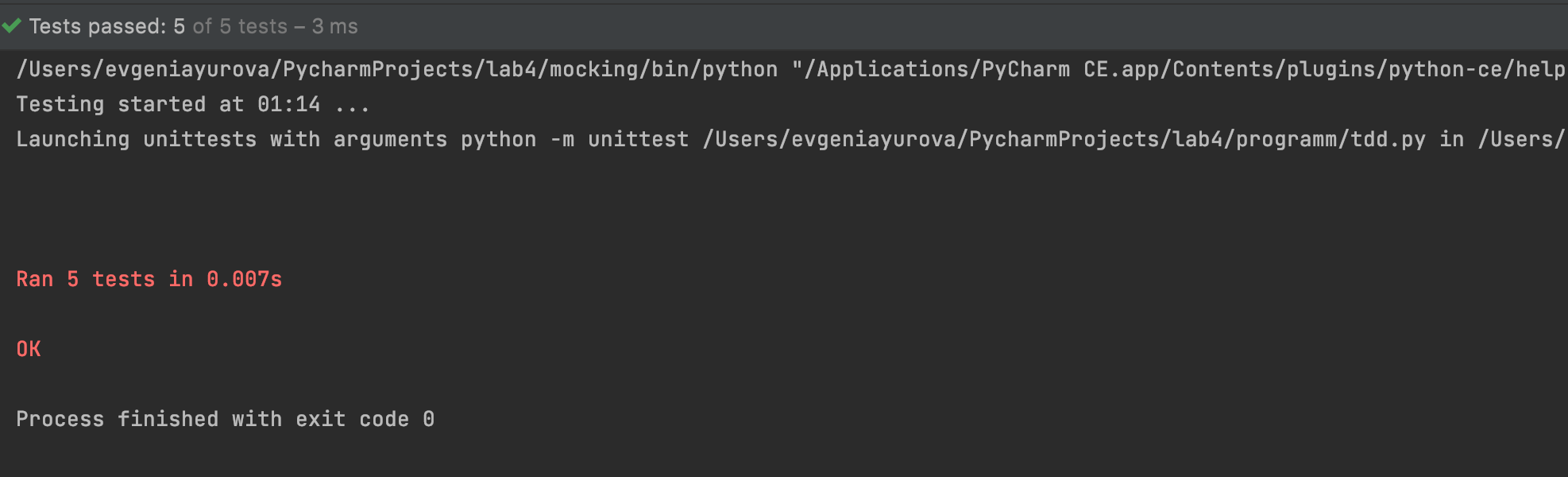
from get\_roots import get\_roots  
  
@given('A = 4, B = -5, C = 1')  
def step\_impl(context):  
 context.A = 4  
 context.B = -5  
 context.C = 1  
 pass  
  
@when('get\_roots execute4')  
def step\_impl(context):  
 context.roots = get\_roots(context.A, context.B, context.C)  
 pass  
  
@then('roots are [-1, 1, 0.5, -0,5]')  
def step\_impl(context):  
 assert len(context.roots) == 4  
 assert -1 in context.roots  
 assert 1 in context.roots  
 assert 0.5 in context.roots  
 assert -0.5 in context.roots

**Экранные формы с примерами выполнения программы:**

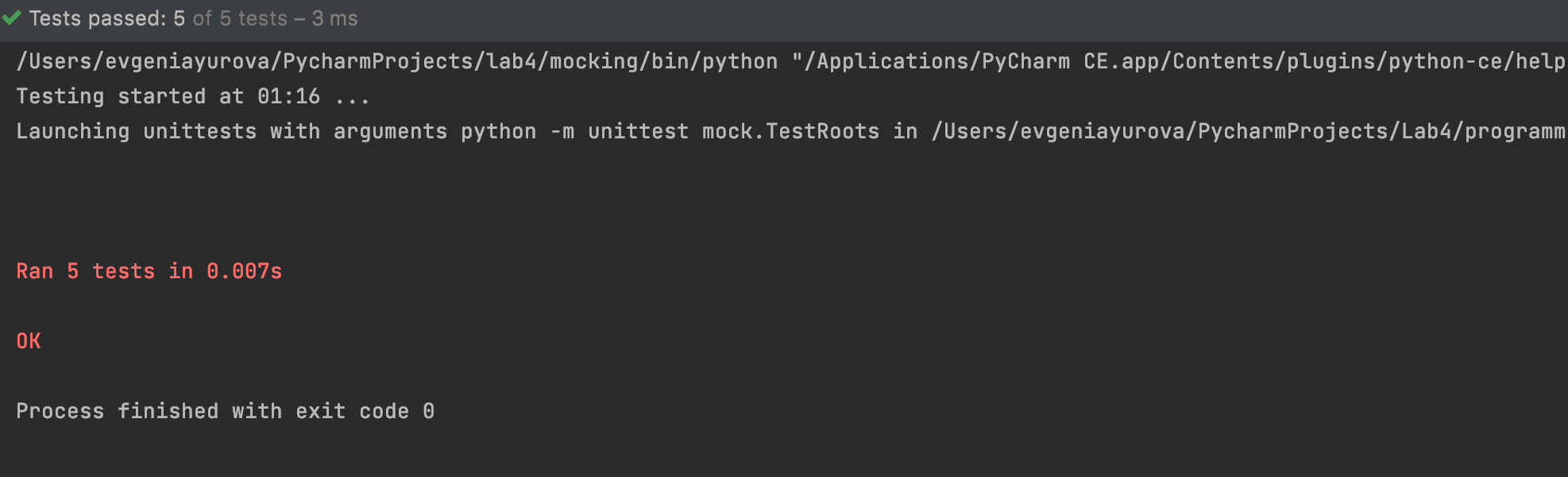
Результаты работы main.py:

****

Результаты работы tdd.py:

****

Результаты работы mock.py:

****

Результаты работы BDD тестов:

