|  |  |
| --- | --- |
|  | **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

Выполнил:

студент группы ИУ5-34Б

Григорян Александр Проверил:

Гапанюк Ю.Е.

2021 г*.*

**Описание**

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.

**Текст программы**

**main.py**

*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:  
 list[float]: Список корней  
 """  
 result = []  
 D = *b* \* *b* - 4 \* *a* \* *c  
  
 if* D == 0:  
 root1 = (-*b* + math.sqrt(D)) / (2.0 \* *a*)  
 *if* root1 > 0:  
 t1 = -math.sqrt(root1)  
 result.append(t1)  
 t2 = math.sqrt(root1)  
 result.append(t2)  
 *elif* root1 == 0:  
 t1 = 0  
 result.append(t1)  
 *elif* D > 0:  
 root1 = (-*b* + math.sqrt(D)) / (2.0 \* *a*)  
 root2 = (-*b* - math.sqrt(D)) / (2.0 \* *a*)  
  
 *if* root1 > 0:  
 t1= -math.sqrt(root1)  
 t2 = math.sqrt(root1)  
 result.append(t1)  
 result.append(t2)  
 *elif* root1 == 0:  
 t1 = 0  
 result.append(t1)  
  
 *if* root2 > 0:  
 t3 = -math.sqrt(root2)  
 t4 = math.sqrt(root2)  
 result.append(t3)  
 result.append(t4)  
 *elif* root1 == 0:  
 t1 = 0  
 result.append(t1)  
 *return* result  
  
l = get\_roots(1, -2, 1)  
print(l)

**test\_file.py**

*from* main *import* \*  
*import* unittest  
*from* unittest.mock *import* patch  
  
*class* TestRoots(unittest.TestCase):  
 *def* test\_get\_roots(self):  
 a = 1  
 b = -2  
 c = 1  
 res = get\_roots(a, b, c)  
 expected\_result = [-1, 1]  
 self.assertEqual(res, expected\_result)  
  
*class* TestRootsMock(unittest.TestCase):  
 @patch('main.get\_roots', return\_value=[-1, 1])  
 *def* test\_get\_roots\_mock(self, *get\_roots*):  
 res = [-1, 1]  
 self.assertEqual(get\_roots(2, 3, 4), res)  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**radish/step.py**

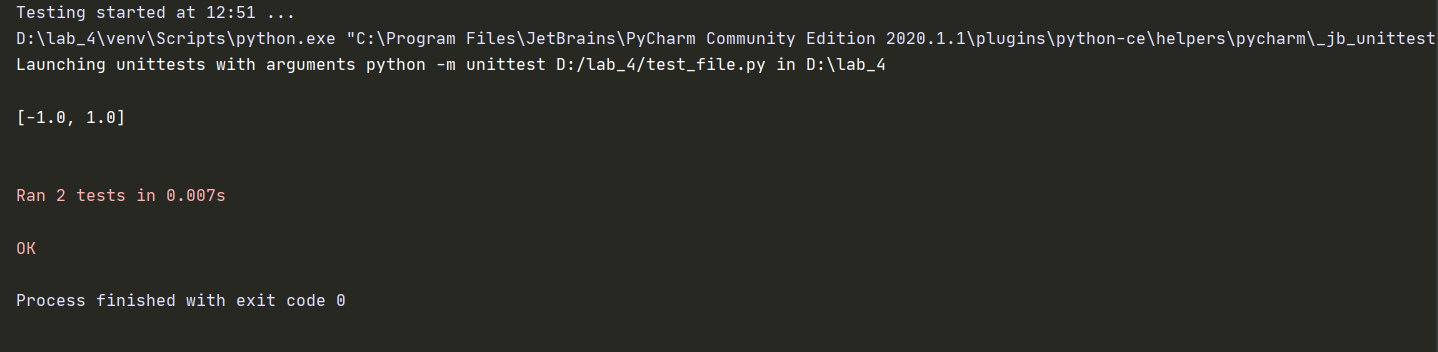
# -\*- coding: utf-8 -\*-  
*from* radish *import* given, when, then  
*import* math  
  
@given('I have the numbers {number1:g}, {number2:g} and {number3:g}')  
*def* have\_numbers(*step*, *number1*, *number2*, *number3*):  
 *step*.context.number1 = *number1  
 step*.context.number2 = *number2  
 step*.context.number3 = *number3*@when("I perform the function")  
*def* solve(*step*):  
 result\_f = []  
 D = *step*.context.number2 \* *step*.context.number2 - 4 \* *step*.context.number1 \* *step*.context.number3  
  
 *if* D == 0:  
 root1 = (-*step*.context.number2 + math.sqrt(D)) / (2.0 \* *step*.context.number1)  
 *if* root1 > 0:  
 t1 = -math.sqrt(root1)  
 result\_f.append(t1)  
 t2 = math.sqrt(root1)  
 result\_f.append(t2)  
 *elif* root1 == 0:  
 t1 = 0  
 result\_f.append(t1)  
 *elif* D > 0:  
 root1 = (-*step*.context.number2 + math.sqrt(D)) / (2.0 \* *step*.context.number1)  
 root2 = (-*step*.context.number2 - math.sqrt(D)) / (2.0 \* *step*.context.number1)  
  
 *if* root1 > 0:  
 t1 = -math.sqrt(root1)  
 t2 = math.sqrt(root1)  
 result\_f.append(t1)  
 result\_f.append(t2)  
 *elif* root1 == 0:  
 t1 = 0  
 result\_f.append(t1)  
  
 *if* root2 > 0:  
 t3 = -math.sqrt(root2)  
 t4 = math.sqrt(root2)  
 result\_f.append(t3)  
 result\_f.append(t4)  
 *elif* root1 == 0:  
 t1 = 0  
 result\_f.append(t1)  
  
 *step*.context.result = result\_f  
  
@then("I expect the result to be {result1:g}, {result2:g}")  
*def* expect\_result(*step*, *result1*, *result2*):  
 res = []  
 res.append(*result1*)  
 res.append(*result2*)  
 *assert step*.context.result == res

**features/function.feature**

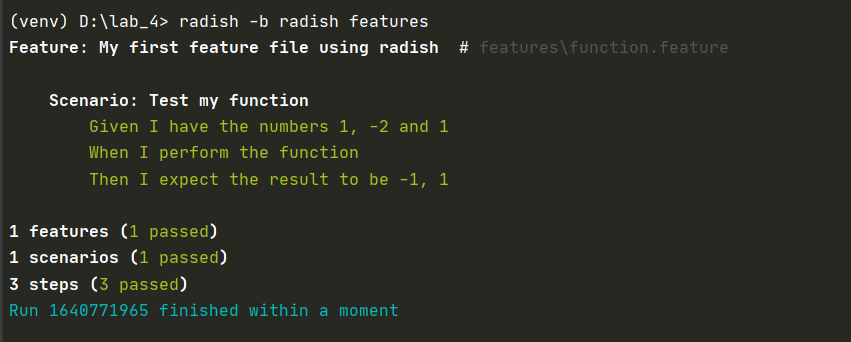
Feature: My first feature file using radish  
 Scenario: Test my function  
 Given I have the numbers 1, -2 and 1  
 When I perform the function  
 Then I expect the result to be -1, 1

**Экранные формы с примерами выполнения программы**

**test\_file.py**

****

**step.py, function.feature**

****