|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 **СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ**

**Отчет по лабораторной работе №5**

**по дисциплине Базовые компоненты интернет технологии**

#### Тема работы: "[Модульное тестирование в Python](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_test)"

**Выполнил:**

Студент группы ИУ5Ц-54Б

Перевощиков Н.Д. 23.11.22 г.

(дата, подпись)

**Проверил:**

Преподаватель

Канев А.И.

(дата, подпись)

Москва, 2022

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ ОТЧЕТА**

[1. Цель лабораторной работы 3](#_Toc115088672)

[2. Описание задания 3](#_Toc115088673)

[3. Текст программы 4](#_Toc115088674)

[4. Результат 12](#_Toc115088675)

## Цель лабораторной работы

Изучение возможностей модульного тестирования в языке Python.

## Описание задания

1. Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4.
2. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.
3. Разработайте модульные тесты. В модульных тестах необходимо применить следующие технологии:

* TDD - фреймворк (не менее 3 тестов).
* BDD - фреймворк (не менее 3 тестов).
* Создание Mock-объектов (необязательное дополнительное задание).

## Текст программы

**Для Unittest**

**Main.py**

from function.get\_roots import get\_roots  
# Основная программа  
def main():  
 while True:  
 try:  
 a = get\_coef(1, 'Введите коэффициент А:')  
 b = get\_coef(2, 'Введите коэффициент B:')  
 c = get\_coef(3, 'Введите коэффициент C:')  
  
 if (a == None or b == None or c == None):  
 print()  
 print('Ошибка заполнения!')  
 break  
  
 # 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(round(roots[0], 2)))  
 elif len\_roots == 2:  
 print('Два корня: {} и {}'.format(round(roots[0], 2), round(roots[1], 2)))  
 elif len\_roots == 3 and roots[0] == 0.0:  
 print('Три корня: {} и {} и {}'.format(round(roots[0], 2), round(roots[1], 2), round(roots[2], 2)))  
 elif len\_roots == 3:  
 print('Два корня: {} и {}'.format(round(roots[1], 2), round(roots[2], 2)))  
 elif len\_roots == 4:  
 print(  
 'Четыре корня: {} и {} и {} и {}'.format(round(roots[0], 2), round(roots[1], 2), round(roots[2], 2),  
 round(roots[3], 2)))  
 else:  
 print('Ошибка! Корней нет!')  
 break  
  
 except ArithmeticError:  
 print('Ошибка! Коэффициент a должен быть натуральным числом!')  
 break  
  
# Если сценарий запущен из командной строки  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

В папке function:

**Init.py**

# print('Подключение функционального файла')

**Get\_coef.py**

import sys  
  
def get\_coef(index, prompt):  
 if (index > 3 or index < 1):  
 print('Индекс вне диапазона. Индексом должен быть 1, 2 и 3')  
 return None  
  
 try:  
 # Пробуем прочитать коэффициент из командной строки  
 coef\_str = sys.argv[index]  
  
 if(coef\_str[0] == '-'):  
 coef\_str = sys.argv[index].replace('-','')  
 # print('coef\_str\_if', coef\_str)  
 else:  
 coef\_str = sys.argv[index]  
 # print('coef\_str\_else', coef\_str)  
  
 if(coef\_str.isdigit() == True):  
 coef\_str = sys.argv[index]  
 # print(f'{coef\_str} явл-ется числом', )  
 else:  
 print('Ошибка! Введите натуральное число!')  
 return None  
  
 except:  
 while True:  
 # Вводим с клавиатуры  
 print(prompt)  
 coef\_str = input()  
  
 # Проверка, есть ли минус числа и нулевой коэффициент?  
 if (coef\_str[0] == '-'):  
 coef\_str\_buff = coef\_str.replace('-', '')  
 if (coef\_str\_buff.isdigit()):  
 break  
 if (coef\_str.isdigit()):  
 break  
 else:  
 print("Ошибка! Введите натуральное число!")  
 return None  
  
 # Переводим строку в действительное число  
 coef = float(coef\_str)  
 return coef

**Get\_goef\_test.py**

import sys  
  
def get\_coef\_test\_no\_cmd(index, value):  
 if (index > 3 or index < 1):  
 print('Индекс вне диапазона. Индексом должен быть 1, 2 и 3')  
 return None  
  
 while True:  
 # Вводим с клавиатуры  
 coef\_str = str(value)  
 # Проверка, есть ли минус числа и нулевой коэффициент?  
  
 if (coef\_str[0] == '-'):  
 coef\_str\_buff = coef\_str.replace('-', '')  
 if (coef\_str\_buff.isdigit()):  
 break  
 if (coef\_str.isdigit()):  
 break  
 else:  
 print("Ошибка! Введите натуральное число!")  
 return None  
  
 # Переводим строку в действительное число  
 coef = float(coef\_str)  
 return coef  
  
  
def get\_coef\_test\_with\_cmd(index, value):  
 if(index > 3 or index < 1):  
 print('Индекс вне диапазона. Индексом должен быть 1, 2 и 3')  
 return None  
  
 # Пробуем прочитать коэффициент из командной строки  
 coef\_str = str(value)  
  
 if (coef\_str[0] == '-'):  
 coef\_str = coef\_str.replace('-', '')  
 # print('coef\_str\_if', coef\_str)  
 else:  
 coef\_str = str(value)  
 # print('coef\_str\_else', coef\_str)  
  
 if (coef\_str.isdigit() == True):  
 coef\_str = str(value)  
 # print(f'{coef\_str} явл-ется числом', )  
 else:  
 print('Ошибка! Введите натуральное число!')  
 return None  
  
 coef = float(coef\_str)  
 return coef

**Get\_roots.py**

import math  
# Функция высчисления дискримината  
# Получение корней  
def get\_roots(a, b, c):  
 result = []  
 D = b \* b - 4 \* a \* c  
  
 # Если дискриминат равен нулю, то корень может быть только один  
 if D == 0.0:  
 root = -b / (2.0 \* a)  
 # result.append(root)  
 if (root > 0.0):  
 root1 = math.sqrt(root)  
 result.append(root1)  
 result.append(-root1)  
  
 # Если дискриминат больше нуля, то корнем может быть четыре  
 elif D > 0.0:  
 sqD = math.sqrt(D)  
 root1 = (-b + sqD) / (2.0 \* a)  
 root2 = (-b - sqD) / (2.0 \* a)  
  
 if (root1 == 0):  
 result.append(abs(root1))  
 elif (root2 == 0):  
 result.append(abs(root2))  
  
 if (root1 > 0.0):  
 root3 = math.sqrt(root1)  
 result.append(root3)  
 result.append(-root3)  
  
 if (root2 > 0.0):  
 root4 = math.sqrt(root2)  
 result.append(root4)  
 result.append(-root4)  
  
 return result

После установки в Python Packages: pytest\_bbd и behave. Должно работать основная программа test.py, в папке feature программа check\_root.feature, а в папке feature есть папка steps: roots.py.

**Test.py**

# Подключаем библиотеку unitetest для тестирования  
import unittest  
import math  
  
'''  
assertEqual(self, first, second)  
first - передаваемое значение  
second - полученное значение (в тело функции должен быть return, если вы там не оставили, тогда прописать здесь как None)  
если передаваемое значение совпадает с полученным значением, то тест пройден успешно  
'''  
  
# Вычисление корня  
from function.get\_roots import get\_roots  
  
# Тест на сумму  
class test\_get\_roots(unittest.TestCase):  
 # https://tutomath.ru/baza-znanij/bikvadratnye-uravneniya.html  
 # Пример №1  
 # Дискриминат больше 0 и 4 корней  
 def test\_example\_1(self):  
 self.assertEqual(  
 get\_roots(1, -5, 6), [math.sqrt(3), -math.sqrt(3), math.sqrt(2), -math.sqrt(2)]  
 )  
  
 # Пример №2  
 # Дискриминат равен 0 и 2 корня  
 def test\_example\_2(self):  
 self.assertEqual(  
 get\_roots(1, -4, 4), [math.sqrt(2), -math.sqrt(2)]  
 )  
  
 # Пример №3  
 # Дискриминат больше 0 и 3 корней  
 def test\_example\_3(self):  
 self.assertEqual(  
 get\_roots(-4, 16, 0), [0, 2, -2]  
 )  
  
 # Пример №4  
 # Дискриминат равен 0 и 2 корней  
 def test\_example\_4(self):  
 self.assertEqual(  
 get\_roots(1, 0, -16), [2.0, -2.0]  
 )  
  
 # Пример №5  
 # Дискриминат равен 0 и нет корней  
 def test\_example\_5(self):  
 self.assertEqual(  
 get\_roots(1, 0, 10), []  
 )  
  
 # Пример №6  
 # Дискриминат больше 0 и 2 корня  
 def test\_example\_6(self):  
 self.assertEqual(  
 get\_roots(1, -5, -36), [3, -3]  
 )  
  
 # Пример №7  
 # Дискриминат больше 0 и 4 корней  
 def test\_example\_7(self):  
 self.assertEqual(  
 get\_roots(1, -5, 4), [2.0, -2.0, 1.0, -1.0]  
 )  
  
  
# Получение коэффициента с комадной строки или ввода  
from function.get\_coef\_test import get\_coef\_test\_no\_cmd, get\_coef\_test\_with\_cmd  
  
class test\_get\_coef\_no\_cmd(unittest.TestCase):  
 # Без командной строки  
  
 # Тест на обычное числа  
 def test\_value\_index\_1(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(1, 1), 1.0  
 )  
  
 # Тест на нулевое числа  
 def test\_value\_index\_2(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(2, 0), 0.0  
 )  
  
 # Тест на обычное числа  
 def test\_value\_index\_3(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(3, 5), 5.0  
 )  
  
 # Тест на индекс  
 def test\_value\_index\_4(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(4, 7), None  
 )  
  
 # Тест на индекс  
 def test\_value\_index\_0(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(0, 7), None  
 )  
  
 # Тест на отрицательное число  
 def test\_value\_index\_negative\_sign(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(3, -5), -5.0  
 )  
  
 # Тест на другие символ  
 def test\_value\_other\_char(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(3, 'a'), None  
 )  
  
 # Тест на другие символ  
 def test\_value\_other\_char\_and\_negative\_sign(self):  
 self.assertEqual(  
 get\_coef\_test\_no\_cmd(3, '-a'), None  
 )  
  
class test\_get\_coef\_with\_cmd(unittest.TestCase):  
 # С командной строки  
  
 # Тест на обычное числа  
 def test\_value\_index\_1(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(1, 1), 1.0  
 )  
  
 # Тест на нулевое числа  
 def test\_value\_index\_2(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(2, 0), 0.0  
 )  
  
 # Тест на обычное числа  
 def test\_value\_index\_3(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(3, 5), 5.0  
 )  
  
 # Тест на индекс  
 def test\_value\_index\_4(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(4, 7), None  
 )  
  
 # Тест на индекс  
 def test\_value\_index\_0(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(0, 7), None  
 )  
  
 # Тест на отрицательное число  
 def test\_value\_negative\_sign(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(3, -5), -5.0  
 )  
  
 # Тест на другие символ  
 def test\_value\_other\_char(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(3, 'a'), None  
 )  
  
 # Тест на другие символ  
 def test\_value\_other\_char\_and\_negative\_sign(self):  
 self.assertEqual(  
 get\_coef\_test\_with\_cmd(3, '-a'), None  
 )  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**Для Behave**

**Check\_root.feature**

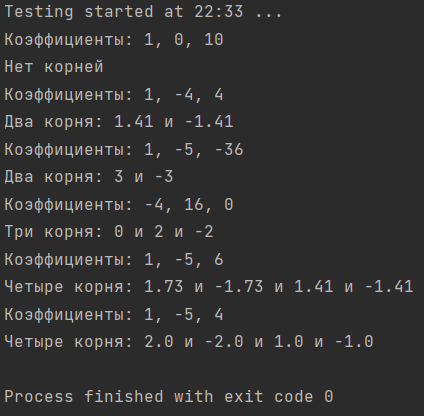
Feature: The biquadrate equation  
  
 *# Нет корней* Scenario Outline: Checking the roots of biquadrate equations: NO ROOT  
 Given I have a function calculation root  
 And I get coefficient: **<A>**, **<B>**, **<C>** When Calculating  
 Then Watch roots: **<root1>**, **<root2>**, **<root3>**, **<root4>** Examples:  
 | **A** | **B** | **C** | **root1** | **root2** | **root3** | **root4** |  
 | **1** | **0** | **10** | **0** | **0** | **0** | **0** |  
  
  
 *# Два корня* Scenario Outline: Checking the roots of biquadrate equations: TWO ROOTS  
 Given I have a function calculation root  
 And I get coefficient: **<A>**, **<B>**, **<C>** When Calculating  
 Then Watch roots: **<root1>**, **<root2>**, **<root3>**, **<root4>** Examples:  
 | **A** | **B** | **C** | **root1** | **root2** | **root3** | **root4** |  
 | **1** | **-4** | **4** | **1.41** | **-1.41** | **0** | **0** |  
 | **1** | **-5** | **-36** | **3** | **-3** | **0** | **0** |  
  
  
 *# Три корня* Scenario Outline: Checking the roots of biquadrate equations: TWO ROOTS  
 Given I have a function calculation root  
 And I get coefficient: **<A>**, **<B>**, **<C>** When Calculating  
 Then Watch roots: **<root1>**, **<root2>**, **<root3>**, **<root4>** Examples:  
 | **A** | **B** | **C** | **root1** | **root2** | **root3** | **root4** |  
 | **-4** | **16** | **0** | **0** | **2** | **-2** | **0** |  
  
  
 *# Четыре корня* Scenario Outline: Checking the roots of biquadrate equations: FOUR ROOTS  
 Given I have a function calculation root  
 And I get coefficient: **<A>**, **<B>**, **<C>** When Calculating  
 Then Watch roots: **<root1>**, **<root2>**, **<root3>**, **<root4>** Examples:  
 | **A** | **B** | **C** | **root1** | **root2** | **root3** | **root4** |  
 | **1** | **-5** | **6** | **1.73** | **-1.73** | **1.41** | **-1.41** |  
 | **1** | **-5** | **4** | **2.0** | **-2.0** | **1.0** | **-1.0** |

**Root.py**

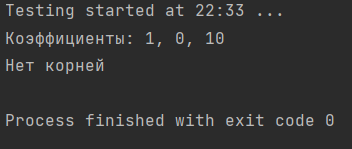
from behave import Given, When, Then  
from function.get\_roots import get\_roots  
  
  
@Given('I have a function calculation root')  
def step\_impl(context):  
 pass  
  
  
@Given("I get coefficient: {A}, {B}, {C}")  
def given\_increment(context, A, B, C):  
 context.A = int(A)  
 context.B = int(B)  
 context.C = int(C)  
 print(f'Коэффициенты: {A}, {B}, {C}')  
  
  
@When("Calculating")  
def given\_increment(context):  
 roots = get\_roots(context.A, context.B, context.C)  
 context.results = roots  
 # print(f'Корни: {roots}')  
  
  
@Then("Watch roots: {root1}, {root2}, {root3}, {root4}")  
def then\_results(context, root1, root2, root3, root4):  
 len\_roots = len(context.results)  
  
 # Вывод корней  
 if len\_roots == 0:  
 # assert (context.results == 0)  
 print('Нет корней')  
 elif len\_roots == 1:  
 assert context.results == float(root1)  
 # print('Один корень {}'.format(round(root1, 2)))  
 elif len\_roots == 2:  
 assert round(context.results[0], 2) == float(root1)  
 assert round(context.results[1], 2) == float(root2)  
 print('Два корня: {} и {}'.format(root1, root2))  
 elif len\_roots == 3 and int(root1) == 0.0:  
 assert round(context.results[0], 2) == float(root1)  
 assert round(context.results[1], 2) == float(root2)  
 assert round(context.results[2], 2) == float(root3)  
 print('Три корня: {} и {} и {}'.format(root1, root2, root3))  
 elif len\_roots == 3:  
 assert round(context.results[0], 2) == float(root1)  
 assert round(context.results[1], 2) == float(root2)  
 print('Два корня: {} и {}'.format(root1, root2))  
 elif len\_roots == 4:  
 assert round(context.results[0], 2) == float(root1)  
 assert round(context.results[1], 2) == float(root2)  
 assert round(context.results[2], 2) == float(root3)  
 assert round(context.results[3], 2) == float(root4)  
 print('Четыре корня: {} и {} и {} и {}'.format(root1, root2, root3, root4))  
 else:  
 print('Ошибка! Корней нет!')

## Результат

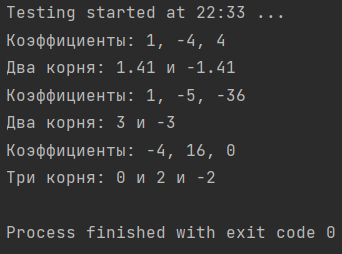
**PyCharm:**

****

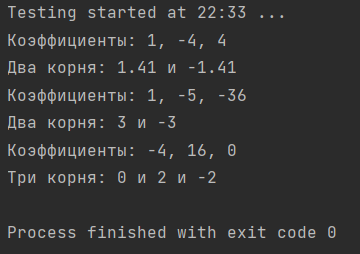
Нет корней:



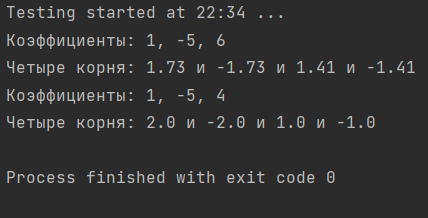
Два корня:



Три корня:



Четыре корня:

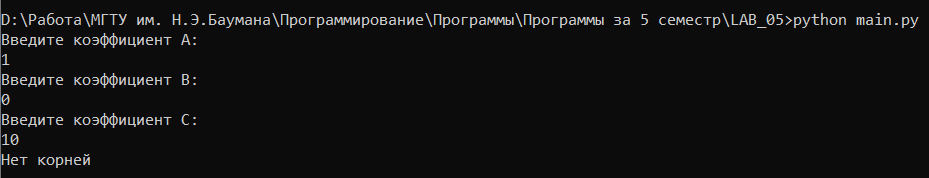


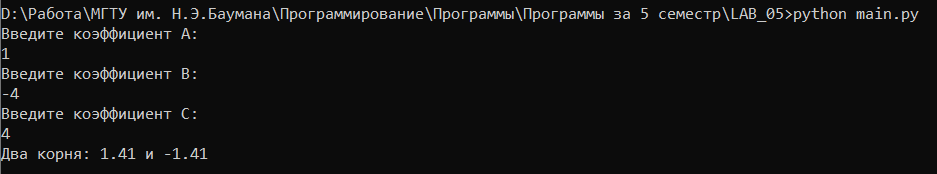
**Командная строка:**

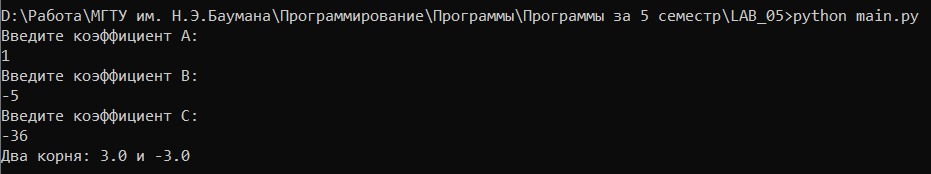
**Для Behave**

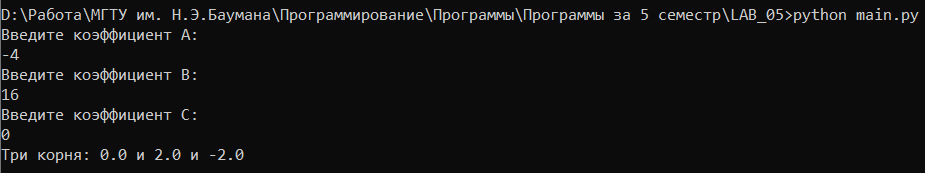
Путь файла для работы программы main.py:

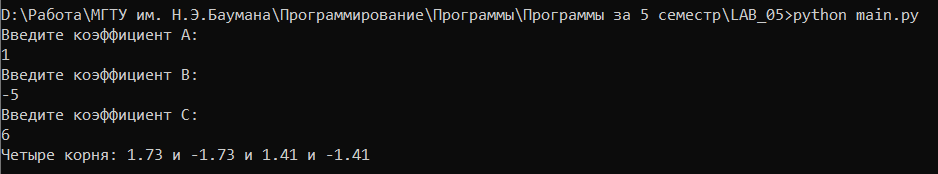
..\LAB\_05>python main.py

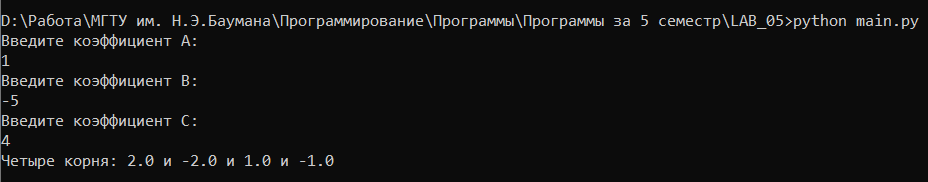












**Для Unittest**

Путь файла для работы программы test.py:

..\LAB\_05>python test.py

