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»

Лабораторная работа №4

«Шаблоны проектирования и модульное тестирование в Python»

по дисциплине «Базовые компоненты интернет-технологий»

Выполнила:

студентка группы ИУ5-35Б

Ищенко А.С.

2021 г.

Задание:

1. 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.
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.

Лабораторная работа была сделана по пункту 2 и 3.

Текст программы:
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 A
        b (float): коэффициент B
        c (float): коэффициент C
    Returns:f
        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 A:', )

    while a == 0:
        print('Коэфициент A не может быть равен 0')
        a = get_coef(1, 'Введите коэффициент A:')


    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
```

TDD:

tdd.py

```
import unittest
from main import get_roots


class TestGetRoots(unittest.TestCase):
    # x^4+2x^2+3 = 0
    def test_no_roots(self):
        self.assertEqual(get_roots(1, 2, 3), [])

    def test_four_roots(self):
        expected_len = 4
        expected_value = [-1, 1, 0.5, -0.5]

        cur_value = get_roots(4, -5, 1)
        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)
        self.assertEqual(expected_value[3] in cur_value, True)

    def test_three_roots(self):
        expected_len = 3
        expected_value = [-1, 0, 1]

        cur_value = 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)

    def test_two_roots(self):
        expected_len = 2
        expected_value = [-2, 2]

        cur_value = 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)

    def test_one_root(self):
        expected_value = [0]

        cur_value = get_roots(1, 1, 0)
        self.assertEqual(len(cur_value), 1)
        self.assertEqual(expected_value[0], cur_value[0], True)


if __name__ == "__main__":
    unittest.main()
```

BDD:

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]
```

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
```

test_one_root.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
```

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
```

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
```

test_four_roots.py

```
from behave import *
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
```

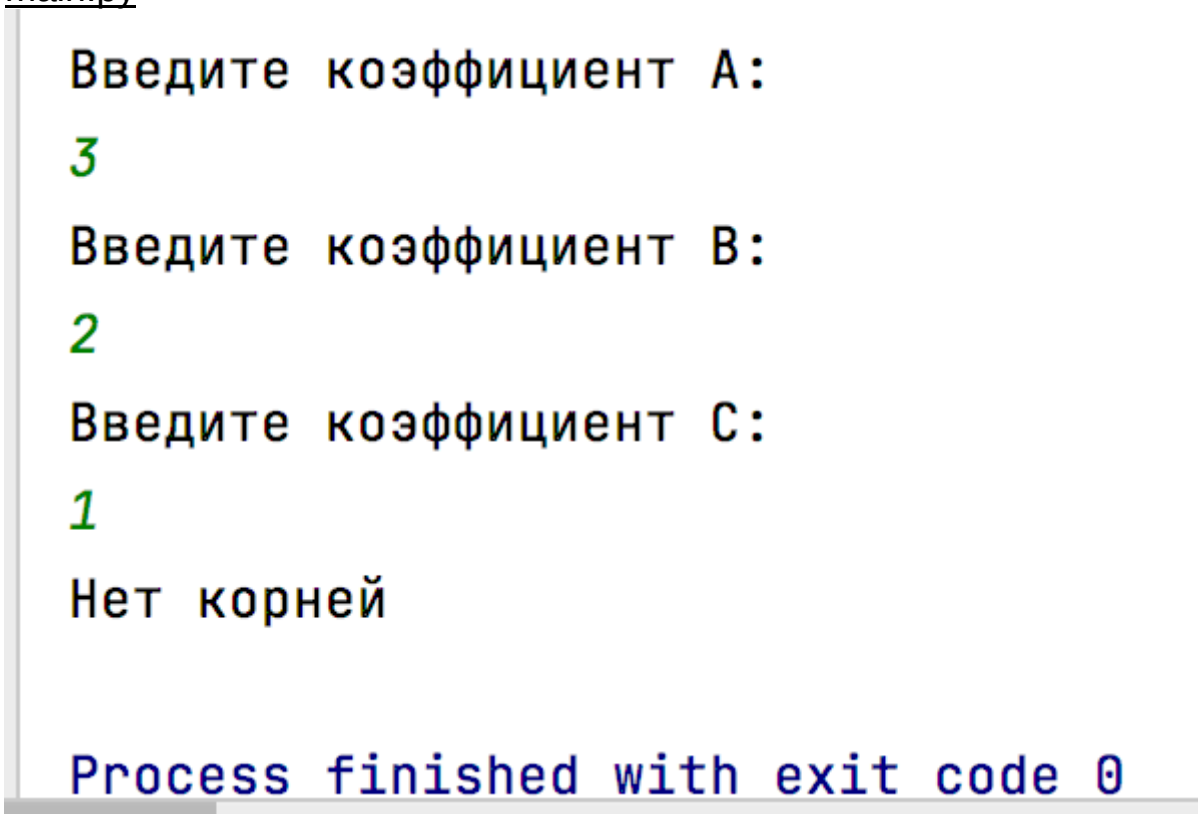
Mock-объекты

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)

if __name__ == "__main__":
    TestRoots.no_roots()
```

Результат выполнения программы:

main.py



TDD

```
Testing started at 5:02 ...
Launching unittests with arguments python -m unittest



Ran 5 tests in 0.003s

OK

Process finished with exit code 0
```

BDD

```
features/steps/test_no_roots.py:18: SyntaxWarning: "is" with a literal. Did you mean "=="?
  assert context.array_len is 0
Feature: get_roots function # features/get_roots.feature:1

  Scenario: test no roots     # features/get_roots.feature:3
    Given A = 1, B = 2, C = 3 # features/steps/test_no_roots.py:4 0.000s
    When get_roots run        # features/steps/test_no_roots.py:11 0.000s
    Then roots array is empty # features/steps/test_no_roots.py:16 0.000s

  Scenario: test one root     # features/get_roots.feature:8
    Given A = 1, B = 1, C = 0 # features/steps/test_one_root.py:4 0.000s
    When get_roots execute1   # features/steps/test_one_root.py:11 0.000s
    Then roots is [0]         # features/steps/test_one_root.py:16 0.000s

  Scenario: test two root       # features/get_roots.feature:13
    Given A = 1, B = -2, C = -8 # features/steps/test_two_roots.py:4 0.000s
    When get_roots execute2     # features/steps/test_two_roots.py:11 0.000s
    Then roots are [-2, 2]      # features/steps/test_two_roots.py:16 0.000s

  Scenario: test three root    # features/get_roots.feature:18
    Given A = 1, B = -1, C = 0 # features/steps/test_three_roots.py:4 0.000s
    When get_roots execute3    # features/steps/test_three_roots.py:11 0.000s
    Then roots are [-1, 0, 1]  # features/steps/test_three_roots.py:16 0.000s

  Scenario: test four root            # features/get_roots.feature:23
    Given A = 4, B = -5, C = 1        # features/steps/test_four_roots.py:4 0.000s
    When get_roots execute4           # features/steps/test_four_roots.py:11 0.000s
    Then roots are [-1, 1, 0.5, -0,5] # features/steps/test_four_roots.py:16 0.000s

1 feature passed, 0 failed, 0 skipped
5 scenarios passed, 0 failed, 0 skipped
15 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.002s
```

```
Testing started at 5:04 ...
Launching unittests with arguments python -m unittest



Ran 5 tests in 0.005s

OK

Process finished with exit code 0
```