

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

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «БКИТ»
Отчет по лабораторной работе №5

Выполнил:

студент группы ИУ5-35Б
Крылов Дмитрий

Подпись и дата:

Проверил:

преподаватель каф. ИУ5
Гапанюк Ю. Е.

Подпись и дата:

Москва, 2022 г.

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

equation.py

```
from math import sqrt
import sys

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 коэффициента
    Returns:
        float: Коэффициент квадратного уравнения
    '''
    try:
        # Пробуем прочитать коэффициент из командной строки
        coef_str = sys.argv[index]
        #coef = float(coef_str)
        coef = coef_str
        not_argv = False
    except:
        # Вводим с клавиатуры
        not_argv = True
        print(prompt)
        pass
    if not_argv:
        flag = True
        while flag:
            try:
                coef = float(input())
                flag = False
```

```
except:
    print('Повторите ввод коэффициента')
    pass
return coef
```

1 6

```
def get_roots(a, b, c):
```

```
    """
    Вычисление корней квадратного уравнения
    Args:
        a (float): коэффициент A
        b (float): коэффициент B
        c (float): коэффициент C
    Returns:
        list[float]: Список корней
    """
```

```
    if type(a) not in [float]:
        raise TypeError("Коэффициент A должен быть положительным float!")
```

```
    if type(b) not in [float]:
        raise TypeError("Коэффициент B должен быть неотрицательным float!")
```

```
    if type(c) not in [float]:
        raise TypeError("Коэффициент C должен быть неотрицательным float!")
```

```
    if a == 0.0 and b == 0.0:
        raise ValueError("Коэффициент A и B должны быть положительными float!")
```

```
    result = []
    under_sqrt_mini = b ** 2 - 4 * a * c
```

```

result = []
under_sqrt_mini = b ** 2 - 4 * a * c
if under_sqrt_mini >= 0:
    under_sqrt_one = (-b + sqrt(under_sqrt_mini)) / 2 * a
    under_sqrt_two = (-b - sqrt(under_sqrt_mini)) / 2 * a
    if under_sqrt_one >= 0:
        result.append(sqrt(under_sqrt_one))
        result.append(-sqrt(under_sqrt_one))
    if under_sqrt_two >= 0:
        result.append(sqrt(under_sqrt_two))
        result.append(-sqrt(under_sqrt_two))
return result

def main():
    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)
    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 корень: {} и {}'.format(roots[0], roots[1]))
    elif len_roots == 4:
        print('Четыре корня: {} и {} и {} и {}'.format(roots[0], roots[1], roots[2], roots[3]))

if __name__ == "__main__":
    main()

```

test_equation.py

```
import unittest
from equation import get_roots

class TestEquation(unittest.TestCase):
    def test_calculate(self):
        self.assertEqual(get_roots(1.0, -10.0, 9.0), [-3.0, -1.0, 1.0, 3.0])
        self.assertEqual(get_roots(-4.0, 16.0, 0.0), [-2.0, 0.0, 2.0])
        self.assertEqual(get_roots(431.0, -123.0, 665.0), [])


    def test_value(self):
        self.assertRaises(ValueError, get_roots(0.0, 0.0, 9.0))

    def test_type(self):
        self.assertRaises(TypeError, get_roots(12.0, "B", 4.0))

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

equation.feature

1 Feature: Scenario Outline

2  This app solve biquatratric equation

4 Scenario Outline: Solve the equation with correct value

5 Given The A coefficient <A>

6 And The B coefficient

7 And The C coefficient <C>

8 When Solve the equation

9 Then I get <D> roots

11 Examples:

12 | A | B | C | D |

14 | 1 | 12 | 36 | 0 |

15 | 6 | 60 | 54 | 0 |

16 | 3 | 31 | 56 | 0 |

18 | 1 | 1 | 0 | 1 |

19 | 5 | 15 | 0 | 1 |

20 | 30 | 18 | 0 | 1 |

22 | 3 | -5 | -28 | 2 |

23 | 3 | -14 | -117 | 2 |

24 | 11 | -86 | -117 | 2 |

test_BDD.py

```
from main import get_roots
from pytest_bdd import scenarios, given, when, then, parsers

scenarios("equation.feature")

@given(parsers.parse("The A coefficient {A:d}"), target_fixture="coefA")
def t_root_input_1(A):
    return A

@given(parsers.parse('The B coefficient {B:d}'), target_fixture="coefB")
def t_root_input_2(B):
    return B

@given(parsers.parse('The C coefficient {C:d}'), target_fixture="coefC")
def t_root_input_3(C):
    return C

@when(parsers.parse('Solve the equation'), target_fixture="equ")
def t_root_solve(coefA, coefB, coefC):
    return get_roots(coefA, coefB, coefC)

@then(parsers.parse("I get {zero:d} roots"))
def t_then(equ, zero):
    assert len(equ) == zero
```


main.py

```
import sys
import math

def get_coef(index, prompt):
    try:
        # Пробуем прочитать коэффициент из командной строки
        coef_str = sys.argv[index]
    except:
        # Вводим с клавиатуры
        print(prompt)
        coef_str = input()
    # Обрабатываем неправильный ввод
    while True:
        try:
            coef = float(coef_str)
        except:
            print("Введены неправильные данные.", prompt)
            coef_str = input()
        else:
            return coef

def get_roots(a, b, c):
    result = []
    D = b * b - 4 * a * c
    if D == 0.0:
        root = -b / (2.0 * a)
        if root >= 0:
            result.append(math.sqrt(root))
            result.append(-math.sqrt(root))
```

```
    elif D > 0.0:
        sqD = math.sqrt(D)
        root1 = (-b + sqD) / (2.0 * a)
        if root1 >= 0:
            if root1 == 0:
                result.append(root1)
            else:
                result.append(math.sqrt(root1))
                result.append(-math.sqrt(root1))
        root2 = (-b - sqD) / (2.0 * a)
        if root2 >= 0:
            if root2 == 0.0:
                result.append(root2)
            else:
                result.append(math.sqrt(root2))
                result.append(-math.sqrt(root2))
        result = set(result)
    return result

def main():
    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', end=" ")
```

```
roots = get_roots(a, b, c)
# Вывод корней
len_roots = len(roots)
if len_roots == 0:
    print('Нет корней', end=" ")
    return
elif len_roots == 1:
    print('Один корень:', end=" ")
elif len_roots == 2:
    print('Два корня:', end=" ")
elif len_roots == 3:
    print('Три корня:', end=" ")
else:
    print('Четыре корня:')
print(*roots, sep=" ")

# Если сценарий запущен из командной строки
if __name__ == "__main__":
    main()
```

Результаты выполнения программы

Test equation.py

```
/Users/dmitrykrylov/PycharmProjects/lab5/venv/bin/python /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_pytest_runner
.py --path /Users/dmitrykrylov/PycharmProjects/lab5/test_equation.py
Testing started at 12:47 ...
Launching pytest with arguments /Users/dmitrykrylov/PycharmProjects/lab5/test_equation.py --no-header --no-summary -q in
/Users/dmitrykrylov/PycharmProjects/lab5

===== test session starts =====
collecting ... collected 3 items

test_equation.py::TestEquation::test_calculate FAILED [ 33%]
test_equation.py:4 (TestEquation.test_calculate)
[-3.0, -1.0, 1.0, 3.0] != [3.0, -3.0, 1.0, -1.0]

Expected :[3.0, -3.0, 1.0, -1.0]
Actual   :[-3.0, -1.0, 1.0, 3.0]
<Click to see difference>

self = <test_equation.TestEquation testMethod=test_calculate>

    def test_calculate(self):
>     self.assertEqual(get_roots(1.0, -10.0, 9.0), [-3.0, -1.0, 1.0, 3.0])

test_equation.py:6: AssertionError

test_equation.py::TestEquation::test_type FAILED [ 66%]
test_equation.py:12 (TestEquation.test_type)

self = <test_equation.TestEquation testMethod=test_type>

    def test_type(self):
>     self.assertRaises(TypeError, get_roots(12.0, "B", 4.0))

test_equation.py:14:
-----

a = 12.0, b = 'B', c = 4.0

    def get_roots(a, b, c):
        """
        Вычисление корней квадратного уравнения
        Args:
            a (float): коэффициент A
            b (float): коэффициент B
            c (float): коэффициент C
        Returns:
            list[float]: Список корней
        """
        if type(a) not in [float]:
            raise TypeError("Коэффициент A должен быть положительным float!")

        if type(b) not in [float]:
>             raise TypeError("Коэффициент B должен быть неотрицательным float!")
E             TypeError: Коэффициент B должен быть неотрицательным float!

equation.py:58: TypeError
```

```

test_equation.py::TestEquation::test_value FAILED [100%]
test_equation.py:9 (TestEquation.test_value)
self = <test_equation.TestEquation testMethod=test_value>

    def test_value(self):
>         self.assertRaises(ValueError, get_roots(0.0, 0.0, 9.0))

test_equation.py:11:
-----

a = 0.0, b = 0.0, c = 9.0

    def get_roots(a, b, c):
        """
        Вычисление корней квадратного уравнения
        Args:
            a (float): коэффициент A
            b (float): коэффициент B
            c (float): коэффициент C
        Returns:
            list[float]: Список корней
        """
        if type(a) not in [float]:
            raise TypeError("Коэффициент A должен быть положительным float!")

        if type(b) not in [float]:
            if a == 0.0 and b == 0.0:
>                 raise ValueError("Коэффициент A и B должны быть положительными float!")
E                 ValueError: Коэффициент A и B должны быть положительными float!

equation.py:56: ValueError

===== 3 failed in 0.03s =====

Process finished with exit code 1

```

Test BDD.py

```

/Users/dmitrykrylov/PycharmProjects/lab5/venv/bin/python /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py
c.py --path /Users/dmitrykrylov/PycharmProjects/lab5/features/test_BDD.py
Testing started at 12:49 ...
Launching pytest with arguments /Users/dmitrykrylov/PycharmProjects/lab5/features/test_BDD.py --no-header --no-summary -q in
/Users/dmitrykrylov/PycharmProjects/lab5/features

===== test session starts =====
collecting ... collected 9 items

test_BDD.py::test_solve_the_equation_with_correct_value[1-12-36-0] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 11%]
test_BDD.py::test_solve_the_equation_with_correct_value[6-60-54-0] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 22%]
test_BDD.py::test_solve_the_equation_with_correct_value[3-31-56-0] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 33%]
test_BDD.py::test_solve_the_equation_with_correct_value[1-1-0-1] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 44%]
test_BDD.py::test_solve_the_equation_with_correct_value[5-15-0-1] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 55%]
test_BDD.py::test_solve_the_equation_with_correct_value[30-18-0-1] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 66%]
test_BDD.py::test_solve_the_equation_with_correct_value[3--5--28-2] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 77%]
test_BDD.py::test_solve_the_equation_with_correct_value[3--14--117-2] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [ 88%]
test_BDD.py::test_solve_the_equation_with_correct_value[11--86--117-2] <- ../venv/lib/python3.11/site-packages/pytest_bdd/scenario.py PASSED [100%]

===== 9 passed in 0.01s =====

Process finished with exit code 0

```