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»

Отчет по лабораторной работе №5
по дисциплине «Базовые компоненты интернет-технологий»
Тема «Модульное тестирование в Python»

Выполнил:
студент группы
ИУ5Ц-52Б Дзауров И.А.
_____, “___” _____ 2022 г.

Проверил:
преподаватель кафедры
ИУ5 - Гапанюк Ю.Е.
_____, “___” _____ 2022 г.

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).

Структура проекта:

```
main.py
TDDTestRoots.py
MockTest.py
```

Основной файл - main.py:

```python
# This is a sample Python script.

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
        coef_str = sys.argv[index]
    except:
        # Вводим с клавиатуры
        print(prompt)
        coef_str = input()

    # Переводим строку в действительное число
    while True:
        try:
            coef = float(coef_str)
        except ValueError:
            print("Неверный ввод. Попробуйте еще раз")
            # Вводим с клавиатуры
            print(prompt)
            coef_str = input()
        else:
            break

    return coef

#Определение знака
def get_sign(number):
    if number >= 0:
        return '+'
    return '-'


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 [int, float]:
        raise TypeError("Коэффициент [a] должен быть положительным вещественным
числом!")
    if type(b) not in [int, float]:
        raise TypeError("Коэффициент [b] должен быть неотрицательным
вещественным числом!")
    if type(c) not in [int, float]:
        raise TypeError("Коэффициент [c] должен быть неотрицательным
вещественным числом!")

    if a == 0:
        raise ValueError("Коэффициент [f] должен быть положительным вещественным
числом!")

    result = []
    D = b * b - 4 * a * c
    if D == 0.0:
        root = -b / (2.0 * a)
        result.append(root)
    elif D > 0.0:
        sqD = math.sqrt(D)
        root1 = (-b + sqD) / (2.0 * a)
        root2 = (-b - sqD) / (2.0 * a)
        result.append(root1)
        result.append(root2)
    return result


def get_roots_biquadratic(roots):
    '''
    Вычисление корней для биквадратного уравнения исходя из результата функции -
[get_roots]
    Args:
        list [float]: массив корней квадратного уравнения

    Returns:
        list [float]: массив корней биквадратного уравнения
    '''

    result = []

    for root in roots:
        if root == 0:
            result.append(root)
        elif root > 0:
            sqRoot = math.sqrt(root)
            result.append(sqRoot)
            result.append(-sqRoot)

    return result


def main():
    '''
    Основная функция
    '''
    a = get_coef(1, 'Введите коэффициент - [a]:')
    while a == 0.0:
        print('Коэффициент - [a] в биквадратном уравнении не может равняться
нулю')
```

```
        a = get_coef(1, 'Введите коэффициент - [a]:')
    b = get_coef(2, 'Введите коэффициент - [b]:')
    c = get_coef(3, 'Введите коэффициент - [c]:')

    # Вычисление корней для квадратного уравнения
    roots = get_roots(a, b, c)

    # Вычисление корней для биквадратного уравнения исходя из результата функции
- [get_roots]
    roots = get_roots_biquadratic(roots)

    # Вывод корней
    len_roots = len(roots)

    if len_roots == 0:
        print('У уравнения {}x^4 {} {}x^2 {} {} нет корней'.format(a,
get_sign(b), abs(b), get_sign(c), abs(c)))

    elif len_roots == 1:
        print('У уравнения {}x^4 {} {}x^2 {} {} один корень: {}'.format(a,
get_sign(b), abs(b), get_sign(c), abs(c), roots[0]))

    elif len_roots == 2:
        print('У уравнения {}x^4 {} {}x^2 {} {} два корня: {}, {}'.format(a,
get_sign(b), abs(b), get_sign(c), abs(c), roots[0], roots[1]))

    elif len_roots == 3:
        print('У уравнения {}x^4 {} {}x^2 {} {} три корня: {}, {}, {}'.format(a,
get_sign(b), abs(b), get_sign(c), abs(c), roots[0], roots[1], roots[2]))

    elif len_roots == 4:
        print('У уравнения {}x^4 {} {}x^2 {} {} четыре корня: {}, {}, {},
{}'.format(a, get_sign(b), abs(b), get_sign(c), abs(c), roots[0], roots[1],
roots[2], roots[3]))


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

TDD-тест - TDDTestRoots.py:

```
import unittest

from main import get_roots, get_roots_biquadratic

class TestGetRoots(unittest.TestCase):
    def testGetRoots(self):
        self.assertEqual(get_roots_biquadratic(get_roots(4, -5, 1)), [1.0, -1.0,
0.5, -0.5])
        self.assertEqual(get_roots_biquadratic(get_roots(1, -2, -8)), [2.0, -
2.0])
        self.assertEqual(get_roots_biquadratic(get_roots(1, 1, 1)), [])

    def testValue(self):
        with self.assertRaises(ValueError) as e:
            get_roots_biquadratic(get_roots(0, 33, 9))

    def testType(self):
        with (self.assertRaises(TypeError)) as e:
```

```
            get_roots_biquadratic(get_roots(7, "D", 4))

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

Mock-тест - MockTest.py:

```
import unittest
from unittest.mock import Mock

from main import get_roots, get_roots_biquadratic

class MOCKtestGetRoots(unittest.TestCase):
   def testGetRoots(self):
      mockRoot = Mock(return_value=5)
      get_roots_biquadratic(get_roots(mockRoot(), 5, 5))

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

Экранные формы с примерами выполнения программы