Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»
Отчёт по лабораторной работе №4
«Шаблоны проектирования и модульное тестирование в Python»

Выполнил:
студент группы ИУ5-31Б
Шимолина Полина
Кирилловна

Подпись:_____

Дата:_____

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий
Евгеньевич

Подпись:_____

Дата:_____

Москва, 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.

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()
    # Переводим строку в действительное число
    try:
        coef = float(coef_str)
    except:
        return get_coef(index, prompt)
    return coef


def get_roots(a, b, c):
    result = []
    D = b * b - 4 * a * c
    if D == 0.0:
        root = -b / (2.0 * a)
        result.append(math.sqrt(root))
        result.append(-math.sqrt(root))
    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(math.sqrt(root1))
            result.append(-math.sqrt(root1))
        elif root1 == 0:
            result.append(root1)
        if root2 >= 0:
            result.append(math.sqrt(root2))
            result.append(-math.sqrt(root2))
        elif root2 == 0:
            result.append(root2)
```

```
    return result


def check_coef(a, b, c):
    if a != 0:
        return get_roots(a, b, c)
    else:
        if b != 0:
            if c != 0:
                return [-c / b]
            else:
                return [0]
        else:
            if c != 0:
                return []
            else:
                return ['infinity']


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 = check_coef(a, b, c)
    # Вывод корней
    len_roots = len(roots)
    if len_roots == 0:
        print('Нет корней')
    elif len_roots == 1:
        print('{}'.format(roots[0]))
    elif len_roots == 2:
        print('{} и {}'.format(roots[0], roots[1]))
    elif len_roots == 3:
        print('{}, {} и {}'.format(roots[0], roots[1], roots[2]))
    elif len_roots == 4:
        print('{}, {}, {} и {}'.format(roots[0], roots[1], roots[2],
roots[3]))


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

TDDtest.py

```
import main
import unittest
from unittest.mock import patch


class Tests(unittest.TestCase):
    def test_2(self):
        roots = main.check_coef(1, -4, 4)
        self.assertAlmostEqual(1.4142, roots[0], 4)
        self.assertAlmostEqual(-1.4142, roots[1], 4)

    def test_3(self):
        roots = main.check_coef(-4, 16, 0)
        self.assertEqual([0, 2, -2], roots)

    def test_4(self):
        roots = main.check_coef(1, -10, 9)
        self.assertEqual([3, -3, 1, -1], roots)
```

```
    def test_0(self):
        roots = main.check_coef(1, 0, 10)
        self.assertEqual([], roots)

    def test_inf(self):
        roots = main.check_coef(0, 0, 0)
        self.assertEqual(['infinity'], roots)

    @patch('main.check_coef', return_value=[9])
    def test_m(self, check_coef):
        self.assertEqual(main.check_coef(1, 2, 3), [9])
```

test.feature

```
Feature: 3 roots

  Scenario: k -4 16 0
    Given I have -4*x**4 + 16*x**2 + 0 = 0
    When I solve the equation
    Then I e
```

steps.py

```
from main import *
from behave import *


@given('I have {a}*x**4 + {b}*x**2 + {c} = 0')
def step_impl(context, a, b, c):
    context.a = float(a)
    context.b = float(b)
    context.c = float(c)


@when('I solve the equation')
def step_impl(context):
    context.roots = check_coef(context.a, context.b, context.c)


@then('I expect 3 roots: {x_1}, {x_2}, {x_3}')
def step_impl(context, x_1, x_2, x_3):
    res = [float(x_1), float(x_2), float(x_3)]
    assert context.roots == res
```

Экранные формы:

-m behave

```
Feature: 3 roots # test.feature:1

  Scenario: k -4 16 0                        # test.feature:3
    Given I have -4*x**4 + 16*x**2 + 0 = 0 # steps/steps.py:5
    When I solve the equation              # steps/steps.py:12
    Then I expect 3 roots: 0, 2, -2        # steps/steps.py:17

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.002s
```

-m unittest TDDtest.Tests

```
Ran 6 tests in 0.009s

OK
```