Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №5 «Модульное тестирование в Python»

Выполнил:
Студент группы ИУ5-32Б:
Арзамасцев Артем
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.

Текст программы

Программа для тестирования взятая из первой лабораторной работы BiSquareRoot.py

```python
import sys
import math


class MismatchError(Exception):
    pass


class SquaredRoot:
    def __init__(self, a, b, c):
        if a == 0.0:
            raise MismatchError("Уравнение с такими коэффициентами " +
                                "не является квадратным")
        self.a = a
        self.b = b
        self.c = c

    def calculate(self):
        descriminant = self.b * self.b - 4 * self.a * self.c
        if descriminant < 0.0:
            return []
        elif descriminant == 0.0:
            return [-self.b / (2 * self.a)]
        descriminant = math.sqrt(descriminant)
        return list(map(lambda x: (-self.b + x) / (2 * self.a),
                    (-descriminant, descriminant)))


class BiSquareRoot:
    def __init__(self, a, b, c):
        if a == 0.0:
            raise MismatchError("Уравнение с такими коэффициентами не " +
                                "является биквадратным")
        self.a = a
        self.b = b
        self.c = c

    def calculate(self):
        equation = SquaredRoot(self.a, self.b, self.c)
        roots = equation.calculate()
        res = []
        for i in list(filter(lambda x: x >= 0.0, roots)):
            if i == 0.0:
                res.append(0.0)
            else:
                root = math.sqrt(i)
                res.extend((-root, root))
        return res


def update_list_factors(lst, val):
    try:
        lst.append(float(val))
    except ValueError:
        return lst
    except Exception:
        return lst
    return lst


def read_factors():
    lst = []
    name_factors = ["a", "b", "c"]
    for i in sys.argv[1:]:
        lst = update_list_factors(lst, i)
        if len(lst) == 3:
            return lst
    while len(lst) != 3:
        lst = update_list_factors(
            lst,
            input(f"введите коэффициент {name_factors[len(lst)]}: ")
                    )
    return lst


def main():
    list_factors = read_factors()
    try:
        equation = BiSquareRoot(*list_factors)
        roots = equation.calculate()
        if len(roots) == 0:
            print("корней нет")
        else:
            print("корни уравнения: ", *roots)
    except MismatchError as err:
        print(err)
    except Exception as err:
        print(err)


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

TDD

```
import unittest.mock
from BiSquareRoot import BiSquareRoot, read_factors
import sys


def compare_list(lst1, lst2):
    return len(lst1) == len(lst2) and set(lst1) == set(lst2)


class TestBiSquareRoot(unittest.TestCase):
    def test_calculate(self):
        tests = [
            ([5, -5, 0], [1, -1, 0]),
            ([1, 1, 9], []),
            ([-5, -2, -34], []),
            ([5, 34, 12], []),
            ([113, 23, 1], []),
            ([-1, -1, -11], []),
            ([1, -25, 144], [3, -3, 4, -4]),
            ([1, -8, 16], [2, -2]),
            ([0.01, -0.0769, 0.09], [-1.2, 1.2, -2.5, 2.5]),
            ([1, -7.69, 9], [-1.2, 1.2, -2.5, 2.5]),
            ([50, 550, -2306.88], [-1.8, 1.8]),
            ([500, 5500, -23068.8], [-1.8, 1.8]),
        ]
        for test in tests:
            equation = BiSquareRoot(*test[0])
            res = equation.calculate()
            self.assertTrue(compare_list(res, test[1]))


class TestMain(unittest.TestCase):
    @unittest.mock.patch("BiSquareRoot.input")
    def test_read_factors(self, input_mock):
        sys.argv.extend("2 sdgg sd / -26 ds".split())
        input_mock.return_value = "12.7"
        self.assertEqual(read_factors(), [2, -26, 12.7])

        sys.argv.append("18")
        self.assertEqual(read_factors(), [2, -26, 18])
        sys.argv = sys.argv[:1]


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

BDD

Feature файл

Feature: Testing a program that solves a biquadrate equation

Scenario: solution of the biquadrate equation 1

Given I have the numbers 5, 4, 6

When I make them the coefficients of the biquadrate equation

Then I expect it to have the following roots: None

Scenario: solution of the biquadrate equation 2

Given I have the numbers 5, 55, 6

When I make them the coefficients of the biquadrate equation

Then I expect it to have the following roots: None

Scenario: solution of the biquadrate equation 3

Given I have the numbers 5, 455, 6

When I make them the coefficients of the biquadrate equation

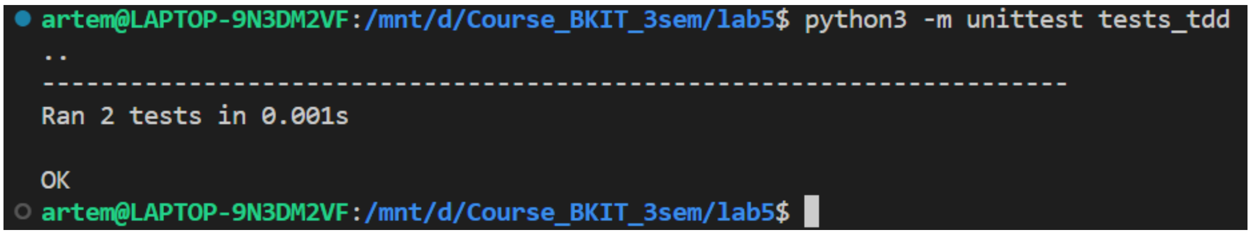
Then I expect it to have the following roots: None

tests_bdd.py

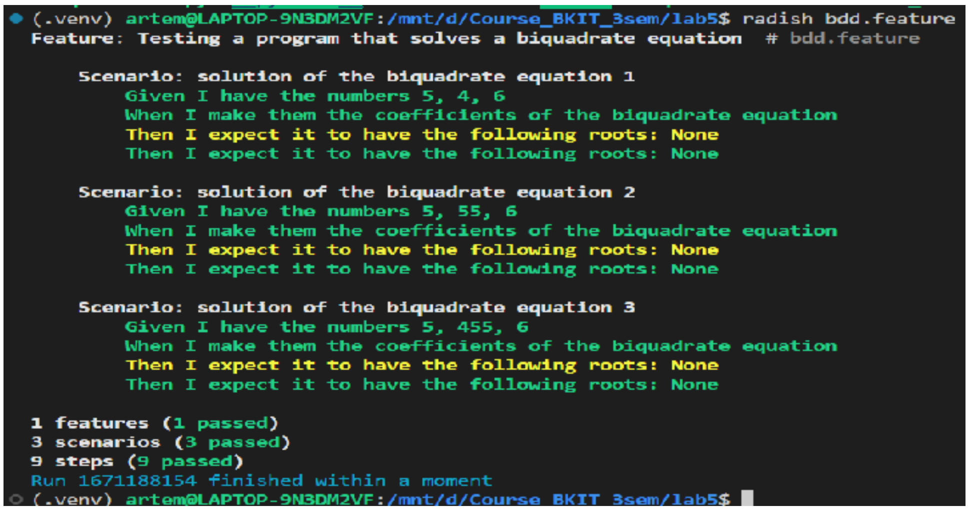
```
from radish import given, when, then, custom_type, register_custom_type,
TypeBuilder
import sys
```

```python
import os

sys.path.append(os.getcwd())
import BiSquareRoot


@custom_type('Number', r"\d*")
def parse_number(text):
    res = None
    try:
        res = int(text)
    except Exception:
        pass
    return res

register_custom_type(NumberList=TypeBuilder.with_many(
    parse_number, listsep=','))

@given('I have the numbers {numbers:NumberList}')
def have_numbers(step, numbers):
    step.context.factors = list(filter(lambda x: isinstance(x, (int, float)),
numbers))


@when("I make them the coefficients of the biquadrate equation")
def sum_numbers(step):
    if len(step.context.factors) != 3:
        step.context.result = [None]
        return
    equation = BiSquareRoot.BiSquareRoot(*step.context.factors)
    step.context.result = equation.calculate()
    if len(step.context.result) == 0:
        step.context.result = [None]

@then("I expect it to have the following roots: {res:NumberList}")
def expect_result(step, res):
    print(res)
    assert step.context.result == res
```

Примеры выполнения программ



```
● artem@LAPTOP-9N3DM2VF:/mnt/d/Course_BKIT_3sem/lab5$ python3 -m unittest tests_tdd
  ..
  ----------------------------------------------------------------------
  Ran 2 tests in 0.001s

  OK
○ artem@LAPTOP-9N3DM2VF:/mnt/d/Course_BKIT_3sem/lab5$
```



```
● (.venv) artem@LAPTOP-9N3DM2VF:/mnt/d/Course_BKIT_3sem/lab5$ radish bdd.feature
  Feature: Testing a program that solves a biquadrate equation  # bdd.feature

      Scenario: solution of the biquadrate equation 1
          Given I have the numbers 5, 4, 6
          When I make them the coefficients of the biquadrate equation
          Then I expect it to have the following roots: None
          Then I expect it to have the following roots: None

      Scenario: solution of the biquadrate equation 2
          Given I have the numbers 5, 55, 6
          When I make them the coefficients of the biquadrate equation
          Then I expect it to have the following roots: None
          Then I expect it to have the following roots: None

      Scenario: solution of the biquadrate equation 3
          Given I have the numbers 5, 455, 6
          When I make them the coefficients of the biquadrate equation
          Then I expect it to have the following roots: None
          Then I expect it to have the following roots: None

  1 features (1 passed)
  3 scenarios (3 passed)
  9 steps (9 passed)
  Run 1671188154 finished within a moment
○ (.venv) artem@LAPTOP-9N3DM2VF:/mnt/d/Course_BKIT_3sem/lab5$
```