Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3
«Модульное тестирование в Python.»

Выполнил:
студент группы ИУ5-31Б
Зобнин Александр
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Подпись и дата:

Постановка задачи

Задание:

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).

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
    '''
    while True:
            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 ValueError:
                print("Некорректный ввод. Попробуйте ещё раз.")
                continue
            else:
                break
    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:
        list[float]: Список корней
    '''
    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_real_roots(result):
    real_result = []
    for root in result:
        if root > 0:
            real_result.append(math.sqrt(root))
            real_result.append(-math.sqrt(root))
        elif root == 0:
            real_result.append(abs(root))
    return real_result


def solve_biquadrate_equation(a, b, c):
    roots = get_roots(a, b, c)
    real_roots = get_real_roots(roots)
    return real_roots


def main():
    '''
    Основная функция
    '''
    a = get_coef(1, 'Введите коэффициент A:')
    b = get_coef(2, 'Введите коэффициент B:')
    c = get_coef(3, 'Введите коэффициент C:')
    real_roots = solve_biquadrate_equation(a, b, c)
    # Вывод корней
    len_roots = len(real_roots)
    if len_roots == 0:
        print('Нет корней')
    elif len_roots == 1:
        print('Один корень: {}'.format(real_roots[0]))
    elif len_roots == 2:
        print('Два корня: {} и {}'.format(real_roots[0], real_roots[1]))
    elif len_roots == 3:
        print('Три корня: {}, {} и {}'.format(real_roots[0], real_roots[1],
real_roots[2]))
    elif len_roots == 4:
        print('Четыре корня: {}, {}, {} и {}'.format(real_roots[0], real_roots[1],
real_roots[2], real_roots[3]))


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

# Примеры запуска
# roots_proc.py 1 0 10 (Нет корней)
# roots_proc.py 1 0 -4 (Два корня)
# roots_proc.py -4 16 0 (Три корня)
# roots_proc.py 1 -13 36 (Четыре корня)
```

tddtests.py

```
import unittest
from main import *

class Test_biquadrate_solution(unittest.TestCase):
    def test_0_roots(self):
        self.assertEqual(len(solve_biquadrate_equation(1, 0, 10)), 0)
    def test_2_roots(self):
        self.assertEqual(len(solve_biquadrate_equation(1, 0, -4)), 2)
        self.assertEqual(solve_biquadrate_equation(1, 0, -4), [1.4142135623730951, -
1.4142135623730951])
    def test_3_roots(self):
        self.assertEqual(len(solve_biquadrate_equation(-4, 16, 0)), 3)
        self.assertEqual(solve_biquadrate_equation(-4, 16, 0), [0.0, 2.0, -2.0])
    def test_4_roots(self):
        self.assertEqual(len(solve_biquadrate_equation(1, -13, 36)), 4)
        self.assertEqual(solve_biquadrate_equation(1, -13, 36), [3.0, -3.0, 2.0, -2.0])


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

bddtests.feature

```
Feature: Testing biquadrate equation solution
  Scenario: Test 0 roots
    Given main is ran
    When a = 1, b = 0, c = 10
    Then we get 0 roots
  Scenario: Test 2 roots
    Given main is ran
    When a = 1, b = 0, c = -4
    Then we get 2 roots
  Scenario: Test 3 roots
    Given main is ran
    When a = -4, b = 16, c = 0
    Then we get 3 roots
  Scenario: Test 4 roots
    Given main is ran
    When a = 1, b = -13, c = 36
    Then we get 4 roots
```

bddtests.py

```
from behave import Given, When, Then
from main import solve_biquadrate_equation


@Given("main is ran")
def given_function(context):
    print("given func is ran")


@When("a = {a}, b = {b}, c = {c}")
```

```
def when_equation(context, a, b, c):
    context.result = len(solve_biquadrate_equation(int(a), int(b), int(c)))


@Then("we get {out} roots")
def then_result(context, out):
    assert(context.result == int(out))
```

Calculations.swift

```
//
//  Calculation.swift
//  Biquadrate
//
//  Created by MacBook on 30.09.2023.
//

import Foundation

struct Calculation {

    // # Linear Equation Solver
    func linearSolve(a: Double, b: Double) -> [Double] {
        if a == 0 {
            return []
        }

        return [Double(-b/a)]
    }


    // # Quadratic Equation Solver
    func quadraticSolve(a: Double, b: Double, c: Double, threshold: Double =
0.0001) -> [Double] {
        if a == 0 { return linearSolve(a: b, b: c) }

        var roots = [Double]()

        var d = pow(b, 2) - 4*a*c

        // Check if discriminate is within the 0 threshold
        if -threshold < d && d < threshold { d = 0 }

        if d > 0 {
            let x_1 = Double((-b + sqrt(d))/(2*a))
            let x_2 = Double((-b - sqrt(d))/(2*a))
            roots = [x_1, x_2]
        } else if d == 0 {
            let x = Double(-b/(2*a))
            roots = [x, x]
        }

        return roots
    }


    // # Biquadratic Equation Solver
    func biquadrateSolve(_ a: Double, _ b: Double, _ c: Double) -> [Double] {
        var result = [Double]()
        let solutions = quadraticSolve(a: a, b: b, c: c)
        for root in solutions {
            if root > 0 {
```

```
                result.append(-root.squareRoot())
                result.append(root.squareRoot())
            } else if root == 0 {
                result.append(0)
            }
        }
        return result
    }
}
```

WorkingBiquadrateTests.swift

```
//
//  CalculationTests.swift
//  BiquadrateTests
//
//  Created by MacBook on 30.09.2023.
//

import XCTest
@testable import Biquadrate

final class WorkingBiquadrateTests: XCTestCase {

    func testZeroRootsInBiqudrateEquation() {
        // Given (Arrange)
        let a: Double = 1
        let b: Double = 0
        let c: Double = 10
        let calculation = Calculation()

        // When (Act)
        let roots = calculation.biquadrateSolve(a, b, c)

        // Then (Assert)
        XCTAssertEqual(roots, [])
    }


    func testTwoRootsInBiqudrateEquation() {
        // Given (Arrange)
        let a: Double = 1
        let b: Double = 0
        let c: Double = -4
        let calculation = Calculation()

        // When (Act)
        let roots = calculation.biquadrateSolve(a, b, c)

        // Then (Assert)
        XCTAssertEqual(roots, [-1.4142135623730951, 1.4142135623730951])
    }


    func testThreeRootsInBiqudrateEquation() {
        // Given (Arrange)
        let a: Double = -4
        let b: Double = 16
        let c: Double = 0
        let calculation = Calculation()

        // When (Act)
        let roots = calculation.biquadrateSolve(a, b, c)

        // Then (Assert)
```

```
        XCTAssertEqual(roots, [0.0, -2.0, 2.0])
    }


    func testFourRootsInBiqudrateEquation() {
        // Given (Arrange)
        let a: Double = 1
        let b: Double = -13
        let c: Double = 36
        let calculation = Calculation()

        // When (Act)
        let roots = calculation.biquadrateSolve(a, b, c)

        // Then (Assert)
        XCTAssertEqual(roots, [-3.0, 3.0, -2.0, 2.0])
    }
}
```

WorkingQuickTests.swift

```
//
//  QuickTests.swift
//  QuickTests
//
//  Created by MacBook on 01.10.2023.
//

import Quick
import Nimble
@testable import Biquadrate

class WorkingQuickTests: QuickSpec {

    override class func spec() {

        var a: Double!
        var b: Double!
        var c: Double!
        var calculation: Calculation!

        beforeEach {
            calculation = Calculation()
        }

        describe("Biquadrate equation") {

            it("zero roots") {
                a = 1
                b = 0
                c = 10
                expect(calculation.biquadrateSolve(a, b, c)).to(equal([]))
            }

            it("two roots") {
                a = 1
                b = 0
                c = -4
                expect(calculation.biquadrateSolve(a, b, c)).to(equal([-
1.4142135623730951, 1.4142135623730951]))
            }

            it("three roots") {
                a = -4
                b = 16
```

```
                c = 0
                expect(calculation.biquadrateSolve(a, b, c)).to(equal([0.0, -2.0,
2.0]))
            }

            it("four roots") {
                a = 1
                b = -13
                c = 36
                expect(calculation.biquadrateSolve(a, b, c)).to(equal([-3.0, 3.0, -
2.0, 2.0]))
            }
        }
    }
}
```

Анализ результатов

```
Ran 4 tests in 0.008s

OK

Process finished with exit code 0
```

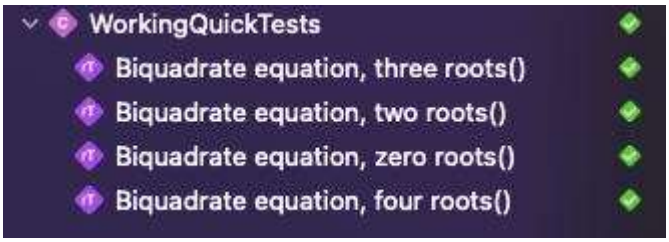
```
Ran 4 tests in 0.005s

FAILED (failures=1)


[1.4143135623730951, -1.4142135623730951] != [1.4142135623730951, -1.4142135623730951]

Expected :[1.4142135623730951, -1.4142135623730951]
Actual   :[1.4143135623730951, -1.4142135623730951]
```

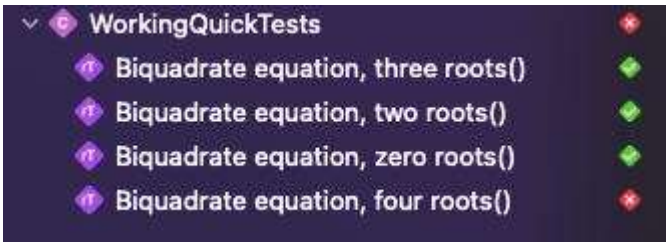
```
given func is ran
given func is ran
given func is ran
given func is ran

Process finished with exit code 0
```

```
given func is ran

Traceback (most recent call last):
  File "C:\Users\matve\AppData\Local\Programs\Python\Python311\Lib\site-packages\behave\model.py", line 1329, in run
    match.run(runner.context)
  File "C:\Users\matve\AppData\Local\Programs\Python\Python311\Lib\site-packages\behave\matchers.py", line 98, in run
    self.func(context, *args, **kwargs)
  File "features\steps\bddtests.py", line 17, in then_result
    assert(context.result == int(out))
           ^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError

given func is ran
given func is ran
given func is ran
```


WorkingBiquadrateTests
testZeroRootsInBiqudrateEquation()
testTwoRootsInBiqudrateEquation()
testThreeRootsInBiqudrateEquation()
testFourRootsInBiqudrateEquation()


WorkingBiquadrateTests
testZeroRootsInBiqudrateEquation()
testTwoRootsInBiqudrateEquation()
testThreeRootsInBiqudrateEquation()
testFourRootsInBiqudrateEquation()


WorkingQuickTests
Biquadrate equation, three roots()
Biquadrate equation, two roots()
Biquadrate equation, zero roots()
Biquadrate equation, four roots()


WorkingQuickTests
Biquadrate equation, three roots()
Biquadrate equation, two roots()
Biquadrate equation, zero roots()
Biquadrate equation, four roots()