**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: | Проверил: |
| студент группы ИУ5-31Б | преподаватель каф. ИУ5 |
| Зобнин Александр |  |
| Подпись и дата: | 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 А

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 А:')

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

# 



 

 

 