**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 №1

«Основные конструкции языка python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Егоров Виктор |  | Гапанюк Юрий |
|  |  |  |
|  |  |  |

Москва, 2024 г.

Постановка задачи

Разработать программу для решения биквадратного уравнения.

Программа должна быть разработана в виде консольного приложения на языке Python.

Программа осуществляет ввод с клавиатуры коэффициентов А, В, С, вычисляет дискриминант и ДЕЙСТВИТЕЛЬНЫЕ корни уравнения (в зависимости от дискриминанта).

Коэффициенты А, В, С могут быть заданы в виде параметров командной строки ( вариант задания параметров приведен в конце файла с примером кода ). Если они не заданы, то вводятся с клавиатуры в соответствии с пунктом 2. Описание работы с параметрами командной строки.

Если коэффициент А, В, С введен или задан в командной строке некорректно, то необходимо проигнорировать некорректное значение и вводить коэффициент повторно пока коэффициент не будет введен корректно. Корректно заданный коэффициент - это коэффициент, значение которого может быть без ошибок преобразовано в действительное число.

Дополнительное задание 1 (\*). Разработайте две программы на языке Python - одну с применением процедурной парадигмы, а другую с применением объектно-ориентированной парадигмы.

Дополнительное задание 2 (\*). Разработайте две программы - одну на языке Python, а другую на любом другом языке программирования (кроме С++).

**Текст программы**

**main.py**

import sys  
import math  
  
  
def get\_coef(index, prompt) -> float:  
 try:  
 try:  
 coef\_str = sys.argv[index]  
 coef = float(coef\_str)  
 return coef  
 except ValueError:  
 pass  
 except:  
 while True:  
 try:  
 print(prompt)  
 coef\_str = input()  
 coef = float(coef\_str)  
 return coef  
 except ValueError:  
 pass  
  
  
def get\_roots(a, b, c) -> list:  
 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 main():  
 a = get\_coef(1, 'Введите коэффициент А:')  
 b = get\_coef(2, 'Введите коэффициент B:')  
 c = get\_coef(3, 'Введите коэффициент C:')  
 roots = get\_roots(a, b, c)  
  
 count = 0  
 for el in roots:  
 if el > 0:  
 count += 1  
  
 finally\_roots = []  
 for el in roots:  
 if el > 0 or el == 1:  
 finally\_roots.append(math.sqrt(el))  
 finally\_roots.append(-1 \* math.sqrt(el))  
 elif el == 0:  
 finally\_roots.append(el)  
  
 len\_roots = len(finally\_roots)  
 if len\_roots == 0:  
 print('Корней нет')  
 elif len\_roots == 1:  
 print(f'Один корень: {abs(finally\_roots[0])}')  
 elif len\_roots == 2:  
 print('Два корня: {} и {}'.format(finally\_roots[0], finally\_roots[1]))  
 elif len\_roots == 3:  
 print('Три корня: {} и {} и {}'.format(finally\_roots[0], finally\_roots[1], finally\_roots[2]))  
 else:  
 print('Четыре корня: {} и {} и {} и {}'.format(finally\_roots[0], finally\_roots[1], finally\_roots[2],  
 finally\_roots[3]))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print('\n============ LAB-01 ============\n')  
 main()  
 print('\n================================\n')

**lab1\_oop.py**

import sys  
import math  
  
class EquationSolver:  
 def \_\_init\_\_(self, a, b, c):  
 self.a = a  
 self.b = b  
 self.c = c  
  
 def discriminant(self):  
 return self.b \*\* 2 - 4 \* self.a \* self.c  
  
 def solve(self):  
 D = self.discriminant()  
 if D < 0:  
 return []  
 elif D == 0:  
 root = -self.b / (2 \* self.a)  
 return [root] if root >= 0 else []  
 else:  
 sqD = math.sqrt(D)  
 root1 = (-self.b + sqD) / (2 \* self.a)  
 root2 = (-self.b - sqD) / (2 \* self.a)  
 return [root1, root2] if root1 >= 0 and root2 >= 0 else [root for root in [root1, root2] if root >= 0]  
  
 def find\_biquadratic\_roots(self):  
 quadratic\_roots = self.solve()  
 biquadratic\_roots = []  
 for root in quadratic\_roots:  
 if root > 0:  
 biquadratic\_roots.append(math.sqrt(root))  
 biquadratic\_roots.append(-math.sqrt(root))  
 elif root == 0:  
 biquadratic\_roots.append(0)  
 return biquadratic\_roots  
  
def get\_coef(index, prompt):  
 while True:  
 try:  
 coef = float(sys.argv[index])  
 return coef  
 except (IndexError, ValueError):  
 try:  
 coef = float(input(prompt))  
 return coef  
 except ValueError:  
 print("Некорректный ввод, попробуйте снова.")  
  
def main():  
 a = get\_coef(1, 'Введите коэффициент А: ')  
 b = get\_coef(2, 'Введите коэффициент B: ')  
 c = get\_coef(3, 'Введите коэффициент C: ')  
  
 solver = EquationSolver(a, b, c)  
 roots = solver.find\_biquadratic\_roots()  
  
 if not roots:  
 print("Действительных корней нет.")  
 else:  
 print(f"Найдены корни: {', '.join(map(str, roots))}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**lab1\_test.py**

import unittest  
from unittest.mock import patch  
import math  
import sys  
  
*# Импортируем функции, которые будем тестировать*from main import get\_coef, get\_roots, main  
  
  
class TestQuadraticEquation(unittest.TestCase):  
  
 def test\_get\_coef\_from\_argv(self):  
 *"""Тестируем получение коэффициента из аргументов командной строки"""* sys.argv = ['script\_name', '1.0', '2.0', '3.0']  
 self.assertEqual(get\_coef(1, "Введите коэффициент A:"), 1.0)  
 self.assertEqual(get\_coef(2, "Введите коэффициент B:"), 2.0)  
 self.assertEqual(get\_coef(3, "Введите коэффициент C:"), 3.0)  
  
 @patch('builtins.input', side\_effect=['4.0'])  
 def test\_get\_coef\_from\_input(self, mock\_input):  
 *"""Тестируем получение коэффициента через ввод, если аргумента нет"""* sys.argv = ['script\_name']  
 self.assertEqual(get\_coef(1, "Введите коэффициент A:"), 4.0)  
  
 def test\_get\_roots\_two\_roots(self):  
 *"""Тестируем получение двух корней"""* self.assertEqual(get\_roots(1, -3, 2), [2.0, 1.0])  
  
 def test\_get\_roots\_one\_root(self):  
 *"""Тестируем получение одного корня"""* self.assertEqual(get\_roots(1, 2, 1), [-1.0])  
  
 def test\_get\_roots\_no\_roots(self):  
 *"""Тестируем случай без вещественных корней"""* self.assertEqual(get\_roots(1, 0, 1), [])  
  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

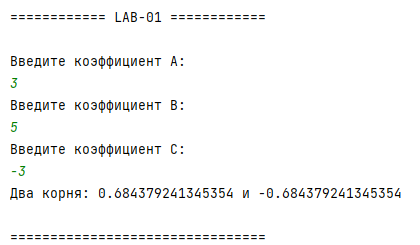
**main.feature**

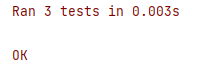
Feature: Solving Bi-Quadratic Equations  
  
 Scenario: No real roots for bi-quadratic equation  
 Given the coefficients a = 1, b = 0, c = 1  
 When I solve the bi-quadratic equation  
 Then the result should be "Корней нет"  
  
 Scenario: One real root for bi-quadratic equation  
 Given the coefficients a = 1, b = 0, c = 0  
 When I solve the bi-quadratic equation  
 Then the result should contain "Один корень: 0"  
  
 Scenario: Four real roots for bi-quadratic equation  
 Given the coefficients a = 1, b = -5, c = 4  
 When I solve the bi-quadratic equation  
 Then the result should contain "Четыре корня: 2.0 и -2.0 и 1.0 и -1.0"

**lab1\_bdd.py**

from behave import given, when, then  
import sys  
import io  
from main import main  
  
*# Mocking input and output for testing*def run\_main\_with\_args(args):  
 sys.argv = ['main.py'] + args  
 captured\_output = io.StringIO()  
 sys.stdout = captured\_output  
 main()  
 sys.stdout = sys.\_\_stdout\_\_  
 return captured\_output.getvalue().strip()  
  
@given('the coefficients a = {a}, b = {b}, c = {c}')  
def step\_given\_coefficients(context, a, b, c):  
 context.coefficients = [a, b, c]  
  
@when('I solve the bi-quadratic equation')  
def step\_when\_solve\_equation(context):  
 context.output = run\_main\_with\_args(context.coefficients)  
  
@then('the result should be "{expected\_result}"')  
def step\_then\_check\_result(context, expected\_result):  
 assert context.output == expected\_result, f"Expected '{expected\_result}', but got '{context.output}'"  
  
@then('the result should contain "{expected\_result}"')  
def step\_then\_check\_result\_contains(context, expected\_result):  
 assert expected\_result in context.output, f"Expected '{expected\_result}' to be in '{context.output}'"

**Обработка результатов**

****

****