|  |  |  |
| --- | --- | --- |
| Защищено:  Гапанюк Ю.Е.    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2023** г. |  | Демонстрация:  Падалко К.Р.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2023** г. |

**Отчет по лабораторной работе № 3 по курсу**

**Парадигмы и конструкции языков программирования**

#### Тема работы: " Модульное тестирование в Python. "

5

(количество листов)

Вариант № **23**

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5Ц-54Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Падалко К.Р. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2023**  г. |

Москва, МГТУ - **2023**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание

[1. Описание задания 3](#_Toc153577991)

[2. Текст программы 3](#_Toc153577992)

[3. Экранные формы с примерами выполнения программы 5](#_Toc153577993)

# **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).

# **Текст программы**

Файл «module.py»

﻿﻿import sys  
import cmath  
#ЛР1  
  
# Функция для ввода коэфицентов и проверки на ошибки  
def get\_coeff(prompt, coef\_name):  
 while True:  
 try:  
 coef = float(input(prompt))  
 return coef  
 except ValueError:  
 print(f"Некорректное значение для коэффициента {coef\_name}. Повторите ввод.")  
  
  
# Функция для вычисления корней уравнения  
def solve\_biquadratic(a, b, c):  
 discriminant = b \*\* 2 - 4 \* a \* c  
  
 if discriminant > 0:  
 root1 = cmath.sqrt((-b + cmath.sqrt(discriminant)) / (2 \* a))  
 root2 = -root1  
 root3 = cmath.sqrt((-b - cmath.sqrt(discriminant)) / (2 \* a))  
 root4 = -root3  
 return root1, root2, root3, root4  
 elif discriminant == 0:  
 root = cmath.sqrt(-b / (2 \* a))  
 return root, -root  
 else:  
 return ()  
  
  
# Функция поиска действительных корней  
def findTrueRoots(roots):  
 trueRoots = []  
 for root in roots:  
 # Проверка на действительность и на дубликатность ( 0 и -0)  
 if (root.imag == 0) and (str(root.real) != "-0.0"):  
 trueRoots.append(root.real)  
 return trueRoots  
  
  
def printTrueRoots(trueRoots):  
 if len(trueRoots) == 4:  
 print(  
 f"Уравнение имеет четыре действительных корня: {trueRoots[0]}, {trueRoots[1]}, {trueRoots[2]}, {trueRoots[3]}")  
 elif len(trueRoots) == 3:  
 print(f"Уравнение имеет три действительных корня: {trueRoots[0]}, {trueRoots[1]}, {trueRoots[2]}")  
 elif len(trueRoots) == 2:  
 print(f"Уравнение имеет два действительных корня: {trueRoots[0]}, {trueRoots[1]}")  
 elif len(trueRoots) == 1:  
 print(f"Уравнение имеет один действительный корень: {trueRoots[0]}")  
 else:  
 print("Уравнение не имеет действительных корней.")  
  
  
def main():  
 # Проверка ввода из консоли и получение аргументов  
 if len(sys.argv) == 4:  
 try:  
 a = float(sys.argv[1])  
 b = float(sys.argv[2])  
 c = float(sys.argv[3])  
 except ValueError:  
 print("Некорректные коэффициенты в командной строке. Пожалуйста, введите их с клавиатуры.")  
 a = get\_coeff("Введите коэффициент A: ", "A")  
 b = get\_coeff("Введите коэффициент B: ", "B")  
 c = get\_coeff("Введите коэффициент C: ", "C")  
 else:  
 a = get\_coeff("Введите коэффициент A: ", "A")  
 b = get\_coeff("Введите коэффициент B: ", "B")  
 c = get\_coeff("Введите коэффициент C: ", "C")  
   
 roots = solve\_biquadratic(a, b, c)  
   
 trueRoots = findTrueRoots(roots)  
 printTrueRoots(trueRoots)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Файл «test.py»

# Тесты TDD  
  
from module import findTrueRoots, solve\_biquadratic  
import math  
  
def test\_solve\_biquadratic\_real\_roots():  
 # Test case with real roots  
 roots = solve\_biquadratic(1, -3, 2)  
  
 expected\_roots = [math.sqrt(2), -math.sqrt(2), 1.0, -1.0]   
 assert findTrueRoots(roots) == expected\_roots  
  
def test\_solve\_biquadratic\_complex\_roots():  
 # Test case with complex roots  
 roots = solve\_biquadratic(1, 1, 1)  
 expected\_roots = () # no real roots  
 assert roots == expected\_roots  
  
def test\_find\_true\_roots():  
 # Test case for find\_true\_roots function  
 roots = [1 + 0j, 2 + 0j, 0 + 1j, -1 - 1j] # mix of real and complex roots  
 true\_roots = findTrueRoots(roots)  
 expected\_true\_roots = [1, 2]  
 assert true\_roots == expected\_true\_roots

# **3.**