**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №1

Основные конструкции языка Python.

| Выполнил: |  | Проверил: |
| --- | --- | --- |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Герасименко Д. Д. |  |  |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2023 г.

**Постановка задачи**

Цель лабораторной работы: изучение основных конструкций языка Python.

**Задание:**

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 biquadratic\_procedure.py.

Задача решена с помощью процедурного подхода.

import sys

import math

Testing: bool

def get\_from\_stdin() -> list:

while True:

try:

a, b, c = map(float, input("enter coefficients a, b and c: ").split())

except:

print("incorrect input, you must provide 3 numbers: ")

continue

else:

break

return [a, b, c]

def get\_from\_cmdln(cli\_args: list) -> list:

return [cli\_args[1], cli\_args[2], cli\_args[3]]

def get\_coeffs(cli\_args: list) -> list:

a, b, c = None, None, None

if (len(cli\_args) == 4):

try:

a, b, c = map(float, get\_from\_cmdln(cli\_args))

except:

print("incorrect arguments provided. reading from stdin.")

return get\_from\_stdin()

else:

return [a, b, c]

print("incorrect or no arguments provided, reading from stdin.")

return get\_from\_stdin()

def try\_get\_pmroot(expr: float) -> list:

if (expr < 0):

return [None, None]

else:

return [-math.sqrt(expr), math.sqrt(expr)]

def solve(cli\_args: list) -> set:

if (not Testing):

a, b, c = get\_coeffs(cli\_args)

else:

a, b, c = map(float, cli\_args)

answer = set()

if (a == 0):

if (b == 0 and c == 0):

return {math.inf}

elif (b == 0 and c != 0):

return set()

elif (b != 0):

x\_1, x\_2 = try\_get\_pmroot((-c)/b)

return {x\_1, x\_2}

else:

discr = b\*\*2 - 4 \* a \* c

if (discr < 0):

return set()

elif (discr == 0):

x\_1, x\_2 = try\_get\_pmroot((-b) / (2 \* a))

return {sol for sol in {x\_1, x\_2} if sol != None}

else:

discr\_root\_1, discr\_root\_2 = try\_get\_pmroot(discr)

x\_1, x\_2 = try\_get\_pmroot((-b + discr\_root\_1) / (2 \* a))

x\_3, x\_4 = try\_get\_pmroot((-b + discr\_root\_2) / (2 \* a))

return {sol for sol in {x\_1, x\_2, x\_3, x\_4} if sol != None}

return answer

if \_\_name\_\_ == "\_\_main\_\_":

Testing = False

answer\_set = solve(sys.argv)

print(\*answer\_set)

Текст программы biquadratic\_oop.py.

Задача решена с помощью объектно ориентированного подхода.

import sys

import math

Testing : bool

def try\_get\_pmroot(expr: float) -> list:

if (expr < 0):

return [None, None]

else:

return [-math.sqrt(expr), math.sqrt(expr)]

class Solution:

solution\_set : set = set()

solution\_amount : int = 0

def add\_solution(self, solution):

self.solution\_set.add(solution)

self.solution\_amount += 1

def \_\_check\_if\_inf(self):

for sol in self.solution\_set:

if (sol == math.inf):

self.solution\_amount = int(math.inf)

class Equation:

a : float = math.nan

b : float = math.nan

c : float = math.nan

solution : Solution = Solution()

def \_\_init\_\_(self, a = 0.0, b = 0.0, c = 0.0):

self.a = a

self.b = b

self.c = c

def \_\_get\_from\_stdin(self):

while True:

try:

a, b, c = map(float, input("enter coefficients a, b and c: ").split())

except:

print("incorrect input, you must provide 3 numbers: ")

continue

else:

break

self.a = a

self.b = b

self.c = c

def \_\_get\_from\_cmdln(self, cli\_args: list):

if not Testing:

self.a = float(cli\_args[1])

self.b = float(cli\_args[2])

self.c = float(cli\_args[3])

else:

self.a = float(cli\_args[0])

self.b = float(cli\_args[1])

self.c = float(cli\_args[2])

def get\_coeffs(self, cli\_args: list):

try:

self.\_\_get\_from\_cmdln(cli\_args)

except:

print("incorrect arguments provided. reading from stdin.")

self.\_\_get\_from\_stdin()

def solve(self):

if (self.a == 0):

if (self.b == 0 and self.c == 0):

self.solution.add\_solution(math.inf)

return

elif (self.b == 0 and self.c != 0):

return

elif (self.b != 0):

x\_1, x\_2 = try\_get\_pmroot((-self.c)/self.b)

self.solution.add\_solution(x\_1)

self.solution.add\_solution(x\_2)

else:

discr = self.b\*\*2 - 4 \* self.a \* self.c

if (discr < 0):

return

elif (discr == 0):

x\_1, x\_2 = try\_get\_pmroot((-self.b) / (2 \* self.a))

if (x\_1 != None): self.solution.add\_solution(x\_1)

if (x\_2 != None): self.solution.add\_solution(x\_2)

else:

discr\_root\_1, discr\_root\_2 = try\_get\_pmroot(discr)

x\_1, x\_2 = try\_get\_pmroot((-self.b + discr\_root\_1) / (2 \* self.a))

x\_3, x\_4 = try\_get\_pmroot((-self.b + discr\_root\_2) / (2 \* self.a))

for i in {x\_1, x\_2, x\_3, x\_4}:

if i != None:

self.solution.add\_solution(i)

def print\_solution(self):

if (self.solution.solution\_amount == 0):

print("no solutions")

elif (self.solution.solution\_amount == math.nan):

print("inf amount of solutions")

else:

print(\*self.solution.solution\_set)

if \_\_name\_\_ == "\_\_main\_\_":

Testing = False

equation : Equation = Equation()

equation.get\_coeffs(sys.argv)

equation.solve()

equation.print\_solution()

Также к каждому варианту решения задачи написаны тесты.   
Запустить их можно с помощью скрипта test.py

Текст программы biquadratic\_meta.cc

Задача решена с помощью т.н метапрограммирования применительно к C++,

то есть по сути, уравнение решено еще до запуска программы, т. к. ответы известны на этапе компиляции.

#include <iostream>

#include <limits>

#include <tuple>

#include <utility>

#ifndef A

#define A (0)

#endif

#ifndef B

#define B (0)

#endif

#ifndef C

#define C (0)

#endif

namespace detail {

double constexpr sqrtNewtonRaphson(double x, double curr, double prev) {

return curr == prev ? curr

: sqrtNewtonRaphson(x, 0.5 \* (curr + x / curr), curr);

}

} // namespace detail

double constexpr ce\_sqrt(double x) {

return x >= 0 && x < std::numeric\_limits<double>::infinity()

? detail::sqrtNewtonRaphson(x, x, 0)

: std::numeric\_limits<double>::quiet\_NaN();

}

template <typename TupleType, std::size\_t... Idxs>

void print(const TupleType& tuple, std::index\_sequence<Idxs...> /\*unused\*/) {

std::cout << "(";

(..., (std::cout << (Idxs == 0 ? "" : ", ") << std::get<Idxs>(tuple)));

std::cout << ")\n";

}

template <typename... T>

void print(const std::tuple<T...>& tuple) {

print(tuple, std::make\_index\_sequence<sizeof...(T)>());

}

template <int N>

static constexpr bool is\_positive = N > 0;

template <int a, int b, int c>

struct solve {

static constexpr std::tuple<double, double, double, double> answer =

{ce\_sqrt((-b + ce\_sqrt(b \* b - 4 \* a \* c)) / (2 \* a)),

ce\_sqrt((-b - ce\_sqrt(b \* b - 4 \* a \* c)) / (2 \* a)),

-ce\_sqrt((-b + ce\_sqrt(b \* b - 4 \* a \* c)) / (2 \* a)),

-ce\_sqrt((-b - ce\_sqrt(b \* b - 4 \* a \* c)) / (2 \* a))};

};

template <int b, int c>

struct solve<0, b, c> {

static constexpr std::tuple<double, double> answer =

is\_positive<c> ? std::tuple<double, double>{std::numeric\_limits<double>::quiet\_NaN(),

std::numeric\_limits<double>::quiet\_NaN()}

: std::tuple<double, double>{ce\_sqrt(static\_cast<double>(-c)/b),

-ce\_sqrt(static\_cast<double>(-c)/b)};

};

template <int c>

struct solve<0, 0, c> {

static constexpr std::tuple<double> answer = {std::numeric\_limits<double>::quiet\_NaN()};

};

template <>

struct solve<0, 0, 0> {

static constexpr std::tuple<double> answer = {std::numeric\_limits<double>::infinity()};

};

int main(int argc, char\*\* argv) {

print(solve<A, B, C>::answer);

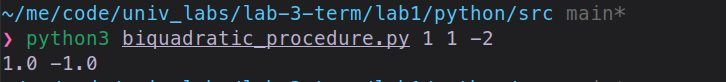
return EXIT\_SUCCESS;

}

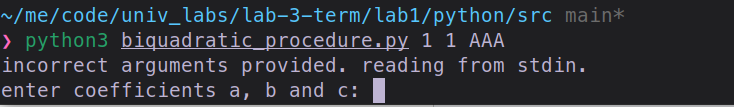
Примеры выполнения программ.

biquadratic\_procedure.py

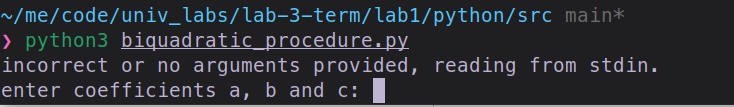
correct cmdline input:



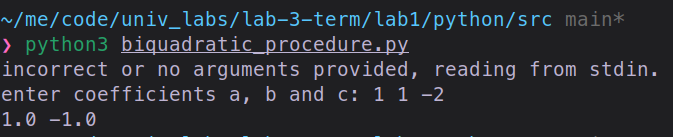
incorrect cmdline input:



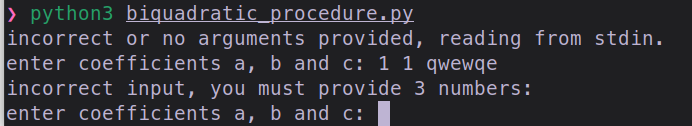
no cmdline arguments:



correct stdin arguments:



incorrect stdin arguments:



все то же самое применимо и к программе biquadratic\_oop.py