**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Информатика»**

Тема: Основные управляющие конструкции языка Python

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Иванов Д. М. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2023

## Цель работы

Изучить основные управляющие конструкции языка Python. С их помощью написать. С их помощью написать программу, состоящую из трех функций, каждя из которых проводит алгебраические операции.

## Задание

Задача 1.

Оформите решение в виде отдельной функции check\_collision. На вход функции подаются два ndarray -- коэффициенты bot1, bot2 уравнений прямых bot1 = (a1, b1, c1), bot2 = (a2, b2, c2) (уравнение прямой имеет вид ax+by+c=0).

Функция должна возвращать точку пересечения траекторий (кортеж из 2 значений), предварительно округлив координаты до 2 знаков после запятой с помощью round(value, 2).

Задача 2.

Оформите задачу как отдельную функцию check\_surface, на вход которой передаются координаты 3 точек (3 ndarray 1 на 3): point1, point2, point3. Функция должна возвращать коэффициенты a, b, с в виде ndarray для уравнения плоскости вида ax+by+c=z. Перед возвращением результата выполнение округление каждого коэффициента до 2 знаков после запятой с помощью round(value, 2).

Задача 3.

Оформите задачу как отдельную функцию check\_surface, на вход которой передаются координаты 3 точек (3 ndarray 1 на 3): point1, point2, point3. Функция должна возвращать коэффициенты a, b, с в виде ndarray для уравнения плоскости вида ax+by+c=z. Перед возвращением результата выполнение округление каждого коэффициента до 2 знаков после запятой с помощью round(value, 2).

## Выполнение работы

Для работы с алгебраическими операциями, массивами и матрицами были использованы библиотеки *numpy* и *math*. Рассмотрим каждую функцию в отдельности.

1. *def check\_collision(bot1, bot2)*: Необходмо найти точку пересечения 2-х прямых. Функция принимает на вход два массива – коэффициенты уравнений прямых. Из элементов этих массивов была составлена система из двух линейных уравнений, которая решается методом *linalg.solve*.
2. *def check\_surface(point1, point2, point3)*: Необходимо найти уравнение плоскости по 3-м точкам. Аргументы функции – массивы, в которых содержатся координаты этих точек. Созданы две матрицы: матрица коэффициентов и вектор свободных членов. Тем же методом *numpy* решаем состему из трех уравнений и находим коэффициенты уравнения плосоксти вида *ax+by+c=z*. Перед этим существование решения системы проверяет метод *linalg.matrix\_rank*.
3. *def check\_rotation(vec, rad)*: Необходимо найти координаты матрицы, повернутой на угол *rad*. Так как известно, что поворот осуществляется вокруг оси z, то составим формулу матрицы поворота (см. рис. 1). Результатом будет умножение этой матрицы на исходную.

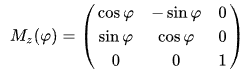


Рисунок 1 - Матрица поврота вокруг оси z

Переменные:

* a1, a2, a3, b1, b2, b3, z1, z2, z3 – элементы, на которые разбиваются переданные в качестве аргумента массивы
* arr\_1, arr\_2, arr

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | make  ./menu  0 1 16 2 -18 -22 15 -3 13 0 -6 1 9 24 1 -18 15 28 20 -17 16 -11 | gcc -c -std=c99 menu.c  gcc -c -std=c99 index\_first\_negative.c  gcc -c -std=c99 index\_last\_negative.c  gcc -c -std=c99 sum\_between\_negative.c  gcc -c -std=c99 sum\_before\_and\_after\_negative.c  gcc -c -std=c99 output.c  gcc -c -std=c99 input.c  gcc menu.o index\_first\_negative.o index\_last\_negative.o sum\_between\_negative.o sum\_before\_and\_after\_negative.o output.o input.o -o menu  3 | Верный вывод без ошибок |
|  | 1 1 16 2 -18 -22 15 -3 13 0 -6 1 9 24 1 -18 15 28 20 -17 16 -11 | gcc -c -std=c99 menu.c  gcc -c -std=c99 index\_first\_negative.c  gcc -c -std=c99 index\_last\_negative.c  gcc -c -std=c99 sum\_between\_negative.c  gcc -c -std=c99 sum\_before\_and\_after\_negative.c  gcc -c -std=c99 output.c  gcc -c -std=c99 input.c  gcc menu.o index\_first\_negative.o index\_last\_negative.o sum\_between\_negative.o sum\_before\_and\_after\_negative.o output.o input.o -o menu  20 | Верный вывод без ошибок |
|  | 2 1 16 2 -18 -22 15 -3 13 0 -6 1 9 24 1 -18 15 28 20 -17 16 -11 | gcc -c -std=c99 menu.c  gcc -c -std=c99 index\_first\_negative.c  gcc -c -std=c99 index\_last\_negative.c  gcc -c -std=c99 sum\_between\_negative.c  gcc -c -std=c99 sum\_before\_and\_after\_negative.c  gcc -c -std=c99 output.c  gcc -c -std=c99 input.c  gcc menu.o index\_first\_negative.o index\_last\_negative.o sum\_between\_negative.o sum\_before\_and\_after\_negative.o output.o input.o -o menu  226 | Верный вывод без ошибок |
|  | 3 1 16 2 -18 -22 15 -3 13 0 -6 1 9 24 1 -18 15 28 20 -17 16 -11 | gcc -c -std=c99 menu.c  gcc -c -std=c99 index\_first\_negative.c  gcc -c -std=c99 index\_last\_negative.c  gcc -c -std=c99 sum\_between\_negative.c  gcc -c -std=c99 sum\_before\_and\_after\_negative.c  gcc -c -std=c99 output.c  gcc -c -std=c99 input.c  gcc menu.o index\_first\_negative.o index\_last\_negative.o sum\_between\_negative.o sum\_before\_and\_after\_negative.o output.o input.o -o menu  30 | Верный вывод без ошибок |

## Выводы

Был разработан проект с использование утилиты *make*, разбитый на несколько файлов и выполняющий считывание с клавиатуры исходных данных, которые добавляются в массив, и команды пользователя. В зависимости от этой команды выполняется тот или иной алгоритм и выводится соответствующий результат. Изучены и проработаны методы сборки программы в Си, компиляция и линоковка программы.

# Приложение А Исходный код программы

Название файла: main.py

import numpy as np

from math import sin, cos

def check\_collision(bot1, bot2):

a1, a2, a3 = map(int, bot1)

b1, b2, b3 = map(int, bot2)

arr\_1 = np.array([[a1, a2], [b1, b2]])

arr\_2 = np.array([-a3, -b3])

result = np.linalg.solve(arr\_1, arr\_2)

return (round(result[0], 2), round(result[1], 2))

def check\_surface(point1, point2, point3):

a1, b1, z1 = map(int, point1)

a2, b2, z2 = map(int, point2)

a3, b3, z3 = map(int, point3)

arr\_1 = np.array([[a1, b1, 1], [a2, b2, 1], [a3, b3, 1]])

arr\_2 = np.array([z1, z2, z3])

if np.linalg.matrix\_rank(arr\_1) < 3:

return None

result = np.linalg.solve(arr\_1, arr\_2)

return np.array([round(result[0], 2), round(result[1], 2), round(result[2], 2)])

def check\_rotation(vec, rad):

arr = np.array([[cos(rad), -sin(rad), 0], [sin(rad), cos(rad), 0], [0, 0, 1]])

result = np.dot(arr, vec)

return np.array([round(result[0], 2), round(result[1], 2), round(result[2], 2)])