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, c 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, c 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 плосокости 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.

$$M_z(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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
1.	check_collision(array([-3, -6, 9]), array([8, -7, 0]))	(0.91, 1.04)	Верный вывод
2.	check_surface(array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1]))	[2. 1. 5.]	Верный вывод
3.	check_rotation(array([1, -2, 3]), 1.57)	[2. 1. 3.]	Верный вывод

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])
    if np.linalg.matrix_rank(arr_1) < 2:
        return None
    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)])
```