**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: Управляющие конструкции языка Python**

Студент гр. 3344 _____ Коршунов П.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

**Цель работы**

Освоение работы с управляющими конструкциями на языке Python.

**Задание.**

Вариант 1. Вариант лабораторной работы состоит из 3 задач, оформите каждую задачу в виде отдельной функции согласно условиям задач. Приветствуется использование модуля *numpy*, в частности пакета *numpy.linalg*. Вы можете реализовывать вспомогательные функции, главное -- использовать те же названия основных функций, что требуются в задании. Сами функции вызывать не надо, это делает за вас проверяющая система.

Задача 1. Два дакибота приближаются к перекрестку. Чтобы избежать столкновения, им необходимо знать точку пересечения их траекторий движения. Траектории -- линейные, и дакиботы уже вычислили коэффициенты этих уравнений. Ваша задача -- помочь ботам вычислить точку потенциального столкновения. 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. Три дакибота начали движение, отъехали от условной точки старта и через некоторое время остановились. Каждый дакибот уже вычислил свою координату относительно точки старта. Дакиботам нужно передать на базу карту местностности, по которой они двигались. Для построения карты местности необходимо знать уравнение плоскости. Ваша задача -- помочь дакиботам найти уравнение плоскости, в которой они двигались. 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. Дакибот выехал на перекресток и готовится к выполнению поворота вокруг своей оси (вокруг оси z), чтобы продолжить движение в другом направлении. Он знает свои координаты и знает угол поворота (в радианах). Помогите дакиботу повернуться в нужное направление для продолжения движения. Оформите решение в виде отдельной функции *check rotation*. На вход функции подаются *ndarray* 3-х координат дакибота и угол поворота. Функция возвращает повернутые *ndarray* координаты, каждая из которых округлена до 2 знаков после запятой с помощью *round(value, 2)*.

**Выполнение работы**

Была импортирована библиотека *numpy*.

Была реализована функция *def check_collision(bot1, bot2)*, принимающая на вход два *ndarray* – коэффициенты *bot1*, *bot2* уравнений прямых *bot1* = (*a1, b1, c1*), bot2 = (*a2, b2, c2*), уравнение прямой имеет вид *ax+by+c=0*. В условном выражении *if* с помощью функции *np.linalg.matrix_rank([bot1, bot2])* было проверено, что ранг матрицы, состоящей из этих уравнений, имеет допустимое для нахождения решения значение. Если условие *if np.linalg.matrix_rank([bot1, bot2]) == 2* не выполнялось, то функция возвращала *None*, иначе с помощью функции *np.linalg.solve([bot1[:2], bot2[:2]], [-bot1[2], -bot2[2]])* была получена точка пересечения траекторий. В функцию *np.linalg.solve()* аргументы переданы именно так, потому что на вход она должна получать матрицу коэффициентов и вектор свободных членов. Далее значения точки были округлены до 2 знаков после запятой с помощью *np.round(),* преобразованы в кортеж и возвращены. *return tuple(np.round(np.linalg.solve([bot1[:2], bot2[:2]], [-bot1[2], -bot2[2]]), 2))*

В функции *def check_surface(point1, point2, point*3), принимающей на вход три *ndarray* – координаты дакиботов, было реализовано нахождение коэффициентов *a, b, c* в виде *ndarray* для уравнения плоскости вида *ax+by+c=z*. Была создана матрица три на три, состоящая из единиц *matrix = np.ones((3, 3))* и пустой массив *vector = []*. Далее циклом *for* была заполнена матрица коэффициентов и вектор свободных членов:

```
for index, point in enumerate([point1, point2, point3]):
    matrix[index, :2] = point[:2]
    vector.append(point[2])
```

В условном выражении *if* с помощью функции *np.linalg.matrix_rank(matrix)* было проверено, что ранг матрицы коэффициентов имеет допустимое для нахождения решения значение. Если условие *if np.linalg.matrix rank(matrix) == 3* не выполнялось, то функция возвращала *None*, иначе с помощью функции *np.linalg.solve()* были получены искомые коэффициенты, которые были округлены до 2 знаков после запятой с помощью *np.round()* и возвращены. *return np.round(np.linalg.solve(matrix, vector), 2)*

Была реализована функция *def check_rotation(vec, rad)*, принимающая на вход *ndarray* 3-х координат дакибота и угол поворота. Были инициализированы значения косинуса и синуса угла поворота с помощью функций *np.sin()* и *np.cos()*: *sin = np.sin(rad), cos = np.cos(rad)*. Была составлена матрица поворота в трёхмерном пространстве оп оси *z*: *rotation_matrix = np.array([[cos, -sin, 0], [sin, cos, 0], [0, 0, 1]])*. Для получения повернутых координат необходимо умножить матрицу поворота на вектор координат дакибота. Для этого был использован оператор *@,* который является аналогом функции *np.matmul()*. Повернутые координаты бы были округлены до 2 знаков после запятой с помощью *np.round()* и возвращены. *return np.round(rotation_matrix @ vec, 2)*

**Тестирование.**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | check_collision(np.array([-3, -6, 9]), np.array([8, -7, 0])) | (0.91, 1.04) | - |
| 2. | check_surface(np.array([1, -6, 1]), np.array([0, -3, 2]), np.array([-3, 0, -1])) | [2. 1. 5.] | - |
| 3. | check_rotation(np.array([1, -2, 3]), 1.57) | [2. 1. 3.] | - |

**Выводы**

Была освоена работа с управляющими конструкциями на языке Python. Были получены базовые навыки работы с пакетом *numpy*. Были освоены функции округления, решения линейных систем уравнения, умножения матриц, получения значений синуса и косинуса, получения ранга матрицы.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Korshunov_Petr_lb1.py

```
import numpy as np


def check_collision(bot1, bot2):
    if np.linalg.matrix_rank([bot1, bot2]) == 2:
        return  tuple(np.round(np.linalg.solve([bot1[:2],  bot2[:2]],  [-
bot1[2], -bot2[2]]), 2))
    else:
        return None


def check_surface(point1, point2, point3):
    matrix = np.ones((3, 3))
    vector = []
    for index, point in enumerate([point1, point2, point3]):
        matrix[index, :2] = point[:2]
        vector.append(point[2])
    if np.linalg.matrix_rank(matrix) == 3:
        return np.round(np.linalg.solve(matrix, vector), 2)
    else:
        return None


def check_rotation(vec, rad):
    sin = np.sin(rad)
    cos = np.cos(rad)
    rotation_matrix = np.array([[cos, -sin, 0], [sin, cos, 0], [0, 0, 1]])
    return np.round(rotation_matrix @ vec, 2)
```