**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**

Студентка гр. 3343 _____ Синицкая Д.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

**Цель работы**

Научиться использовать основные управляющие конструкции языка программирования *python*, оформлять функции в языке программирования *python*, работать с модулем *numpy*.

**Задание**

Вариант 2. 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. Содержательная постановка задачи: дакибот приближается к перекрестку. Он знает 4 координаты, соответствующие координатам углов перекрестка (координаты образуют прямоугольник), и свои координаты. По правилам движения дакибот должен остановиться сразу, как только оказывается на перекрестке. Ваша задача -- помочь дакиботу понять, находится ли он на перекрестке (внутри прямоугольника).

Формальная постановка задачи: оформите задачу как отдельную функцию: *def check_crossroad(robot, point1, point2, point3, point4)*. На вход функции подаются: координаты дакибота *robot* и координаты точек, описывающих перекресток: *point1, point2, point3, point4*. Точка - это кортеж из двух целых чисел (x, y). Функция должна возвращать True, если дакибот на перекрестке, и False, если дакибот вне перекрестка.

Задача 2. Содержательная часть задачи: несколько дакиботов прибыли на базу, но их корпуса оказались поврежденными. В логах ботов программисты нашли сведения про их траектории движения, которые задаются линейными уравнениями вида: ax+by+c=0. В логах хранятся коэффициенты этих уравнений a, b, c. Ваша задача -- вывести список номеров ботов (кортежи), которые столкнулись с друг другом (боты нумеруются с нуля, порядок следования коэффициентов уравнений соответствует порядку ботов).

Формальная постановка задачи: оформите решение в виде отдельной функции *check_collision()*. На вход функции подается матрица ndarray Nx3 (N

- количество ботов, может быть разным в разных тестах) коэффициентов уравнений траекторий *coefficients*. Функция возвращает список пар - номера столкнувшихся ботов (если никто из ботов не столкнулся, возвращается пустой список).

Задача 3. Содержательная часть задачи: при перемещении по дакитауну дакибот должен регулярно отправлять на базу сведения, среди которых есть длина пройденного пути. Дакиботу известна последовательность своих координат (x, y), по которым он проехал. Ваша задача -- помочь дакиботу посчитать длину пути.

Формальная постановка задачи: оформите задачу как отдельную функцию *check_path()*, на вход которой передается последовательность (список) двумерных точек (пар) *points_list*. Функция должна возвращать число - длину пройденного дакиботом пути (выполните округление до 2 знака с помощью *round(value, 2)).*

**Выполнение работы**

Задача 1. В решении были использованы: условная конструкция *if-else*; логические операторы *and, or*; условные операторы <=, >=.

Задача 2. В решении были использованы: список *list_pars*; цикл *for*; оператор *continue*; матрицы *M1*, *M2*.

Логика программы построенна на том, что в матрице *M1* содержатся коэффициенты двух дакиботов, где каждая строка матрицы соответсвует одному дакиботу, а столбцы содержат коэффициенты a и b, так как траектория линейная, то коэфициенты в матрице формируют систему уравнений, представляющую их траектории. В матрице *M2* каждая строка содержит коэффициент c для соответствующего дакибота.

Коэффициент c необходим для проверки на то, что траетории дакиботов линейны. Если ранг матрицы *M1* меньше или равен рангу матрицы *M2*, то система уравнений, представляющая собой линейные траектории дакиботов, не имеет решений, а значит, траектории не пересекаются, соответсвенно эта пара дакиботов не подходит. Если ранг матрицы *M1* больше ранга матрицы *M2*, то система уравнений, представляющая собой линейные траектории дакиботов, имеет решение, а значит, траектории пересекаются, соответсвенно эта пара дакиботов подходит.

В функции был использован модуль *numpy*, а также функции модуля *numpy: np.array()* - используется для создания матрицы; *np.linalg.matrix_rank()* - используется для получения ранга матриц; *np.any()* - используется для определения содержат ли матрицы хотя бы один элемент отличный от нуля.

Задача 3. В решении были использованы: списки *lenght, points_list*; цикл *for*; функция *sqrt* модуля *math*; встроенная функция *round()*.

Разработанный программный код см. в приложении А.

**Выводы**

Приобрела навыки использования основных управляющих конструкций языка программирования *python*, оформления функций в языке программирования *python*, работе с модулем *numpy*.

В лабораторной работе было реализованно три функции. Функция *check_crossroad()*, определяющая нахождение дакибота на перекрестке. Функция *check_collision()*, определяющая какие дакиботы столкнулись. Функция *check_path()*, определяющая длину пути дакибота.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


def check_crossroad(robot, point1, point2, point3, point4):
       if (((point1[0] or point4[0])<=robot[0]) and ((point3[0] or
point2[0])>=robot[0]))          and          (((point1[1]          or
point2[1])<=robot[1])and((point3[1] or point4[1])>=robot[1])):
        return True
    else:
        return False


def check_collision(coefficients):
    list_pars=[]
    for i in range(len(coefficients)):
        for j in range(len(coefficients)):
            if i==j:
                continue
            x1,y1,z1 = coefficients[i]
            x2,y2,z2 = coefficients[j]
            M1 = np.array([[x1,x2],[y1,y2]])
            M2 = np.array([[z1,z2]])
              if np.linalg.matrix_rank(M1) <= np.linalg.matrix_rank(M2)
or not np.any(M1) and not np.any(M2):
                continue
            list_pars.append(tuple([i,j]))
    return list_pars

def check_path(points_list):
    lenght=0
    for i in range(len(points_list)-1):
        x0=points_list[i][0]
        y0=points_list[i][1]
        x=points_list[i+1][0]
        y=points_list[i+1][1]
        from math import sqrt
        lenght+=sqrt((x0-x)**2+(y0-y)**2)
         return round(lenght,2)
```