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. 3343 _____ Пухов А.Д.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Научиться пользоваться основными управляющими конструкциями python.

Задание.

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).

Пример ситуации:

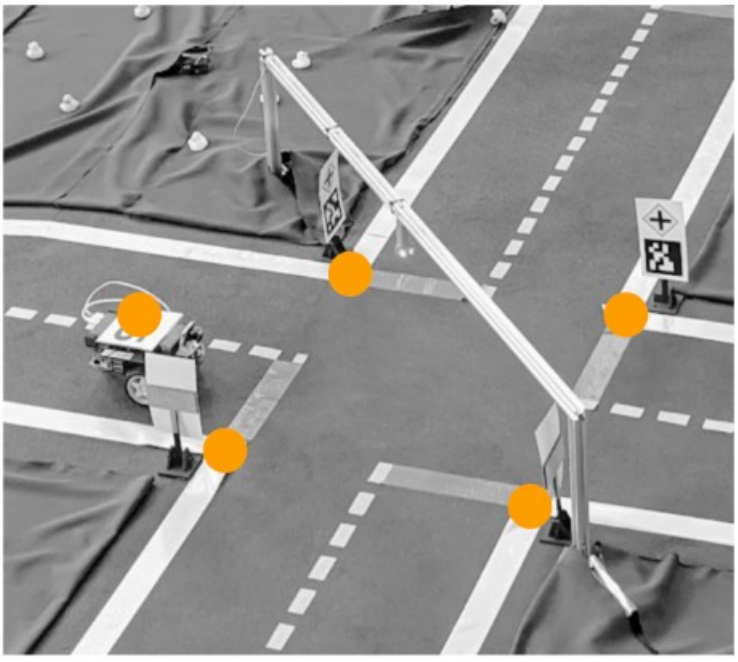

Рисунок 1 – Задача 1

Геометрическое представление (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями; обратите внимание, как пронумерованы точки):

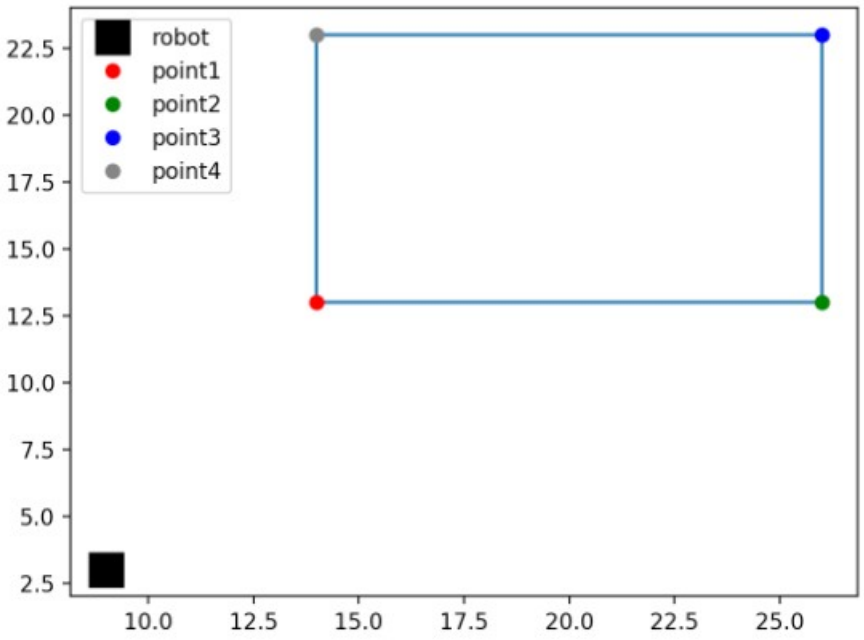

Рисунок 2 – Задача 1

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).

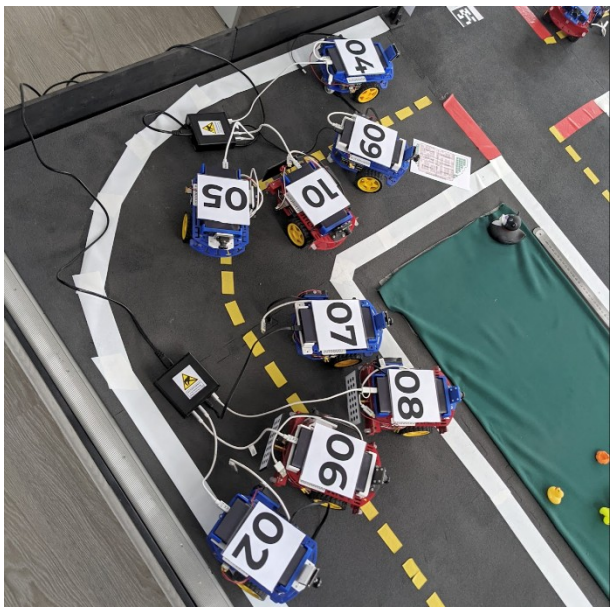

Рисунок 3 – Задача 2

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).

Примечание: помните про ранг матрицы и как от него зависит существование решения системы уравнений. В случае, если ни одного решение не было найдено (например, из-за линейно зависимых векторов), функция должна вернуть 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.

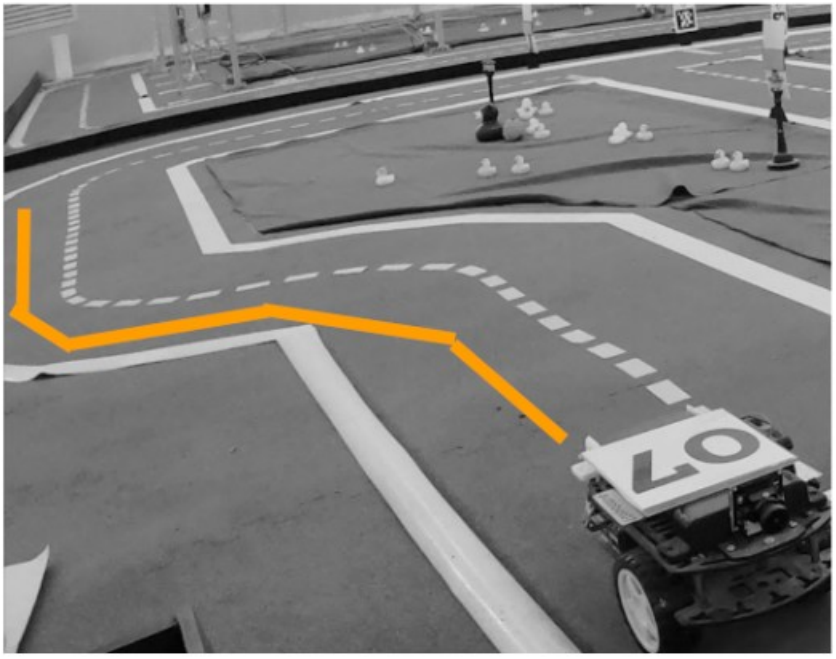

Рисунок 4 – Задача 3

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

Для каждой задачи были разработаны функции (см. приложение А).

Функция check_crossroad(), сравнивая координаты дикибота и координаты перекрёстка, определяет находиться ли дикибот внутри перекрёстка.

Функция check_collisions(), перебирая пары дикиботов, из координат этих пар создаёт массив numpy с помощью функции np.array(), а потом используя функцию np.linalg.matrix_rank() ищет ранг матрицы. Если ранг равен 2, это означает что данная пара дикиботов столкнулась друг с другом.

Функция check_path() ищет пройденное расстояние дикиботом с помощью
формулы поиска расстояния между координатными точками ($r=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$).

Переменные:

- rb – координаты дакибота
- pt1, pt2, pt3, pt4 – координаты точек описывающих перекрёсток
- res – список номеров ботов которые столкнулись с руг другом
- cof – коэффициенты уравнений
- i, j – переменные счётчика
- b – длина пути, пройденная дакиботом

Функции:

- check_crossroad() – функция определения нахождения дакибота на перекрёстке
- check_collision() – функция поиска столкнувшихся дакиботов
- check_path() – функция поиска пройденного расстояния дакиботом

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	check_crossroads input: (19, 9) (7, 5) (17, 5) (17, 18) (7, 18) check_collision input: [[6 5 10] [-3 -5 9] [3 4 4] [-6 -1 0] [3 -8 5] [3 -7 8] [6 2 5] [1 9 10]] check_path input: [(1.0, 2.0), (2.0, 3.0)]	False [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (1, 0), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 0), (2, 1), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (3, 0), (3, 1), (3, 2), (3, 4), (3, 5), (3, 6), (3, 7), (4, 0), (4, 1), (4, 2), (4, 3), (4, 5), (4, 6), (4, 7), (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 6), (5, 7), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 7), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6)] 1.41	OK

Выводы

В данной лабораторной работе были изучены и применены на практике основные управляющие конструкции языка python.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: app-001.py

```
import numpy as np

def check_crossroad(rb, p1, p2, p3, p4):
    if rb[0] >= p1[0] and rb[1] >= p1[1] and rb[0] <= p2[0] and rb[1] >= p2[1] and rb[0] <= p3[0] and rb[1] <= p3[1] and rb[0] >= p4[0] and rb[1] <= p4[1]:
        return True
    else:
        return False

def check_collision(cof):
    res = []
    for i in range(len(cof)):
        for j in range(len(cof)):
            if (i != j) and np.linalg.matrix_rank(np.array([[cof[i][0],cof[i][1]] , [cof[j][0],cof[j][1]]])) == 2:
                res.append((i,j))
    return res

def check_path(pl):
    b = 0
    for i in range(len(pl)-1):
        b = b + ((pl[i+1][0]-pl[i][0])**2 + (pl[i+1][1]-pl[i][1])**2)**(1/2)
    return round(b,2)
```