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 3343 Волох И.О.

Преподаватель Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Научиться создавать простые программы на языке программирования Python с использованием условий, циклов, списков, а также с модулем numpy.

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).

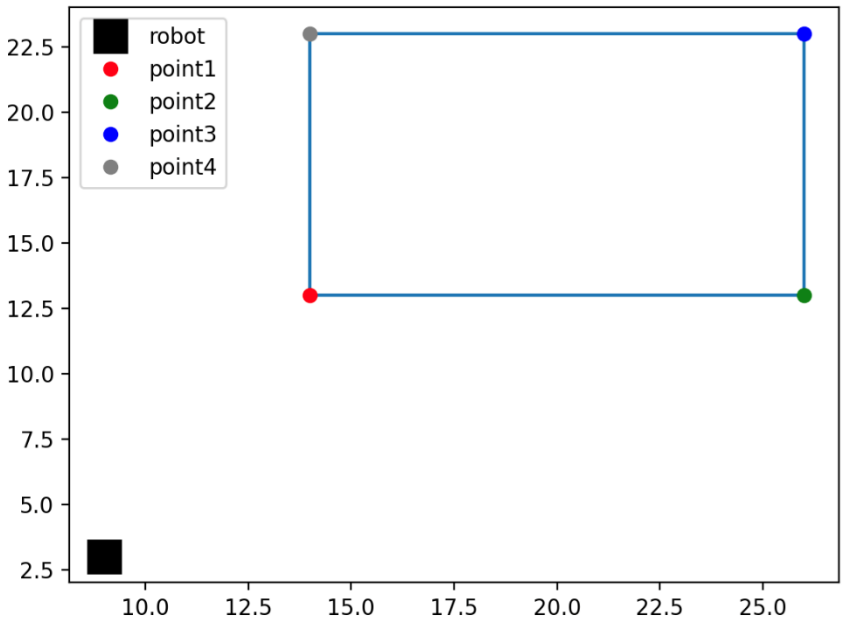

Рисунок 1 – Расположение точек перекрёстка

Формальная постановка задачи

Оформите задачу как отдельную функцию: def check_crossroad(robot, point1, point2, point3, point4). 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).

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.

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. Для того чтобы проверить, находится ли дакибот на перекрестке, нужно проверить условие, что его расстояние от каждой из точек не превышает максимально возможного, то есть длину диагонали прямоугольника и расстояние от каждой из точек до дакибота не превышат его ширину и длину. Для этого сначала вызывается функция check_crossroad, которая проверяет оба этих условия. Для этого вызывается функция check_if_longest, которая находит расстояние между каждой точкой и дакиботом, сохраняет их в переменных c, g, z, d после чего возвращает количество неподходящих расстояний. Ещё вызывается функция check_if_width_length, которая проверяет не превышает ли расстояние между дакиботом(расстояние между координатами дакибота и координатами точек прямоугольника хранится в списке alot) максимально возможного расстояния между соседними точками прямоугольника, то есть его длину и ширину, которые хранятся в переменных length и width соответсвенно.

Задача 2. Для того чтобы проверить, столкнулись ли дакиботы, нужно проверить, пересекались ли их траектории движения. Это можно сделать с помощью модуля nupmy.linalg, который используется в функции thats_collision_man. Сперва, был создан пустой список collisions, который будет хранить кортежи с номерами столкнувшихся дакиботов, а также созданы списки azs, bzs, czs из массива (ndarray) coefficients (чтобы иметь возможность получить номер дакибота по параметрам траектории его движения). С помощью цикла for проверены движения всех дакиботов на возможность их

перечение. Для этого вызывалась функция thats_collision_man, которая через решение системы уравнение, при помощи np.linalg.solve, проверяла есть ли у системы решение или нет, после чего возвращала 1 и 0 соответственно. Если функция возвращала , то номера дакиботов добавлялись в список collisions в виде кортежа из их номеров. После выполнения цикла функция вернет список collisions, отсортированный по возрастанию первого элемента.

Задача 3. Для того чтобы посчитать длину траектории, имея координаты ее точек, можно использовать формулу для нахождения длины вектора, через корень из суммы квадратов его координат в функции distance . Для этого, создана переменная full_path, которая будет содержать посчитанную длину траектории. Через цикл for (от 1 до длины списка точек) проходятся все пары соседствующих точек следующим образом: в переменные x0, y0 и x1, y1 записываются координаты соответственно предыдущей и текущей точек. По формуле считается длина между этими точками, переменная full_path увеличивается на эту длину. После выполнения цикла функция вернет значение path, округленное с помощью функции round до 2-ух знаков после запятой.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(9, 3) (14, 13) (26, 13) (26, 23) (14, 23)	False	Функция check_crossroad работает корректно
2.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	Функция check_crossroad работает корректно
3.	[[-1 -4 0] [-7 -5 5] [1 4 2] [-5 2 2]] (в виде ndarray)	[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]	Функция check_collision работает корректно
4.	[(1.0, 2.0), (2.0, 3.0)]	1.41	Функция check_path работает корректно
5.	[(2.0, 3.0), (4.0, 5.0)]	2.83	Функция check_path работает корректно

Выводы

Были изучены основные управляющие конструкции языка Python и некоторые функции модуля numpy. Разработана программа, разделенная на независимые функции, выполняющая обработку данных.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

def check_if_longest(point1,point2, point3, point4, robot, longest):

    coun = 0
    c = (abs(point1[0] - robot[0]), abs(point1[1] - robot[1]))
    g = (abs(point2[0] - robot[0]), abs(point2[1] - robot[1]))
    z = (abs(point3[0] - robot[0]), abs(point3[1] - robot[1]))
    d = (abs(point4[0] - robot[0]), abs(point4[1] - robot[1]))
    ln1, ln2, ln3, ln4 = np.linalg.norm(c), np.linalg.norm(g), np.linalg.norm(z), np.linalg.norm(d)
    if ln1 > longest:
        coun += 1
    if ln2 > longest:
        coun += 1
    if ln3 > longest:
        coun += 1
    if ln4 > longest:
        coun += 1
    return coun

def check_if_width_length(point1,point2, point3, point4, robot, width, length):
    alot = [[abs(point1[0] - robot[0]), abs(point1[1] - robot[1])],[abs(point2[0] - robot[0]), abs(point2[1] - robot[1])], [abs(point3[0] - robot[0]), abs(point3[1] - robot[1])], [abs(point4[0] - robot[0]), abs(point4[1] - robot[1])] ]
    coun = 0
    x1, y1, x2, y2, x3, y3, x4, y4 = alot[0][0], alot[0][1], alot[1][0], alot[1][1], alot[2][0], alot[2][1], alot[3][0], alot[3][1]
    if x1 > width or y1 > length:
        coun += 1
    if x2 > width or y2 > length:
        coun += 1
    if x3 > width or y3 > length:
        coun += 1
    if x4 > width or y4 > length:
        coun += 1
    return coun

def check_crossroad(robot, point1, point2, point3, point4):
    c = (abs(point1[0] - point3[0]), abs(point1[1] - point3[1]))
    longest, width, length = np.linalg.norm(c), abs(point1[0] - point2[0]), abs(point4[1] - point1[1])
    count = 0
    if check_if_longest(point1,point2, point3, point4, robot, longest) > 1:
```

```
            return False
        else:
            count += 1
        if check_if_width_length(point1,point2, point3, point4, robot,
width, length) > 0:
            return False
        else:
            count += 1
        if count == 2:
            return True
        else:
            return False

    def thats_collision_man(a1, b1, c1, a2, b2, c2):
        A = np.array([[a1, b1], [a2, b2]])
        B = np.array([-c1, -c2])
        try:
            np.linalg.solve(A, B)
            return 1
        except:
            return 0

    def check_collision(coefficients):
        azs, bzs, czs = [], [], []
        coefficients = coefficients.tolist()
        for i in coefficients:
                azs.append(int(i[0]))
                bzs.append(int(i[1]))
                czs.append(int(i[2]))
        collisions = []
        N = len(coefficients)
        for i in range(N - 1):
            for j in range(i + 1, N):
                if thats_collision_man(azs[i], bzs[i], czs[i],azs[j],
bzs[j], czs[j]) == 1:
                    collisions.append((i, j))
                    collisions.append((j, i))
        collisions = sorted(collisions, key=lambda x: x[0])
        return collisions

    def distance(x1, x2, y1, y2):
        dist = np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
        return dist
    def check_path(points_list):
        full_path = 0.0
        for i in range(len(points_list) - 1):
            x1, y1 = points_list[i]
            x2, y2 = points_list[i+1]
            if (x1 != x2 or y1 != y2) and ((x1, y1) != (x2, y2)):
                full_path += distance(x1, x2, y1, y2)
        return round(full_path, 2)
```