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 Лобова Е. И.

Преподаватель Иванов Д. В.

Санкт-Петербург

2023

Цель работы

Целью работы является освоение работы с основными управляющими конструкциями на языке Python, включая модуль numpy, в частности его пакет numpy.linalg.

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).

Пример ситуации:

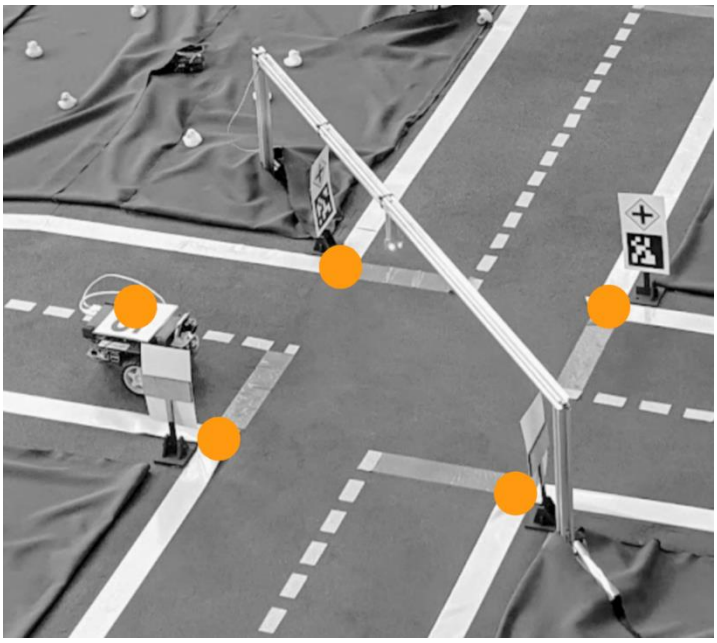

Рисунок 1 – Задача 1

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

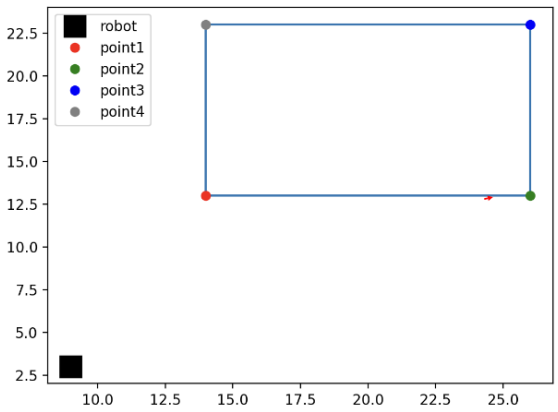

Рисунок 2 – Задача 1

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.

Примеры входных аргументов и результатов работы функции:

1. Входные аргументы: (9, 3) (14, 13) (26, 13) (26, 23) (14, 23)

Результат: False

2. Входные аргументы: (5, 8) (0, 3) (12, 3) (12, 16) (0, 1)

Результат: True

Задача 2. Содержательная часть задача

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).

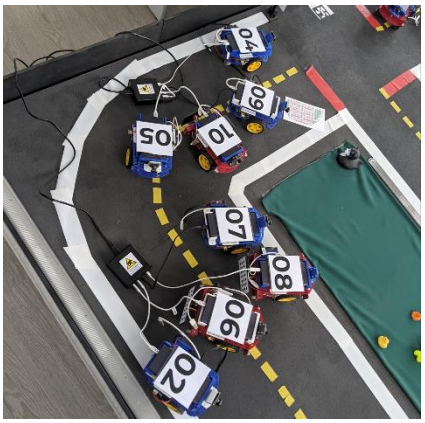

Рисунок 3 – Задача 2

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).

Пример входного аргумента ndarray 4x3 :

[[-1 -4 0]

 [-7 -5 5]

 [1 4 2]

 [-5 2 2]]

Пример выходных данных:

[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]

Первая пара в этом списке (0, 1) означает, что столкнулись 0-й и 1-й боты (то есть их траектории имеют общую точку).

В списке отсутствует пара (0, 2), можно сделать вывод, это боты 0-й и 2-й не сталкивались (их траектории НЕ имеют общей точки).

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.

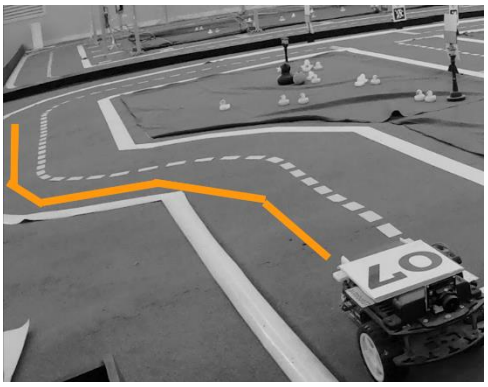

Рисунок 4 – Задача 3

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)).

Пример входных данных:

[(1.0, 2.0), (2.0, 3.0)]

Пример выходных данных:

1.41

Пример входных данных:

[(2.0, 3.0), (4.0, 5.0)]

Пример выходных данных:

2.83

Основные теоретические положения

- Функция округления вещественных чисел *round(value[, signs])*, которая принимает на вход 2 аргумента:
    1. вещественное число *value*, которое нужно округлить
    2. номер знака после запятой *signs*, до которого нужно округлить (если этот аргумент отсутствует, то округление происходит до ближайшего целого)
- Методы модуля numpy:
    1. *numpy.linalg.norm(vector)* - данный метод из пакета *linalg* модуля numpy позволяет вычислить норму (модуль, длину) вектора *vector*(в общем случае — матрицы), переданного на вход.

        Длина вектора — это число, равное корню из суммы квадратов координат вектора. Например: $r_1=(x_1,y_1,z_1)$ — вектор, тогда его норма вычисляется так:

        $$r_1 = \sqrt{(x_1^2 + y_1^2 + z_1^2)}$$

    2. *numpy.array(object)* – данный метод модуля numpy позволяет преобразовать список *object* в массив.
    3. *numpy.linalg.matrix_rank(matrix)* - данный метод из пакета *linalg* позволяет посчитать ранг квадратной матрицы *matrix*.

        Ранг — это количество линейно-независимых строк или столбцов квадратной матрицы. Два вектора r_1 r_2 считаются линейно-независимыми, если ни один из них нельзя представить в виде линейной комбинации другого. Если матрица коэффициентов линейной системы уравнений имеет ранг равный числу строк, то система имеет решение.

Выполнение работы

Функции, реализованные в программе:

- Функция *check_crossroad(robot, point1, point2, point3, point4)* принимает на вход координаты дакибота *robot* и координаты точек, описывающих перекресток: *point1, point2, point3, point4*. Точка - это кортеж из двух целых чисел (x, y). Она проверяет, находится ли координата x между координатами x первой и второй точки (т. к. у них различны координаты x по рисунку, и у второй точки она больше), включая границы, и аналогичную проверку делает для координаты y и точек 2, 3. Если оба условия выполнены, то функция возвращает «True», иначе – «False».
- Функция *check_collision(coefficients)* принимает на вход матрицу *ndarray Nx3* (N - количество ботов, может быть разным в разных тестах) коэффициентов уравнений траекторий *coefficients*. Далее создается пустой список *pairs_of_colliding_bots*, куда будут записываться пары номеров столкнувшихся ботов. С помощью вложенного цикла перебираются все возможные пары дакиботов (но не с одинаковыми номерами, т. к. это невозможно), и записываются в переменные *a_i, b_i, a_j, b_j* коэффициенты при x и y строк матрицы с номерами i и j соответственно. Далее создается матрица из этих коэффициентов *coeff_matrix* с помощью метода *numpy.array()*, а с помощью метода *numpy.linalg.matrix_rank()* проверяется ранг матрицы, из значения которого делается вывод о том, имеет ли система двух уравнений решение и следовательно столкнутся ли два дакибота. Функция возвращает список пар -номера столкнувшихся ботов (если никто из ботов не столкнулся, возвращается пустой список).
- Функция *check_path(points_list)* принимает на вход список) двумерных точек последовательных пар координат дакибота *points_list*. Далее создается переменная *lenght*, в которой будет

накапливаться длина пройденного пути. Цикл со счетчиком перебирает пары соседних точек и на каждой итерации в переменную vector записывается кортеж из координат вектора, который образуют две соседние точки. Далее к переменной *length* прибавляется длина очередного вектора, полученная с помощью метода *numpy.linalg.norm()*. Функция возвращает длину пройденного дакиботом пути, округленную до двух знаков после запятой с помощью *round(value, 2))*.

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.	check_crossroads input: (2, 10) (2, 8) (16, 8) (16, 23) (2, 23) check_collision input: [[3 1 1] [2 -2 2] [4 -3 5] [6 2 8] [-6 -1 10] [8 9 8] [-9 -8 6] [-9 -6 9] [1 3 3]] check_path input: [(3.69, 0.7), (3.66, 0.91), (3.54, 1.21), (3.54, 1.69), (3.51, 1.87)]	True [(0, 1), (0, 2), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (1, 0), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (2, 0), (2, 1), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (3, 1), (3, 2), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (4, 0), (4, 1), (4, 2), (4, 3), (4, 5), (4, 6), (4, 7), (4, 8), (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 6), (5, 7), (5, 8), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 7), (6, 8), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 8), (8, 0), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7)] 1.2	Для границ перекрестка первая функция работает верно. Вторая функция нашла все пары номеров столкнувшихся дакиботов даже при большом их количестве. Для более чем двух точек третья функция работает исправно.
2.	check_crossroads input: (7, 12) (10, 0) (25, 0) (25, 10) (10, 10) check_collision input: [[-4 -6 7] [-10 2 0] [8 8 7]] check_path input: [(1.0, 2.0), (2.0, 3.0)]	False [(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)] 1.41	Первая функция работает, если дакибор строго внутри перекрестка. Вторая функция работает верно для небольшого количества пар столкнувшихся дакиботов. Для минимального числа пар координат(2 пары) третья функция работает исправно.

Выводы

Были изучены основные управляющие конструкции языка, особое внимание было уделено работе с модулем numpy и его пакету *numpy.linalg*.

Разработана часть программы, содержащая в себе 3 функции. Для реализации математических задач использовался модуль numpy, а именно методы *numpy.linalg.norm(vector)*, *numpy.array(object), numpy.linalg.matrix_rank(matrix).* Для перебора пар элементов массива использовались циклы *for*. Для проверки выполнения заданного условия использовался условный оператор *if – else*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


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


def check_collision(coefficients):
    pairs_of_colliding_bots =[]
    for i in range(len(coefficients)):
        for j in range(len(coefficients)):
            if i!=j:
                a_i,b_i=coefficients[i][0],coefficients[i][1]
                a_j,b_j=coefficients[j][0],coefficients[j][1]
                coeff_matrix=np.array([[a_i,b_i],[a_j,b_j]])
                if np.linalg.matrix_rank(coeff_matrix)==2:
                    pairs_of_colliding_bots.append((i,j))
    return pairs_of_colliding_bots


def check_path(points_list):
    lenght=0;
    for i in range(len(points_list)-1):
        vector=(points_list[i+1][0]-
points_list[i][0],points_list[i+1][1]-points_list[i][1])
        lenght+=np.linalg.norm(vector)
    return round(lenght,2)
```