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 Python

Студент гр. 3341 _____ Моисеева А.Е.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург

2023

Цель работы

Изучить основные управляющие конструкции языка Python, а также модуль numpy, в частности, пакет numpy.linalg. Используя полученные знания, реализовать 3 функции в соответствии с заданием.

Задание.

Вариант 2.
Задача 1. Содержательная часть 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).

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).

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.

Выполнение работы

Был подключен модуль numpy. Для решения математических задач были реализованы 4 фукнции с помощью numpy.linalg.solve и numpy.array.

Функция check_crossroad(robot, point1, point2, point3, point4):
На вход подаются пять кортежей, каждый из которых содержит координаты x, y соответствующей точки. Первый содержит координаты робота, остальные задают координаты перекрёстка. Поскольку согласно условию перекрёсток является прямоугольником, необходимо проверить, чтобы координата робота по x находилась между координатами по x точек 1 и 2, а также чтобы координата робота по y находилась между координатами по y точек 1 и 3. После проверки условия функция подаёт на вывод значение True, если дакибот находится на перекрёстке, и False в противоположном случае.

Функция check_collision(coefficients):
На вход подаётся матрица ndarray размером Nx3, где N – количество дакиботов. Далее через вложенные циклы происходит перебор всех пар уравнений исходной матрицы. С помощью вспомогательной функции solve_system() определяется, имеет ли данная пара уравнений общее решение. В соответстующем случае кортеж из индексов уравнений добавляется в список result[]. Функция выводит итоговый список result[].

Функция check_path(points_list):
На вход подаётся список кортежей points_list. Через цикл осуществляется проход каждой пары подряд идущих координат и вычисляется длина пути от первой точки до второй. Сумма прибавляется к итоговому результату result. Функция выводит итоговое значение переменной result с округлением до 2 значащих цифр через round.

Функция solve_system(equation1, equation2):

На вход подаются пара уравнений, затем для решения системы с помощью numpy.linalg.solve создаётся матрица коэффициентов (из первых и вторых коэффициентов уравнений) и вектор свободных членов (из третьих коэффициенты уравнений). Затем при нахождении решения функция возвращает значение True, иначе - False.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(10, 5) (14, 15) (28, 15) (28, 20) (14, 20)	False	Дакибот находится вне перекрёстка
2.	(6, 8) (5, 2) (19, 2) (19, 20) (5, 20)	True	Дакибот находится на перекрёстке
3.	[[-1, -4, 0] [-7, 5, 5] [1, 4, 2] [1, 4, 3]]	[(0, 1), (1, 0), (1, 2), (1, 3), (2, 1), (3, 1)]	Программа выводит номера столкнувшихся дакиботов в виде массива
4.	[(3.69, 0.7), (3.67, 0.99), (3.47, 1.31), (3.63, 1.47), (3.5, 1.82)]	1.27	Программа выводит длину пути, пройденного дакиботом
5.	[(1.0, 2.7), (3.6, 2.99)]	2.62	Программа выводит длину пути, пройденного дакиботом

Выводы

Изучены основные управляющие конструкции языка Python, такие как условия, функции, циклы. Исследованы методы библиотеки numpy и пакета numpy.linalg, например, создание матрицы, вычисление длины вектора, умножение матриц друг на друга.

С помощью полученных знаний был написан программный код, содержащий в себе четыре функции:

check_crossroad() – функция, помогающая определить, находится ли дакибот на перекрёстке;

check_collision() – функция, определяющая столкновение дакиботов;

check_path() – функция, определяющая длину пути дакибота;

solve_system() – вспомогательная функция для выполнения 2 задачи, решающая системы двух линейных уравнений, заданных матрицами, с помощью numpy.linalg.solve и numpy.array.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb1.py

```
import numpy as np

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

def solve_system(equation1, equation2):
    coefficients = np.array([equation1[:2], equation2[:2]])
    constants = np.array([-equation1[-1], -equation2[-1]])
    try:
        solution = np.linalg.solve(coefficients, constants)
        return(True)
    except:
        return(False)

def check_collision(coefficients):
    size_line = coefficients.shape[0]
    result = []
    for i in range(size_line):
        system_loc = coefficients[i]
        for j in range(size_line):
            if i != j:
                system_new = coefficients[j]
                if solve_system(system_loc, system_new):
                    ans = (i, j)
                    result.append(ans)
    return result
    pass

def check_path(points_list):
    num_points = len(points_list)
    result = 0
    for i in range(num_points-1):
           len_path = ((points_list[i+1][0] - points_list[i][0])**2 +
(points_list[i+1][1] - points_list[i][1])**2)**0.5
        result += len_path

    return round(result, 2)
    pass
```