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. 2381 Долотов Н.А.

Преподаватель Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Изучить основные управляющие конструкции Python, а также подключаемый модуль *numpy*, в частности, пакет *numpy.linalg*. Реализовать 3 функции, используя теоретические знания.

Задание.

Задача 1.

Оформите задачу как отдельную функцию: *def check_rectangle(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)

Резлультат: False

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

Результат: True

Задача 2.

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.

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

Выполнение работы.

Для решения поставленных задач были написаны 3 функции с использованием подключаемого модуля numpy, а именно, с помощью пакета numpy.linalg.

Функция *check_crossroad(robot, point1, point2, point3, point4)*

Функция принимает 5 кортежей, которые являются координатами точек перекрёстка (прямоугольника) и самого робота. Так как по условию известно, что перекрёсток является прямоугольником, то передача функции четырёх точек является избыточной: достаточно двух диагональных точек, например, левой нижней (*point1*) и правой верхней (*point3*). В функции с помощью операторов сравнения проверяется, принадлежит ли координата робота по определённой оси (x и y) отрезку, на котором располагается прямоугольник, и, если да – возвращает *True*, иначе – *False*.

Функция *check_collision(coefficients)*

Функция должна проверять, пересекаются ли уравнения траектории двух ботов. Так как уравнение траектории задаётся линейным уравнением вида ax + by + c = 0, то функция должна посчитать определитель матрицы; если он не равен нулю, то система имеет единственное нулевое решение, а значит, что траектория ботов пересеклась и они столкнулись друг с другом.

Функция принимает матрицу ndarray Nx3 (N – количество ботов) коэффициентов уравнения траектории *coefficients*. С помощью метода *numpy.delete()* из начальной матрицы удаляется третий столбец свободных членов и новая матрица *fl_matrix* становится размером Nx2. Далее реализован цикл *for* в цикле *for*, где *i* – индекс строки первого уравнения коэффициентов, а *j* – индекс строки второго уравнения коэффициентов (*i != j*). Матрица *matrix* – матрица размером 2x2, полученная с помощью конкатенации двух строк благодаря методу *numpy.concatenate()*, который объединяет последовательность массивов вдоль заданной оси. Далее с помощью метода *numpy.linalg.det()* вычисляется определитель квадратной матрицы 2x2. И если он не равен нулю, то кортеж из индексов двух строк (индексы строк являются одновременно и порядковыми номерами роботов) добавляется в массив кортежей столкнувшихся роботов *out*. Функция возвращает массив *out*.

Функция *check_path(points_list)*

Функция принимает список двумерных точек (кортежей) *points_list*. Цикл *for* проходит по каждой паре точек и вычисляет координаты каждого вектора

vector. Далее с помощью метода *numpy.linalg.norm()* вычисляется длина каждого вектора (т.е. расстояние между двумя точками), которая суммируется в переменную *path*. Функция возвращает длину пройденного роботом пути *path*, округлённую с помощью функции *round()* до двух знаков после запятой.

Разработанный код см. в приложении А.

Выводы.

Были изучены основные управляющие конструкции языка Python, такие как функции, циклы, условия. Также были исследованы методы библиотеки *numpy* и пакета *numpy.linalg*, такие как конкатенация матриц, вычисление определителя матрицы, а также длины вектора.

Была написана программа, содержащая три функции, предназначенные для использования в разработке алгоритмов автономного движения дакиботов: *check_crossroad()* – функция, определяющая, находится ли робот на перекрёстке или нет; *check_collision()* – функция, выводящая порядковые номера столкнувшихся роботов; *check_path* – функция, считающая длину пройденного роботов пути.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

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

def check_collision(coefficients):
    fl_matrix = np.delete(coefficients, 2, 1)
    out = []
    for i in range(len(fl_matrix)):
        for j in range(i + 1, len(fl_matrix)):
            matrix = np.concatenate(([fl_matrix[i]], [fl_matrix[j]]),
axis=0)
            if np.linalg.det(matrix) != 0:
                out.append((i, j))
                out.append((j, i))
    return sorted(out)

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