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 _____ Коршков А.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы.

Научится писать простейшие программы на Python с использованием циклов, условий, преобразований типов, а также с библиотекой 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).

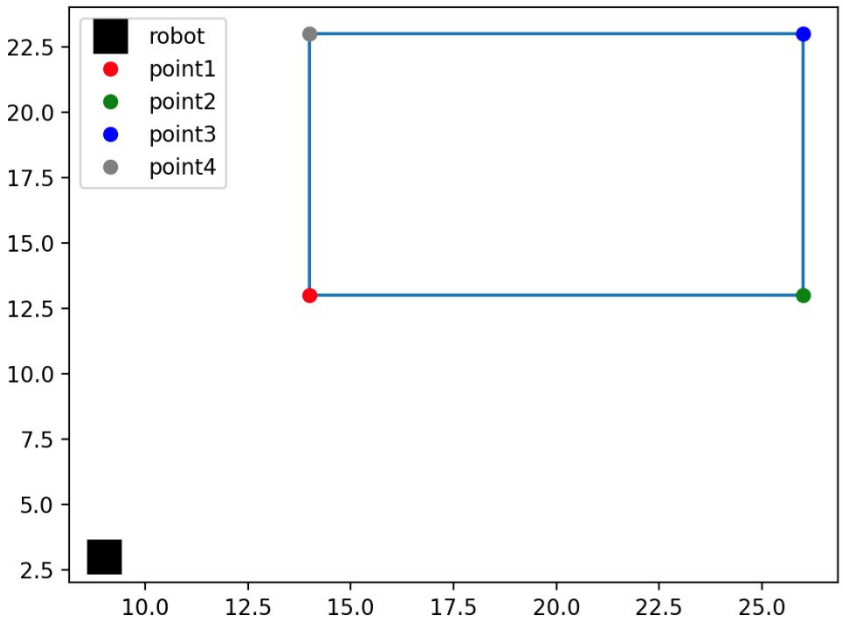

Рисунок 1 – Расположение точек перекрёстка

Формальная постановка задачи

Оформите задачу как отдельную функцию: def check_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.

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).

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.

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

Выполнение работы.

Задача 1. Для того чтобы определить, находится ли дакибот, в прямоугольнике были выписаны координаты x0, y0 (начальные координаты), x1, y1 (конечные координаты). Далее идёт проверка, находятся ли координаты дакибота в пределах начальных и конечных координат, и если это условие выполняется, то выводится true, иначе false.

Задача 2. В данной задаче необходимо перебрать всевозможные комбинации коэффициенты для двух уравнений траекторий. Для каждого случая получается система уравнений:

$$\begin{cases} a_1x + b_1y + c_1 = 0 \ (* a_2) \\ a_2x + b_2y + c_2 = 0 \ (* a_1) \end{cases} \begin{cases} a_1x + b_1y + c_1 = 0 \ (* b_2) \\ a_2x + b_2y + c_2 = 0 \ (* b_1) \end{cases}$$

Необходимо создать две системы уравнений, чтобы избавиться от одного из коэффициентов и найти формулу для второго:

$$\begin{cases} a_1a_2x + b_1a_2y + c_1a_2 = 0 \ (-) \\ a_1a_2x + b_2a_1y + c_2a_1 = 0 \end{cases} \begin{cases} a_1b_2x + b_1b_2y + c_1b_2 = 0 \\ a_2b_1x + b_2b_1y + c_2b_1 = 0 \ (-) \end{cases}$$

$$(b_1a_2 - b_2a_1)\,y + c_1a_2 - c_2a_1 = 0$$

$$(b_1a_2 - b_2a_1)\,x + c_2b_1 - c_1b_2 = 0$$

$$y = \frac{c_2a_1 - c_1a_2}{b_1a_2 - b_2a_1} \quad x = \frac{b_2c_1 - c_2b_1}{a_1b_2 - a_1b_2}$$

В данном случае у x и у получается одинаковый знаменатель. Если точек пересечения нет, то он становится нулевым. Именно поэтому проверяется, что $b_1a_2 - b_2a_1$!= 0, ведь на ноль делить нельзя. В этом случае можно сказать, что точка пересечения есть.

Задача 3. Чтобы вычислить, какое расстояние проехал дакибот, необходимо воспользоваться теоремой Пифагора. В цикле берутся координаты двух точек, вычитаем из координат второй точки координаты первой точки, затем полученные результаты возводим в квадрат, складываем их, и извлекаем корень из полученного результата.

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
2.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	Функция check_crossroad
3.	[[-1 -4 0] [-7 -5 5] [1 4 2] [-5 2 2]]	[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]	Функция check_collision
4	[(1.0, 2.0), (2.0, 3.0)]	1.41	Функция check_path
5	[(2.0, 3.0), (4.0, 5.0)]	2.83	Функция check_path

Выводы.

Были изучены основные управляющие конструкции языка Python и основные функции библиотеки numpy. Разработана программа, выполняющая обработку данных через функции.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

def check_crossroads(robot, point1, point2, point3, point4):
    x0, x1, y0, y1 = point1[0], point3[0], point1[1], point3[1]
    if x0 <= robot[0] <= x1 and y0 <= robot[1] <= y1:
        return True
    return False


def check_collision(coefficients):
    pars = []
    for i in range(len(coefficients)):
        for j in range(len(coefficients)):
            if i != j:
            a1, b1, c1 = coefficients[i][0], coefficients[i][1],
coefficients[i][2]
            a2, b2, c2 = coefficients[j][0], coefficients[j][1],
coefficients[j][2]
                if (a2 * b1) - (a1 * b2) != 0:
                    pars.append(tuple([i ,j]))
    return pars


def check_path(points_list):
    path_len = 0
    for i in range(len(points_list) - 1):
        path_len += ((points_list[i + 1][0] - points_list[i][0])
** 2
                + (points_list[i + 1][1] - points_list[i][1])
** 2) ** 0.5
    return round(path_len, 2)
```