МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информатика»

Тема: Управляющие конструкции языка Python

Студент гр. 3341 _____ Костромитин М.М.

Преподаватель _____ Рябов М.Л.

Санкт-Петербург
2023

Цель работы

Целью лабораторной работы является изучение модуля numpy в языке python, а также решение практической задачи с его использованием.

Задание

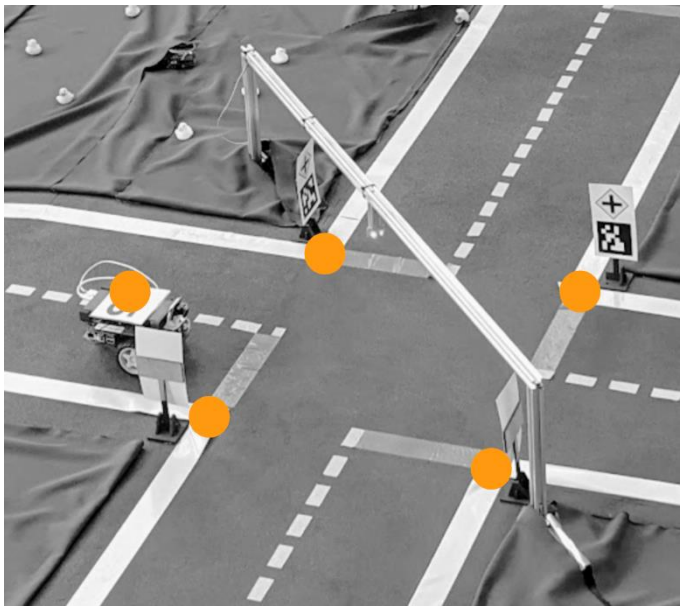
Вариант 2

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.

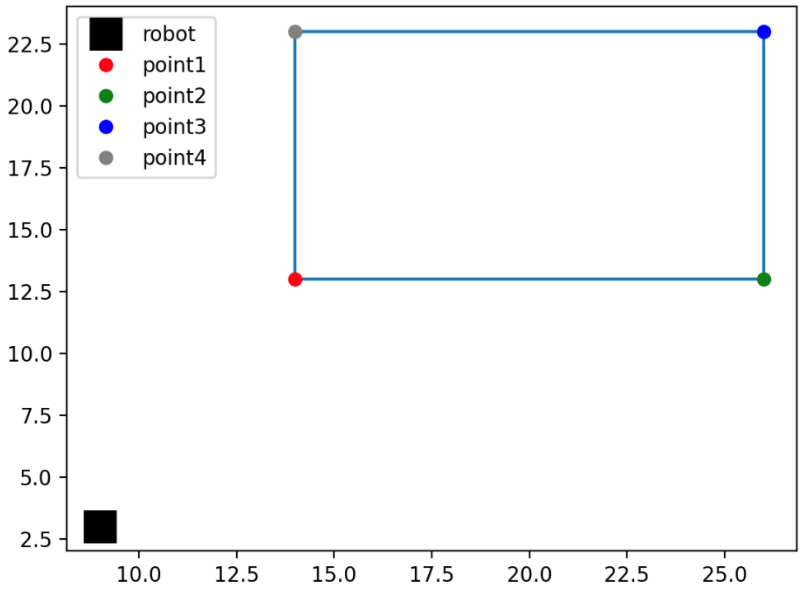
Задача 1. Содержательная постановка задачи

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:



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



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.

Примеры входных аргументов и результатов работы функции:

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

Резлультат: False

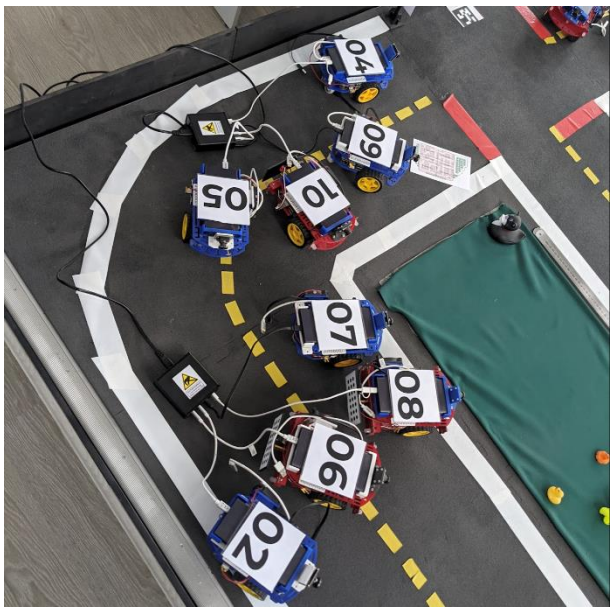
2. Входные аргументы: (5, 8) (0, 3) (12, 3) (12, 16) (0, 16)

Результат: True

Задача 2. 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]]

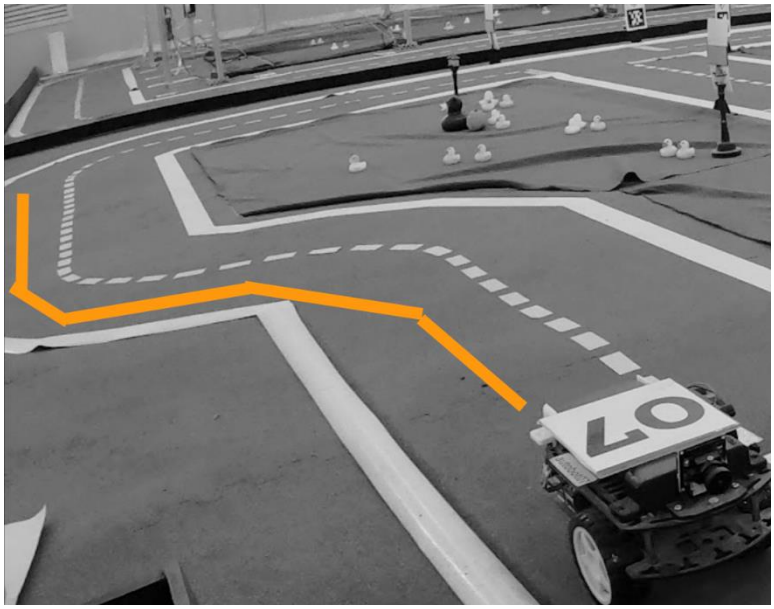
Пример выходных данных:
[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]

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

Импортируем модуль NumPy.

Задача 1.

В первой функции check_crossroad реализовано два дополнительных условия под названием condition1 и condition2, в них идет проверка вхождения точки робота на площадь прямоугольника. Codition1 проверяет, больше ли координата робота, чем point1, если да, то возвращается True, в противном случае False. Аналогично происходит для переменной condition2, которая сравнивает point3 и координаты робота. Если codition1 и condition2 равны True, то это значит, что робот находится внутри прямоугольника и функция возвращает True, в противном случае False.

Задача 2.

Во второй функции check_collision реализованы два цикла, которые проходят по всем парам дакиботов и сравнивает их коэффициенты при x y y следующего уравнения: y = -(a/b)x + c, и если коэффициенты равны, это означает, что боты никогда не пересекутся и их траектории движения не имеют общих точек. Если коэффициенты не равны, то записываем порядковые номера столкнувшихся дакиботов в массив, после окончания циклов возвращаем получившийся список.

Задача 3.

В третей функции под названием check_path при помощи цикла передвигаемся по всем точкам, кроме последней. Внутри цикла берем координату текущей точки (x, y) и следующий (x1, y1) и по теореме Пифагора вычисляем модуль длины между ними при помощи следующей формулы: sqrt((x1-x)**2 + (y1-y)**2). Результат получившейся длины между двумя точками складываем в общий путь дакибота, после завершения цикла возвращаем итоговый путь.

Тестирование

Результаты тестирования представлены в Таблице 1.

Таблица 1.

№ п/п	Входные данные	Выходные данные	Комментарии
1	(5, 6) (3, 5) (9, 5) (9, 11) (3, 11)	True	Задача 1
2	(15, 2) (3, 5) (9, 5) (9, 11) (3, 11)	False	Задача 1
3	[[5, 3, 4], [8, 1, 0], [10, 6, 3], [5, 7, 1]]	[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3,1), (3, 2)]	Задача 2
4	[(1.0, 2.5), (2.0, 3.0), (5.6, 4.0), (8.0, 6.3), (9.8, 7.6)]	10.4	Задача 3

Выводы

В результате выполнения лабораторной работы были изучены и применены некоторые функции модуля NumPy в Python при решении практической задачи, а так же реализовано 3 функции для решения поставленных задач.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Исходный файл: main.py

```
import numpy as np


def check_crossroad(robot, point1, point2, point3, point4):
    condition1 = (point1[0] <= robot[0]) and (point1[1] <= robot[1])
    condition2 = (point3[0] >= robot[0]) and (point3[1] >= robot[1])
    if condition1 and condition2:
        return True
    return False


def check_collision(coefficients):
    total_collision = []
    for i in range(len(coefficients)):
        for j in range(len(coefficients)):
            if (-(coefficients[i][0] / coefficients[i][1])) != (-
(coefficients[j][0] / coefficients[j][1])):
                total_collision.append((i, j))
    return total_collision


def check_path(points_list):
    length = 0
    for i in range(len(points_list)-1):
        x, y = points_list[i]
        x1, y1 = points_list[i+1]
        length += np.sqrt(((x1 - x)**2) + ((y1 - y)**2))
    return round(length, 2)
```