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: Управляющие конструкции языка Python

Студент гр. 3341 _____ Пчелкин Н.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Цель лабораторной работы заключается в решении задач, включающих использование основных управляющих конструкций языка Python. В результате лабораторной работы необходимо разработать и протестировать 3 функции, каждая из которых решает свою задачу.

Задание

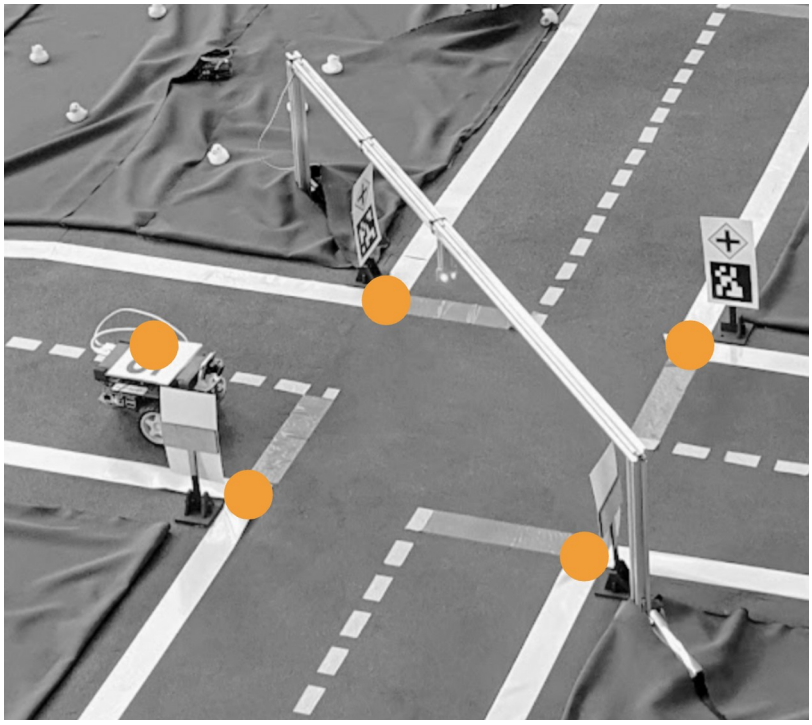
Вариант 2

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.

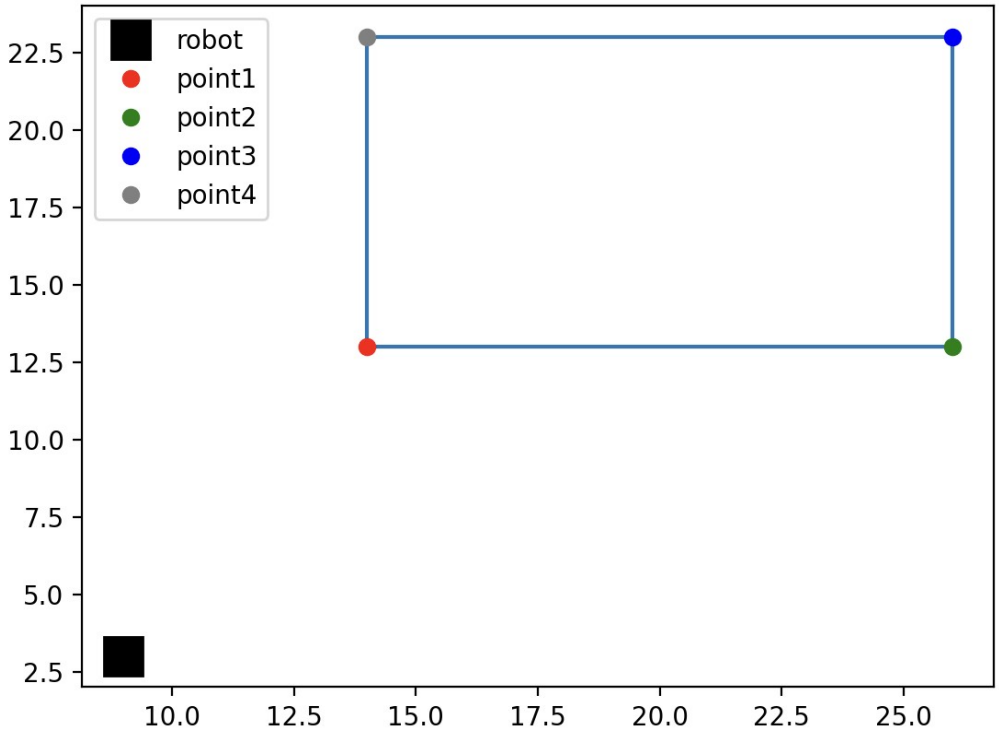
Задача 1. Содержательная постановка задачи

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

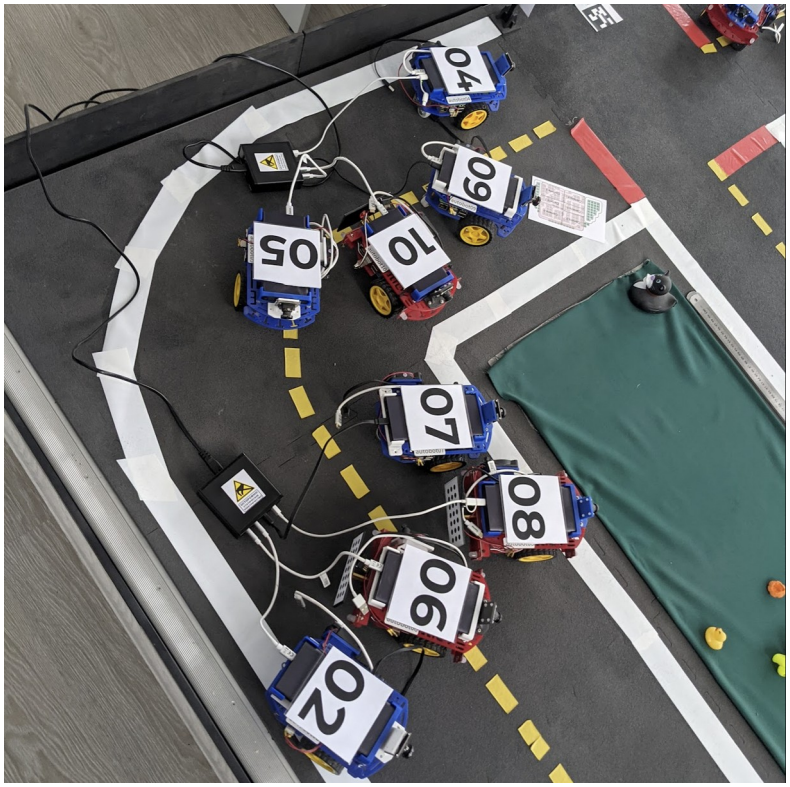
2. Входные аргументы: (5, 8) (0, 3) (12, 3) (12, 16) (0, 16)

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)]

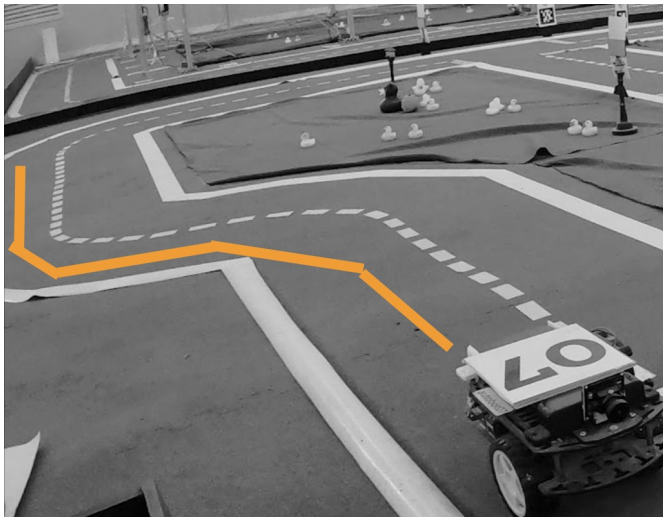
Первая пара в этом списке (0, 1) означает, что столкнулись 0-й и 1-й боты (то есть их траектории имеют общую точку).

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

В задании требуется оформить каждую из 3 задач в виде отдельной функции согласно условиям.

1. Решение задачи 1:

   check_crossroad(robot, point1, point2, point3, point4):

   На вход функции подаются координаты робота и координаты точек – границ перекрестка. С помощью координат перекрестка вычисляем коэффициенты k и b уравнений прямых типа y = kx + b. Эти прямые проходят через стороны перекрестка, а значит если робот будет находиться в области, заданной этими прямыми, то наш бот находится на перекрестке. В функции существует 2 проверки: первая – на то, что стороны перекрестка расположены параллельно осям координат, вторая – наклонен ли на прямоугольник в правую или в левую сторону.

2. Решение задачи 2:

   check_collision(coefficients)

   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).

   Создаем 2 цикла с параметром (один вложен в другой) и проверяем каждую пару на наличие столкновений. Аналогично работе в первой функции ищем коэффициент k для каждой траектории. Столкновений не будет, если траектории параллельны, а следовательно их k будут равны.

   Решение задачи 3:

   def check_path(points_list)

   На вход передается последовательность (список) двумерных точек (пар) *points_list*. Функция должна возвращать число – длину

пройденного дакиботом пути (с округлением до 2 знака с помощью *round(value, 2)*).

Создаем 2 цикла с параметром (один вложен в другой) и вычисляем путь для каждой пары координат. Аккумулируем пути в переменную result – эта же переменная в конце итераций и есть наш искомый путь.

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.	(9, 3) (14, 13) (26, 13) (26, 23) (14, 23)	False	-
2.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	-
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)]	-
4.	[(1.0, 2.0), (2.0, 3.0)]	1.41	-
5.	[(2.0, 3.0), (4.0, 5.0)]	2.83	-

Выводы

В результате выполнения данной лабораторной работы были достигнуты следующая цели: решение задач, включающих использование основных управляющих конструкций языка Python, а также разработка и тестирование трех функций, каждая из которых предназначена для решения конкретной задачи.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
def check_crossroad(robot, point1, point2, point3, point4):
     if point1[0] != point4[0]:
         k1 = (point4[0] - point1[0])/(point4[1]-point1[1])
         k3 = k1
         k2 = -1/k1
         k4 = k2
         b1 = point1[1] - k1*point1[0]
         b2 = point2[1] - k2*point2[0]
         b3 = point3[1] - k3 * point3[0]
         b4 = point4[1] - k4 * point4[0]
         if b1 > b3:
                 if robot[1] <= k1*robot[0] + b1 and robot[1] <=
k4*robot[0] + b4 and robot[1] >= k3*robot[0] + b3 and robot[1] >=
k2*robot[0] + b2:
                     return True
         else:
             if robot[1] >= k1 * robot[0] + b1 and robot[1] <= k4 *
robot[0] + b4 and robot[1] <= k3 * robot[0] + b3 and robot[1] >= k2
* robot[0] + b2:
                     return True
     else:
          if robot[1] <= point3[1] and robot[1] >= point1[1] and
robot[0] >= point1[0] and robot[0] <= point3[0]:
             return True
     return False

def check_collision(coefficients):
    result = []
    for i in range(len(coefficients)):
        k1 = -coefficients[i][0]/coefficients[i][1]
        for j in range(len(coefficients)):
            if i == j:
                continue
            k2 = -coefficients[j][0]/coefficients[j][1]
            if k1 != k2:
                result.append((i, j))
                result.append((j, i))
    return result


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