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. 3344 _____ Охрименко Д.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы.

Цель – освоение работы с управляющими конструкциями на языке Python на примере использующей их программы. Для достижения поставленной цели требуется решить следующие задачи:

1) Импортировать модуль numpy;
2) Научиться работать с матрицами, представленных в виде списков;
3) Применить основные управляющие конструкции языка Python в программе.

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 (см. рис. 1):

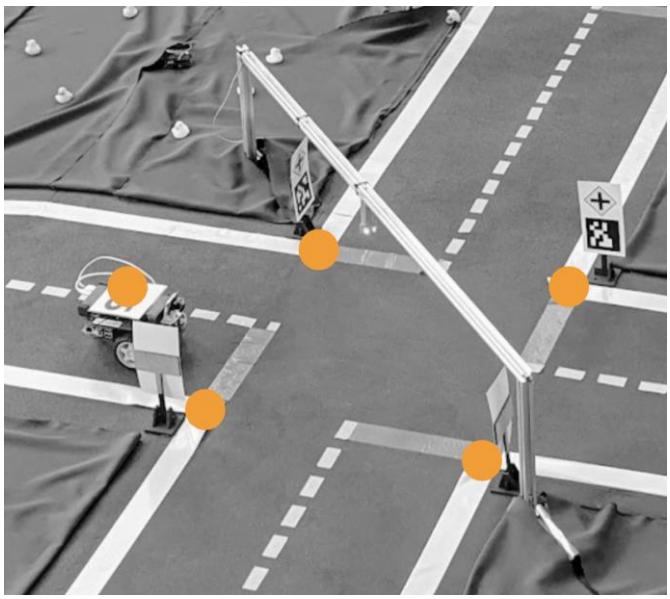

Рисунок 1 – Пример ситуации

Геометрическое представление (см. рис. 2) (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями; обратите внимание, как пронумерованы точки):

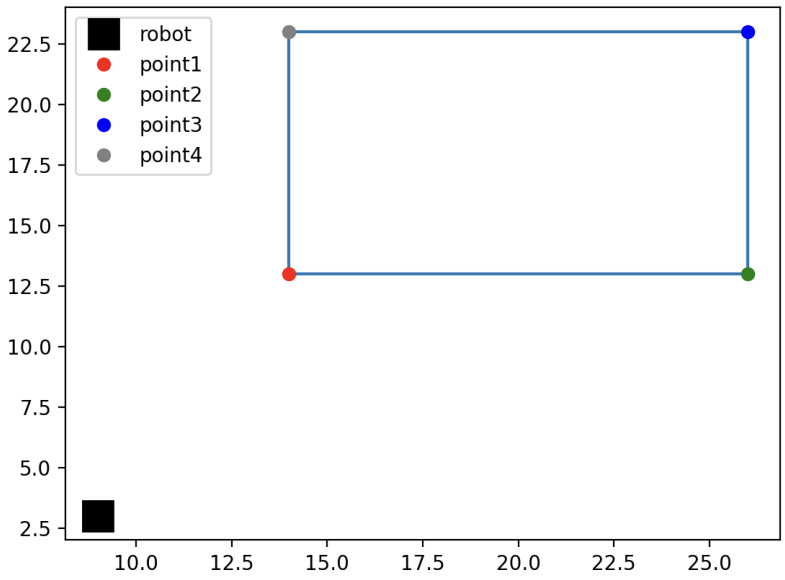

Рисунок 2 – Геометричекое представление

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 (см. рис. 3) (боты нумеруются с нуля, порядок следования коэффициентов уравнений соответствует порядку ботов).

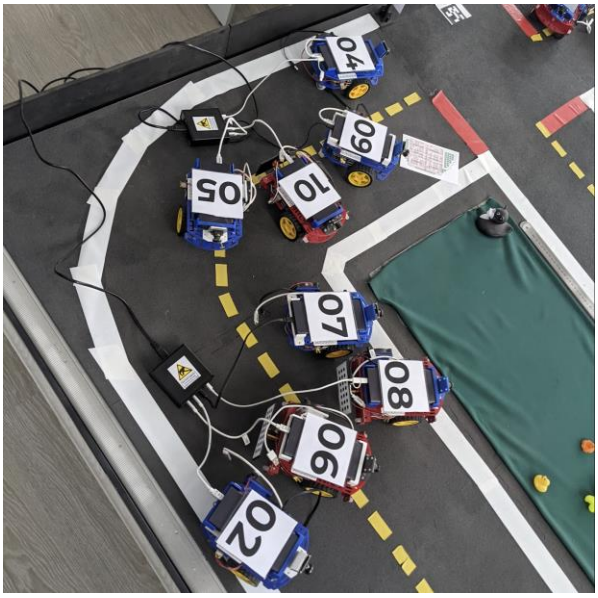

Рисунок 3 – Столкновение дакиботов

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 (см. рис. 4).

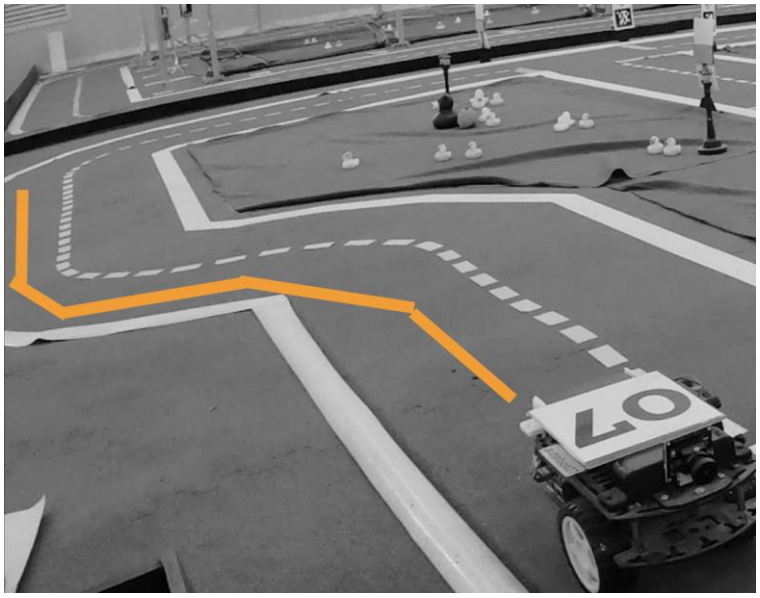

Рисунок 4 – Тректория движения дакибота

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.

Для решения трёх поставленных задач были созданы три соответсвующие функции: (1)check_rectangle(robot, point1, point2, point3, point4), (2)check_collision(coefficients), (3)check_path(points_list).

Рассмотрим их по отдельности:

Функция check_rectangle принимает аргументы robot, point1, point2, point3, point4 – координаты робота и прямоугольника. В ходе работы функции создаётся массив из координат прямоугольника для удобства использования, затем проверятся входит ли координата робота по OX в прямоугольник, если да, то входит ли координата робота по OY в прямоугольник, если да, функция возвращает значение True, во всех иных случаях возвращается False.

Функция check_collision принимает на вход матрицу в виде двумерного массива coefficients. На выход идет массив answer из кортежей типа (a, b), где a и b – порядковые номера столкнувшихся дакиботов. Поскольку столкнуться могут любые два дакибота, начинаем перебирать варианты вложенным циклом:

```
for ind_first in range(len(coefficients)):
    for ind_second in range(len(coefficients)):
        if ind_first != ind_second: #Исключаем случаи столкновение дакибота с самим собой
```

Далее необходимо понять, при каком условии 2 дакибота могут столкнуться. Поскольку траектория движения каждого из них задана уравнением прямой, то они либо имеют одинаковый угол наклона, идут параллельно и никогда не

пересекуться, либо неизбежно столкнуться. В связи с этим мы должны проверить равны ли коэффициенты двух дакиботов перед x, равные a1/b1 и a2/b2 при уравнениях траектории дакиботов: a1x + b1y + c =0 и a2x + b2y + c = 0, y = a1/b1x - c/b1 и y = a2/b2 - c/b2 и при их равенстве исключить таких дакиботов из ответа.

Для нахождения пути функция check_path использует формулу sqrt((x1 – x2) ** 2 + (y1 – y2) ** 2) для каждой пары из двух последовательных точек. Например, при входных данных: [(2.0, 3.0), (4.0, 5.0), (5.0, 6.0)] находится расстояние от (2.0, 3.0) до (4.0, 5.0), а затем от (4.0, 5.0) до (5.0, 6.0). Ответ является суммой всех найденных расстояний. Для реализации используем цикл for, работающий до того, как массив точек points_list не будет полностью проитерирован. Каждую очередная итерация выдаёт точку finish. Начальная точка — start = pop(0), после очередного прохода заменяется на значение finish. Результат округляем с помощью функции round(<array>, 2) из библиотеки numpy.

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.	*(0,2), (1,2), (5,2), (5,4), (1,4)* *[[9,5,7], [-3,3,10]]* *[(1, 2), (2, 3)]*	*False* *[(0, 1), (1, 0)]* *1.41*	Верный вывод
2.	*(5,16), (5,1), (14,1), (14,12), (5,12)* *[[1,2,3], [-5,6,8]]* *[(1, -2), (-2, 3)]*	*False* *[(0, 1), (1, 0)]* *5.83*	Верный вывод
3.	*(5, 11), (5, 1), (14, 1), (14, 12), (5, 12)* *[[20, 40, 32], [-1, -*	*True* *[]*	Верный вывод

	2, 832]] [(0, 0), (4, 3)]	5.0	

Выводы.

1) Была исследована библиотека numpy, применена функция numpy.round
2) Проведена работа с массивами как с матрицами; функции принимали, создавали и обрабатывали списки чисел.
3) Изучены такие управляющие функции языка Python, как встроенные функции pop(), round(), append(), len(); использованы вложенные циклы for; реализованы функции и безымянные (lambda) функции.

   Разработана программа, берущая на вход данные в виде списка и кортежей. Для обработки данных созданы функции. В основе решения каждой подзадачи лежит анализ математической составляющей. С помощью условного оператора if, вложеннных циклов for, безымянной функции lambda задуманные идеи о пересечении прямых, нахождении пути, были преобразованы в программный код.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


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


def check_collision(coefficients):
    answer = []
    for ind_first in range(len(coefficients)):
        for ind_second in range(len(coefficients)):
            if ind_first != ind_second:
                if not (
coefficients[ind_first][0]    /    coefficients[ind_first][1]    ==
coefficients[ind_second][0] / coefficients[ind_second][1]
                ):
                    answer.append((ind_first, ind_second))
    return answer


def check_path(points_list):
    answer = 0
    S = lambda x, y: np.sqrt((x[0] - y[0]) ** 2 + (x[1] - y[1])
     ** 2)
    start = points_list.pop(0)
    for points in points_list:
        answer += S(start, points)
        start = points
```