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. 3341 _____ Кудин А.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью работы было изучение и практическое применение принципов программирования на языке Python, включая использование списков, циклов, словарей, множеств, функций, и основ работы модуля numpy.

Задание

Вариант 2

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).

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)

Результат: 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

Программа, написанная на языке Python, реализует три задачи, которые описаны в разделе “задание” как функции.

Первая функция (check_crossroad) проверяет, пересекает ли заданная точка robot прямоугольник, определенный точками point1, point2, point3 и point4. Если точка находится внутри этого прямоугольника, функция возвращает True, в противном случае - False.

Вторая функция (check_collision) принимает массив coefficients, который содержит коэффициенты. Она проверяет, есть ли пересечения между прямыми, заданными этими коэффициентами. Если есть, то эти прямые пересекаются, и их индексы добавляются в список collisions, который затем возвращается.

Третья операция (check_path) принимает список точек points_list. Она вычисляет общую длину пути, который проходит через эти точки, используя формулу для расстояния между двумя точками в двумерном пространстве. Функция возвращает округленное значение общей длины пути.

Переменные, используемые в этой программе:

- robot: Это точка, представленная как список с двумя координатами (x, y), которую мы проверяем на пересечение с прямоугольником.
- point1, point2, point3, point4: Эти переменные представляют координаты четырех точек, определяющих прямоугольник для проверки пересечения с точкой robot.

- coefficients: Это массив, который содержит коэффициенты прямых для каждой точки в заданной системе координат.
- Vi и Vj: Это векторы, вычисленные с помощью функции get_vector для пары коэффициентов coefficients[i] и coefficients[j].
- M: Это матрица, которая содержит векторы Vi и Vj в качестве ее строк. Матрица используется для проверки их линейной независимости.
- collisions: Это список, в который добавляются пары индексов (i, j), для которых обнаружено пересечение.
- points_list: Это список точек, через которые проходит путь. Каждая точка представлена парой координат (x, y).
- distance: Это переменная, используемая для накопления длины пути при вычислении общей длины пути в функции check_path.
- total_distance: Это переменная, в которой сохраняется общая длина пути после вычислений в функции check_path.

Функции, используемые в этой программе:

- check_crossroad(robot, point1, point2, point3, point4): Проверяет, находится ли точка robot внутри заданного прямоугольника.
- get_vector(cf): Вычисляет вектор, проходящий через две точки на основе коэффициентов cf.
- check_collision(coefficients): Проверяет наличие пересечений между прямыми, заданными коэффициентами, и возвращает список пересекающихся пар.
- check_path(points_list): Вычисляет общую длину пути, проходящего через заданные точки points_list.

Эта программа демонстрирует управляющие конструкции языка Python, функции и модули, а также основы работы с библиотекой 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.	(9, 3) (14, 13) (26, 13) (26, 23) (14, 23)	False	ф-ия check_crossroad
2.	[[-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)]	ф-ия check_collision
3.	[(1.0, 2.0), (2.0, 3.0)]	1.41	ф-ия check_path
4.	[(2.0, 3.0), (4.0, 5.0)]	2.83	ф-ия check_path

Выводы

Были изучены управляющие конструкции языка Python, функции и модули, основы работы с библиотекой NumPy.

Создана и описана программа для решения 3 задач, которые возникают при управлении дакиботом.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np
import math


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



def get_vector(cf):
    dot1 = np.array( [0, -cf[2]/cf[1]] )
    dot2 = np.array( [1, (-cf[0] - cf[2]) / cf[1]] )
    return dot2 - dot1

def check_collision(coefficients):
    collisions = []
    for i in range(coefficients.shape[0]):
        for j in range(coefficients.shape[0]):
            if (i != j):
                Vi = get_vector(coefficients[i])
                Vj = get_vector(coefficients[j])
                M = np.array([Vi, Vj])


                if (np.linalg.matrix_rank(M) == 2):
                    collisions.append( (i, j) )
    return collisions
```

```
def check_path(points_list):
    total_distance = 0.0

    for i in range(1, len(points_list)):
        x1, y1 = points_list[i-1]
        x2, y2 = points_list[i]
        distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
        total_distance += distance

    return round(total_distance, 2)
```