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

Студентка гр. 2381 _____ Малюская Д.И.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Изучить основные управляющие конструкции языка Python, а также модуль numpy.

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).

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).

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.

Выполнение работы.

Функция check_crossroad(robot, point1, point2, point3, point4).

Функция принимает на вход кортежи содержащие в себе координаты работа и точек, ограничивающих перекресток. Функция проверяет, находится ли робот внутри прямоугольника путем сравнения его координат с координатами вершин прямоугольника. Функция возвращает True, если условие выполняется и False в противном случае.

Функция 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.

Двумя циклами for перебираются все пары линейных уравнений без повторений. В m записываются np-массив содержащий в себе коэффициенты линейных уравнений. В v записываются свободные члены. С помощью функции try-except выявляем, пересеклись ли боты в пути. Если пересеклись, то записываем кортеж, включающий их номера в прямом и обратном порядке, в массив ans. С помощью функции sorted() упорядочиваем номер ботов, которые пересеклись.

Функция check_path(points_list).

Функция принимает на вход список кортежей, которые содержат в себе координаты точек, через которые прошел робот. С помощью цикла for программа по теореме Пифагора считает длину отрезков между двумя координатами дакибота, а затем прибавляет ее к общему пути len_path. Функция возвращает округленное до 2 знаков с помощью функции round() значение пути, пройденного дакиботом.

Выводы.

Были изучены основные управляющие конструкции языка Python. Изучены методы библиотеки numpy.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


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

def check_collision(coefficients):
        ans = []
        for i in range(0, len(coefficients)):
            for j in range(i + 1, len(coefficients)):
                m = np.array([[coefficients[i][0], coefficients[i][1]],
[coefficients[j][0], coefficients[j][1]]])
                v = np.array([coefficients[i][2], coefficients[j][2]])
                try:
                    np.linalg.solve(m, v)
                    ans.append((i, j))
                    ans.append((j, i))

                except:
                    pass
        return sorted(ans)

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