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. 2381 _____ Тищенко А. М.

Преподаватель _____ Жангиров Т. А.

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.

Основные теоретические положения.

Задача 1: используется метод numpy prod, возвращающий произведение элементов массива.

Задача 2: используются метод numpy vstack для получения матрицы 2x2 и метод linalg det для нахождения определителя матрицы.

Задача 3: используется метод linalg norm для нахождения длины вектора.

Выполнение работы.

Задача 1: переменные: *vec1, vec2* – векторы проведенные из point*1* в *robot* и из *robot* в *point3* соответственно.

Программа находит векторы, проверяет их координаты на знак и возвращает результат.

Задача 2: переменные: *ans* для хранения ответа, *i, j* для хождения по массиву, *eq1, eq2* для хранения коэффициентов уравнения прямой.

Программа проходиться по всем парам дакиботов в массиве, и проверяет на параллельность их траектории, записывая пары непараллельно идущих дакиботов в *ans*, а после возвращает их.

Задача 3: переменные: *path* хранящий длину пути, *vec1*,*vec2* для хранения соседних точек траектории.

Программа поочередно находит вектора, по которым движется дакибот, считает их длину и сохраняет её в *path*, после возвращает общую длинну.

Разработанный программный код см. в приложении А.

Выводы.

Были изучены основные управляющие конструкции языка Python, а также модуль numpy.

Разработаны функции проверки попадание на перекресток дакибота, нахождения номеров столкнувшихся дакиботов, и пути пройдённый дакиботом. Первая проверяла вектора на положительные координаты, и если все оказывались больше нуля, то возвращала истину. Вторая проверяла траектории на параллельность с помощью определителя матрицы, если он равен нулю, то прямые параллельны. Третья искала сумму длин векторов между точками траектории.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


def check_crossroad(robot, point1, point2, point3, point4) -> bool:
    robot, point1, point3 = map(np.array,[robot, point1, point3])
    vec1, vec2 = (robot - point1) >= 0, (point3 - robot) >= 0
    return bool(np.prod(vec1) and np.prod(vec2))


def check_collision(coefficients) -> list:
    ans = []
    for i in range(len(coefficients)):
        for j in range(len(coefficients)):
            if i == j: continue
            eq1 = np.array(coefficients[i][:2])
            eq2 = np.array(coefficients[j][:2])
            if np.linalg.det(np.vstack((eq1,eq2))):
                ans.append((i,j))
    return ans


def check_path(points_list) -> float:
    path = 0
    for i in range(1,len(points_list)):
        vec1,       vec2       =       np.array(points_list[i-1]),
np.array(points_list[i])
        path += np.linalg.norm(vec2-vec1,ord=2)
    return round(path,2)
```