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. 3344 Волков А.А.

Преподаватель Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Освоить основные управляющие конструкции языка программирования Python.

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. 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). Оформите задачу как отдельную функцию: *def check_cross-road(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.

Задача 2. 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). 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).

Задача 3. 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. 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)*).

Выполнение работы

Подключается библиотека *numpy*.

Функции:

1. Функция *check_crossroad*():

Получает на вход координаты робота и координаты четырёх точек, образующих перекрёсток. Переменные *robot_x* и *robot_y* хранят в себе координаты робота x и y соответственно. С помощью условного оператора проверяет, что координаты робота лежат внутри прямоугольника. Возвращает *True*, если робот находится внутри перекрёстка, и *False* в противном случае.

2. Функция *collision():*

Получает на вход матрицу коэффициентов уравнения ax+by+c=0 *ndrray* размером Nx3. В *count_robot* с помощью метода *shape* записывается количество строк в матрице, что равно количеству роботов. Далее функция проверяет параллельность прямых. Для этого она сравнивает отношение коэффициентов (перед и y). Если прямые параллельны – столкновений между двумя роботами не было, в противном случае столкновение было. Функция возвращает пары индексов, чьи прямые пересекаются.

3. Функция *check_path()*:

Получает на вход список точек, через которые проходил робот. Для каждый пары по теореме Пифагора вычисляется пройденный путь робота. Переменная *length* отвечает за общий путь, который определяется как сумма всех вычисленных расстояний. Функция возвращает длины пути, пройденного роботом, округлённую до сотых с помощь *round(length, 2)*.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	check_crossroad((9, 3) (14, 13) (26, 13) (26, 23) (14, 23))	False	Верный ответ
2.	check_crossroad((5, 8) (0, 3) (12, 3) (12, 16) (0, 16))	True	Верный ответ
3.	collision([[-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)]	Верный ответ
4.	check_path([(1.0, 2.0), (2.0, 3.0)])	1.41	Верный ответ
5.	check_path([(2.0, 3.0), (4.0, 5.0)])	2.83	Верный ответ

Выводы

Было изучено применение основных управляющих конструкций языка программирования Python.

Была разработана программа, которая включает в себе функции, позволяющие определять нахождение роботов внутри заданных областей, проверять столкновение роботов между собой и вычислять длину пути, пройденного роботом.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb1.py

```
import numpy as np


def check_crossroad(robot, point1, point2, point3, point4):
    robot_x, robot_y = robot
    point1_x, point1_y = point1
    point2_x, point2_y = point2
    point4_x, point4_y = point4
    if point1_x <= robot_x <= point2_x and point1_y <= robot_y <= point4_y:
        return True
    else:
        return False


def check_collision(coefficients):
    count_robot = coefficients.shape[0]  # количество строк = количество
дакиботов
    collision = []
    for i in range(count_robot):
        for j in range(count_robot):
            if i != j:
                # проверка на параллельность прямых
                ratio_x = coefficients[i][0] / coefficients[j][0]
                ratio_y = coefficients[i][1] / coefficients[j][1]
                if ratio_x != ratio_y:
                    collision.append((i, j))
    return collision


def check_path(points_list):
    # по теореме Пифагора
    length = 0
    for i in range(len(points_list) - 1):
        x1, y1 = points_list[i]
        x2, y2 = points_list[i + 1]
        length += ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
    length = round(length, 2)
    return length
```