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. 3343 _____ Иванов П.Д.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2023

Цель работы

Изучить основные управляющие конструкции языка Python, а также библиотеку NumPy

Задание

Вариант лабораторной работы состоит из 3 задач, требуется оформить каждую задачу в виде отдельной функции согласно условиям задач.

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. Задача - помочь дакиботу понять, находится ли он на перекрестке (внутри прямоугольника).

Формальная постановка задачи

Оформить задачу как отдельную функцию: def check_crossroad(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. 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.

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).

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. 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)).

Выполнение работы

Была написана программа на языке Python, где каждая задача реализована в отдельной функции.

Для реализации первой задачи была написана функция check_crossroad, которая возвращает True либо False в зависимости от того, находится дакибот на перекрестке или нет соответственно.

Для второй задачи была написана функция check_collision. Данная функция проверяет траектории движения ботов и в случае если они пересеклись добавляет номера этих ботов кортежем в массив res. Для проверки пересечения траекторий используется функция solve() из модуля NumPy. Данная функция при отсутствии столкновения (решения у системы уравнений) возвращает ошибку linalg.LinAlgError. Чтобы избежать завершения программы в этом случае была использована конструкция try/exept.

Третья задача была решена в функции check_path, которая вычисляет пройденный дакиботом путь с помощью т.Пифагора. Для этого расчитывается разница между начальной точкой по оси Y и оси X. Затем эти значения возводятся в квадрат и вычисляется корень из их суммы. Функция округляет значение пути до 2-х цифр после запятой с помощью round() и возвращает это значение.

Также в программе использовались следующие функции и методы из модуля NumPy: tolist() - позволяет преобразовать массив ndarray во вложенный список; array() - преобразует вложенный список в массив ndarray.

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	Работа функции *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)]	Работа функции *check_collision()*.
3.	[(1.0, 2.0), (2.0, 3.0)]	1.41	Работа функции *check_path()*.

Выводы

Была написана программа, которая позволяет обрабатывать данные, получаемые от дакиботов. Для этого использовались встроенные инструменты языка Python и библиотеки NumPy, а именно:

1. Условные конструкции if/else, циклы for, вложенные списки, ввод и вывод данных с клавиатуры. Благодаря им возможна гибкая настройка обработки данных.

2. Объявление функций и работа с ними. Благодаря функциям код становится проще читать, а также уходит проблема переписывания больших частей одинакового кода.

3. Функции библиотеки NumPy, которые упрощают работу с матрицами и помогают в решении задач линейной алгебры.

Разработанная программа позволяет считывать данные, которые поступают из потока ввода, выполнять с ними различные операции (вычисление траекторий, подсчет пройденного пути, проверка столкновений) и выводить результат вычислений в поток вывода.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

def check_crossroad(robot, point1, point2, point3, point4):
    horizontal_down = point1[1]
    horizontal_up = point4[1]

    vertical_left = point1[0]
    vertical_right = point2[0]

         between_horizontal  =  horizontal_down  <=  robot[1]  <=
horizontal_up
    between_vertical = vertical_left <= robot[0] <= vertical_right

    return between_horizontal and between_vertical

def check_collision(coefficients_np):
    res = []
    coefficients = coefficients_np.tolist()
    for first in range(len(coefficients)):
        for second in range(len(coefficients)):
            if first == second:
                continue

            a = np.array([coefficients[first][0:-1]\
coefficients[second][0:-1]])
           b = np.array([coefficients[first][-1]\
coefficients[second][-1]])

            try:
                np.linalg.solve(a, b)
                res.append((first, second))
            except np.linalg.LinAlgError:
                continue
    return res

def check_path(points_list):
    way = 0
    for i in range(len(points_list) - 1):
        first_point = points_list[i]
        second_point = points_list[i+1]

        y_diff = first_point[1] - second_point[1]
        x_diff = first_point[0] - second_point[0]

        way += (y_diff**2 + x_diff**2)**0.5
    return round(way, 2)
```