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. 3342 _____ Попадюк И.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью работы является освоение работы с функциями в языке python и с библиотекой numpy, а также написание программы, состоящей из трех функций.

Задание

Вариант 2.

Задача 1.

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.

Задача 2.

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.

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

Для работы с матрицами была использована библиотека *numpy*. В программе были реализованы следующие функции:

Первая функция *check_crossroad* принимает координаты дакибота и координаты четырех точек перекрестка. Функция сравнивает координаты дакибота и точек перекрестка и возвращает *True* если дакибот находится внутри него. Иначе функция возвращает *False*.

Вторая функция *check_collision*. Функция возвращает список пар в виде кортежей - номера столкнувшихся ботов (если никто из ботов не столкнулся, возвращается пустой список). Для ее реализации были использованы два цикла, переменные-итераторы которых являются индексами строк матрицы с коэффициентами линейных уравнений. Внутри циклов создаются массивы с коэффициентами соответствующих строк матрицы. Затем с помощью функции из библиотеки *numpy array* создается матрица, которая содержит в себе эти два массива. С помощью функции из библиотеки *numpy linalg.matrix_rank* вычисляется ранг матрицы, чтобы понять, если ли решение у системы. Если ранг матрицы равен двум, значит решение есть и роботы столкнутся. Если роботы столкнулись, то в массив *collisions* записываются кортежи с номерами столкнувшихся роботов. После всех итераций функция возвращает массив *collisions.*

Третья функция *check_path* принимает список двумерных точек *points_list* и вычисляет длину пути, проходящего через эти точки. Для этого используется формула расстояния между двумя точками на плоскости.

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Результат вычислений округляется до двух знаков после запятой и возвращается в виде числа с плавающей точкой.

Переменные, используемые в программе:

- *collisions* – список из кортежей с номерами столкнувшихся дакиботов.
- *result* – сумма длин путей дакибота.

Данная программа демонстрирует использование функций библиотеки *numpy* и работу функций на языке *Python* для выполнения различных математический операций.

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.	(5, 10), (2, 9), (9, 9), (9, 15), (2, 15)	True	
2.	[[2 4 8] [-1 5 3] [9 -3 8]]	[(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]	
3.	[(1.0, -3.0), (3.0, 7.0), (3.6, 11)]	14.24	

Выводы

Были изучены правила работы с функциями в языке *python* и работа с библиотекой *numpy*.

Разработаны функции, возвращающие решения определенных математических заданий.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


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


def check_collision(coefficients):
    collisions = []
    for i in range(len(coefficients)):
        arr = coefficients[i][0:2]
        for j in range(len(coefficients)):
            arr_insert = coefficients[j][0:2]
            matrix = np.array([arr, arr_insert])
            rank_of_matrix = np.linalg.matrix_rank(matrix)
            if rank_of_matrix == len(arr):
                collisions.append((i, j))
    return collisions


def check_path(points_list):
    result = 0
    for i in range(len(points_list)-1):
        x0, x1 = points_list[i][0], points_list[i+1][0]
        y0, y1 = points_list[i][1], points_list[i+1][1]
        result += ((x1 - x0) ** 2 + (y1 - y0) ** 2)**0.5
    return round(result, 2)
```