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. 3342 _____ Иванов С.С.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью работы является освоение работы с функциями и функционалом библиотеки numpy.

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

Данная программа написана на языке Python с использованием библиотеки numpy. Она состоит из 3-функций, которые вызываются сразу на сайте https://e.moevm.info.

1. check_crossroad:

Эта функция принимает пять аргументов: robot и четыре кортежа point. Она проверяет, находится ли робот в пределах прямоугольника, определенного четырьмя точками. Сначала создаются списки, содержащие значения x и y координат вершин прямоугольника. Затем вычисляются максимальные и минимальные значения x и y. Наконец, функция проверяет, находятся ли координаты x и y робота в пределах границ прямоугольника. Если оба условия выполняются, функция возвращает True, указывая на то, что робот находится на перекрестке.

2. check_collision:

Эта функция принимает список коэффициентов в качестве входных данных. Она выполняет итерацию по всем парам коэффициентов из входного списка и формирует матрицу размером 2x2 для каждой пары. С использованием возможностей линейной алгебры в NumPy, функция затем проверяет, равен ли ранг сформированной матрицы 2. Если это так, это означает, что два робота, определенных этими коэффициентами, находятся на курсе столкновения. Затем функция добавляет индексы сталкивающихся роботов в список collisions. Наконец, она возвращает список пар столкновений.

3. check_path:

Эта функция принимает список точек в качестве входных данных. Она выполняет итерацию по списку, вычисляя расстояние между соседними точками и накапливая общее расстояние. Она использует формулу расстояния для вычисления расстояния между каждой парой соседних точек. Вычисленные

расстояния затем суммируются для получения общей длины пути. Функция возвращает общую длину пути, округленную до двух десятичных знаков, указывая на общее расстояние, пройденное вдоль пути.

Разработанный программный код см. в приложении А.

Выводы

В результате выполнения лабораторной работы были изучены правила работы с функциями и библиотекой numpy, они же применены на практике, которая заключалась в реализации нескольких функций, которые выполняли определённые математические действия.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np
from itertools import product

def check_crossroad(robot, point1, point2, point3, point4):
    rec_points = [point1, point2, point3, point4]

    x_vals = [pt[0] for pt in rec_points]
    y_vals = [pt[1] for pt in rec_points]

    max_x = max(x_vals)
    max_y = max(y_vals)

    min_x = min(x_vals)
    min_y = min(y_vals)

    return min_x <= robot[0] <= max_x and min_y <= robot[1] <= max_y


def check_collision(coefficients):
    collisions = []

    for (i, robot1), (j, robot2) in product(enumerate(coefficients),
repeat=2):
        matrix = np.array([robot1, robot2])

        if np.linalg.matrix_rank(matrix) == 2:
            collisions.append((i, j))

    return collisions


def check_path(points_list):
    sum_path = 0

    for i in range(1, len(points_list)):
        x1, y1 = points_list[i-1]
        x2, y2 = points_list[i]

        dx = x2 - x1
        dy = y2 - y1
        sum_path += ((dx*dx + dy*dy) ** .5)

    return round(sum_path, 2)
```