**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. 3341 |  | Романов А.К. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2022

## Цель работы

Отработка навыков работы с основными управляющими конструкциями языка Python и библиотекрй numpy.

Для достижения поставленной цели требуется решить следующие

задачи:

1) Ознакомиться с существующими управляющими конструкциями.

2) Научиться их использовать.

3) Написать программу, решающую задачу в соответствии с заданием.

## Задание

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.

**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))*.

Вариант работы - №2.

## Выполнение работы

Для решения задачи было реализовано три функции для каждой из задач, указанных в условии.

В программе реализованы следующие функции:

* *check\_crossroad(robot, point1, point2, point3, point4)* — данная функция принимает на вход пять кортежей: в первом (*robot*) указаны координаты робота, а в остальных четырех (*point1, point2, point3, point4)* — координаты угловых точек перекрёстка. Точка принадлежит квадрату, если ее координата х находится в промежутке [x1, x2], а точка у — в промежутке [у1, у2], где точки х1, х2 — крайние точки квадрата по оси Ох, а у1, у2 — по оси Оу. Данная проверка в функции реализовнана при помощи двойного неравенства. Если точка принадлежит квадрату, функция возвращает *True*, иначе — *False*. Примечание: в условие указано, что функцию надо назвать *check\_rectangle*, однако при проверке выяснилось, что ее следует назвать *check\_crossroad.*
* *check\_collision(coefficients) —* данная функция принимает на вход матрицу *coefficients* размером N\*3. (N — количество ботов, может варьироваться). Функция попарно проверяет факт столкновения ботов. Для этого реализован перебор строчек матрицы двумя циклами. На каждом шаге с использованием функции *np.array()* мы получаем матрицу из двух строчек и трех столбцов — расширенную матрицу системы. Далее, если ранг расширенной матрицы функции равен рангу матрицы функции и равен двум (посскольку мы работаем с системой из двух уравнений), мы можем делать вывод о том, что прямые пересекаются, соответственно данная пара роботов сталкивалась. (Проверка ранга функции осуществлялась при помощи *matrix\_rank()* из модуля *linalg* библиотеки *numpy*). После в массив результатов *res* добавляется два кортежа с номерами столкнувшихся роботов. (Например, если столкнулись роботы А и Б, то необходимо добавить как факт столкновения робота А с роботом Б, так и факт столкновения Б с А). Функция возвращает отсортированный по возрастанию массив *res*.
* *check\_path(points\_list) —* данная функция принимает на вход массив *points\_list*, состоящийй из кортежей, в которых хранятся координаты точек. Массив преобразуется в *np.array.* Далее с помощью цикла осуществляется проход по всем элементам (кортежам) массива *points\_list.*  На каждом шаге цикла определяются координаты вектора перемещения путем вычитания координат предыдущей точки их соответствующих координат текущей точки *([points\_list[i, 0] - points\_list[i-1, 0], points\_list[i, 1] - points\_list[i-1, 1]]).* Затем при помощи функции *norm* из модуля *linalg* библиотеки *numpy* вычисляется длина вектора перемещения. Она прибавляется к итоговому результату *res*. Функция возвращает значение *res*, округленное до двух символов после точки.

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 |
|  | check\_crossroads input:  (4, 17) (4, 0) (17, 0) (17, 10) (4, 10)  check\_collision input:  [[ 6 4 8]  [ 9 -7 1]  [ 3 6 8]  [ 2 2 10]  [-6 -9 0]  [-1 -1 5]]  check\_path input:  [(1.0, 2.0), (2.0, 3.0)] | False  [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 0), (1, 2), (1, 3), (1, 4), (1, 5), (2, 0), (2, 1), (2, 3), (2, 4), (2, 5), (3, 0), (3, 1), (3, 2), (3, 4), (4, 0), (4, 1), (4, 2), (4, 3), (4, 5), (5, 0), (5, 1), (5, 2), (5, 4)]  1.41 |  |
|  | check\_crossroads input:  (14, 4) (13, 14) (25, 14) (25, 26) (13, 26)  check\_collision input:  [[ 1 -8 0]  [-5 -6 1]  [-6 -3 1]  [-1 9 5]  [ 9 9 2]  [-3 5 9]  [ 4 -6 0]]  check\_path input:  [(3.64, 1.07), (3.59, 1.13), (3.48, 1.27), (3.53, 1.76)] | False  [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 0), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 0), (2, 1), (2, 3), (2, 4), (2, 5), (2, 6), (3, 0), (3, 1), (3, 2), (3, 4), (3, 5), (3, 6), (4, 0), (4, 1), (4, 2), (4, 3), (4, 5), (4, 6), (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 6), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5)]  0.75 |  |

## Выводы

Были отработаны навыки работы с основными управляющими конструкциями языка Python и библиотекрй numpy.

Были изучены правила работы с некоторыми функциями библиотеки numpy.

Разработаны функции:

1. Проверяющая принадлежность точки квадрату
2. Проверяющая пересечение двух прямх
3. Рассчитывающая длину вектора перемещения

# Приложение А Исходный код программы

Название файла: main.py

Программный код:

import numpy as np

def check\_crossroad(robot, point1, point2, point3, point4):

x1, x2 = point1[0], point3[0]

y1, y2 = point1[1], point3[1]

if (x1 <= robot[0] <= x2) and (y1 <= robot[1] <= y2):

return True

return False

pass

def check\_collision(coefficients):

res = []

for i in range(0, len(coefficients) - 1):

for j in range(i + 1, len(coefficients)):

syst = np.array((coefficients[i], coefficients[j]))

if np.linalg.matrix\_rank(syst[:, :-1]) == np.linalg.matrix\_rank(syst) == 2:

res.append((i, j))

res.append((j, i))

else:

continue

res.sort()

return res

pass

def check\_path(points\_list):

points\_list = np.array(points\_list)

res = 0

for i in range(1, len(points\_list)):

vec = np.array([points\_list[i, 0] - points\_list[i - 1, 0], points\_list[i, 1] - points\_list[i - 1, 1]])

res += np.linalg.norm(vec)

res = round(res, 2)

return res