**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 |  | Стрижков И.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2023

## Цель работы

Научиться создавать простые программы на языке программирования Python с использованием условий, циклов, списков, а также с модулем numpy.

## 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. 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. 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. Ваша задача – помочь дакиботу понять, находится ли он на перекрестке (внутри прямоугольника).

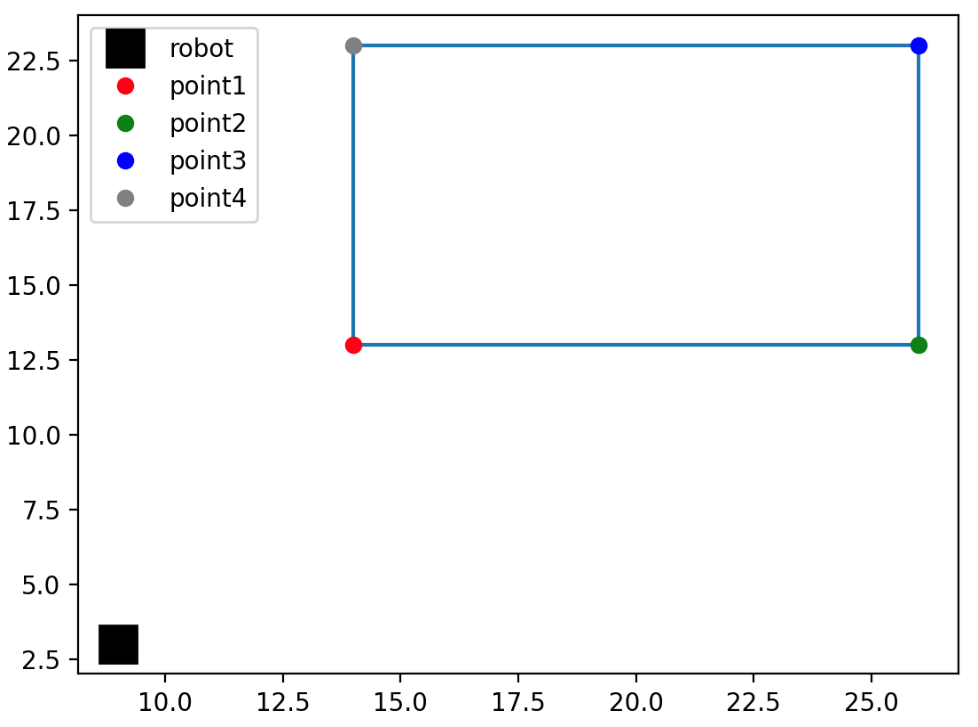


Рисунок 1 – Расположение точек перекрёстка

*Формальная постановка задачи*

Оформите задачу как отдельную функцию: def check\_rectangle(robot, point1, point2, point3, point4). 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.

Ваша 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).

Примечание: помните про ранг матрицы и как от него зависит существование решения системы уравнений. В случае, если ни одного решение не было найдено (например, из-за линейно зависимых векторов), функция должна вернуть 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. Ваша 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

Задача 1. Для того чтобы проверить, находится ли дакибот на перекрестке, нужно проверить условие, что его x-координата находится между x-координатами первой и третьей точек, а его y-координата находится между y-координатами первой и третьей точек. Для этого в функцию передаются аргументы: координаты робота и координаты четырех точек первого, второго, третьего и четвертого. Координаты робота и точек извлекаются из переданных аргументов. Выполняется проверка условия, если оба условия выполняются, это означает, что координаты робота находятся внутри перекрестка. Если условие выполняется, функция возвращает `True`, что означает, что робот находится в перекрестке. Если условие не выполняется, функция возвращает `False`, что означает, что робот не находится в перекрестке. Таким образом, эта функция проверяет, находится ли робот в заданном перекрестке на основе его координат и координат четырех точек, и возвращает соответствующий булевский результат.

Задача 2. Для того чтобы проверить, столкнулись ли дакиботы, нужно проверить, пересекались ли их траектории движения. Для этого в функции check\_collision() создаем пустой список `collisions` для хранения пар номеров столкнувшихся ботов. Дальше во внешнем и внутреннем цикле перебирает все боты в матрице `coefficients`. Если индексы `i` и `j` не равны (чтобы избежать сравнения бота с самим собой), выполняется следующий код:

a. Получаем коэффициенты a и b для текущих ботов `a1, b1 = coefficients[i][0], coefficients[i][1]` и `a2, b2 = coefficients[j][0], coefficients[j][1]`.

b. Вычисляем значение `(a2 \* b1) - (a1 \* b2)`. Если это значение не равно нулю, это означает, что уравнения ботов имеют точку пересечения и боты сталкиваются друг с другом.

c. Если столкновение обнаружено, добавляем пару номеров столкнувшихся ботов в список `collisions` в виде кортежа `(i, j)`. По завершении внутреннего цикла возвращаем список столкновений `collisions`

Задача 3. Для того чтобы посчитать длину траектории, имея координаты ее точек, можно использовать теорему Пифагора для каждой пары соседних (в траектории) точек. Создается переменная `distance` и инициализируется нулем. В эту переменную будет накапливаться общее расстояние между точками. В цикле `for` перебираются индексы `i` от 1 до `len(points\_list) - 1`. Перебор начинается со второй точки, так как для вычисления расстояния требуется предыдущая и текущая точки. Извлекаются координаты предыдущей точки `(x1, y1)` и текущей точки `(x2, y2)` из списка `points\_list`. Вычисляется расстояние между двумя точками с помощью формулы расстояния между двумя точками на плоскости. Вычисленное расстояние `segment\_distance` добавляется к общему расстоянию `distance`. По завершении цикла `for` общее расстояние `distance` возвращается с использованием функции `round()` для округления до двух десятичных знаков.

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 | Функция check\_crossroad  работает корректно |
| 2. | (5, 8) (0, 3) (12, 3) (12, 16) (0, 16) | True | Функция check\_crossroad  работает корректно |
| 3. | [[-1 -4 0]  [-7 -5 5]  [ 1 4 2]  [-5 2 2]]  (в виде ndarray) | [(0, 1), (0, 3), (1, 0),  (1, 2), (1, 3), (2, 1),  (2, 3), (3, 0), (3, 1),  (3, 2)] | Функция check\_collision  работает корректно |
| 4. | [(1.0, 2.0), (2.0, 3.0)] | 1.41 | Функция check\_path  работает корректно |
| 5. | [(2.0, 3.0), (4.0, 5.0)] | 2.83 | Функция check\_path  работает корректно |

## Выводы

Были изучены основные управляющие конструкции языка Python и некоторые функции модуля numpy. Разработана программа, разделенная на независимые функции, выполняющая обработку данных.

# Приложение А Исходный код программы

Название файла: main.py

import numpy as np

import math

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

x\_robot, y\_robot = robot

x1, y1 = point1

x2, y2 = point2

x3, y3 = point3

if x1 <= x\_robot <= x3 and y1 <= y\_robot <= y3:

return True

else:

return False

def check\_collision(coefficients):

collisions = []

for i in range(len(coefficients)):

for j in range(len(coefficients)):

if i != j:

a1, b1 = coefficients[i][0], coefficients[i][1]

a2, b2 = coefficients[j][0], coefficients[j][1]

if (a2 \* b1) - (a1 \* b2) != 0:

collisions.append(tuple([i ,j]))

return collisions

def check\_path(points\_list):

distance = 0

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

x1, y1 = points\_list[i-1]

x2, y2 = points\_list[i]

# Вычисляем расстояние между двумя точками

segment\_distance = math.sqrt((x2 - x1)\*\*2 + (y2 - y1)\*\*2)

distance += segment\_distance

return round(distance, 2)