**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 |  | Антонов Н. Д. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2023

# Цель работы:

Цель данной работы: это разработка программы на языке *Python*, с использованием модуля *numpy* и пакета *numpy.linalg*.

# 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).

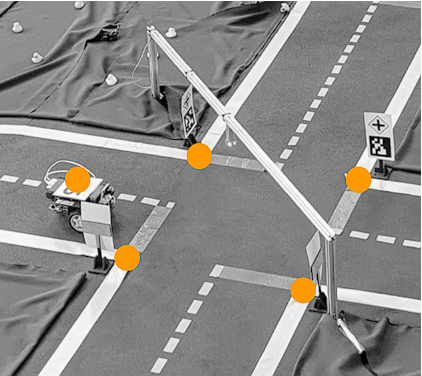


Рисунок 1 – Задача 1

Геометрическое представление (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями; обратите внимание, как пронумерованы точки):

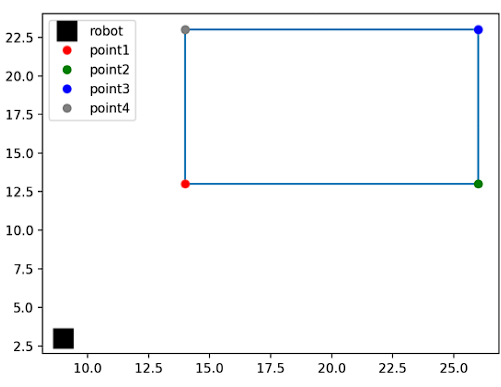


Рисунок 2 – Задача 1

##### **Формальная постановка задачи**

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. 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).

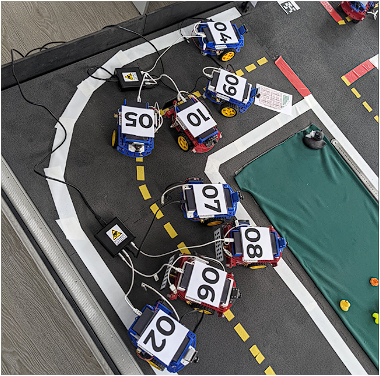


Рисунок 3 – Задача 2

##### **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.

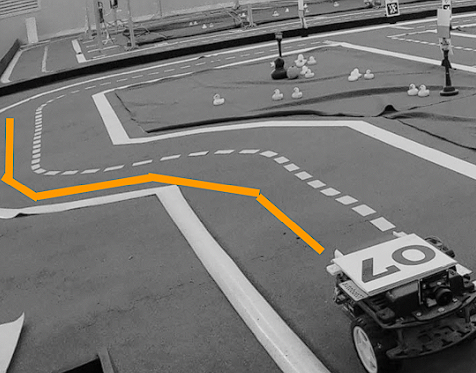


Рисунок 4 – Задача 3

##### **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

Написана программа на языке Python. Выполняет функции для работы и управления дакиботами. Программа состоит из 3 функций, каждая из которых выполняет определённую задачу.

Первая задача, реализованная в функции *check\_crossroad*, функция получает на вход координаты дакибота, а также крайние точки перекрестка.

Функция возвращает ***True***, если дакибот на перекрестке, и ***False***, если дакибот не на перекрестке.

Вторая задача, реализованная в функции *check\_collision*, возвращает индекс последнего отрицательного числа в массиве. На вход подается матрица ***ndarray Nx3*** (*N* - количество ботов, может быть разным в разных тестах) коэффициентов уравнений траекторий *koef*. Функция возвращает список пар - номера столкнувшихся ботов

Третья задача, реализованная в функции *check\_path*, а вход функции подается матрица***ndarray Nx3*** (*N* - количество ботов, может быть разным в разных тестах) коэффициентов уравнений траекторий *koef*. Функция возвращает номера столкнувшихся ботов попарно в виде списка, а если никто из ботов не столкнулся, то возвращается пустой список.

*np.array()* - функция, которая создает массив (матрицу) из указанных данных. В коде она используется для создания матрицы коэффициентов x и y для двух прямых в функции *check\_collision.*

*np.linalg.matrix\_rank()* - функция из модуля NumPy. Она вычисляет ранг матрицы, и в коде она используется для того, чтобы определить, пересекаются ли две прямые. Если ранг матрицы равен 2 – прямые пересекаются.

*np.array([])* - способ создания массива (вектора) из указанных данных. В коде он используется для создания массива разностей между соседними точками в функции *check\_path*.

*np.diff()* - функция NumPy, которая вычисляет разности между элементами массива вдоль указанной оси. В коде она используется для вычисления разности между координатами соседних точек.

*np.linalg.norm()* - функция NumPy, которая вычисляет норму вектора. В коде она используется для вычисления евклидовых расстояний между точками.

*np.sum()* - функция NumPy, которая вычисляет сумму элементов массива. В коде она используется для вычисления общего расстояния между точками, найденного в функции *check\_path*.

Созданная программа позволяет наблюдать за работой функций из модуля *NumPy*.

Программный код находится в приложении А.

# Тестирование:

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | (9, 3) (14, 13) (26, 13) (26, 23) (14, 23) | False | Тестирование функции *check\_rectangle* |
|  | [[-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)] | Тестирование функции *check\_collision* |
|  | [(1.0, 2.0), (2.0, 3.0)] | 1.41 | Тестирование функции *check\_path* |

# Выводы:

Была разработана программа для работы с данными и их вычислениями на языке Python и использованием библиотеки NumPy. Во время разработки программы были изучены и применены

* 1. Вычисления с NumPy: Для решения задач и вычислений расстояний между точками использовались функции из библиотеки NumPy: *np.linalg.norm()* - для вычисления расстояний.

*np.array()* - для работы с матрицами и векторами.

* 1. Циклы *for* и условные конструкции *if* – позволяющие обрабатывать данные и выполнять повторяющиеся операции определённое количество раз, а также принять правильное решение при проверке известных данных.
  2. Функции, которые позволяют нам сделать код более понятным и простым для чтения.

Разработанная программа позволяет пользователю выполнить различные операции с данными, вычислять расстояния и проверять пересечения прямых, используя конструкции и функций из библиотеку NumPy.

# Приложение А Исходный код программы

**Название файла: main.py**

import numpy as np

def check\_crossroad(robot, tk1, tk2, tk3, tk4) -> bool:

return ( robot[0] >= tk1[0] and robot[1] >= tk1[1] and robot[0] <= tk2[0] and robot[1] >= tk2[1] and robot[0] <= tk3[0] and robot[1] <= tk3[1] and robot[0] >= tk4[0] and robot[1] <= tk4[1])

def check\_collision(koef) -> list:

korp = []

for i in range(len(koef)):

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

x1, y1, z1 = koef[i]

x2, y2, z2 = koef[j]

matr = np.array([[x1, y1], [x2, y2]])

if np.linalg.matrix\_rank(matr) == 2:

korp.append((i, j))

korp.append((j, i))

korp.sort()

return korp

def check\_path(tk) -> float:

if len(tk) < 2:

return 0.0

tksp = np.array(tk)

dels = np.diff(tksp, axis=0)

distances = np.linalg.norm(dels, axis=1)

distance = np.sum(distances)

return round(distance, 2)