**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 |  | Гребнев Е.Д. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2023

## Цель работы

Целью работы являлось изучение и практическое применения принципов программирования на языке Python, при этом используя модуль *numpy*, в частности пакет *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).

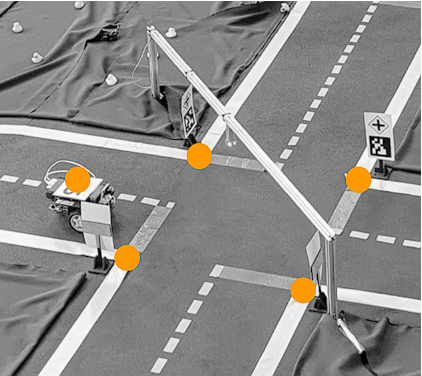


Рисунок 1 – Задача 1

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

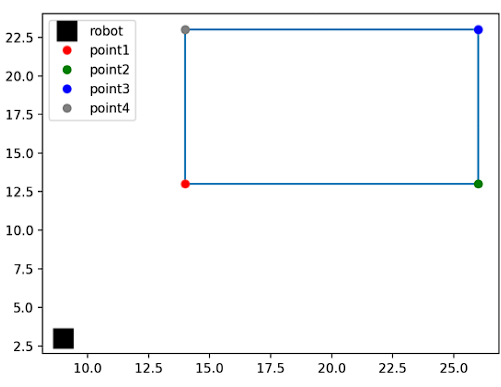


Рисунок 2 – Задача 1

##### **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).

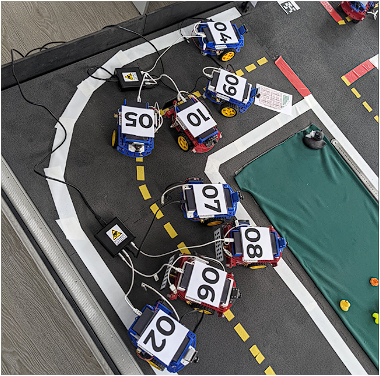


Рисунок 3 – Задача 2

##### **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.

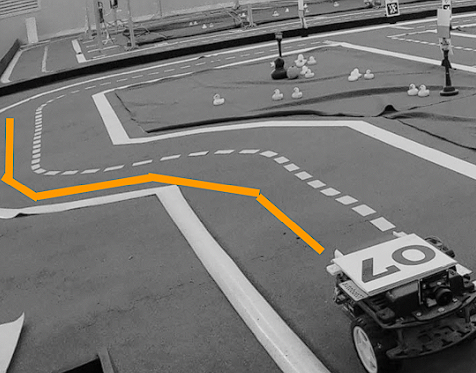


Рисунок 4 – Задача 3

##### **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

Написанная программа написана на языке Python. Выполняет функции для работы и управления дакиботами. Всего в программе представлено 3 функции, для выполнения 3 задач.

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

Функция должна возвращать ***True***, если дакибот на перекрестке, и ***False***, если дакибот вне перекрестка.

Вторая задача, реализованная в функции *check\_collision*, возвращает индекс последнего отрицательного числа в массиве. 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

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 *coefficients*. Функция возвращает список пар - номера столкнувшихся ботов (если никто из ботов не столкнулся, возвращается пустой список).

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

*np.linalg.matrix\_rank():* Это функция из модуля NumPy, которая вычисляет ранг матрицы. Ранг матрицы - это число линейно независимых строк (или столбцов) в матрице. В коде она используется для определения, пересекаются ли две прямые. Если ранг матрицы равен 2, это означает, что две прямые пересекаются.

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

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

*np.linalg.norm():* Это функция NumPy, которая вычисляет норму (длину) вектора. В коде она используется для вычисления евклидовых расстояний между точками. В данном контексте, это помогает определить расстояние между точками.

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

Данная программа демонстрирует использование встроенных конструкций языка Python, а также работу с библиотекой NumPy.

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 |
|  | (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 обработки геометрических данных и вычисления расстояний с использованием Python и библиотеки NumPy. В процессе разработки программы были изучены и применены следующие концепции и элементы языка:

1. Управляющие конструкции: Программа включает в себя конструкции условия (`if`) и циклы (`for`). Условные операторы используются для проверки условий и принятия решений, а циклы для обработки данных и выполнения повторяющихся операций.

2. Функции: В программе объявлены и использованы функции для выполнения конкретных операций. Функции упрощают структуру кода и делают его более читаемым и модульным.

3. Вычисления с использованием NumPy: Для решения задач геометрии и вычисления расстояний между точками использовались функции из библиотеки NumPy, такие как *np.linalg.norm()* для вычисления расстояний и *np.array()* для работы с матрицами и векторами.

4. Ввод и вывод данных: Программа считывает данные с клавиатуры пользователя и выводит результаты обработки на экран.

Таким образом, разработанная программа позволяет пользователям выполнять различные операции с геометрическими данными, вычислять расстояния и проверять пересечения прямых, используя управляющие конструкции, функции и библиотеку NumPy. Это обеспечивает гибкость и точность в решении задач, связанных с геометрией и численными вычислениями.

# Приложение А Исходный код программы

Название файла: main.py

import numpy as np

# Функция check\_crossroad проверяет, пересекает ли робот какой-либо перекресток, заданный 4 точками.

# Она принимает координаты робота (robot) и координаты 4 точек перекрестка (point1, point2, point3, point4).

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

    # Проверяем, находится ли робот внутри перекрестка, используя условия пересечения прямоугольников.

    return (

        robot[0] >= point1[0] and robot[1] >= point1[1] and

        robot[0] <= point2[0] and robot[1] >= point2[1] and

        robot[0] <= point3[0] and robot[1] <= point3[1] and

        robot[0] >= point4[0] and robot[1] <= point4[1]

    )

# Функция check\_collision проверяет пересечение двух прямых, заданных коэффициентами a, b, c.

# Она принимает список коэффициентов coefficients, где каждый элемент - кортеж (a, b, c) для одной прямой.

def check\_collision(coefficients) -> list:

    collisions = []

    for i in range(len(coefficients)):

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

            a1, b1, c1 = coefficients[i]

            a2, b2, c2 = coefficients[j]

            # Создаем матрицу из коэффициентов a и b для двух прямых.

            matrix = np.array([[a1, b1], [a2, b2]])

            # Проверяем ранг матрицы, чтобы определить, пересекаются ли прямые.

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

                collisions.append((i, j))

                collisions.append((j, i))

    collisions.sort()

    return collisions

# Функция check\_path вычисляет общее расстояние между точками в списке points\_list.

def check\_path(points\_list) -> float:

    if len(points\_list) < 2:

        return 0.0

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

    deltas = np.diff(points\_array, axis=0)

    # Используем np.linalg.norm для вычисления евклидовых расстояний между точками.

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

    total\_distance = np.sum(distances)

    return round(total\_distance, 2)