**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. 3344 _____ Хангулян С. К.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург

2023

**Цель работы**

Целью работы является изучение основных управляющих конструкций языка Python и библиотеки numpy.

**Задание**

**Вариант 2**

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).

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).

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).

**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**

Вначале была импортирована библиотека numpy как np.

Функция check_crossroad. Входные данные:

- robot – кортеж, содержащий координаты робота в двухмерной системе координат;
- point1, …, point4 – кортежи, содержащие координаты вершин прямоугольника (перекрестка);

Объявлена следующая переменная:

- arr – список, в который записаны отсортированные по возрастанию координаты вершин прямоугольника.

С помощью функции sorted сортируются координаты вершин прямоугольника, где первый элемент – ближайший к началу координат кортеж, последний элемент – дальний от начала координат кортеж. Далее с помощью условного оператора if проверяется, находятся ли координаты робота внутри перекрестка, учитывая линии перекрестка. В случае попадания функция возвращает True, в противном случае – False.

Функция check_collision. Входные данные:

- coefficients - матрица ndarray Nx3 коэффициентов уравнений траекторий.

Объявлены следующие переменные:

- otv – список ответов, который будет возвращать функция;
- mtr – матрица, содержащая коэффициенты уравнений траекторий двух ботов;
- c – coefficients (укороченная запись для удобства).

С помощью двух циклов for перебираются все комбинации коэффициентов траекторий ботов. Для того, чтобы избежать рассмотрение одного и того же бота, используется условный оператор if. Так как сравниваются списки, используется функция any, находящая хотя бы одно

несоответствие и возвращающая в данном случае False. Далее создается матрица mtr с помощью метода array из коэффициентов уравнений траекторий двух ботов. С помощью условного оператора if проверяются два условия: ранг матрицы >= 2 (так как в матрице 2 строки) и отношения коэффициентов a1/a2 != b1/b2. Последнее условие было выявлено экспериментальным путем. Иной вариант второго условия – расширить матрицу или перевернуть ее на 90 градусов для повторной проверки ранга. В случае, если матрица имеет решения (а это значит, что боты столкнулись), в ответ добавляются номера этих ботов два раза. Сначала вида (бот 0, бот 1), затем вида (бот 1, бот 0). В конце функция возвращает отсортированный (с помощью команды sorted) по возрастанию список ответов. В случае, если столкновений не происходило, функция вернет пустой список.

Функция chech_path. Входные данные:

- point_list – список начальных/промежуточных/конечных координат робота.

Объявлены следующие переменные:

- otv – ответ, путь, пройденный роботом;
- x0 – начальные координаты робота при рассмотрении пар кортежей;
- x – конечные координаты робота при рассмотрении пар кортежей;
- s – путь, пройденный роботом при рассмотрении пар кортежей.

С помощью цикла for идет прогон всех пар кортежей, содержащих координаты. i-й кортеж – начальные координаты – x0, i+1-й кортеж – конечные координаты – x. Применив теорему Пифагора, находим путь s и прибавляем его к ответу otv. После завершения выполнения цикла, возвращаем ответ, округленный до 2 знака с помощью функции round.

**Тестирование**

Результаты тестирования представлены в таблице 1.

Таблица 1 – Результаты тестирования

| Функция | № | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|---|
| check_crossroad | 1 | (9, 3), (14, 13), (26, 13), (26, 23), (14, 23) | False | Корректно |
| check_crossroad | 2 | (5, 8), (0, 3), (12, 3), (12, 16), (0, 16) | True | Корректно |
| check_collision | 1 | [[-1 -4 0],<br>[-7 -5 5],<br>[ 1 4 2],<br>[-5 2 2]] | [(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)] | Корректно |
| chech_path | 1 | [(1.0, 2.0), (2.0, 3.0)] | 1.41 | Корректно |
| chech_path | 2 | [(2.0, 3.0), (4.0, 5.0)] | 2.83 | Корректно |

**Выводы**

Были изучены основные управляющие конструкции языка Python и применены на практике некоторые методы библиотеки numpy, условный оператор if, цикл for. Также были проведены знакомство и работа с массивами данных.

# ПРИЛОЖЕНИЕ А

# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Khangulyan_Sargis_lb1.py

```
import numpy as np

def check_crossroad(robot, point1, point2, point3, point4):
    arr = sorted([point1, point2, point3, point4])
    if arr[0][0] <= robot[0] <= arr[3][0] and arr[0][1] <= robot[1] <=
arr[3][1]:
        return True
    else:
        return False

def check_collision(coefficients):
    otv = []
    c = coefficients
    for i1 in range(len(c)):
        for i2 in range(len(c)):
            if (c[i1] != c[i2]).any():
                mtr = np.array([c[i1], c[i2]])
                if  np.linalg.matrix_rank(mtr)  >=  2  and  (c[i1][0]  /
c[i2][0]) != (c[i1][1] / c[i2][1]):
                    otv.append((i1, i2))
    return sorted(otv)

def check_path(points_list):
    otv = 0
    for i in range(len(points_list)-1):
        x = points_list[i+1]
        x0 = points_list[i]
        s = ((x[0]-x0[0])**2 + (x[1]-x0[1])**2)**(1/2)
        otv += s
    return round(otv, 2)
```