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: Управляющие конструкции языка Python

Студент гр. 3341 _____ Шуменков А.П.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2023

Цель работы

Целью работы является освоение управляющих конструкций на языке Python, а также модуля NumPy на примере программы, в которой они применяются.

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

Оформите задачу как отдельную функцию: *def check_crossroad(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))*.

Основные теоретические положения

В лабораторной работе была применена библиотека NumPy, используемая для разнообразных математических вычислений.

Методы модуля NumPy:

1. *numpy.radians(array)*

   Данных метод позволяет перевести последовательность значений углов (в геометрическом смысле) *array* (типа *ndarray*) из градусов в радианы. Возвращает новый *ndarray* со значениями тех же углов в радианах.

2. *numpy.ones(shape))*

   Данный метод позволяет создать матрицу из единиц заданного размера. Размер задается с помощью кортежа *shape*, где через запятую передаются размеры матрицы.

3. *numpy.vstack(arr1,arr2,[arrN])*

   Данный метод позволяет дописать матрицы последовательного друг к другу.

4. *numpy.linalg.norm(vector)*

   Данный метод из пакета *linalg* модуля *numpy* позволяет вычислить норму (модуль, длину) вектора *vector*(в общем случае — матрицы), переданного на вход.

5. *numpy.linalg.matrix_rank(matrix)*

   Данный метод из пакета *linalg* позволяет посчитать ранг квадратной матрицы matrix.

6. *numpy.linalg.solve(A,v)*

   Данный метод из пакета *linalg* позволяет найти решение линейной системы уравнений, которая представлена матрицей коэффициентов *A* и вектором свободных членов *v*.

Выполнение работы

Подключается модуль NumPy: import numpy as np.

Далее каждая из 3 подзадач оформлена в виде отдельной функции.

Задача 1. Функция def check_crossroad(robot, point1, point2, point3, point4). Даны координаты углов перекрёстка. Для решения задачи использовался следующий алгоритм: с помощью if мы проверяем случай, когда координата робота находится между координатами оси 0Y точек 1 и 3 и оси 0X точек 1 и 2. В этом случае на вывод подаётся значение True, иначе False.

Задача 2. Функция def check_collision(coefficients) 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).

Создаётся пустой список collisions, в него будут записываться пары столкнувшихся ботов.

Запускается цикл с ещё одним вложенным:

for i in range(coefficients.shape[0]):

for k in range(coefficients.shape[0]): …

Внутри цикла создаются переменные vI и vK, в которые записываются коэффициенты уравнений. Следом создаются массивы типа ndarray: A(для записи первых двух коэффициентов каждого из уравнений системы) и B(для записи последних коэффициентов каждого из уравнений системы). Далее, в функции try используется функция np.linalg.solve(A,B) для решения системы уравнений. В случае нахождения в список collisions записывается кортеж из значений i и k, являющихся номерами ботов. Функция try вызывается потому, что, если у системы уравнений нет решений (боты не столкнулись) появляется ошибка numpy.linalg.LinAlgError: Singular matrix. В except стоит оператор pass, так как системы уравнений, не имеющие решения не нужны. Функция возвращает список кортежей с номерами ботов collisions.

Задача 3. Функция check_path принимает на вход список points_list двумерных точек, а возвращает вещественное число – длину пройденного дакиботом пути. Инициализируется переменная p_len=0 – в ней будет храниться длина пройденного пути. Далее создаётся переменная pnum, которой присваивается значение len(points_list) – количество переданных функции точек. Также переменная plist преобразовывается к ndarray. Далее в цикле for x in range(1, pnum) в переменную napr записывается разность x-го и (x-1)-го элемента points_list (napr=plist[x]-plist[x-1]), что задаёт координаты вектора из точки plist[x-1] в plist[x], после чего к значению переменной p_len прибавляется длина этого вектора: path_len+=np.linalg.norm(napr). Функция возвращает значение переменной path_len, округленное до 2 знаков после запятой с помощью функции round(p_len, 2).

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(17, 13) (12, 7) (24, 7) (24, 17) (12, 17)	True	Задача 1
2.	(18, 18) (6, 4) (20, 4) (20, 16) (6, 16)	False	Задача 1
3.	[[-5 3 7] [-8 2 4] [4 -9 1] [-8 -3 3] [-1 7 0]]	[(0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 2), (1, 3), (1, 4), (2, 0), (2, 1), (2, 3), (2, 4), (3, 0), (3, 1), (3, 2), (3, 4), (4, 0), (4, 1), (4, 2), (4, 3)]	Задача 2
4.	[(1.0, 2.0), (2.0, 3.0)]	1.41	Задача 3
5.	[(2.0, 3.0), (4.0, 5.0)]	2.83	Задача 3

Выводы

В результате работы были освоены основные управляющие конструкции языка Python, а так же получены практические навыки использования модуля NumPy.

Были разработаны 3 функции, каждая из которых решает свою поставленную задачу. В функциях применялись пакеты модуля NumPy, что значительно облегчило решение поставленных задач.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

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

def check_collision(coefficients):
    collisions=[]
    for i in range (coefficients.shape[0]):
        for k in range (coefficients.shape[0]):
            vI=coefficients[i]
            vK=coefficients[k]
            B=np.array([vI[2], vK[2]])
            A=np.array([[vI[0], vI[1]], [vK[0], vK[1]]])
            try:
                np.linalg.solve(A, B)
                collisions.append((i,k))
            except:
                pass
    return collisions

def check_path(points_list):
    p_len=0
    pnum=len(points_list)
    plist=np.array(points_list)
    for x in range(1, pnum):
        napr=plist[x]-plist[x-1]
        p_len+=np.linalg.norm(napr)
    p_len = round(p_len, 2)
    return(p_len)
    pass
```