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 Мильхерт А.С.

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)* проверяет, пересекает ли робот определенный перекресток. Она принимает пять аргументов: robot (позиция робота) и четыре точки point1, point2, point3, и point4, которые представляют углы перекрестка. Если робот находится внутри этого перекрестка, функция возвращает True, иначе возвращает False. Для решения задачи достаточно сравнить координаты робота и *point1* и *point3*: *if (robot[0] >= point1[0] and robot[0] >= point4[0] and robot[0] <= point2[0] and robot[0] <= point3[0]) and (robot[1] >= point1[1] and robot[1] >= point2[1] and robot[1] <= point3[1] and robot[1] <= point4[1]).* Если условие выполняется, то функция возвращает *True*, а если нет – то *False*.

Задача 2. Функция *def check_collision(coefficients)* принимает на вход матрицу (*ndarray*) *coefficients* коэффициентов уравнений траекторий ботов, а возвращает список пар номеров столкнувшихся ботов *collisions*. Создаётся пустой список *answer*, а также двумерный список *matrix = [[ell for ell in el]for el in coefficients]*, в котором находятся коэффициенты уравнений траекторий, заполненные из *coefficients* при помощи генераторов списков. Далее в двух вложенных циклах for, пробегающих с помощью переменных-итераторов i и j элементы матрицы *coefficients* (индексы элементов получаются с помощью *coefficients.shape[0]*), вычисляется ранг матрицы, состоящей из элементов матрицы *coefficients* с индексами *i* и *j* (дополнительно проверяется, что i не равен j), и если этот ранг равен 2 (необходимое и достаточное условие существования решений), то в *answer* добавляется пара *(x, y).*

Задача 3. Функция *check_path* вычисляет длину пути, пройденного по списку точек. Она принимает список points_list, где каждая точка представлена двумя координатами [x, y]. Функция вычисляет сумму длин отрезков между последовательными точками и возвращает округленное значение этой длины до двух знаков после запятой. Инициализируется переменная *answer=0* – в ней

будет храниться длина пройденного пути. Далее в цикле *for i in range(1, len(points_list)* в переменную *vector* записывается разность *i*-го и (*i-1*)-го элемента *points_list* (vector = [points_list[i][0] - points_list[i-1][0], points_list[i][1] - points_list[i-1][1]]), что задаёт координаты вектора из точки *points_list[x-1]* в *points_list[x]*, после чего к значению переменной *answer* прибавляется длина этого вектора: *answer += math.sqrt((vector[0]**2) + (vector[1]**2)).* Функция возвращает значение переменной *answer,* округленное до 2 знаков после запятой с помощью функции *round(answer, 2).*

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	Задача 1
2.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	Задача 1
3.	[[-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)]	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 также получены практические навыки использования модуля NumPy.

Были разработаны 3 функции, каждая из этих функций выполняет разные проверки и вычисления в контексте роботов и траекторий движения. 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
import math


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


def check_collision(coefficients):
    matrix = [[ell for ell in el]for el in coefficients]
    answer = []
    length = coefficients.shape[0]
    for i in range(length):
        coefficient_a_b_1 = np.array(matrix[i][:2])
        for j in range(length):
            if i != j:
                coefficient_a_b_2 = np.array(matrix[j][:2])
                if      np.linalg.matrix_rank([coefficient_a_b_1,
coefficient_a_b_2]) >= 2:
                    answer.append((i, j))
    return answer


def check_path(points_list):
    answer = 0
    for i in range(1, len(points_list)):
        vector   =   [points_list[i][0]   -   points_list[i-1][0],
points_list[i][1] - points_list[i-1][1]]
        answer += math.sqrt((vector[0]**2) + (vector[1]**2))
    return round(answer, 2)
```