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. 2381 _____ Ильясов М.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Ознакомится с основными управляющими конструкциями языка Python, изучить библиотеку numpy и использовать полученные знания на примере практического задания

Задание.

Вариант 1

Задача 1. Оформите решение в виде отдельной функции *check_collision*. На вход функции подаются два ndarray – коэффициенты *bot1*, *bot2* уравнений прямых *bot1 = (a1, b1, c1)*, *bot2 = (a2, b2, c2)* (уравнение прямой имеет вид *ax+by+c=0*).

Функция должна возвращать точку пересечения траекторий (кортеж из 2 значений), предварительно округлив координаты до 2 знаков после запятой с помощью *round(value, 2)*.

Пример входных данных:

array([-3, -6, 9]), array([8, -7, 0])

Пример возвращаемого результата:

(0.91, 1.04)

Задача 2. Оформите задачу как отдельную функцию *check_surface*, на вход которой передаются координаты 3 точек (3 *ndarray* 1 на 3): *point1*, *point2*, *point3*. Функция должна возвращать коэффициенты *a*, *b*, *c* в виде ndarray для уравнения плоскости вида *ax+by+c=z*. Перед возвращением результата выполнение округление каждого коэффициента до 2 знаков после запятой с помощью *round(value, 2)*.

Например, даны точки: *A(1, -6, 1)*; *B(0, -3, 2)*; *C(-3, 0, -1)*. Подставим их в уравнение плоскости:

Составим матрицу коэффициентов:

$$a \cdot 1 + b(-6) + c = 1$$
$$a \cdot 0 + b(-3) + c = 2$$
$$a(-3) + b \cdot 0 + c = -1$$

Вектор свободных членов:

$$\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

Для такой системы уравнение плоскости имеет вид: *z = 2x + 1y + 5*

Пример входных данных:

array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])

Возвращаемый результат:

[2. 1. 5.]

Задача 3. Оформите решение в виде отдельной функции *check_rotation*. На вход функции подаются *ndarray* 3-х координат дакибота и угол поворота. Функция возвращает повернутые ndarray координаты, каждая из которых округлена до 2 знаков после запятой с помощью *round(value, 2)*..

Пример входных аргументов:

array([1, -2, 3]), 1.57

Пример возвращаемого результата:

[2. 1. 3.]

Выполнение работы.

Для решения представленных задач были написаны две вспомогательные функции: *round_matrix(array)* и *solve(*arrays)*. Первая функция выполняет округление элементов numpy-массива array с использованием встроенных функций *map* и *round* через промежуточный перевод в список:

```
result = numpy.array(list(map(lambda x:round(x,2), array)))
```

Результат сохраняется в переменной result (тип: *ndarray*) и возвращается оператором return. Вторая функция – *solve(*arrays)* – принимает на вход в качестве аргументов массивы, из которых составляется расширенная матрица matrix, содержащая все коэффициенты системы линейных уравнений. Далее

посредством взятия срезов матрица разбивается на две — *coefficients* (матрица всех коэффициентов, кроме свободных членов уравнений) и *free_terms* (матрица свободных членов), и те в свою очередь подаются на вход функции *numpy.linalg.solve*, производящую решение системы с сохранением в *result* и возвратом через return. Так как система уравнений не всегда имеет решения, используется конструкция *try-except* для обработки соотвествующего исключения *numpy.linalg.LinAlgError*. При этом в случае неудачи *result* инициализируется значением *None*.

Функция *check_collision(bot1, bot2)* принимает два аргумента типа ndarray, содержащих, согласно тексту задания, коэффициенты уравнений траекторий движения двух роботов. Чтобы получить искомую точку потенциального столкновения, требуется найти решение системы, состоящей из введённых уравнений прямых, а для этого можно воспользоваться ранее написанной функцией *solve*, что и реализованно в программе. После вызова *solve(bot1, bot2)* и присваивания соотвествующего значения переменной *result* идёт обработка значений, отличных от *None*: матрица *result* умножается на -1 и подаётся на вход встроенной функции *tuple* для получения кортежа. Указанное умножение имеет место быть вследствие нестандартной записи свободных коэффициентов в уравнении траектории движения (их знак противоположен ожидаемому функцией *solve*, так как вместо *a*x+b*y=c*, уравнения имеют вид *a*x+b*y+c=0*).

Функция *check_surface(point1, point2, point3)* получает три точки в виде одномерных numpy-массивов. Для вычисления коэффициентов уравнения плоскости составляется система уравнений из входных троек координат, с учетом чётвертого необходимого для задания плоскости значения c, которое по иллюстрациям из задания, имеет числовой коэффициент равный единице в каждом уравнении системы. Для добавления соотвествующего третьего столбца единиц, используется функция *numpy.insert* для всех строк

двумерного массива, после чего вызывается функция *solve*, результат которой и служит ответом на указанную подзадачу.

Функция *check_rotation* принимает на вход координаты position (ndarray) и угол поворота *angle*. Так как поворот происходит вокруг оси *z*, происходят изменения только *x* и *y* координат, которые путём среза отделяются в переменную *position_x_y*. Далее через угол строится матрица поворота, которая домножается на *position_x_y* с последующим округлением вспомогательной функцией *round_matrix()* в переменную *new_position_x_y*. К результату добавляется ранее срезанная координата *z* через *numpy.hstack()*. Полученная таким образом новая позиция (*new_position*) возвращается в качестве выходного значения.

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
1.	array([-3, -6, 9]), array([8, -7, 0])	(0.91, 1.04)	Фунция check_collision возвращает правильное значение, в соответсвии с примером
2.	array([1, 1, 9]), array([2, 2, 0])	None	Нерешаемая система не приводит первую функцию к ошибке
3.	array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])	[2. 1. 5.]	check_surface() правильно определяет уравнение плоскости
4.	array([0, 1, 0]), array([0, 0, 2]), array([0, 3, 0])	None	Вторая также функция успешно обрабатывает нерешаемые системы
5.	array([1, -2, 3]), 1.57	[2. 1. 3.]	Функция поворота работает корректно

Выводы.

Мной были изучены основные конструкции языка Python, такие как циклы, ветвления, функции, анонимные лямбда-функции и оператор *try-except*. Также были исследованы и задействованы ключевые инструменты библиотеки *numpy* для работы с матрицами. Была написанна программа, содержащая три функции, предназначенные для использовании в разработке дакиботов: *check_collision* — находящая точку потенциального столкновения двух ботов, *check_surface* – определяющая уравнение плоскости по трём координатам и *check_rotation*, возвращающая координаты после поворота. Решения этих трёх подзадач задействуют методы библиотеки numpy и вспомогательные функции, позволяя тем самым отработать полученные знания на практике.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main_lb1.py

```
import numpy

# вспомогательная функция для округления элементов numpy-массива
def round_matrix(array):
    result = numpy.array(list(map(lambda x: round(x, 2), array)))
    return result

# вспомогательная функция для решения системы линейных уравнений
def solve(*arrays):
    matrix = numpy.array(arrays)
    coefficients = matrix[:, :-1]
    free_terms = matrix[:, -1]
    try:
        solvetion = numpy.linalg.solve(coefficients, free_terms)
        result = round_matrix(solvetion)
    except numpy.linalg.LinAlgError:
        result = None
    return result

def check_collision(bot1, bot2):
    result = solve(bot1, bot2)
    if type(result) != type(None):
        result = tuple(-result)
    return result

def check_surface(point1, point2, point3):
    return solve(numpy.insert(point1, 2, 1),
                 numpy.insert(point2, 2, 1),
                 numpy.insert(point3, 2, 1))

def check_rotation(position, angle):
    position_x_y = position[:-1]
    rotation_matrix = numpy.array([[numpy.cos(angle),
                                    -numpy.sin(angle)],
                                   [numpy.sin(angle),
                                    numpy.cos(angle)]])
    new_position_x_y = numpy.matmul(rotation_matrix, position_x_y)
    new_position_x_y = round_matrix(new_position_x_y)
    new_position = numpy.hstack([new_position_x_y, position[-1]])
    return new_position
```