Multidimensional Arrays [CO1]

[Each method carries 5 marks]

Instructions for students:

- Complete the following methods on 2D Arrays
- You may use any language to complete the tasks.
- All your methods must be written in one single .java or .py or .pynb file. DO NOT CREATE separate files for each task.
- If you are using JAVA, you must include the main method as well which should test your other methods and print the outputs according to the tasks.
- If you are using PYTHON, then follow the coding templates shared in this folder.

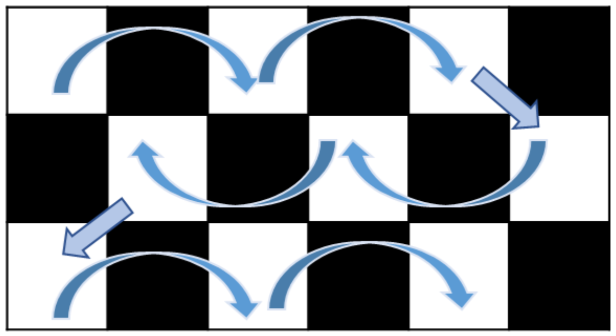
NOTE:

- **YOU CANNOT USE ANY BUILT-IN FUNCTION EXCEPT** len **IN PYTHON. [negative indexing, append is prohibited]**
- **YOU HAVE TO MENTION SIZE OF ARRAY WHILE INITIALIZATION**
- **YOUR CODE SHOULD WORK FOR ANY VALID INPUTS. [Make changes to the Sample Inputs and check whether your program works correctly]**

2D Array Tasks:

1. Zigzag Walk:

As a child, you often played this game while walking on a tiled floor. You walked avoiding a certain color, for example black tiles (it almost felt like if you stepped on a black tile, you would die!). Now you are in a room of m x n dimension. The room has m*n black and white tiles. You step on the white tiles only. Your movement is like this-



Now suppose you are given a 2D array which resembles the tiled floor. Each tile has a number associated with its color. B stands for Black whereas W stands for White. Can you write a method that will print your walking sequence on the floor? **Constraint: The first tile of the room is always white and the number of columns is always even.**

Sample Input:

```
-----------------------------------------
| 3W | 8B | 4W | 6B | 1W | 5B |
-----------------------------------------
| 3B | 2W | 1B | 6W | 3B | 8W |
-----------------------------------------
| 9W | 0B | 7W | 5B | 3W | 8B |
-----------------------------------------
| 2B | 1W | 3B | 6W | 0B | 4W |
-----------------------------------------
| 1W | 4B | 2W | 8B | 6W | 6B |
-----------------------------------------
```

Sample Output:

```
3W 4W 1W
8W 6W 2W
9W 7W 3W
4W 6W 1W
1W 2W 6W
```

2. **Landscape Screen:**

   You know what your smartphone screen actually is? For simplicity's sake, we can say that it is a grid structure (2D matrix) where every cell (i.e. every pixel) consists of a color. Most often we use auto-rotate in our smartphone. Auto rotation basically converts the screen from portrait to landscape. Behind the scenes, auto rotation changes the 2D matrix. The grid structure's rows become the columns and the columns become the rows.
   Suppose you are given a grid structure or 2D array where each cell represents a number. Write a method that performs auto rotate operation on the 2D matrix as below

Sample Input:	**Sample Output:**	**Explanation:**
----------------- | 7 | 11 | 8 | ----------------- | 6 | 9 | 14 | ----------------- | 0 | 4 | 7 | ----------------- | 2 | 0 | 8 | -----------------	--------------------- | 7 | 6 | 0 | 2 | --------------------- | 11 | 9 | 4 | 0 | ---------------------- | 8 | 14 | 7 | 8 | ----------------------	First row 7,11,8 becomes first column 7 11 8 Second row 6,9,14 becomes second column 7 6 11 9 8 14 And so on and so forth

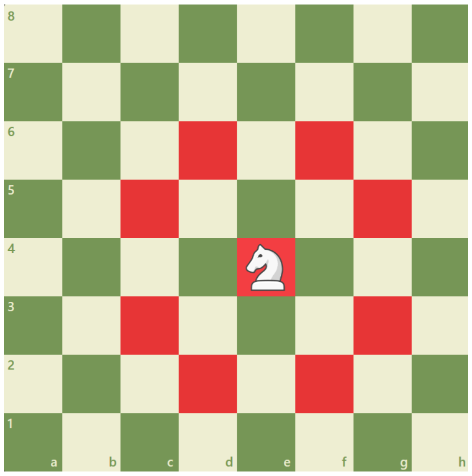
3. **Seating Arrangement:**

   Your university has an eccentric anime club. The seating arrangement of the clubroom has equal numbers of rows and columns. Each year, the club recruits enough members such that no seat remains vacant or no extra seats are required. Now a special rule based on the number of watched anime has been established for the seating arrangement. The rule is- the difference in the number of watched anime between an anime fan sitting in a chair in **i^{th} row and j^{th} column** and an anime fan sitting in a chair in **j^{th} row and i^{th} column** must be equal to a certain number. The club president will choose the certain number. Now you being the tech guy in the club, the club president tells you to write a program that will verify if the seating arrangement is correct provided the current seating arrangement and the specific number.

Sample Input:	Sample Output:	Explanation:
---------------------- | 8 | 15 | 7 | 12 | ---------------------- | 13 | 2 | 18 | 11 | ---------------------- | 9 | 20 | 5 | 2 | ---------------------- | 14 | 9 | 0 | 10 | ---------------------- 2	True	Member sitting in the chair of the 1st row and 2nd column has watched 15 anime. Member sitting in the chair of the 2nd row and 1st column has watched 13 anime. The difference is 15~13 = 2, the given number Member sitting in the chair of the 2nd row and 3rd column has watched 18 anime. Member sitting in the chair of the 3rd row and 2nd column has watched 20 anime. The difference is 18~20 = 2, the given number
| 7 | 13 | 9 | 14 | --------------------- | 12 | 8 | 15 | 11 | --------------------- | 10 | 17 | 3 | 2 | --------------------- | 15 | 10 | 1 | 4 | --------------------- 1	False	Member sitting in the chair of the 2nd row and 3rd column has watched 15 anime. Member sitting in the chair of the 3rd row and 2nd column has watched 17 anime. The difference is 15~17 ≠ 1, the given number

4. **Chess Piece:**

The chess board is 8x8 size. We will only deal with the knight and the rook piece here.
The knight piece moves like this-



From its position, it can move in 8 ways

a. 2 cells in upward direction and 1 cell in right direction.
b. 2 cells in upward direction and 1 cell in left direction.
c. 2 cells in downward direction and 1 cell in right direction.
d. 2 cells in downward direction and 1 cell in left direction.
e. 2 cells in the left direction and 1 cell in upward direction.
f. 2 cells in the left direction and 1 cell in downward direction.
g. 2 cells in the right direction and 1 cell in upward direction.
h. 2 cells in the right direction and 1 cell in downward direction.

Given the position of a knight in a tuple, your task is to write a method that calculates all the possible moves of the knight and return a 8x8 chessboard where empty cells are 0 and the probable knight moves are denoted as 3. The given knight cell has 33 in it.
Be very careful with CORNER CASES.

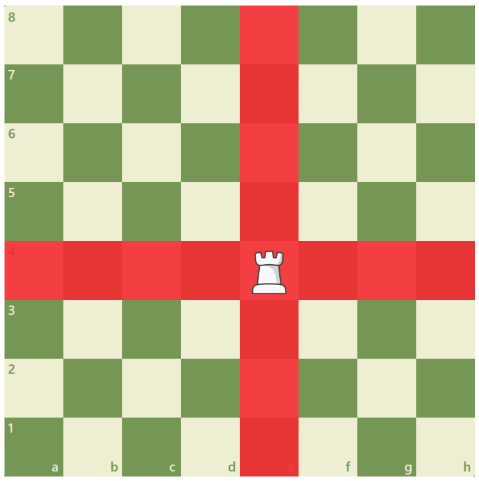
Sample Input:	**Sample Output:**
knight = (3,4)	see below

Sample Output:

```
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
-----------------------------------------
| 0 | 0 | 0 | 3 | 0 | 3 | 0 | 0 |
-----------------------------------------
| 0 | 0 | 3 | 0 | 0 | 0 | 3 | 0 |
-----------------------------------------
| 0 | 0 | 0 | 0 | 33 | 0 | 0 | 0 |
-----------------------------------------
| 0 | 0 | 3 | 0 | 0 | 0 | 3 | 0 |
-----------------------------------------
| 0 | 0 | 0 | 3 | 0 | 3 | 0 | 0 |
-----------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
-----------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
-----------------------------------------
```

Bonus Task:

This is an extended version of task4. Now you have to incorporate another chess piece ‘rook’ as well. A rook moves like this-



The rook moves horizontally or vertically, through any number of squares.

Given the position of a knight in a tuple and the position of a rook in another tuple, your task is to write a method that calculates if the knight can kill the rook or if the rook can kill the knight or if they cannot kill each other. In the 8x8 chessboard, the empty cells are 0 and the probable knight moves are denoted as 3. The given knight cell has 33 in it and the given rook cell has 5.

Sample Input
knight = (3,4)
rook = (4,1)

Sample Output:
```
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
------------------------------------------
| 0 | 0 | 0 | 3 | 0 | 3 | 0 | 0 |
------------------------------------------
| 0 | 0 | 3 | 0 | 0 | 0 | 3 | 0 |
------------------------------------------
| 0 | 0 | 0 | 0 | 33 | 0 | 0 | 0 |
------------------------------------------
| 0 | 5 | 3 | 0 | 0 | 0 | 3 | 0 |
------------------------------------------
| 0 | 0 | 0 | 3 | 0 | 3 | 0 | 0 |
------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
-----------------------------------------

Cannot Kill
```

knight = (5,6)
rook = (5,1)

```
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
-------------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
-------------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
-------------------------------------------------
```

```
| 0 | 0 | 0 | 0 | 0 | 3 | 0 | 3 |
-------------------------------------------------
| 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
-------------------------------------------------
| 0 | 5 | 0 | 0 | 0 | 0 | 33| 0 |
-------------------------------------------------
| 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
-------------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 3 | 0 | 3 |
-------------------------------------------------
Rook can kill
```