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:

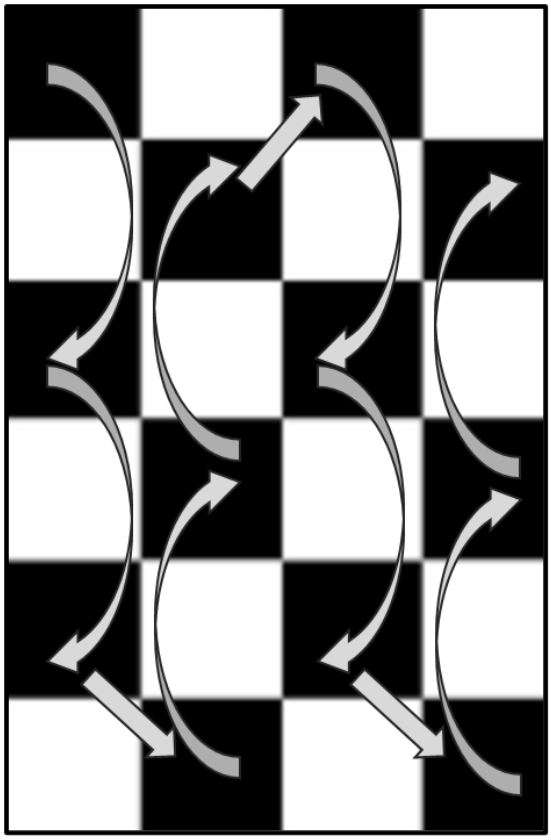
- **YOU CANNOT USE ANY BUILT-IN FUNCTION EXCEPT** len **IN PYTHON. [negative indexing, append is prohibited]**
- **You can use the attribute 'shape' of numpy arrays**
- **YOU HAVE TO MENTION SIZE OF ARRAY WHILE INITIALIZATION**
- **YOUR CODE SHOULD WORK FOR ANY VALID INPUTS. [Make changes to the Sample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

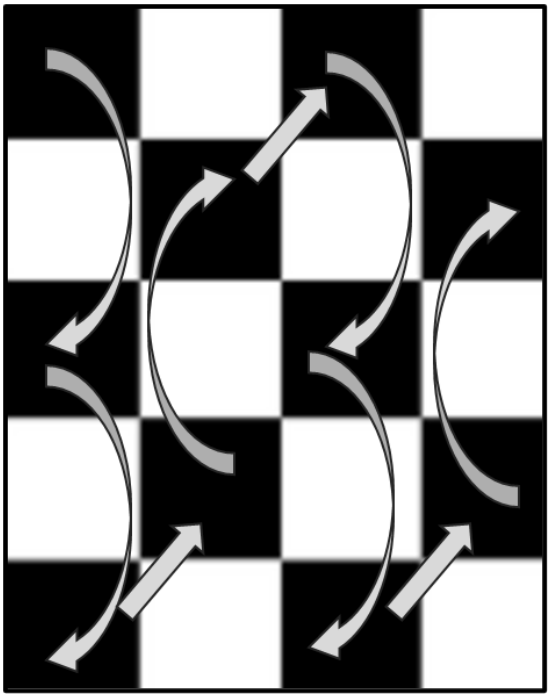
2D Array tasks:

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 white tiles (it almost felt like if you stepped on a white 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 black tiles only. Your movement is like this:

OR





Now suppose you are given a 2D array which resembles the tiled floor. Each tile has a number. Can you write a method that will print your walking sequence on the floor?

Constraint: The first tile of the room is always black.

Hint: Look out for the number of rows in the matrix [Notice the transition from column 0 to column 1 in the above figures]

Sample Input	Sample Output
---------- | 3 | 8 | 4 | 6 | 1 | ---------- | 7 | 2 | 1 | 9 | 3 | ---------- | 9 | 0 | 7 | 5 | 8 | ---------- | 2 | 1 | 3 | 4 | 0 | ---------- | 1 | 4 | 2 | 8 | 6 | ----------	3 9 1 1 2 4 7 2 4 9 1 8 6
---------- | 3 | 8 | 4 | 6 | 1 | ---------- | 7 | 2 | 1 | 9 | 3 | ---------- | 9 | 0 | 7 | 5 | 8 | ---------- | 2 | 1 | 3 | 4 | 0 | ----------	3 9 1 2 4 7 4 9 1 8

2. Wall Up Trost District:

You live in the Trost District. There was a wall (named Wall Rose) around your district for protection of the attacks from giants (named Titans). One day, the wall got destroyed and the district council decided to build a new wall around your district. The design of the district is rather intriguing. It is shaped like a 2D matrix. To create a wall around the district is equivalent to padding the array all around with a certain number. In the council, it was decided that the padding number would be 8 and the depth of the wall would be decided by the council head.

Now, given the matrix representation of your district and the depth of the wall, write a method that will create a new 2D array and print the resultant array [the district array surrounded by padding of 8].

Sample Input	Sample Output
district = ---------- | 2 | 3 | 4 | ---------- | 3 | 4 | 6 | ---------- | 2 | 1 | 4 | ---------- depth = 1	| 8 | 8 | 8 | 8 | 8 | ---------- | 8 | 2 | 3 | 4 | 8 | ---------- | 8 | 3 | 4 | 6 | 8 | ---------- | 8 | 2 | 1 | 4 | 8 | ---------- | 8 | 8 | 8 | 8 | 8 | ----------
district = ---------- | 2 | 3 | 4 | 1 | ---------- | 3 | 4 | 6 | 5 | ---------- | 2 | 1 | 4 | 7 | ---------- depth = 2	| 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | ---------- | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | ---------- | 8 | 8 | 2 | 3 | 4 | 1 | 8 | 8 | ---------- | 8 | 8 | 3 | 4 | 6 | 5 | 8 | 8 | ---------- | 8 | 8 | 2 | 1 | 4 | 7 | 8 | 8 | ---------- | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | ---------- | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | ----------

3. Crows vs Cats:

In your prefecture, there are two volleyball teams who are dubbed as Crow and Cat. They often engage in volleyball matches in their clubroom. The clubroom is square in shape and the volleyball net is along the primary diagonal. In the match, Crows stand in the upper right corner of the net and cats in the lower left corner. To exchange greetings in the beginning, they stand in a matrix position.
Now given a 2D matrix ‘clubroom’, where the primary diagonal is the net, upper triangle is the Crow, lower triangle is the Cat; write a method that calculates the difference in strength between the Crow player and its mirrored Cat Player and store them in a array named ‘strength_diff’.
Note: The length of ‘strength_diff’<=100. You can take the array size 100.

BONUS [1.5 marks] : Find out the size of the ‘strength_diff’ array during initialization **without** using any loop.

Sample Input	Sample Output	Explanation
clubroom = | 1 | 2 | 9 | 7 | ---------- | 4 | 5 | 1 | 8 | ---------- | 3 | 6 | 2 | 7 | ---------- | 2 | 8 | 6 | 3 | ----------	strength_diff = [-2, 6, 5, -5, 0, 1]	1, 5, 2, 3 are the elements along the prime diagonal. The (0,1) element is 2 and its mirror element (1,0) is 4. Therefore, the difference is 2 - 4 = -2. The (1,3) element is 8 and its mirror element (3,1) is 8. Therefore, the difference is 8 - 8 = 0

4. ATM's Triangle:

ATM's triangle is the right triangular array of numbers that begins with 1 on the top. In the ATM's right triangle, in the nth row, there are n numbers and the first and last element of the row is n itself. The other numbers (i,j) between these two numbers are the summation of the numbers above its row (i-1 row) up to the j column.

Given the number of rows in the ATM's right triangle, create a **matrix** that stores the right triangle and print it.

Sample Input	Sample Output	Explanation
5	1 2 2 3 4 3 4 7 10 4 5 11 21 25 5	The 1st row has 1 number, that is 1 The 2nd row has 2 numbers. The first and last elements are 2 itself. The 3rd row has 3 numbers. The first (2,0) and the last number (2,2) are 3. The middle number (2,1) is the summation of the numbers in its previous row upto column 1. That is 2+2 = 4. The 4th row has 4 numbers. The first (3,0) and the last number (3,3) are 4. The number (3,1) is the summation of the numbers in its previous row up to column 1. That is 3+4 = 7. The number (3,2) is the summation of the numbers in its previous row up to column 2. That is 3+4+3 = 10.

5. Trace the Bot:

Your part of the BRACU NANOBOTS team. Your Nanobot only works on flat surfaces. It always maps its surroundings into a grid. The grid is always a 7x7 grid with the Nanobot in the middle. Using a wireless remote you can pass various combinations of 12 commands.

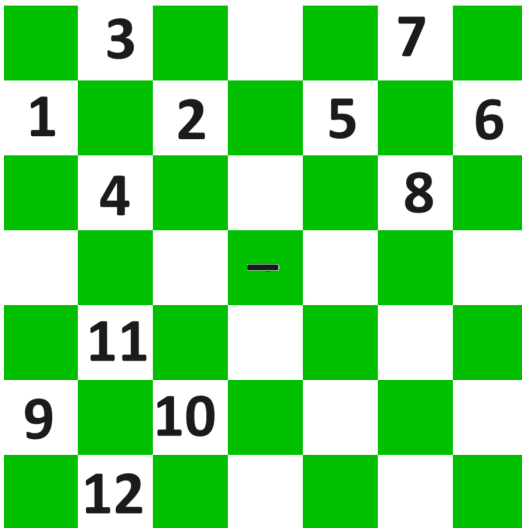
- 1 for moving 2 cells diagonally in the **upper left** and moving 1 cell **left**.
- 2 for moving 2 cells diagonally in the **upper left** and moving 1 cell **right**.
- 3 for moving 2 cells diagonally in the **upper left** and moving 1 cell **up**.
- 4 for moving 2 cells diagonally in the **upper left** and moving 1 cell **down**.
- 5 for moving 2 cells diagonally in the **upper right** and moving 1 cell **left**.
- 6 for moving 2 cells diagonally in the **upper right** and moving 1 cell **right**.
- 7 for moving 2 cells diagonally in the **upper right** and moving 1 cell **up**.
- 8 for moving 2 cells diagonally in the **upper right** and moving 1 cell **down**.
- 9 for moving 2 cells diagonally in the **lower left** and moving 1 cell **left**.
- 10 for moving 2 cells diagonally in the **lower left** and moving 1 cell **right**.
- 11 for moving 2 cells diagonally in the **lower left** and moving 1 cell **up**.
- 12 for moving 2 cells diagonally in the **lower left** and moving 1 cell **down**.

These sequences of commands will be executed using the bot's **moving_around()** function. The other members of BRACU NANOBOTS team have already created a print_matrix() function to draw the grid. Now your task is to implement the moving_around() function to visualize its movement in the grid.

Note:

- The initial position of the bot is always (3,3) and When the bot starts its icon is '-'.
- Whenever the bot moves to a particular cell, it leaves a '*'.
- When the bot stops its icon is changed to '/'.

HINT: LOOK OUT FOR CORNER CASES



Sample Input	Sample Output	Explanation
cmds = np.array([5,11,2,9])	---------- | . | / | . | . | . | . | . | ---------- | . | . | . | . | * | . | . | ---------- | . | . | * | . | . | . | . | ---------- | . | . | . | - | . | . | . | ---------- | . | . | . | . | . | . | . | ---------- | . | . | . | . | . | . | . | ---------- | . | . | . | . | . | . | . | ----------	The bot starts as (3,3). The first cmd is 5. It moves 2 cells diagonally in the **upper right (1,5)**, then 1 cell **left** and lands at (1,4) The second cmd is 11. It moves 2 cells diagonally in the **lower left (3,2),** then moves 1 cell **up** and lands at (2,2) The third cmd is 2. It moves 2 cells diagonally in the **upper left (0,0),** then moves 1 cell **right** and lands at (0,1) The fourth cmd is 9. Since it moves outside the board, the command is ignored and the final position is (0,1) where a / is set.