SCALER

2D Arrays

TABLE OF CONTENTS

A 2D matrix is a specific type of 2D array that has a rectangular grid of numbers, where each number is called an element.

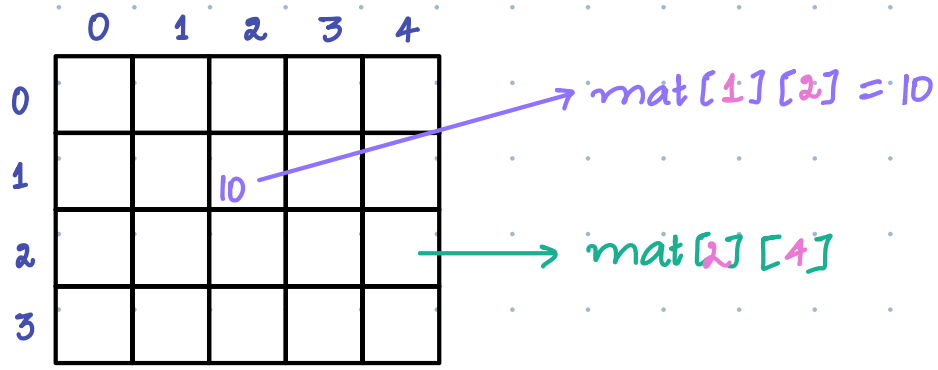
It is a mathematical structure that consists of a set of numbers arranged in rows and columns.

Declaration :

```
int[][] mat = new int[N][M]
```

- `int[][]` — Datatype
- `mat` — Name
- `N` — No of Rows (Horizontal)
- `M` — No of Cols. (Verticals)

mat[4][5] =



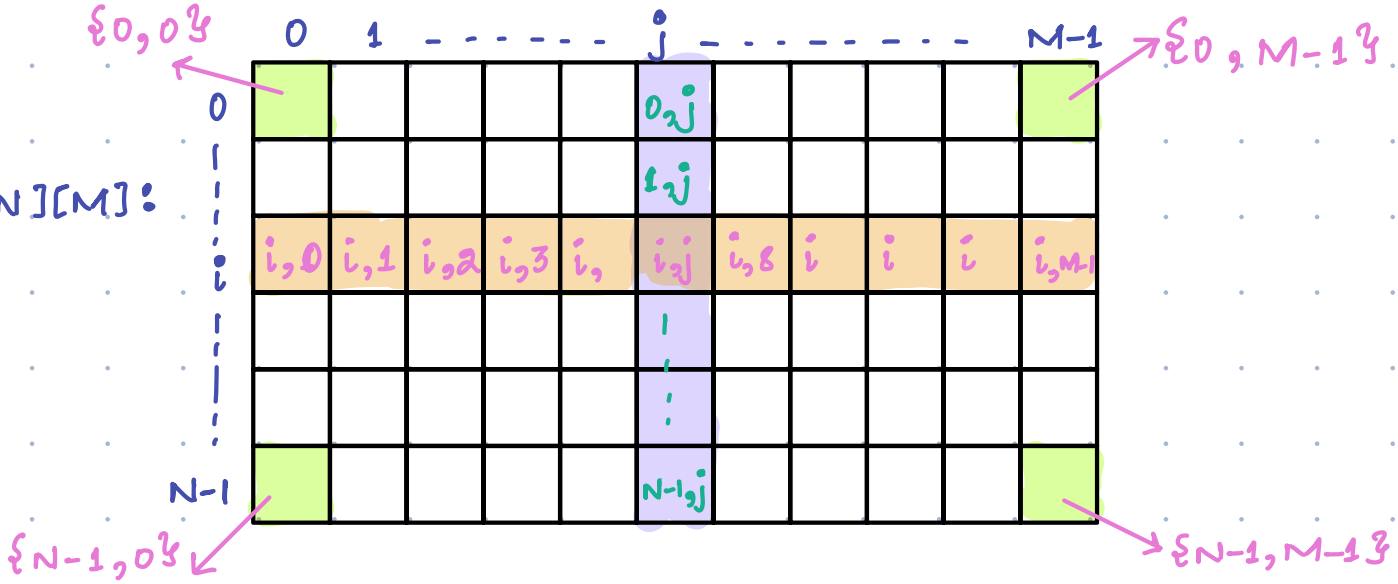
To access a cell in matrix : `mat[row][col]`

No of elements in mat[N][M] : N * M

No of Rows in mat[][] : `mat.length`

No of Cols in mat[][] : `mat[0].length`

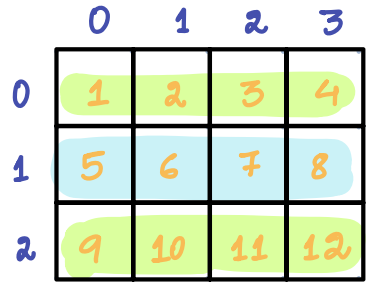
QUIZ!



Observations:

1. Iterate on row, col. changes [0 M-1]
2. Iterate on col, row changes [0 N-1]

< Question > : Given arr[N][M], print row-wise sum.



Output

10

26

42

i

0 : Iterate j : 0 to 3 cal sum & print it

1 : Iterate j : 0 to 3 cal sum & print it

2 : Iterate j : 0 to 3 cal sum & print i

3 : Stop

Idea : For every row, iterate on all M cols, calculate sum & print it.

Rows ↑ Cols ↑

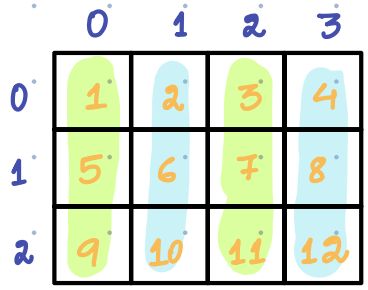
```
void Row_sum(int mat[][], int N, int M){
    for(r = 0; r < N; r++){
        int sum = 0;
        for(c=0, c< M; c++){
            sum = sum + mat[r][c];
        }
        print(sum);
    }
}
```

QUIZ!

T.C : O(N*M)

S.C : O(1)

< Question > : Given arr[N][M], print column-wise sum.



Output 15 18 21 24

c

0 : iterate r [0 : 2] cal sum & print it

1 : iterate r [0 : 2] cal sum & print it

2 : iterate r [0 : 2] cal sum & print it

3 : iterate r [0 : 2] cal sum & print it

4 : Stop

Idea : For every Col , Iterate on all N Rows , cal sum & print it.

```
                                  Rows      Cols
                                   ↑         ↑
void Col_sum(int mat[][], int N, int M){

    for (c=0 ; c<M ; c++){
        int sum = 0 ;
        for (r=0 ; r<N ; r++) {
            sum = sum + mat[r][c] ;
        }
        print(sum) ;
    }
}
```

T.C : O(N*M)

S.C : O(1)

No of Rows = = No of Cols

< Question > : Given square matrix mat[N][N], print diagonals

O/p

	0	1	2	3
0	1	5	8	9
1	3	6	7	1
2	6	5	2	4
3	3	8	9	5

Principal Diagonal

1 → mat[0][0]
6 → mat[1][1]
2 → mat[2][2]
5 → mat[3][3]

```
void PrincipalDiagonal(mat[][],N){
    for(i=0; i<N; i++){
        print(mat[i][i]);
    }
}
```

T.C : O(N) S.C : O(1)

O/p

	0	1	2	3
0	1	5	8	9
1	3	6	7	1
2	6	5	2	4
3	3	8	9	5

Anti Diagonal

9 → mat[0][3]
7 → mat[1][2]
5 → mat[2][1]
3 → mat[3][0]
Stop 4 -1

N=4

[0][4-0-1]
[1][4-1-1]
[2][4-2-1]
[3][4-3-1]

mat[i][N-i-1]

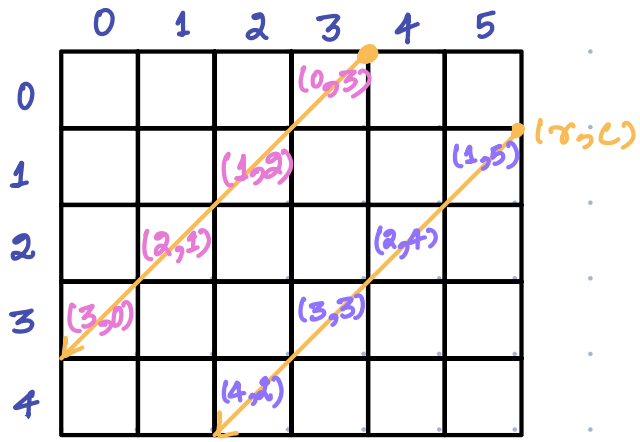
```
void AntiDiagonal(mat[][],N){
    int r=0, c=N-1;
    while(r<N && c>=0){
        print(mat[r][c]);
        r++;
        c--;
    }
}
```

T.C : O(N)
S.C : O(1)

Starting from mat[0][N-1], print elements.
Inc Row by 1 & Dec Col by 1. till row & col indices are valid.

```
void AntiDiagonal (mat[][], int N) {
    for(i=0 ; i<N ; i++) {
        print (mat[i][N-i-1]);
    }
}
```

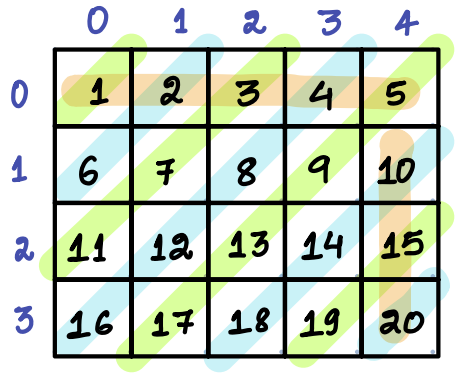
< Question > : Given mat[N][M] and a point S {r, c}. Print a diagonal from right to left and top to bottom which starts from S.



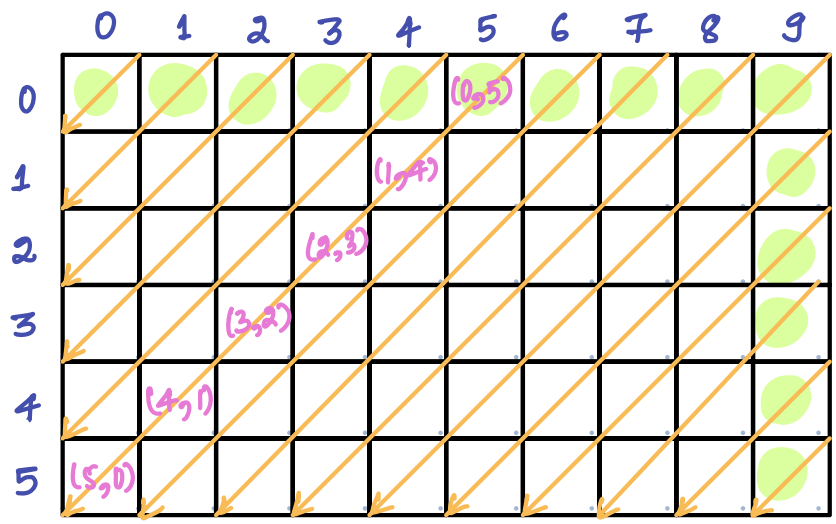

```
void StartDiagonal ( int mat[N][M], int r, int c) {
    while(r<N && c>=0) {
        print (mat[r][c]);
        r++;
        c--;
    }
}
```

T.C : O(N)
S.C : O(1)

< *Question* > : Given mat[N][M]. Print all the diagonals from right to left and top to bottom.



Output	Starting Pt
1	[0 0]
2 6	[0 1]
3 7 11	[0 2]
4 8 12 16	[0 3]
5 9 13 17	[0 4]
10 14 18	[1 4]
15 19	[2 4]
20	[3 4]



Total No of Diagonals: M+N-1

Observations:

- Right → Left diagonal start at 0^{th} Row & $(M-1)^{th}$ col.
- Row is INC by 1.
  Col is DEC by 1.

QUIZ!

< / > Code

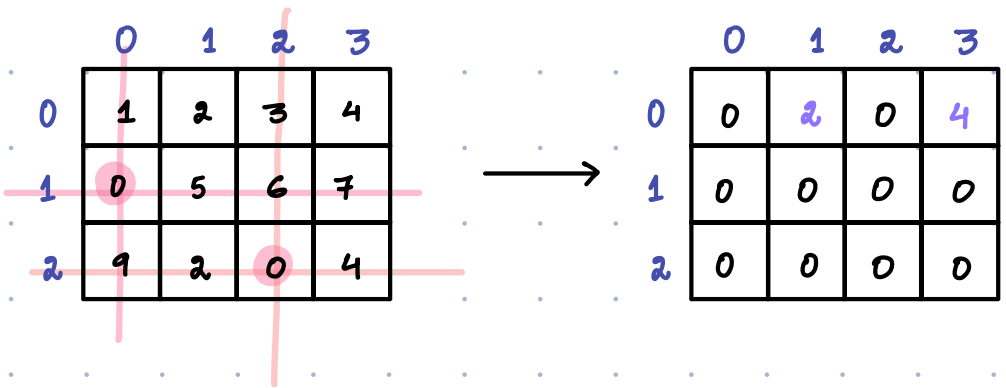
```
void PrintDiagonals( int mat[][], int N, int M){

    // Print Diagonals starting from 0th Row.
    for( c=0; c<M; c++) {
        StartDiagonal( mat, 0, c);
    }

    // Print Diagonals starting from last col.
    for( r=1; r<N; r++) {
        StartDiagonal( mat, r, M-1);
    }

}
```

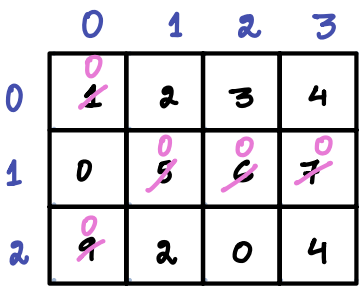
T.C : O(N*M)

S.C : O(1)

< *Question* > : Given array [N] [M].

Make all elements in a row and column zero if arr[i][j] == 0. Specifically make entire ~~other~~ ith row and jth column zero. All elements are positive.





We cann't mark elements as "0" on the go since we will lose track of original "0".

```
void RowToColZero(int mat[][], int N, int M) {

    // Iterate on Row & Mark Col
    for (i=0; i<N; i++) {
        int flag = 0;
        for (j=0; j<M; j++) {
            if (mat[i][j] == 0) {
                flag = 1;
                break;
            }
        }
        if (flag == 1) {
            for(k=0; k<M; k++) {
                if( mat[i][k] != 0) {
                    mat[i][k] = -1;
                }
            }
        }
    }

    // Iterate on Cols & Mark Rows
    for (j=0; j<M; j++) {
        int flag = 0;
        for(i=0; i<N; i++) {
            if( mat[i][j] == 0) {
                flag = 1;
            }
        }
        if (flag == 1) {
            for(k=0; k<N; k++) {
                if( mat[k][j] != 0) {
                    mat[k][j] = -1;
                }
            }
        }
    }
```

```
// change the marked elements to 0
for(i=0; i<N; i++) {
    for(j=0; j<M; j++) {
        if (mat[i][j] == -1) {
            mat[i][j]=0;
        }
    }
}
```

T.C : O (3 N*M) = O(N*M)

S.C : O(1)

	0	1	2	3
0	~~-1~~ 1	2	~~-1~~ 3	4
1	0	~~-1~~ 5	~~-1~~ 6	~~-1~~ 7
2	~~-1~~ 9	~~-1~~ 2	0	~~-1~~ 1

flag = 1

	0	1	2	3
0	0	2	0	4
1	0	0	0	0
2	0	0	0	0