Arrays : 2D Matrices

6th April, 2022

AGENDA :

- 2-dimensional arrays (matrices)
- Basic operations on matrices
  - Accessing an element
  - Traversing through the matrix
- Some interesting problems.
- Applications of Matrices.

Element / Arrays / 2D-Arrays / 3D-Arrays

• → Point → 0 dimensions (Length, Breadth or Height → All are absent)

↓

Element (Integer, character).

✓ 5, 2, 3 (0 dimensional)

——— → Line → 1 dimension (Length)

↓

Array/List

↓

[5, 2, 3, 4, 6] (1 dimensional)

→ Square/Rectangle. → 2 dimension (Length, Breadth)

↘ 2d arrays/Matrix/2D list.

↘
```
[[5, 2, 3, 4, 6]
 [1, 2, 3, 4, 1]
 [2, 0, 1, 0, 1]]
```

Cube/Cuboid.

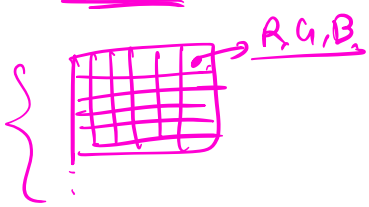
↘ collection of squares.

↘ 3D Matrix.

List of list of lists.

```
[ [[5,2,3], [2,1,2]],
  [[2,1,0], [3,2,0]] ]
```
← 3D array

↘ Image.



* If each element in an array is an array, it will give rise to a 2D array.

* How to create a 2D Matrix in Python ?

arr = [[1,2], [3,4], [6,8]] →

0 1 2

2D Array is a list of lists.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 6 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 6 & 8 \end{bmatrix}$$

$$\begin{matrix} 1 & 3 & 6 \\ 2 & 4 & 8 \end{matrix}$$ X

* How to access the element in a 2D Matrix ?

arr[0][0]

→ arr[0] → [1,2]

arr[0][0] = 1

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 5 & 3 & 2 \\ 1 & 2 & 0 \\ 2 & 6 & 8 \\ 4 & 3 & 2 \end{bmatrix} \end{array}$$

M x N.

M: no. of rows.
N: no. of columns.

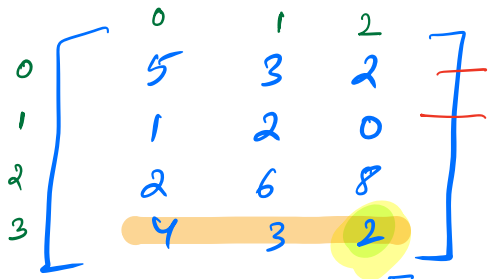
↳ M = 4
N = 3

arr[3][0]

[4,3,2] → 4

arr = [[5,3,2], [1,2,0], [2,6,8], [4,3,2]]

0 1 2 3



arr[3][2]

= arr[M-1][N-1]

* → arr = [[...], [...], [...], [...], ...]

Contains same no. of elements.

M = no. of rows = len(arr)

N = no. of column s = len(arr[0])

[2 3 4 ...
2 3 4 ...
⋮]

[[1 1 1] ,
[ ] ,
[ ] ,
[ ]]

1 1 1

[[1, 3, 5],
[2, 3, 4],
[3, 7, 9]]

No. of rows = len(arr)

No. of cols = len(arr[0])

= len(arr[1])

= len(arr[2])

* How do we access the Rth row of matrix?

arr[R-1] 1-indexed.

* How do we acces the Rth column of matrix?
↳ 1-indexed.

a[r-1] → Rth row. ✓
Not Rth column.

arr[:][r] → X

arr[r-1][r] → (r-1, r)th element.

arr[:][R] → X

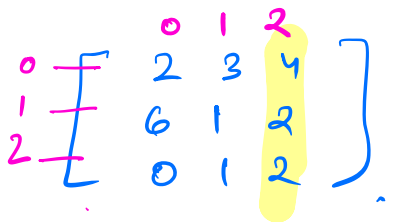
↳ Can be done in Numpy!

Cannot be done without iteration!

* Given a matrix M (mxn), column no. C, print the Cth column.
↳ 0-indexed.

```
for i in range(len(arr)):
    print(arr[i][c])
```

c=2



arr[0][2]
arr[1][2]
arr[2][2].

* #arr.

n = len(arr).

TC ?

O(N) X

TC: O(1) only.

len(arr)

↓

O(1) time.

** Given a 2D Matrix M, return an array containing the row wise sum.

Inp =

	0	1	2	
0	5	3	2	→ = 10
1	1	2	0	→ = 3
2	2	6	8	→ = 16
3	4	3	2	→ = 9

Output = → [10 3 16 9]

TC: O(M*N)

```
row_sums = []
# m,n is known.
for row in range(m):
    sum = 0
    for col in range(n):
        sum += arr[row][col]
    row_sums.append(sum)

return row_sums
```

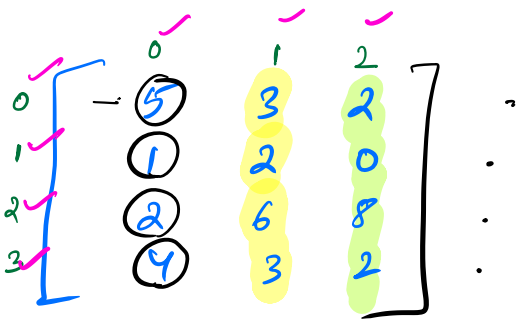
row_sums.append (sum(row))

xx.

```
return [sum(row) for row in arr]
```

List comprehension

** Given a matrix M, return column-wise sum.



Inp =

(0,0)
(1,0)

Output = [12 14 12]

```
col_sums = []
# m,n is known.
for col in range(n):
    sum = 0
    for row in range(m):
        sum += arr[row][col]
    col_sums.append(sum)

return row_sums
```

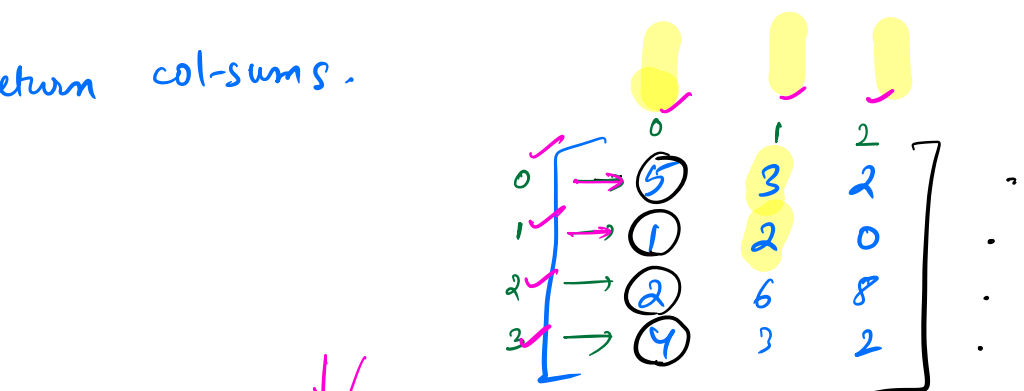
Challenge:

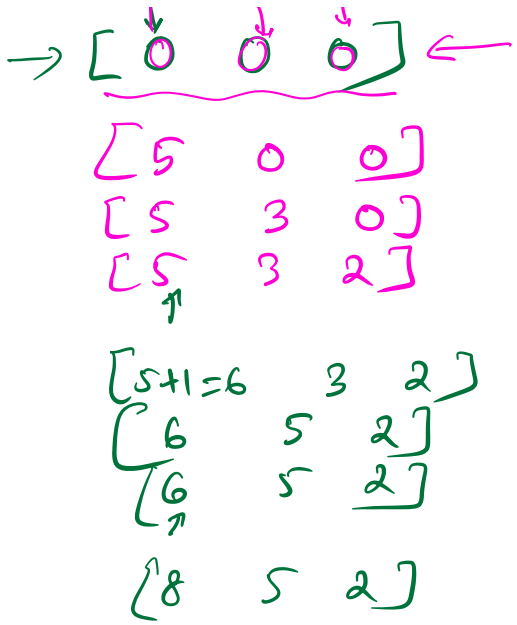
* Do the same operation as above, but, first iterate over rows, and then iterate over columns.

```
# m,n is known.
for row in range(m):

    for col in range(n):
```

```
return col_sums.
```





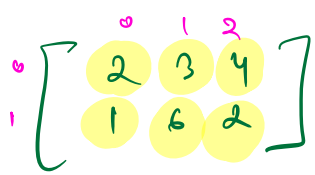
$(m*n)$

TC: $O(m*n)$

Pseudo-code.

```
col_sums = [0] * n
# m, n is known.
for row in range(m):
    for col in range(n):
        col_sums[col] += arr[row][col]
```

$O(m*n)$



Break till 10:20.

result = [0 0 0]
[2 0 0]
[2 3 0]
[2 3 4]
[3 3 4]
[3 9 4]

Result = [3 9 6].

* Given two matrices M1 and M2, return a matrix which is the sum of two matrices.

↓ M1 and M2 has same dimensions.

m, n is known.

res = [] ✗

res = [[]] ✗

res = [[0]*n]*m ✗

res = [[0]*n for _ in range(m)]

[[0]*n]*m = [0000] [0000] [0000] [0000]

res = []
for _ in range(m):
    res.append([0]*n)

for i in range(m):
    for j in range(n):
        res[i][j] = M1[i][j] + M2[i][j]

n →
m ↓
[0000]
[0000]
[0000]

m*n matrix filled with zeroes.

[(m,n)]*N.

Tuples are immutable.

arr = [ ]
A = arr.

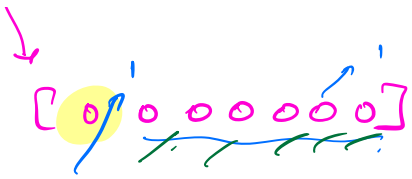
Copying by reference

[.]*n.
[↓:]
[:]
[ ]

→ All lists will point to the same memory location.

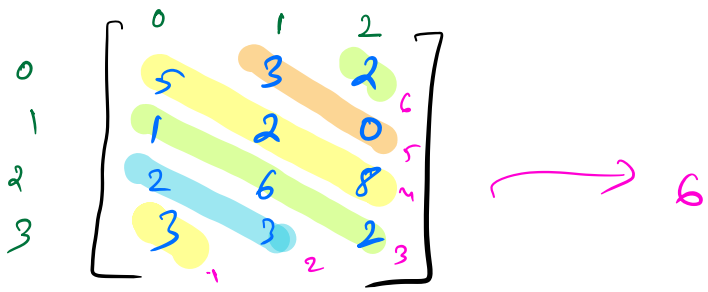
[0]*n.

(

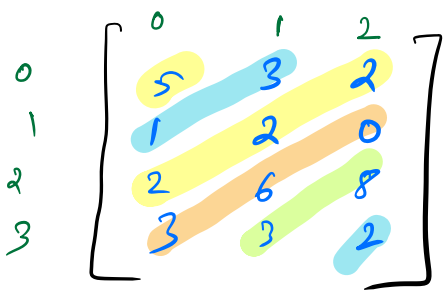


Integers are immutable.
Lists are mutable.

* Given a matrix mxn, print the diagonals of the matrix.



$\times$ { i==j → Valid for main diagonal in a square matrix.

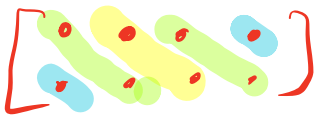


Anti-diagonals.

† 6 diagonals. → left to right
6 Anti-diagonals. → right to left

Count no. of diagonals in a Matrix. (MXN).

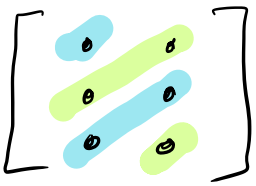
2x4



→ 5 diagonals.

2+4-1=5

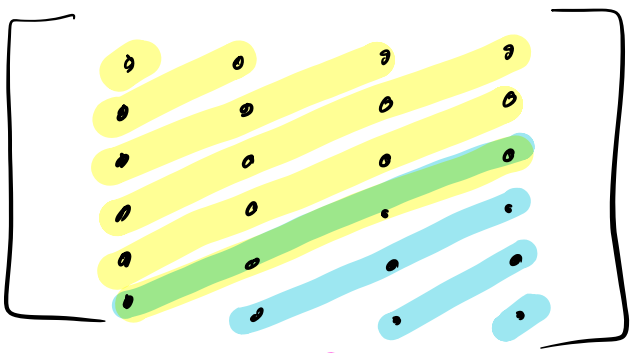
3x2



→ 4 anti-diagonals.

3+2-1=4.

M x N.

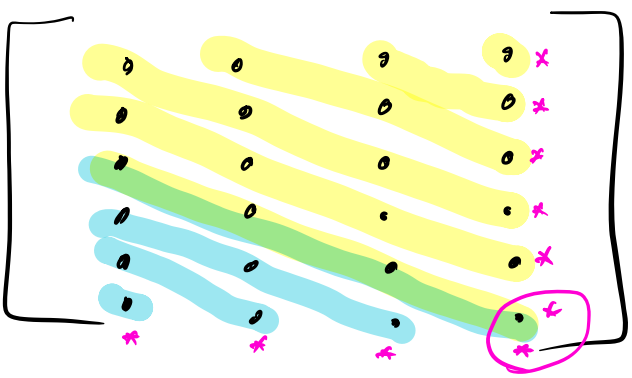


Anti

m

n

m+n-1

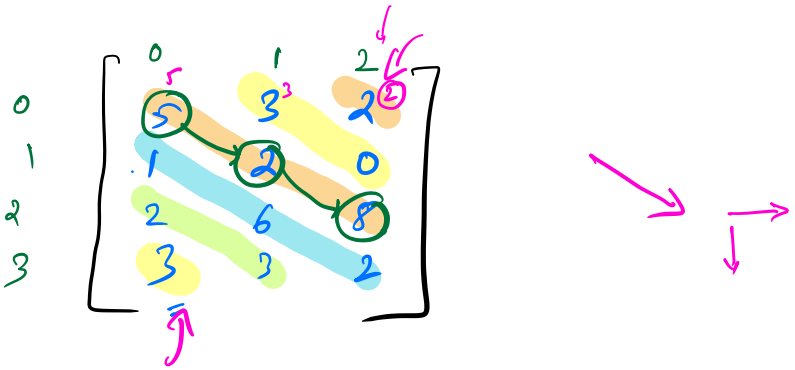


Diagonal.

m+n-1

Q. Print the diagonals of a matrix.

Inp =



Output =

```
3
2 3
1 6 2
5 2 8
3 0
2
```

(i, j)

1: ✓ (3,0) ✓

2: ✓ (2,0) , (3,1)

3: ✓ (1,0) , (2,1) , (3,2)

4: ✓ (0,0) , (1,1) , (2,2)

5: ✓ (0,1) , (1,2)

6: ✓ (0,2)

Observe some pattern!

Observations.

1. In a diagonal, if current cell is (i,j), next cell will be $(i+1, j+1)$

2. Diagonal continues, until we hit the boundary.
   ↳ $i < m$ && $j < n$.
   ** { $i == m$ or $j == n$, } ← Crossed the boundary.

3. Start index of 1st diagonal = $(m-1, 0)$

   $(m-2, 0)$

   $(m-3, 0)$

   $(m-4, 0)$

   ⋮

   $(0, 0)$

   $(0, 1)$

   $(0, 2)$

   $(0, 3)$

   ⋮

   $(0, n-1)$

4. To change diagonal → first move up (decrement row), then move right. (increment col)

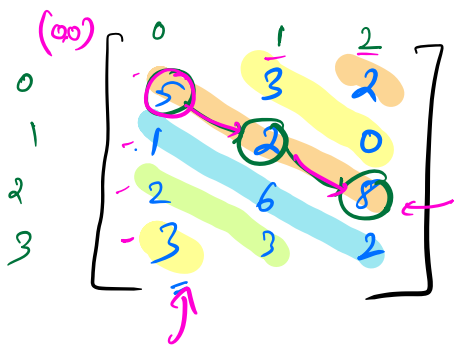
   $(m-1, 0)$
   ↓
   $(m-2, 0)$
   ↓
   $(m-3, 0)$
   ⋮
   $(0, 0) \rightarrow (0, 1) \rightarrow (0, 2) \rightarrow (0, 3) \ldots (0, n-1)$

Pseudo-code.

```
r = m-1
c = 0
while (c < n):          # c = 0
    # Use i, j to traverse through 1 diagonal.
    diag = []
    i = r               // r and c are the starting indices
    j = c               // of this diagonal.
    while (i < m && j < n):
        diag.append(arr[i][j])
        i += 1
        j += 1
    print(diag)
    if r > 0:
        r -= 1
    else:
        c += 1
```

m+n-1

TC: $O(m * n)$



SC:

↳ Max space that diag = [] takes up.

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

2 x 8

Max no. of elements present in 1 single diagonal.
= min (M, N)

SC: O(min(M,N))

HW: Print all the anti-diagonals.

Applications of Matrices.

1. Image representation — 3D matrix. if R, G, B values.
2. Bunch of images → 4D matrix.
3. Graph data structures.
4. Solving linear equations

$$2x + 3y = 5$$
$$3x + 2y = 7$$

$$\begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

5. Video → 4D matrix.

* arr = [[1, 2, 3], [3, 4], [5]]

↓

Not a matrix.

Diff: List of lists ≠ 2D Array / Matrix.

↓ ✗

↓ same no. of columns. ✓

Color: [232, 5, 126]

↓ ↓ ↓

R G B.

[(R,G,B) (R,G,B)] ← 3D matrix.

Image.

From Leetcode / Interview Bit /

Topic: Arrays