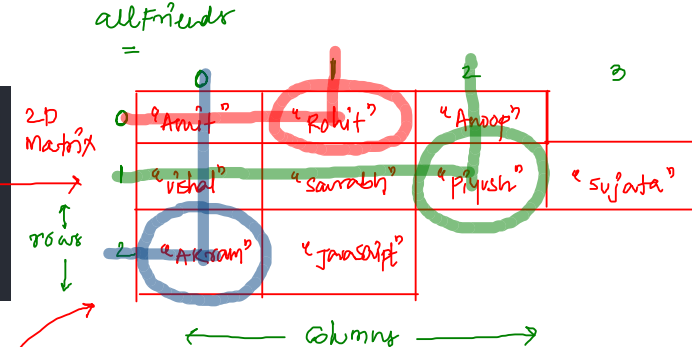
★ Array of Arrays (2D Arrays):

```
684  const friends1 = ["Amit", "Rohit", "Anoop"];
685  const friends2 = ["Vishal", "Saurabh", "Piyush", "Sujata"];
686  const friends3 = ["Akram", "JavaScript"];
687
688  const allFriends = [friends1, friends2, friends3];
689
690  console.log(allFriends);
691  console.log(allFriends[0]);
692  console.log(allFriends[1]);
693  console.log(allFriends[2]);
```




```
[
0 [ "Amit", "Rohit", "Anoop" ],          → 3
1 [ "Vishal", "Saurabh", "Piyush", "Sujata" ],   → 4
2 [ "Akram", "JavaScript" ]              → 2
]
```

friends2[2] → "Piyush" (allFriends[1][2] — 1st row, 2nd col)

friends3[0] → "Akram" (allFriends[2][0] — 2nd row, 0th col)

friends1[1] → "Rohit" (allFriends[0][1] — 0th row, 1st col)

★ If the smaller arrays or the rows do not have equal length they are called as "Jagged Arrays".

allFriends =

	0	1	2	3
0	"Amit"	"Rohit"	"Anoop"	
1	"vishal"	"saurabh"	"piyush"	"sujata"
2	"Akram"	"javaScript"		

* no. of rows
⇒ no. of smaller arrays

allFriends = [[3], [4], [2]].length
(0, 1, 2)

* no. of cols
→ no. of ele's in a smaller Array (each row)

No. of rows = 3 = allFriends.length

No. of cols ⇒ row-0 → 3 = allFriends[0].length = ["Amit", "Rohit", "Anoop"].length
row-1 → 4 = allFriends[1].length
row-2 → 2 = allFriends[2].length

Given a n x n matrix, print it Accordingly

n → rows, n → cols

if n = 3

mat =

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

3 x 3

op:

1 2 3
4 5 6
7 8 9

mat =

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

4 x 4

op:

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

```
for(let i = 0; i < mat[0].length; i++) {   // (no. of cols in row 0)
    process.stdout.write(mat[0][i] + " ");
}
for(let i = 0; i < mat[1].length; i++) {   // (no. of cols in row 0)
    process.stdout.write(mat[1][i] + " ");
}
```

* i is being used to refer the column number.

① i = 0 → mat[0][0]
② i = 1 → mat[0][1]
③ i = 2 → mat[0][2]
④ i = 3 ✗

```
725   const rows = mat.length;
726   // as the no.of cols are same for every row
727   const cols = mat[0].length;
728
729   // go to every row
730   for (let r = 0; r < rows; r++) {
731     // iterate on cols of that row
732     // when in case of jagged arrays (no.of cols in each row dif
733     // const cols = mat[r].length;
734     for (let c = 0; c < cols; c++) {
735       process.stdout.write(mat[r][c] + " ");
736     }
737     console.log();
738   }
```

mat =

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

3 × 3

rows = [[1,2,3], [4,5,6], [7,8,9]] . length

= 3

cols = [1, 2, 3] . length = 3

① r = 0
- → c = 0 → mat[0][0] → 1
- → c = 1 → mat[0][1] → 2
- → c = 2 → mat[0][2] → 3
- → ~~c = 3~~ (3 < 3)

② r = 1
- → c = 0 → mat[1][0] → 4
- → c = 1 → mat[1][1] → 5
- → c = 2 → mat[1][2] → 6
- → ~~c = 3~~ (3 < 3)

③ r = 2
- → c = 0 → mat[2][0] → 7
- → c = 1 → mat[2][1] → 8
- → c = 2 → mat[2][2] → 9
- → ~~c = 3~~ (3 < 3)

④ ~~r = 3~~ (3 < 3)

Create n x n matrix in the following pattern :

1. Create a smaller array i.e, a row
2. add it to the bigger array i.e, matrix

If n = 3

mat =

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

3 x 3

if n = 4

mat =

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

4 x 4

```
const mat = [];

const arr1 = [];
for(let i = 0; i < 3; i++) {
    arr1.push(i+1);
}
```

i=0 i=1 i=2

arr1 → ~~[]~~ ~~[1]~~ ~~[1,2]~~ [1,2,3]

```
mat.push(arr1);
```

mat → ~~[]~~ [[1,2,3]]

```
const arr2 = [];
for(let i = 0; i < 3; i++) {
    arr2.push(i+1);
}
```

i=0 i=1 i=2

arr2 → ~~[]~~ ~~[1]~~ ~~[1,2]~~ [1,2,3]

```
mat.push(arr2);
```

mat → ~~[[1,2,3]]~~ → [[1,2,3], [1,2,3]]

n = 3

```
function createMatrix(n) {
  const mat = [];
  for (let r = 0; r < n; r++) {
    const arr = [];
    for (let c = 0; c < n; c++) {
      arr.push(c + 1);
    }
    mat.push(arr);
  }
  return mat;
}
```

mat = [ ]

① r = 0
→ arr = [ ]
→ c = 0 → [ ] → [1]
→ c = 1 → [1] → [1, 2]
→ c = 2 → [1, 2] → [1, 2, 3]
→ ~~c = 3~~ (3 < 3)
→ [ ] → [[1, 2, 3]]

② r = 1
→ arr = [ ]
→ c = 0 → [ ] → [1]
→ c = 1 → [1] → [1, 2]
→ c = 2 → [1, 2] → [1, 2, 3]
→ ~~c = 3~~ (3 < 3)
→ ~~[[1, 2, 3]]~~
[[1, 2, 3], [1, 2, 3]]

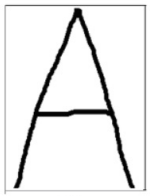
③ r = 2
→ arr = [ ]
→ c = 0 → [ ] → [1]
→ c = 1 → [1] → [1, 2]
→ c = 2 → [1, 2] → [1, 2, 3]
→ ~~c = 3~~ (3 < 3)
→ ~~[[1, 2, 3], [1, 2, 3]]~~

④ ~~r = 3~~ (3 < 3)

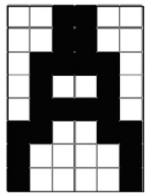
mat = [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

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

★ Some use cases :



(a)



(b)

0	0	1	1	0	0
0	0	1	1	0	0
0	1	1	1	1	0
0	1	0	0	1	0
0	1	1	1	1	0
1	1	0	0	1	1
1	0	0	0	0	1
1	0	0	0	0	1

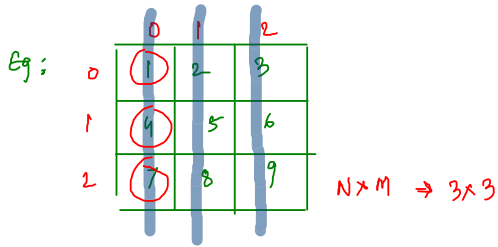
(c)

→ matrices are used in representing images

→ can be used to design a chess game

9:25 PM
— 9:40 PM
BREAK

Q: Print Matrix column wise :




```
function printMatrixColumnwise(mat, n, m) {
  // Write code here and print output

  // go to every column
  for (let c = 0; c < m; c++) {
    // iterate on rows
    for (let r = 0; r < n; r++) {
      process.stdout.write(mat[r][c] + " ");
    }
  }
}
```

* go to each column and iterate on row elements

mat[0][0] → 1
mat[1][0] → 4
mat[2][0] → 7
Col = 0
row = 0 → 2

mat[0][1] → 2
mat[1][1] → 5
mat[2][1] → 8
Col = 1
row = 0 → 2

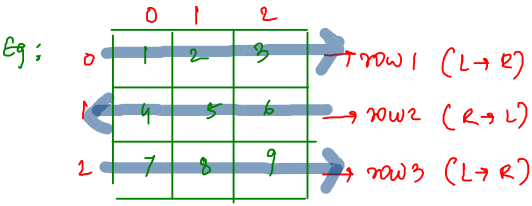
① c = 0
→ r = 0 → mat[0][0] → 1
→ r = 1 → mat[1][0] → 4
→ r = 2 → mat[2][0] → 7

② c = 1
→ r = 0 → mat[0][1] → 2
→ r = 1 → mat[1][1] → 5
→ r = 2 → mat[2][1] → 8

③ c = 2
→ r = 0 → mat[0][2] → 3
→ r = 1 → mat[1][2] → 6
→ r = 2 → mat[2][2] → 9

Q: Alternate Matrix Traversal :

★ odd row → print L to R
even row → print R to L



row idx (r) = even (L→R)
= odd (R→L)

rows
cols

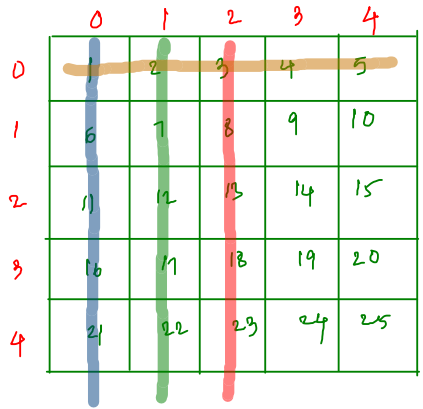
```
function printElementsAlternately(mat, m, n) {
  // Write code here and print output
  for (let r = 0; r < m; r++) {
    if (r % 2 == 0) {
      for (let c = 0; c < n; c++) {
        process.stdout.write(mat[r][c] + " ");
      }
    } else {
      for (let c = n - 1; c >= 0; c--) {
        process.stdout.write(mat[r][c] + " ");
      }
    }
  }
}
```

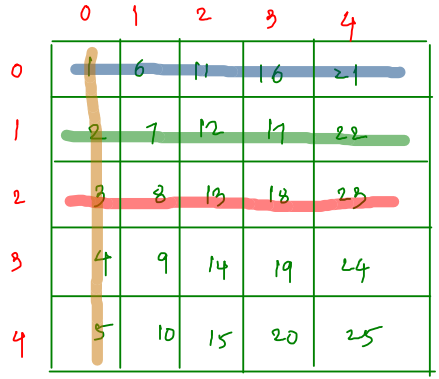
op: 1 2 3 6 5 4 7 8 9

① r = 0
r%2 == 0 (even)
print L→R
→ c=0 → mat[0][0] → 1
→ c=1 → mat[0][1] → 2
→ c=2 → mat[0][2] → 3

② r = 1
r%2 == 0 (odd)
print R→L
→ c=2 → mat[1][2] → 6
→ c=1 → mat[1][1] → 5
→ c=0 → mat[1][0] → 4

Q: Transpose of matrix :





rows ⟷ cols

⇒ 2 concepts → iterating colwise as seen in a previous problem

↘ creating matrix that we had in discussion

```
function matrixTranspose(mat, n) {
  //Write your code here
  const transMat = [];
  for (let c = 0; c < n; c++) {
    const arr = [];
    for (let r = 0; r < n; r++) {
      arr.push(mat[r][c]);
    }
    transMat.push(arr);
  }

  return transMat;
}
```

mat =

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

* TC : $O(N^2)$

SC : $O(N^2)$

① c = 0

arr = []

→ r = 0 → [1]

→ r = 1 → [1,4]

→ r = 2 → [1,4,7]

→ transMat = [[1,4,7]]

② c = 1

arr = []

→ r = 0 → [2]

→ r = 1 → [2,5]

→ r = 2 → [2,5,8]

→ transMat = [[1,4,7], [2,5,8]]

③ c = 2

arr = []

→ r = 0 → [3]

→ r = 1 → [3,6]

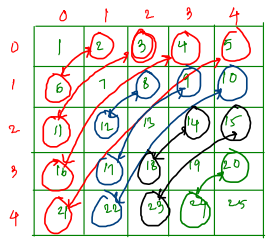
→ r = 2 → [3,6,9]

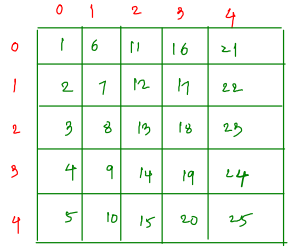
→ transMat = [[1,4,7], [2,5,8], [3,6,9]]

[[1,4,7], [2,5,8], [3,6,9]]

1	4	7
2	5	8
3	6	9

Improve SC : O(N²) → O(1) [Inplace] :





[r, c] ↔ [c, r]

mat[1][0] ↔ mat[0][1]

mat[2][0] ↔ mat[0][2]

mat[3][0] ↔ mat[0][3]

mat[4][0] ↔ mat[0][4]

mat[2][1] ↔ mat[1][2]

mat[3][1] ↔ mat[1][3]

mat[4][1] ↔ mat[1][4]

```
for(let r = 0; r < n; r++) {
    for(let c = 0; c < n; c++) {
        swap(mat[r][c], matrix[c][r]);
    }
}
```

⟹ This fails due to double swaps

① r = 0

→ c = 0 → swap(mat(0)(0), mat(0)(0))

→ c = 1 → swap(mat(0)(1), mat(1)(0))

→ c = 2 → swap(mat(0)(2), mat(2)(0))

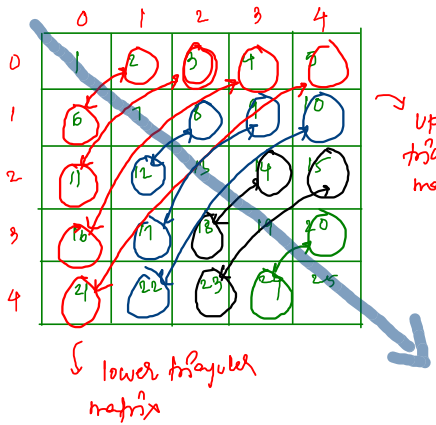
→ c = 3 → swap(mat(0)(3), mat(3)(0))

→ c = 4 → swap(mat(0)(4), mat(4)(0))

② r = 1 (double times swap)

→ c = 0 → swap(mat(1)(0), mat(0)(1))

→ avoid double swaps,



r = 0 → stop = 0
r = 1 → stop = 1
r = 2 → stop = 2
} stop = r

★ perform swap(mat[r][c], mat[c][r])
either on upper triangular matrix / lower triangular matrix

```
for (let r = 0; r < n; r++) {
    for (let c = 0; c < r; c++) {
        swap(mat[r][c], mat[c][r]);
    }
}
```

★ TC: $\frac{n(n+1)}{2} = O(N^2)$

SC: O(1)

★ only works for N × N matrix