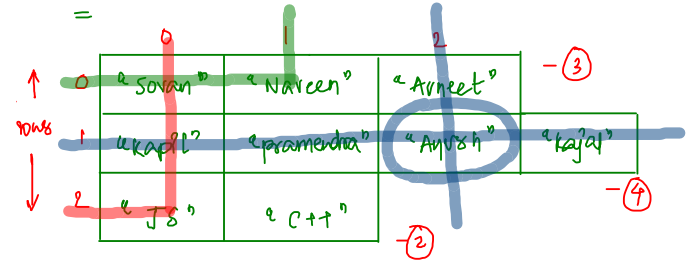
★ Array of Arrays (2D Arrays):

```
const friends1 = ["Sovan", "Naveen", "Avneet"];
const friends2 = ["Kapil", "Pramendra", "Ayush", "Kajal"];
const friends3 = ["JavaScript", "C++"];

const allFriends = [friends1, friends2, friends3];

console.log(allFriends);
console.log(allFriends[0]);
console.log(allFriends[1]);
console.log(allFriends[2]);
```

allFriends = Matrix representation




```
[
0 ["Sovan", "Naveen", "Avneet"],
1 ["Kapil", "Pramendra", "Ayush", "Kajal"],
2 ["JavaScript", "C++"]
]
```

allFriends[1][2]
friends2[2] 1st Row, 2nd col
→ "Ayush"

allFriends[0][1]
friends1[1] 0th Row, 1st col
→ "Naveen"

allFriends[2][0]
2nd Row, 0th col
friends3[0]
→ "JS"

★ If the smaller Arrays or the rows have unequal lengths they are called "Jagged Arrays".

allFriends =

	0	1	2	
0	"Sovan"	"Naveen"	"Avneet"	
1	"kapil"	"pramendra"	"Ayush"	"kajal"
2	"JS"	"C++"		

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

0 1 2

★ No. of rows = 3 = No. of Smaller Arrays
= allFriends.length

→ ["Sovan", "Naveen", "Avneet"].length = 3

★ No. of cols => row-0 = 3 = allFriends[0].length
= No. of ele's in a smaller Array (each Row)
row-1 = 4 = allFriends[1].length
row-2 = 2 = allFriends[2].length

Q: Given a $n \times n$ matrix, print it as follows

rows x cols

Let $n = 3$

	0	1	2
0	10	11	12
1	15	13	14
2	16	17	18

3 x 3

op: 10 11 12

15 13 14

16 17 18

```
903  function printMatrix(mat) {
904    const cols = mat[0].length;
905
906    // print 0th row
907    for (let i = 0; i < cols; i++) {
908      process.stdout.write(mat[0][i] + " ");
909    }
910    console.log();
911
912    // print 1st row
913    for (let i = 0; i < cols; i++) {
914      process.stdout.write(mat[1][i] + " ");
915    }
916    console.log();
917
918    // print 2nd row
919    for (let i = 0; i < cols; i++) {
920      process.stdout.write(mat[2][i] + " ");
921    }
922  }
```

row-0

row-1

row-2

→ this works only for 3 Rows

→ this is a repetitive task we can use nested loops where after loop is for every row.

```
924  function printMatrix(mat) {
925    const rows = mat.length;
926    const cols = mat[0].length;
927
928    for (let r = 0; r < rows; r++) {
929      // Iterate on the cols of the current row
930      for (let c = 0; c < cols; c++) {
931        process.stdout.write(mat[r][c] + " ");
932      }
933      console.log();
934    }
935  }
```

mat =

	0	1	2
0	10	11	12
1	15	13	14
2	16	17	18

3 × 3

mat
↳ bigga Array
mat[0], mat[1], mat[2] } elements Smaller Array

cols = mat[r].length
= Jagged Arrays.

rows = [[10, 11, 12], [15, 13, 14], [16, 17, 18]].length
(0) (1) (2)

= 3

cols = [10, 11, 12].length

= 3

① r = 0
→ c=0 → mat(0)(0) → 10
→ c=1 → mat(0)(1) → 11
→ c=2 → mat(0)(2) → 12
→ c=3 ✗

② r = 1
→ c=0 → mat(1)(0) → 15
→ c=1 → mat(1)(1) → 13
→ c=2 → mat(1)(2) → 14
→ c=3 ✗

③ r = 2
→ c=0 → mat(2)(0) → 16
→ c=1 → mat(2)(1) → 17
→ c=2 → mat(2)(2) → 18
→ c=3 ✗

Q: Creat a n x n matrix in the following pattern

Eg: n = 3

mat =

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

3x3

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

4x4

[[1, 2, 3], [1, 2, 3], [1, 2, 3]]

focus on how to build this smaller Array (Single Row)

[1, 2, 3] n = 3

[1, 2, 3, 4] n = 4

```
// How to build a single row
const smallArr = [];
for (let c = 0; c < cols; c++) {
  smallArr.push(c + 1);
}
```

```
function createMatrix(n) {
  const rows = n;
  const cols = n;
  const mat = []; // bigger array

  // Build smaller arrays for every row
  for (let r = 0; r < rows; r++) {
    // How to build a single row
    const smallArr = [];
    for (let c = 0; c < cols; c++) {
      smallArr.push(c + 1);
    }
    mat.push(smallArr);
  }

  return mat;
}
```

$n = 3$

$\Rightarrow$ rows = cols = $n = 3$

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

[[1,2,3],[1,2,3],[1,2,3]]

0 1 2

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

① r = 0

smallerArr = []

→ c = 0, ~~[]~~ [1]

→ c = 1, ~~[1]~~ [1,2]

→ c = 2, ~~[1,2]~~ [1,2,3]

→ c = 3 ×

② r = 1

smallerArr = []

→ c = 0, ~~[]~~ [1]

→ c = 1, ~~[1]~~ [1,2]

→ c = 2, ~~[1,2]~~ [1,2,3]

→ c = 3 ×

② r = 2

smallerArr = []

→ c = 0, ~~[]~~ [1]

→ c = 1, ~~[1]~~ [1,2]

→ c = 2, ~~[1,2]~~ [1,2,3]

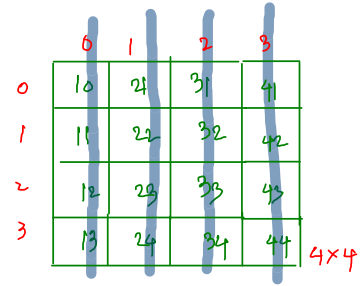
→ c = 3 ×

③ r = 3

×

Q: print matrix column wise :

eg:

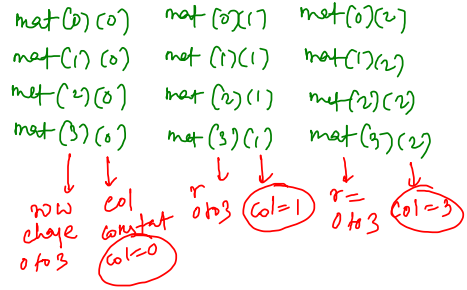


op: 10 11 12 13 21 22 23 24

31 32 33 34 41 42 43 44

Earlier, go to row iterate on col,

Now, go to col iterate on Row.




```
924  function printMatrixColumnwise(mat, n, m) {
925    const rows = n;
926    const cols = m;
927
928    for (let c = 0; c < cols; c++) {
929      // Pick the column, Iterate on row
930      for (let r = 0; r < rows; r++) {
931        process.stdout.write(mat[r][c] + " ");
932      }
933    }
934  }
```

① c = 0

→ r = 0 → m(0)(0)
→ r = 1 → m(1)(0)
→ r = 2 → m(2)(0)
→ r = 3 → m(3)(0)

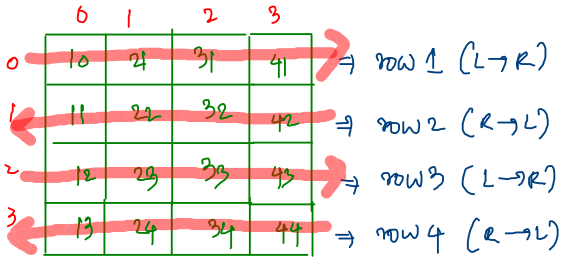
② c = 1

→ r = 0 → m(0)(1)
→ r = 1 → m(1)(1)
→ r = 2 → m(2)(1)
→ r = 3 → m(3)(1)

Q : Alternate Matrix traversal :

* odd Row → print L to R
even Row → print R to L

eg ;



op; 10 21 31 41 42 32 22 11
12 23 33 43 44 34 24 13

→ even Row Idx → LtoR
odd Row Idx → RtoL

```
function printElementsAlternately(mat, m, n) {
  const rows = m;
  const cols = n;

  for (let r = 0; r < rows; r++) {
    if (r % 2 == 0) {
      for (let c = 0; c < cols; c++) {
        process.stdout.write(mat[r][c] + " ");
      }
    } else {
      for (let c = cols - 1; c >= 0; c--) {
        process.stdout.write(mat[r][c] + " ");
      }
    }
  }
}
```

① r = 0 ,
r % 2 == 0
→ c = 0 → m(0)(0)
→ c = 1 → m(0)(1)
→ c = 2 → m(0)(2)
→ c = 3 → m(0)(3)

② r = 1
r % 2 == 1
→ c = 3 → m(1)(3)
→ c = 2 → m(1)(2)
→ c = 1 → m(1)(1)
→ c = 0 → m(1)(0)