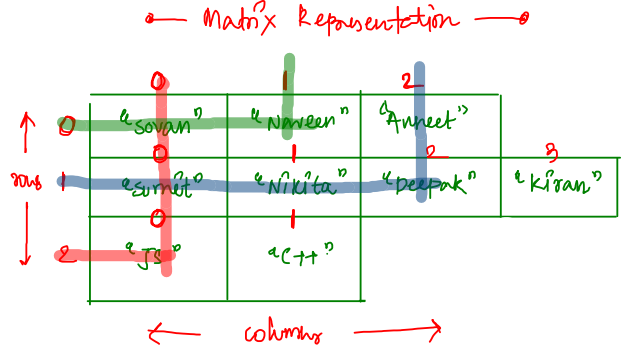
* Array of Arrays (2D Array):

```
const friends1 = ["Sovan", "Naveen", "Avneet"];
const friends2 = ["Sumit", "Nikita", "Deepak", "Kiran"];
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 =

[["Sovan", "Naveen", "Avneet"], ["Sumit", "Nikita", "Deepak", "Kiran"], ["JavaScript", "C++"]]

0 1 2 — 0 1 2 3 — 0 1

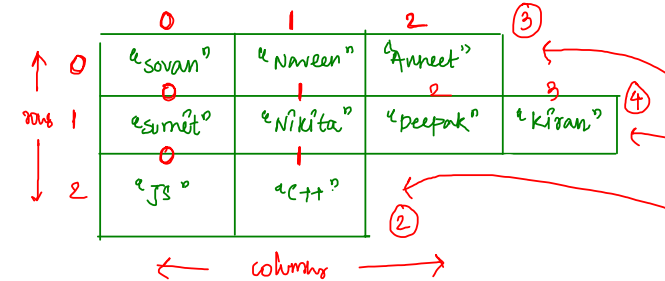
0 → (0)(1) — 1 → (1)(2) — 2 → (2)(0)

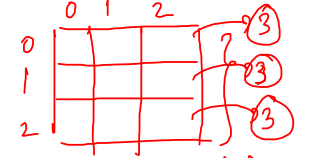
eg: allFriends[1][2] → "Deepak"
1st Row, 2nd col

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

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

allFriends =





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

0 1 2

* No. of ROWS = 3 = No. of smaller Arrays = no. of elements in allFriends Array
  = allFriends.length

["sovan", "Naveen", "Anmol"].length = 3

* No. of cols = No. of ele's in a smaller Array

row-0 = allFriends[0].length = 3
row-1 = allFriends[1].length = 4
row-2 = allFriends[2].length = 2

} allFriends[row].length

Q: Given a n x n matrix, print it as follows

rows cols

```
for(let r=0; r<rows; r++){
  for(let c=0; c<cols; c++){
    psw(mat[r][c] + " ");
  }
  Console.log();
}
```

IP: 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

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

Q: Create a N x N matrix in the following pattern

Ip: N = 3

Op:

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

3x3

Ip: N = 4

Op:

	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

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

→ To build an entire matrix,

= first smaller arrays are needed

```
const arr1 = [1, 2, 3];
const arr2 = [1, 2, 3];
const arr3 = [1, 2, 3];
```

} build smaller Arrays

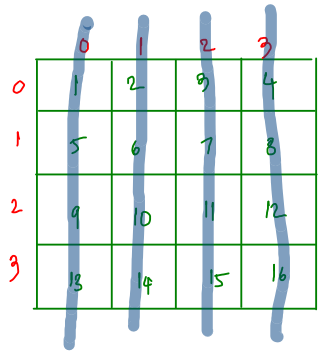
```
const mat = [arr1, arr2, arr3];
```

} store the smaller Arr in a bigger Array

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

* print column wise :

Eg:



4×4

previously,

→ goto each row
→ goto each col

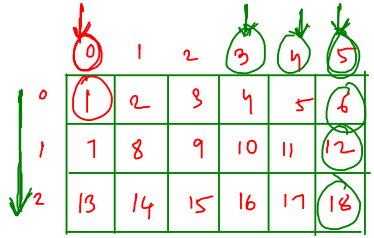
now,

→ go to each col,
→ go to each row

```
for(let c = 0; c < cols; c++)
    for(let r = 0; r < rows; r++) {
        psw(mat[r][c] + " ");
    }
}
```

① c = 0
→ r = 0, mat(0)(0)
→ r = 1, mat(1)(0)
→ r = 2, mat(2)(0)
→ r = 3, mat(3)(0)
→ r = 4 ✗

② c = 1
→ r = 0 → mat(0)(1)
→ r = 1 → mat(1)(1)
→ r = 2 → mat(2)(1)
→ r = 3 → mat(3)(1)



Mat[i][j]
→ Mat[r][c]

r = 0 → 2
c = 0 → 5
} always only follow mat(r)(c)

Mat[j][i]
→ mat[c][r]
0 → 5 0 → 2

mat[4][1]
} Never do this
X

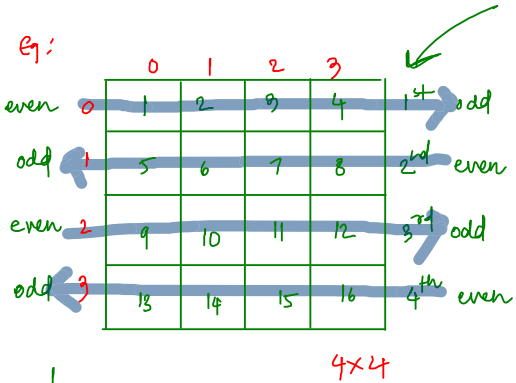
(0)(0)
(1)(0)
(2)(0)
r=0→2 c=0

(0)(1)
(1)(1)
(2)(1)
0 to c=1

r (0)(5)
(1)(5)
(2)(5)

★ Alternate Matrix Traversal :

Eg:



r is even L → R

r is odd R → L

odd Rows (L → R)

1st Row,
3rd Row,
5th Row etc...

even Rows (R → L)

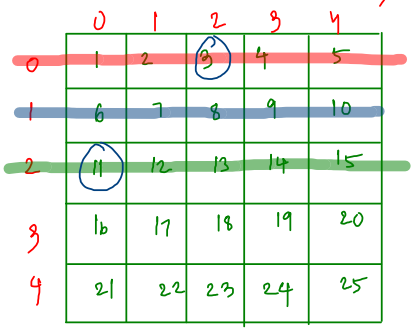
2nd Row,
4th Row
6th Row etc....

```
1121  function printElementsAlternately(mat, m, n) {
1122    //Write code here and print output
1123    for (let r = 0; r < m; r++) {
1124      if (r % 2 == 0) {
1125        for (let c = 0; c < n; c++) {
1126          process.stdout.write(mat[r][c] + " ");
1127        }
1128      } else {
1129        for (let c = n - 1; c >= 0; c--) {
1130          process.stdout.write(mat[r][c] + " ");
1131        }
1132      }
1133    }
1134  }
```

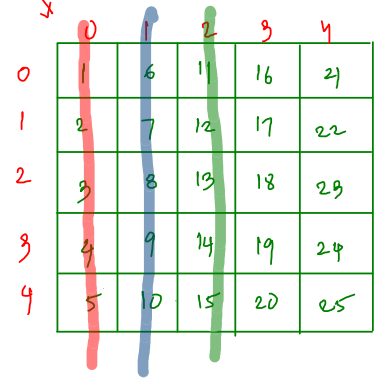
r = 0 → c = 0 1 2 3 ⊗ (L → R)

r = 1 → c = 3 2 1 0 ⊗ (R → L)

* Transpose of a Matrix :

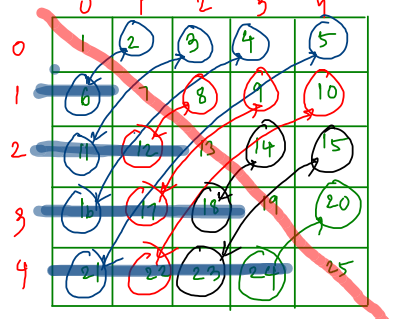


transpose



→ Interchanging rows and cols.

upper triangular matrix



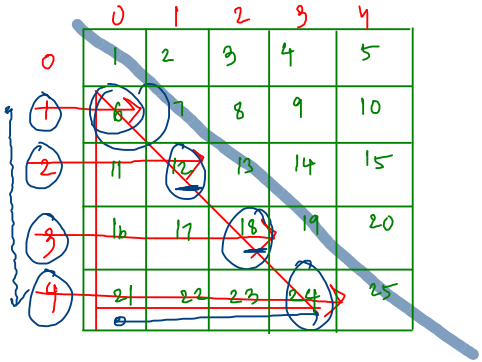
lower triangular matrix

r c c r

Code should perform these operations:

- mat(1)(0) ↔ mat(0)(1)
- mat(2)(0) ↔ mat(0)(2)
- mat(2)(1) ↔ mat(1)(2)
- mat(4)(0) ↔ mat(0)(4)
- mat(4)(1) ↔ mat(1)(4)

- mat(3)(0) ↔ mat(0)(3)
- mat(3)(1) ↔ mat(1)(3)
- mat(3)(2) ↔ mat(2)(3)
- mat(4)(2) ↔ mat(2)(4)
- mat(4)(3) ↔ mat(3)(4)




```
for (let r = 1; r < rows; r++) {
    for (let c = 0; c < r; c++) {
        let temp = mat[r][c];
        mat[r][c] = mat[c][r];
        mat[c][r] = temp;
    }
}
```

row = 1 → c = ~~0~~ (c < 1)

row = 2 → c = 0 1 ~~2~~ (c < 2)

row = 3 → c = 0 1 2 ~~3~~ (c < 3)

row = 4 → c = 0 1 2 3 ~~4~~ (c < 4)

$(r, c) \longleftrightarrow (c, r)$

```
function hw1(n) {
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < i; j++) {
      console.log("*");
      break;
    }
  }
}

function hw2(n) {
  let i = 1;
  while (i ** 2 <= n) {
    i = i + 1;
  }
}

function hw3(m, n) {
  while (m != n) {
    if (m > n) {
      m = m - n;
    } else {
      n = n - m;
    }
  }
}
```

→ O(N)

→ Inner loop → (1)

$i^2 \le N$

$\Rightarrow i \le \sqrt{N}$

$O(\sqrt{N})$ $i = 1 \xrightarrow{+1} \sqrt{N}$

m = 4, n = 1

4 % 1 = 1, m = 3

3 % 1 = 1, m = 2

2 % 1 = 1, m = 1

1 % 1 = 1 ✗

m = 4, n = 1, O(m)

m = 1, n = 4, O(N)

} O(max(m,n)) (or) O(m+n)

```
function hw4(n) {
  let i = 1;
  while (i < n) {
    let j = n;
    while (j > 0) {
      j = parseInt(j / 2);
    }
    i = i * 2;
  }
}

function hw5(n) {
  for (let i = 0; i < parseInt(n / 2); i++) {
    for (let j = 1; j < n - parseInt(n / 2); j++) {
      let m = 1;
      while (m <= n) {
        m = m * 2;
      }
    }
  }
}
```

$i = 1 \xrightarrow{*2} n$ $O(\log_2 n)$

$j = n \xrightarrow{/2} 1$ $O(\log_2 n)$

$O((\log_2 n)^2)$

$i = 0 \xrightarrow{+1} n/2$ → O(N)

$j = 1 \xrightarrow{+1} n - n/2 = n/2$ → O(N)

$m = 1 \xrightarrow{*2} n$ $O(\log_2 n)$

$O(n * n * \log_2 n)$

$\Rightarrow O(n^2 \log_2 n)$