* Right Angle Triangle pattern :

Eg: n = 4

op:

```
*
* *
* * *
* * * *
```

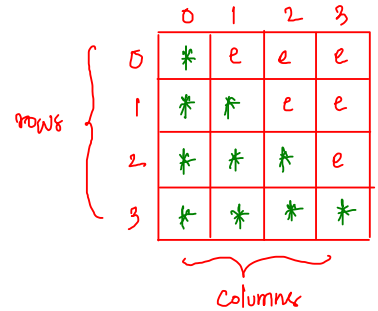
Eg: n = 2

```
*
* *
```

Eg: n = 3

```
*
* *
* * *
```

① observe / Analyze the pattern



Eg: N = 4

No. of Rows = 4 = N

No. of cols in each Row = rowNum + 1

row - 0 ⇒ cols = 1

row - 1 ⇒ cols = 2

row - 2 ⇒ cols = 3

row - 3 ⇒ cols = 4

② Code

1. you have to print every Row

```
for (let r = 0; r < rows; r++) {
```

N

2. In a Row print every column

```
    for(let c = 0; c < cols; c++) {
        psw("*");
    }
    console.log();
}
```

r+1

```
readline.question("", (n) => {
  //Write your code here
  for (let r = 0; r < n; r++) {
    for (let c = 0; c < r + 1; c++) {
      process.stdout.write("*");
    }
    console.log();
  }

  readline.close();
});
```

c < r+1
⇒ c <= r
(both same)

n = 4

```
*
* *
* * *
* * * *
```

① r = 0, 0 < 4
- a. c = 0, 0 < 0+1 ⇒ 0 < 1
- b. c = 1, 1 < 1 X

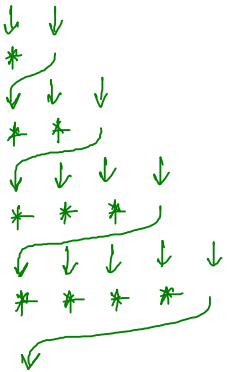
② r = 1, 1 < 4
- a. c = 0, 0 < 1+1 ⇒ 0 < 2
- b. c = 1, 1 < 2
- c. c = 2, 2 < 2 X

③ r = 2, 2 < 4
- a. c = 0, 0 < 2+1 ⇒ 0 < 3
- b. c = 1, 1 < 3
- c. c = 2, 2 < 3
- d. c = 3, 3 < 3 X

④ r = 3, 3 < 4
- a. c = 0, 0 < 4
- b. c = 1, 1 < 4
- c. c = 2, 2 < 4
- d. c = 3, 3 < 4
- e. c = 4, 4 < 4

⑤ r = 4, 4 < 4 X

op:



★ Staircase pattern :

eg: n = 4

op:

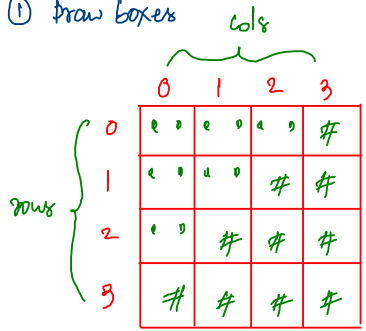
```
   #
  # #
 # # #
# # # #
```

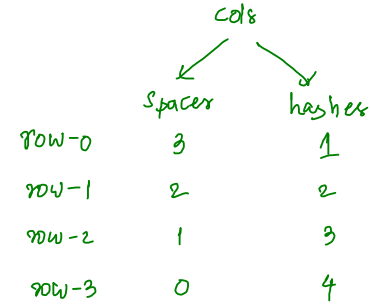
eg: n = 3

op:

```
  #
 # #
# # #
```

① Draw boxes





No. of Rows = 4 = N

No. of cols in each row = 4 = N → spacer = n-r-1, hasher = r+1

→ no. of hashes = rowNum + 1

→ no. of spaces = cols - hashes

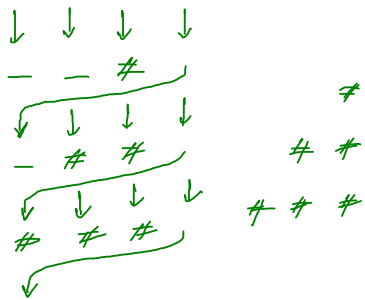
= N - hashes

= N - (rowNum + 1) = N - rowNum - 1

```
readline.question("", (n) => {
  n = parseInt(n);
  //write code here
  for (let r = 0; r < n; r++) {
    const hashes = r + 1;
    const spaces = n - hashes;
    for (let sp = 0; sp < spaces; sp++) {
      process.stdout.write(" ");
    }

    for (let ha = 0; ha < hashes; ha++) {
      process.stdout.write("#");
    }

    console.log();
  }
  readline.close();
});
```



Eg: N = 3

① r = 0, 0 < 3

→ h = 0+1 = 1

→ s = 3-1 = 2

a. sp = 0, 0 < 2

b. sp = 1, 1 < 2

c. sp = 2, 2 < 2 ✗

---

a. ha = 0, 0 < 1

b. ha = 1, 1 < 1 ✗

② r = 1, 1 < 3

→ h = 1+1 = 2

→ s = 3-2 = 1

a. sp = 0, 0 < 1

b. sp = 1, 1 < 1 ✗

---

a. ha = 0, 0 < 2

b. ha = 1, 1 < 2

c. ha = 2, 2 < 2 ✗

③ r = 2, 2 < 3

→ h = 2+1 = 3

→ s = 3-3 = 0

a. sp = 0, 0 < 0 ✗

a. ha = 0, 0 < 3

b. ha = 1, 1 < 3

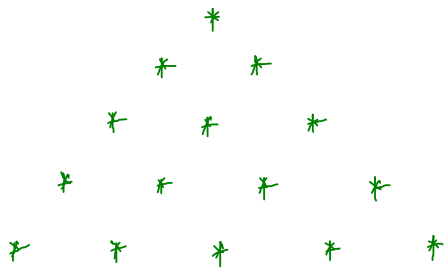
c. ha = 2, 2 < 3

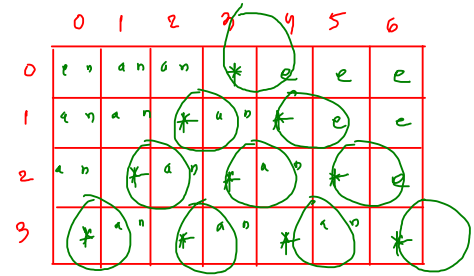
d. ha = 3, 3 < 3 ✗

④ r = 4, 4 < 4 ✗

★ star pyramid :

Eg: n = 5





Eg: N=4

no. of rows = 4
= N

No. of stars
= rowNum + 1

No. of spaces
= n - stars

cols

	spaces	pattern ("* ")
row = 0	3	1
row = 1	2	2
row = 2	1	3
row = 3	0	4

* Number pattern :

eg: n = 5

op:

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

* here we need to take care both shape and the changing pattern to print.

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

```
for(let r=0; r<n; r++) {
    let start = r+1;
    for(let c=0; c<r+1; c++) {
        psw(start);
        start--;
    }
    cl();
}
```

→ we can a row is starting with some number and decrease as it moves to next position.

	start Num
row=0	1
row=1	2
row=2	3
row=3	4

⇒ StartNum = row+1

No. of rows = 4 = N

No. of cols = rowNum+1

row=0 → cols=1

row=1 → cols=2

row=2 → cols=3

row=3 → cols=4

★ character pattern :

Eg: n = 4

op:

A

B B

C C C

D D D D

Eg; n = 3

op:

A

B B

C C C

	0	1	2	3
0	A	e	e	e
1	B	B	e	e
2	C	C	C	e
3	D	D	D	D

No. of rows = 4 = N

No. of cols = rowNum + 1

row = 0 ⇒ ch = 'A'

row = 1 ⇒ ch = 'B'

row = 2 ⇒ ch = 'C'

row = 3 ⇒ ch = 'D'

```
let ch = 'A'; // let ascii = 65
for(let r = 0; r < n; r++) {
  for(let c=0; c < r+1; c++) {
      psw(ch); // string.fromCharCode(ascii);
  }
  console.log();
  ch++; // ascii++;
}
```

↳ not possible look next slide

#1:

ch = 'A' —(65+1)→ 'B' —(66+1)→ 'C' —(67+1)→ 'D'

```
const ascii = ch.charCodeAt(0);
ch = String.fromCharCode(ascii + 1);
ch++;
```

1. get the Ascii of current character
2. get the character for Ascii + 1.

#2:

ascii = 65 → 66 → 67 → 68

→ increment ascii as you move to next row

→ when printing ⇒ String.fromCharCode(ascii)

65 → 'A'

66 → 'B' and so on ....

#3:

row	ascii
row = 0	65 (65+0)
row = 1	66 (65+1)
row = 2	67 (65+2)
row = 3	68 (65+3)

⇒ ascii = 65 + rowNum

```
readline.question("", (n) => {
  // Write your code here
  for (let r = 0; r < n; r++) {
    const ascii = 65 + r;
    for (let c = 0; c < r + 1; c++) {
      process.stdout.write(String.fromCharCode(ascii));
    }
    console.log();
  }
  readline.close();
});
```

★ Armstrong Numbers :

→ given a number check armstrong or not.

eg: n = 153

sum of (digit)$^{\text{no. of digits}}$ == number

⇒ $1^3 + 5^3 + 3^3 == 153$

⇒ $1 + 125 + 27 == 153$

⇒ $153 == 153$

op: True

eg: 1634

⇒ $1^4 + 6^4 + 3^4 + 4^4$

$= 1 + 1296 + 81 + 256$

$= 1634$

eg: 371

⇒ $3^3 + 7^3 + 1^3$

⇒ $27 + 343 + 1$

⇒ 371

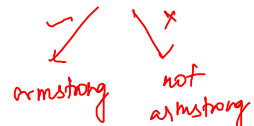
1. no. of digits in the given number "n"
(Digit Extraction)

```
let cnt = 0;
while (n != 0) {
  n = parseInt(n/10);
  cnt++;
}
```

1634 ⟶ 163 ⟶ 16 ⟶ 1 ⟶ (0) ×

cnt = 0 1 2 3 4

2. Sum == n



3. Sum of (digit)^numDigits

1634 ⇒ $1^4 + 6^4 + 3^4 + 4^4$

```
let sum = 0;
while (n != 0) {
  const lastDigit = n % 10;
  n = parseInt(n/10);
  sum += (lastDigit ** cnt);
}
```

1634 ⟶ 163 ⟶ 16 ⟶ 1 ⟶ 0

sum = 0
+ 4^4 + 6^4
+ 3^4 + 1^4

* Armstrong Numbers B/w [M, N] :

fq: m = 0 n = 380

op: 0 1 2 3 4 5 6 7 8 9 153 371

fq: m = 100 n = 400

op: 153 371

m = 100, n = 400

① num = 100 → check → if yes → print(100)

② num = 101 → check → if yes → print(101)

⋮

⑧ num = 400 → check → if yes → print(400)

```
for (let num = m; num <= n; num++) {
    /*
       check num is
          armstrong or not
    */
}
```