★ Right Angled Triangle Stars :

Eg : n = 4

op :

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

Eg : n = 2

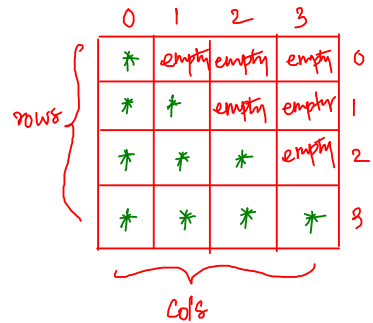
op :

```
*
* *
```

Eg : n = 3

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

① Observe / Analyze the pattern



Rows = 4 = N

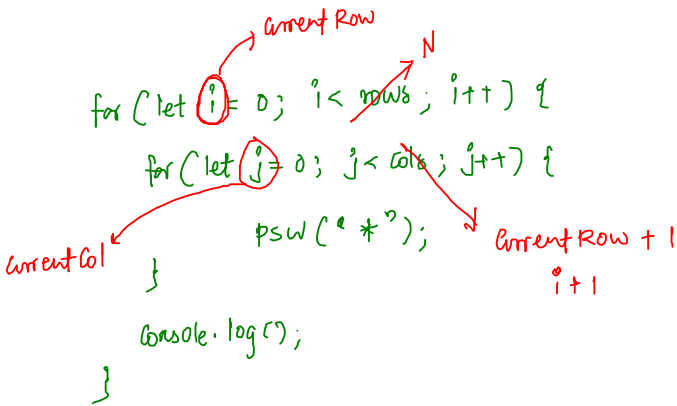
Cols (Ignore empty columns),

row = 0 → # Cols = 1
row = 1 → # Cols = 2
row = 2 → # Cols = 3
row = 3 → # Cols = 4

★ # Cols is dependent on row number,

cols = row + 1

② Code, (A Fixed code template for most of the problems)

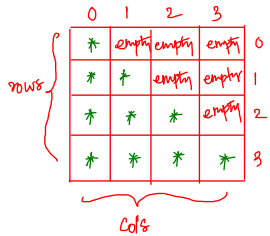


* change rows, cols according to your problem/pattern

```
for (let r = 0; r < n; r++) {
  for (let c = 0; c < r + 1; c++) {
    process.stdout.write("*");
  }
  console.log();
}
```

outer loop

inner loop



→ outer loop is going to every row

(r = 0; r < n; r++)

⇒ r = 0 1 2 3 4 . . . . n-1

→ Inner loop is going to every col w.r.t row

⇒ c = 0 1 2 . . . r

* go to every row print the patter of that row

① r = 0, 0 < 4, true

→ c = 0, 0 < r+1, 0 < 1, True

→ c = 1, 1 < 1, False

② r = 1, 1 < 4, True

→ c = 0, 0 < r+1, 0 < 2, True

→ c = 1, 1 < r+1, 1 < 2, True

→ c = 2, 2 < 2, False

③ r = 2, 2 < 4, true

→ c = 0, 0 < r+1, 0 < 3, True

→ c = 1, 1 < 3, True

→ c = 2, 2 < 3, True

→ c = 3, 3 < 3, False

④ r = 3, 3 < 4, True

→ c = 0, 0 < r+1, 0 < 4, True

→ c = 1, 1 < 4, True

→ c = 2, 2 < 4, True

→ c = 3, 3 < 4, True

→ c = 4, 4 < 4, False

⑤ r = 4, 4 < 4, false

★ StairCase :

Eg: n = 4

Op:

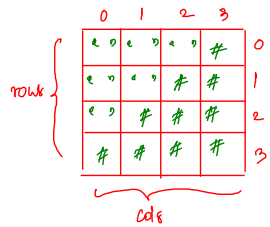
```
   #
  # #
 # # #
# # # #
```

Eg: n = 9

Op:

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

① observe / Analyze - Draw Boxes



★ If empty boxes before some character → spaces
empty boxes after characters → empty

Rows = 4 = N

★ Spaces, hashes are dependent on the current Row

	Cols	Spaces (n-r-1)	hashes
r=0	3	(4-0-1)	1
r=1	2	(4-1-1)	2
r=2	1	(4-2-1)	3
r=3	0	(4-3-1)	4

no. of hashes = r + 1

= current Row + 1

no. of Cols = no. of spaces + no. of hashes

N = no. of spaces + (r+1)

no. of spaces = N - (r+1)

= N-r-1

② Code,

```
for(let r = 0; r < rows; r++) {
    for(let sp = 0; sp < spaces; sp++) {
        psw(" ");
    }
    for(let ha = 0; ha < hasher; ha++) {
        psw("#");
    }
    console.log();
}
```

N

Iterating for no of spaces
iterate for N-r-1 times

N-r-1

r+1

cols

Iterating for r+1 times

```
360   for (let r = 0; r < n; r++) {
361     for (let sp = 0; sp < n - r - 1; sp++) {
362       process.stdout.write(" ");
363     }
364     for (let ha = 0; ha < r + 1; ha++) {
365       process.stdout.write("#");
366     }
367     console.log();
368   }
```

op:

① r = 0 (4-0-1)

→ sp = 0, 0 < n-r-1, 0 < 3, True

→ sp = 1, 1 < 3, True

→ sp = 2, 2 < 3, True

→ sp = 3, 3 < 3, False

→ ha = 0, 0 < r+1, 0 < 1, True

→ ha = 1, 1 < 1, False

② r = 1 (4-1-1)

→ sp = 0, 0 < n-r-1, 0 < 2, True

→ sp = 1, 1 < 2, True

→ sp = 2, 2 < 2, False

→ ha = 0, 0 < r+1, 0 < 2, True

→ ha = 1, 1 < 2, True

→ ha = 2, 2 < 2, False

③ r = 2

→ sp = 0 1

1 < n-r-1, 1 < 1, False

→ ha = 0 1 2 3

3 < r+1, 3 < 3, False

④ r = 3

sp = 0

0 < n-r-1

0 < 0, False

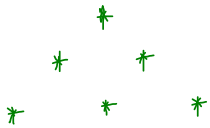
ha = 0 1 2 3 4

4 < r+1

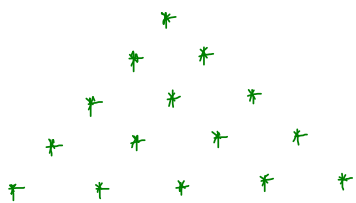
4 < 4, False

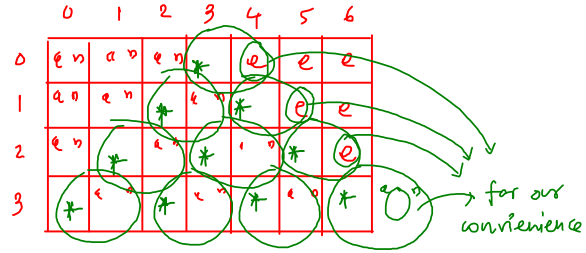
★ Star pyramid :

eg: n = 3



eg: n = 5





★ Instead considering your character as '*'
consider it as "*_"

	#cols	
	#spaces	#star_sp
r = 0	3	1
r = 1	2	2
r = 2	1	3
r = 3	0	4

⇒ exactly similar to staircase just the character was "#" here it is "*_"

★ Number pattern :

Eg : n = 5

op :

1

2 1

3 2 1

4 3 2 1

5 4 3 2 1

★ psw only accepts strings ~~or buffers~~, but we are passing a number, so to avoid error do type conversion

num → string(num)

psw(1234) → psw("1234")

(error)

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

start_num = r+1

and it gradually decreases with column

Rows = 4 = N

cols = r+1

→ how to handle number

	start_num
r=0	1
r=1	2
r=2	3
r=3	4

```
for(let r = 0; r < n; r++) {
    let start_num = r+1;
    for(let c = 0; c < r+1; c++) {
        psw(start_num);
        start_num--;
    }
    console.log();
}
```

★ Character Pattern :

Eg: n = 4

op:

A

B B

C C C

D D D D

Eg: n = 3

op:

A

B B

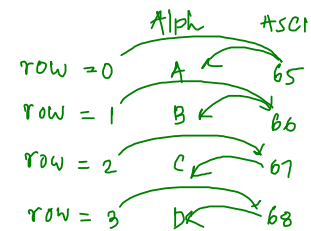
C C C

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

Rows = 4 = N

Cols = r + 1

→ how to handle alphabet



Ascii = 65 + row

```
for(let r=0; r<n; r++){
    Const alph = String.fromCharCode(65 + row);
    for(let c=0; c< r+1; c++) {
        psw(alph);
    }
    Console.log();
}
```

→ other way to write,

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

you can make the ascii global instead of calculating using current row

r=0 ascii = 65 → A
r=1 ascii = 66 → B (+1)
r=2 ascii = 67 → C (+1)
r=3 ascii = 68 → D (+1)

★ Can I write let alph = "A"

⇒ alph++ → alph = "B"

⇒ alph++ → alph = "C"

} wrong assumption

```
let alph = "A"
alph = alph + 1
console.log(alph);
```

"A" + 1 ⇒ "A1"

⇒ Need to see, why? JS is weird

```
alph++;
console.log(alph)
```

↓

NaN

★ Armstrong Numbers :

eg: $n = 153$

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

$\Rightarrow 1^3 + 5^3 + 3^3 = 1 + 125 + 27$

$= 153$

eg: $n = 1634$

$\Rightarrow 1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256$

$= 1634$

eg: $n = 371$

$\Rightarrow 3^3 + 7^3 + 1^3 = 27 + 343 + 1$

$= 371$

eg: $n = 9474$

$\Rightarrow 9^4 + 4^4 + 7^4 + 4^4$

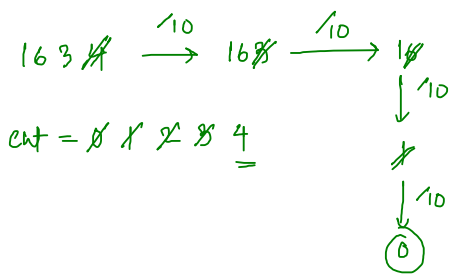
$= 6561 + 256 + 2401 + 256$

$= 9474$

1. No. of digits in the given number "n"

(digit extraction/Iteration)

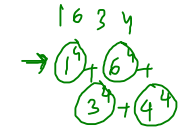
```
let cnt = 0;
while (n > 0) {
    n = parseInt(n/10);
    cnt++;
}
```



9. Sum == n

yes ✓ NO ✗

2. Sum of (digit)^no. of digits




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

1634 % 10 = 4

↓

163 % 10 = 3

↓

16 % 10 = 6

↓

1 % 10 = 1

sum = 0

+ 4^4

+ 3^4

+ 6^4

+ 1^4

⋆ Armstrong numbers in Range [m, n] :

m = 20, n = 10,000

20 → 10^4 (print all armstrong in between)

```
for (let num = m ; num <= n ; n++) {

    /* Code for check whether
       num is armstrong or not
    */

}
```

20 → print if it is armstrong
21 → armstrong
22
23
24
.
.
.
10^4

```
444   for (let num = m; num <= n; num++) {
445     // 1. count no.of digits
446     let cnt = 0;
447     while (num > 0) {
448       num = parseInt(num / 10);
449       cnt++;
450     }
451
452     // 2. sum of digit to the power no.of.digits
453     let sum = 0;
454     while (num > 0) {
455       const digit = num % 10;
456       sum += digit ** cnt;
457       num = parseInt(num / 10);
458     }
459
460     // 3. check for armstrong
461     if (sum == num) {
462       console.log(num);
463     }
464   }
```

let temp = num

num → temp

temp = num

num → temp

num = 1634

① 1634 → 163 → 16 → 1 → 0

cnt = 0 1 2 3 4

② while (num > 0)

↳ But num = 0 due to the 1st loop

↳ hence 2nd loop will not run.

③ sum == num

0 == 0

(every num will be a armstrong according to this code)

④ num++ in the for loop, (Infinite loop)

num = 0 1

★ Do not change the original num in the entire process, Copy / store it in another variable and use this for your process.

$[m = 1634, \quad n = 2000]$

① num = 1634

⇒ temp = ~~1634~~ ~~163~~ ~~16~~ ~~1~~ 0 ✗

cnt = ~~0 + 1 + 2 + 3~~ 4

⇒ temp = ~~1634~~ ~~163~~ ~~16~~ ~~1~~ 0 ✗

Sum = $0 + 4^4 + 3^4 + 6^4 + 1^4$

= 1634

⇒ Sum == num

1634 == 1634

② num = 1635

③ num = 1636 . . . .

num = 1998

, num = 1999

num = 2000

~~num = 2001~~

* even if $m > n$,

simply swap them,

m = ~~2000~~ ↖ 1634

n = ~~1634~~ ↙ 2000

⇒ Now run the code that we discussed.