* Right Angled Triangle stars :

eg: n = 4

op:

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

eg: n=2

op:

```
*
* *
```

eg: n=3

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

```
for(let i=0; i<(1); i++) {
    psw("*");
}
console.log();
for(let i=0; i<(2); i++) {
    psw("*");
}
console.log();
for(let i=0; i<(3); i++) {
    psw("*");
}
console.log();
for(let i=0; i<(4); i++) {
    psw("*");
}
```

based on this number

repititive task

* If we need to print the pattern for n = 1000;

Then we require 1000 loops, which is impossible

i = 0 → 1 , i = 0 → 2 , i = 0 → 3, ... i = 0 → 1000

① ② ③ 1000

* ** *** *** ... **

⇒ printing stars is a repitative task, every loop has same task of printing "*" based on some number.

Nested Loops

```
for (let i=1, i<=n; i++) {
    for(let j=0; j<i; j++) {
        psw ("*");
    }
    console.log();
}
```

① i = 1

`j=0; j<1; j++` *

② i = 2

`j=0; j<2; j++` * *

③ i = 3

`j=0; j<3; j++` * * *

④ i = 4

`j=0; j<4; j++` * * * *

n = 4

```
for (let i=1, i<=n; i++) {
    for(let j=0; j<i; j++) {
        psw ("*");
    }
    console.log();
}
```

1. i=1, 1<=4
   → j=0, 0<1
   1, 1<1 X

2. i=2, 2<=4
   → j=0, 0<2
   1, 1<2
   2, 2<2 X

3. i=3, 3<=4
   → j=0, 0<3
   1, 1<3
   2, 2<3
   3, 3<3 X

4. i=4, 4<=4
   → j=0, 0<4
   1, 1<4
   2, 2<4
   3, 3<4
   4, 4<4 X

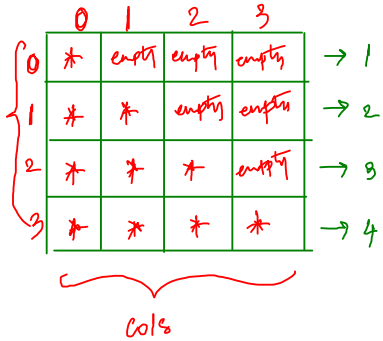
5. i=5, 5<=4 X

op:

Template for patterns :

1. observe / analyse the pattern

n = 4,



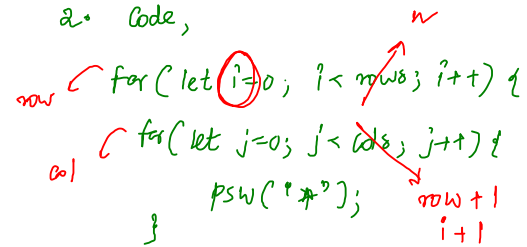
#rows = 4 = n

#cols = are dependent on rows = row + 1

row = 0 → #cols = 1 row = 1 → #cols = 3

row = 1 → #cols = 2 row = 3 → #cols = 4

2. Code,



* going to every row and print the respective pattern

2. Code,

n

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

col `for(let j=0; j<cols; j++){`

`psw("*");`

`}`

`Console.log();`

`}`

row+1

i+1

```
for (let row = 0; row < n; row++) {
  for (let col = 0; col < row + 1; col++) {
    process.stdout.write("*"); // prints in the same line
  }
  console.log(); // move to next line
}
```

n=4

1. i=0, 0<4
   → j=0, 0<1
   1, 1<1 X

2. i=1, 1<4
   → j=0, 0<2
   1, 1<2
   2, 2<2 X

3. i=2, 2<4
   → j=0, 0<3
   1, 1<3
   2, 2<3
   3, 3<3 X

4. i=3, 3<4
   → j=0, 0<4
   1, 1<4
   2, 2<4
   3, 3<4
   4, 4<4 X

5. i=4, 4<4 X

* Staircase :

Eg: n = 4

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

Eg: n = 3

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

1. observe / analyse

	0	1	2	3
0	" "	" "	" "	#
1	" "	" "	#	#
2	" "	#	#	#
3	#	#	#	#

rows

Cols

rows = 4 = n

cols = 4 = n

cols = n

spaces + # hashes = n

spaces = n − # hashes

spaces = n − (row + 1) = n − row − 1

spaces

row = 0 ⇒ n − row − 1
⇒ 4 − 0 − 1 ⇒ (3)

row = 1 ⇒ n − row − 1
⇒ 4 − 1 − 1 ⇒ (2)

row = 2 ⇒ n − row − 1
⇒ 4 − 2 − 1 ⇒ (1)

cols

spaces | # hashes

	# spaces	# hashes
row = 0	3	1
row = 1	2	2
row = 2	1	3
row = 3	0	4

hashes = row + 1

spaces = n − row − 1

2. code;

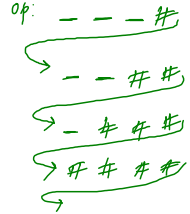
```
for (let row = 0; row < n; row++) {
    for (let space = 0; space < n-row-1; space++) {
        psw (" ");
    }
    for (let hash = 0; hash < row+1; hash++) {
        psw ("#");
    }
    console.log();
}
```

s+h = cols

```
for (let row = 0; row < n; row++) {
  for (let space = 0; space < n - row - 1; space++) {
    process.stdout.write(" ");
  }
  for (let hash = 0; hash < row + 1; hash++) {
    process.stdout.write("#");
  }
  console.log();
}
```

n = 4

op:



1. row = 0, 0 < 4

→ space = 0, 0 < n-row-1
⇒ 0 < 4-0-1
⇒ 0 < 3

→ space = 0, 1 < 3
1, 2 < 3
2, 3 < 3 ✗

→ hash = 0, 0 < row+1
⇒ 0 < 1
1 ⇒ 1 < 1 ✗

2. row = 1, 1 < 4

→ space = 0, 0 < 2
1, 1 < 2
2, 2 < 2 ✗

→ hash = 0, 0 < 2
1, 1 < 2
2, 2 < 2 ✗

3. row = 2, 2 < 4

→ space = 0, 0 < 1
1, 1 < 1 ✗

→ hash = 0, 0 < 3
1, 1 < 3
2, 2 < 3
3, 3 < 3 ✗

4. row = 3, 3 < 4

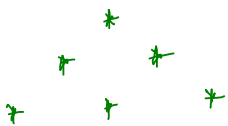
→ space = 0, 0 < 0 ✗

→ hash = 0, 0 < 4
1, 1 < 4
2, 2 < 4
3, 3 < 4
4, 4 < 4 ✗

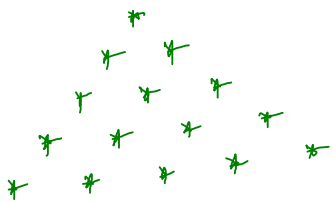
5. row = 4, 4 < 4 ✗

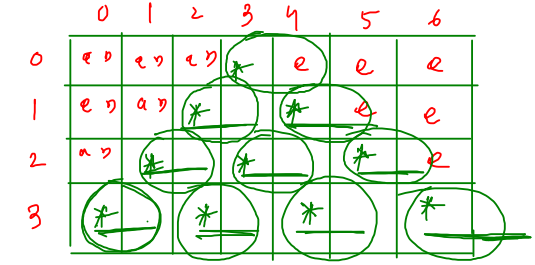
* Star pyramid : (same as staircase but a space difference)

eg: n = 3



eg: n = 5





" * "

↓ idea

" * _ "

rows = 4 = n

Cols

spaces

stars_space

#spaces = n - row - 1

#stars_sp = row + 1

	# spaces	# stars_space
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

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

	num
row = 0	1
row = 1	2
row = 2	3
row = 3	4

row+1

rows = 4 = n

cols = row + 1

```
for (let row = 0; row < n; row++) {
    let num = row + 1;
    for (let col = 0; col < row + 1; col++) {
        psw (num);
        num--;
    }
    console.log();
}
```

* Character Pattern :

Eg: n = 4

A
B B
C C C
D D D D

Eg: n = 3

A
B B
C C C

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

row =	alphabet	Ascii
0	A	65
1	B	66
2	C	67
3	D	68

65 + row

rows = 4 = n

cols = row + 1

```
for(let row = 0; row < n; row++) {
    const alphabet = String.fromCharCode(65+row);
    for(let col = 0; col < row+1; col++) {
        psw(alphabet);
    }
    console.log();
}
```

* Armstrong Numbers :

Eg : $n = 153$

$$\text{sum of } [\text{digit}]^{\text{no. of digits}} = \text{number}$$

$$\Rightarrow 1^3 + 5^3 + 3^3 = 1 + 25 + 27$$
$$= 153$$

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 : $n = 9474$

$$\Rightarrow 9^4 + 4^4 + 7^4 + 4^4$$
$$= 6561 + 256 + 2401 + 256$$
$$= 9474$$

* print all armstrong [m, n] :

```
for( let num = m; num <= n; num++) {
    Check num is armstrong
}
```

1. no of digits in given number 'n'
(extract digits Technique)

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

1634
↓ /10 cnt = 0 1
163
↓ /10 cnt = 1 2
16
↓ /10 cnt = 2 3
1 → 0 /10 cnt = 3 4

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
↓
0 ✗

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

3. sum == N
↓ yes ↘ NO

```
394   // Write the code here
395   for (let num = m; num <= n; num++) {
396     // 1. Count no.of digits
397     let cnt = 0;
398     while (num != 0) {
399       num = parseInt(num / 10);
400       cnt++;
401     }
402
403     // 2. sum of every digit to power no.of digits
404     let sum = 0;
405     while (num != 0) {
406       const lastDigit = num % 10;
407       sum += lastDigit ** cnt;
408       num = parseInt(num / 10);
409     }
410
411     if (sum == num) {
412       console.log(num);
413     }
414   }
```

temp = num

} change num to temp

temp = num

} change num to temp

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

num = 1634

① cnt = 0 1 2 3 4

1634 → 163 → 16 → 1 → 0

② while (num != 0)

↳ But num = 0 due to the 1st step

↳ hence 2nd loop will not run

③ sum == num ⇒ 0 == 0 ⇒ armstrong

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

④ num++ ⇒ num = 0 1

↳ this leads to Infinite loop