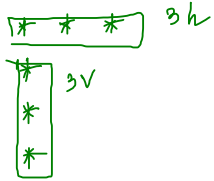
Q: N stars :

fg: n = 3

op:




```
for(let i = 0; i<n; i++) {
    process.stdout.write("*"); - (h)
}
for(let i = 0; i<n; i++) {
    console.log("*"); - (v)
}
```

n = 3

① i = 0
② i = 1
③ i = 2
④ i = 3 < 3 ✗

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

Q: Right Angle Triangle stars :

Eg : n = 4

Op:

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

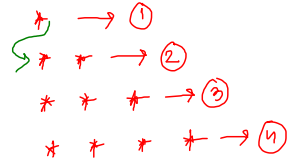
Eg: n = 2

Op:

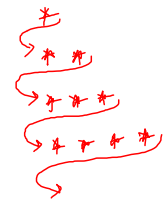
```
*
* *
```

Eg: n = 3

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



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



* If we need to print pattern for n = 1000,
then we need write 1000 for loops, which is impossible

i = 0→1, 0→2, 0→3, 0→4, ... 0→1000
for ① for ② for ③ for ④ for 1000

→ If you observe every loop has same task that is printing '*' based on some number (repetitive task) (for loop itself)

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

i = 1, 2, 3, 4

i start

1000 pushup

```
for ( ) { → 1000 times
    console.log ( pushup 1 ) → num
}
```

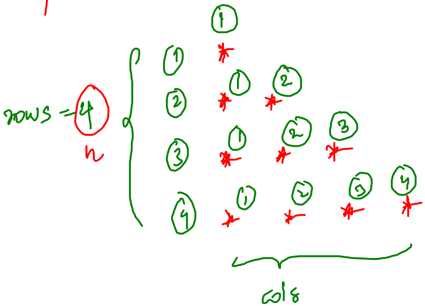
i = 1
j = 0 → 1

i = 2
j = 0 → 2

i = 3
j = 0 → 3

i = 4
j = 0 → 4

n = 4



row = 1 ⇒ cols = 1

2 ⇒ cols = 2

3 ⇒ cols = 3

4 ⇒ cols = 4

* cols = row number

rows = n

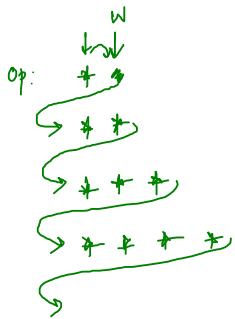
```
for (let i=1; i <= rows n; i++) {        → outer loop helps us do to repeat the inner task

    for (let j=1; j <= cols i; j++) {

        process.stdout.write("*");
    }
}
```

```
readline.question("", (n) => {
  //Write your code here
  const rows = Number(n);
  for (let i = 1; i <= rows; i++) {
    const cols = i;
    for (let j = 1; j <= cols; j++) {
      process.stdout.write("*");
    }
    console.log(); // after printing a row move to new line
  }
  readline.close();
});
```

repeating the task (repeat for n rows)



rows = n = 4

① i = 1
- → j=1, j<=i, j<=1, 1<=1
- → j=2, 2<=1 ✗

② i = 2
- → j=1, 1<=2
- → j=2, 2<=2
- → j=3, 3<=2 ✗

③ i = 3
- → j=1, 1<=3
- → j=2, 2<=3
- → j=3, 3<=3
- → j=4, 4<=3 ✗

④ i = 4
- → j=1, 1<=4
- → j=2, 2<=4
- → j=3, 3<=4
- → j=4, 4<=4
- → j=5, 5<=4 ✗

⑤ i=5, 5<=4 ✗

Q: staircase

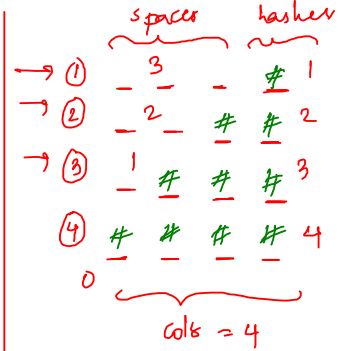
Eg: n = 4

Op:

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

Eg: n = 5

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



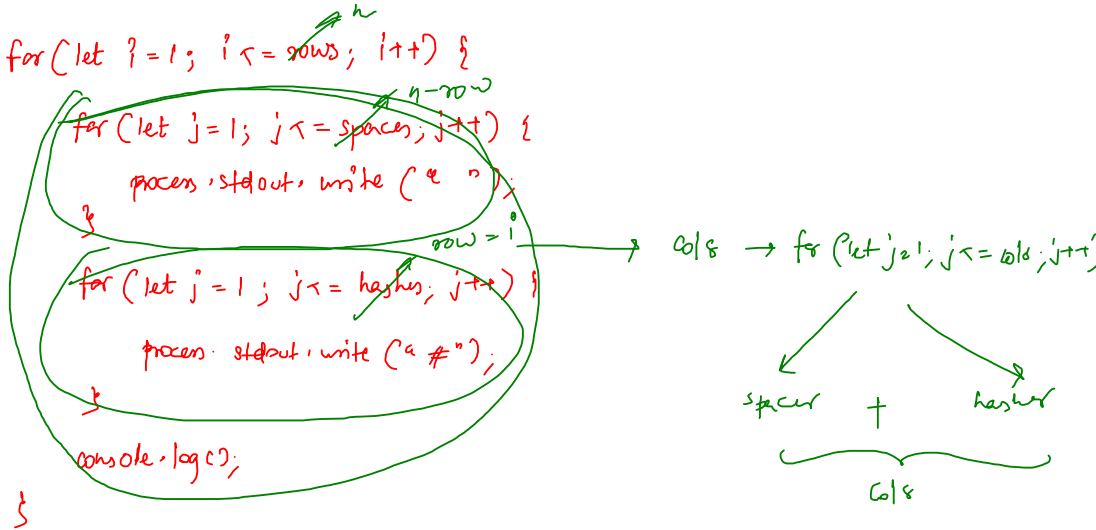
rows = 4 = n

row 1 → 3 spaces, 1 hash
2 → 2 spaces, 2 hash
3 → 1 space, 3 hash
4 → 0 space, 4 hash

★ spaces = n − row
hashes = row

row = 1
spaces = n − row
= 4 − 1 = 3

row = 2
spaces = n − row
= 4 − 2 = 2


for (let i = 1; i <= rows; i++) {
n
for (let j = 1; j <= spaces; j++) {
n-row
process.stdout.write(" ");
}
row = 1
for (let j = 1; j <= hashes; j++) {
process.stdout.write("#");
}
console.log();
}
cols → for (let j=1; j<= cols; j++)
spacer
+
hasher
cols

```
readline.question("", (n) => {
  n = parseInt(n);
  //write code here
  for (let row = 1; row <= n; row++) {
    for (let spaces = 1; spaces <= n - row; spaces++) {
      process.stdout.write(" ");
    }
    for (let hashes = 1; hashes <= row; hashes++) {
      process.stdout.write("#");
    }
    console.log(); // after printing a row move to new line
  }
  readline.close();
});
```

row ka inner

row ka inner

n = 4

① row = 1

→ spaces = 1, 1 < = (4-1), 1 <= 3

→ spaces = 2, 2 < = 3

→ spaces = 3, 3 < = 3

→ spaces = 4, 4 < = 3 ✗

→ hashes = 1, 1 < = 1

→ hashes = 2, 2 < = 1 ✗

② row = 2

→ spaces = 1, 1 < = (4-2), 1 <= 2

→ spaces = 2, 2 < = 2

→ spaces = 3, 3 < = 2 ✗

→ hashes = 1, 1 < = 2

→ hashes = 2, 2 < = 2

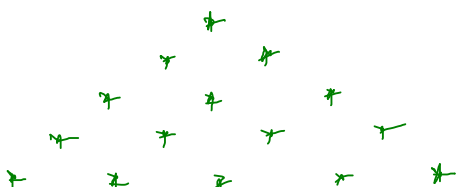
→ hashes = 3, 3 < = 2 ✗

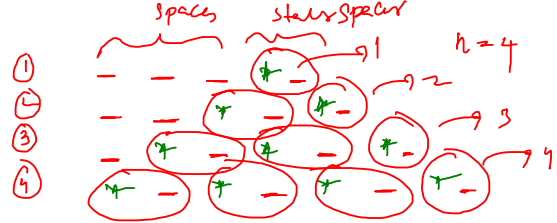
op

Q: star pyramid:

Eg: n = 3

Eg: n = 5





rows = n = 4

row = 1 → spaces = 3 stars = 1

2 → spaces = 2 stars = 2

3 → spaces = 1 stars = 3

4 → spaces = 0 stars = 4

↓

starSpace

* stars = row

spaces = n − row

a ' ' → a * _ '

Q: Number pattern :

Eg: n = 5

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

* psw cannot print numbers it can only print strings.
so convert num → string
~~String(num)~~
num + ""
23 + "" ⇒ "23"
(type coercion)

① 1
② 2 1
③ 3 2 1
④ 4 3 2 1

* rows = 4 = n
* col = row

row = 1 → (1 → 1) 1
2 → (2 → 1) ✗ 1
3 → (3 → 1) ✗ ✗ 1
4 → (4 → 1) ✗ ✗ ✗

* startnum = row

```
for(let row = 1; row <= n; row++) {
    for(let num = row; num >= 1; num--) {
        process.stdout.write(num);
    }
    c|c();
}
```

Character pattern :

eg: n = 4

op:

A

B B

C C C

D D D D

① A

② B B

③ C C C

④ D D D D

rows = n = 4

cols = row

fromCharCode

row = 1 → A (65) → 64+1

2 → B (66) → 64+2

3 → C (67) → 64+3

4 → D (68) → 64+4

ASCII → String/alpha

String.fromCharCode(65)

char of row = 64+row

Q: Armstrong numbers :

Eg: $n = 153$

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

153

$\Rightarrow 1^3 + 5^3 + 3^3$

$= 1 + 125 + 27$

$= 153$

371

$\Rightarrow 3^3 + 7^3 + 1^3$

$= 27 + 343 + 1$

$= 371$

1634

$\Rightarrow 1^4 + 6^4 + 3^4 + 4^4$

$= 1 + 1296 + 81 + 256$

$= 1634$

9474

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

$= 6561 + 256 + 2401 + 256$

$= 9474$

1. no. of digits in 'n'

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

123 4
↓ /10
123
↓ /10
12
↓ /10
1 → 0
/10

ans = ∅ X X ∅ (4)

Sum = 0
sum = sum + (4 ** 4)
= 0 + 4^4
= 0 + 4^4 + (3 ** 4)
= 0 + 4^4 + 3^4 + (2 ** 4)
= 0 + 4^4 + 3^4 + 2^4 + 1^4
= ?

123 (4) % 10 = (4)
↓ /10
123 % 10 = (3)
↓ /10
12 % 10 = (2) /10

0 X
↑ /10
1 % 10 = (1)

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

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

```
if (sum == n) {
    ✓ armstrong
} else {
    X not armstrong
}
```

n = 1234

cnt = 0 1 2 3 4

① Cnt digits,

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

②
```
sum = 0
while (n > 0) {
    digit = n % 10
    sum = sum + (digit ** cnt);
    n = n/10
}
```

① n = 1234 > 0

n = n/10 = 1234/10 = 123

cnt++

② n > 0

123 > 0

n = n/10 = 123/10 = 12

cnt++

③ n > 0

12 > 0

n = n/10 = 1/10 = 1

cnt++

sum = 0

④ n > 0

1 > 0

n = n/10 = 1/10 = 0

cnt++

⑤ n > 0

0 > 0 ✗

① n > 0

0 > 0 ✗

✱ 2nd loop is not running only because n = 0 due 1st loop, To avoid this do not change original num, store it in temp = n and perform operations on temp.