* Right Angled Triangle Stars :

eg: n = 4

op:

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

eg: n = 2

op:

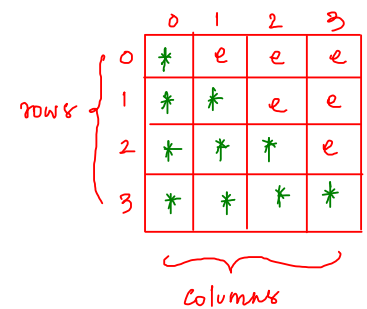
```
*
* *
```

eg: n = 3

op:

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

① observe / Analyze the pattern



eg: N = 4

Rows = 4 = N

Cols (are dependent on rows)

row = 0 → # cols = 1

row = 1 → # Cols = 2

row = 2 → # Cols = 3

row = 3 → # Cols = 4

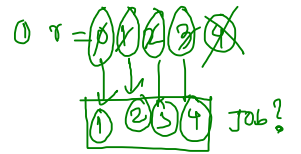
Cols = row + 1

② Code,

rows = 4

⇒ you need to go to every row,

row = 0 → print 1 star
row = 1 → print 2 star
row = 2 → print 3 star




```
for (let r = 0; r < rows; r++) {        → helps to go to every row
    for (let c = 0; c < cols; c++) {    } helps to go to every col w.r.t current row
        psw("*")
    }
    console.log();
}
```

N

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();
});
```

→ Outer loop is going to every row,

$r = 0\ 1\ 2\ 3\ \ldots\ n-1$

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

⇒ $c = 0\ 1\ 2\ 3 \ldots\ r$

Eg: n = 4

① r = 0, 0 < 4 ✓

→ c = 0, 0 < 0+1, 0 < 1 ✓

→ c = 1, 1 < 0+1, 1 < 1 ✗

② r = 1, 1 < 4 ✓

→ c = 0, 0 < 1+1, 0 < 2 ✓

→ c = 1, 1 < 2 ✓

→ c = 2, 2 < 2 ✗

③ r = 2, 2 < 4

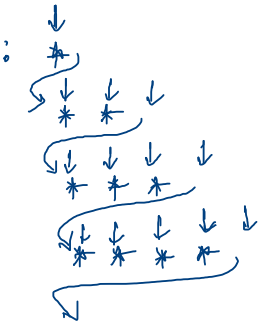
→ c = 0, 0 < 2+1, 0 < 3 ✓

→ c = 1, 1 < 3 ✓

→ c = 2, 2 < 3 ✓

→ c = 3, 3 < 3 ✗

op:



④ r = 3, 3 < 4

→ c = 0, 0 < 3+1, 0 < 4 ✓

→ c = 1, 1 < 4 ✓

→ c = 2, 2 < 4 ✓

→ c = 3, 3 < 4 ✓

→ c = 4, 4 < 4 ✗

⑤ r = 4, 4 < 4 ✗

* Staircase :

Eg: n = 4

Op:

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

Eg: n = 3

Op:

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

① Draw boxes

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

ROWS = 4 = N

no. of hashes = row + 1

no. of spaces = N − hashes

= N − (row + 1)

= N − row − 1

Cols

Spaces → N − row − 1

hashes → row + 1

r	Spaces	hashes (r+1)
r = 0	3	(0+1) 1
r = 1	2	(1+1) 2
r = 2	1	(2+1) 3
r = 3	0	(3+1) 4

no. of cols in a row = no. of spaces + no. of hashes

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

⇒ no. of spaces = N − (row + 1)

= N − row − 1

```
441   for (let r = 0; r < n; r++) {
442     const hashes = r + 1;
443     const spaces = n - hashes;
444     for (let sp = 0; sp < spaces; sp++) {
445       process.stdout.write(" ");
446     }
447     for (let ha = 0; ha < hashes; ha++) {
448       process.stdout.write("#");
449     }
450     console.log();
451   }
```

Eg: $n = 4$

① $r = 0,\ 0 < 4$ ✓

→ hashes = 0+1 = 1

→ spaces = 4−1 = 3

→ $sp = 0,\ 0 < 3$

→ $sp = 1,\ 1 < 3$

→ $sp = 2,\ 2 < 3$

→ $sp = 3,\ 3 < 3$ ✗

→ $ha = 0,\ 0 < 1$

→ $ha = 1,\ 1 < 1$ ✗

② $r = 1,\ 1 < 4$ ✓

→ hashes = 1+1 = 2

→ spaces = 4−2 = 2

→ $sp = 0$ ✓ ✓ $(2 < 2)$

→ $ha = 0$ ✓ ✓ $(2 < 2)$

③ $r = 2,\ 2 < 4$

→ hashes = 2+1 = 3

→ spaces = 4−3 = 1

→ $sp = 0$ ✓ $(1 < 1)$ ✗

→ $ha = 0$ ✗ 2 3 $(3 < 3)$ ✗

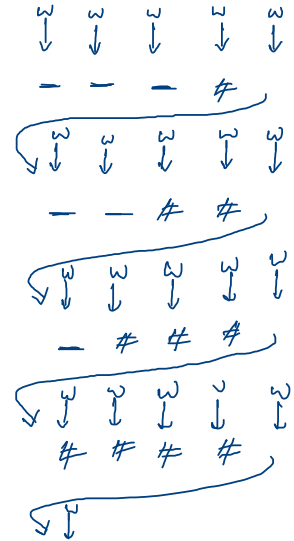
④ $r = 3,\ 3 < 4$

→ hashes = 3+1 = 4

→ spaces = 4−4 = 0

→ $sp = 0\ (0 < 0)$ ✗

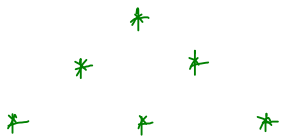
→ $ha = 0$ ✓ ✓ ✓ 4 $(4 < 4)$ ✗



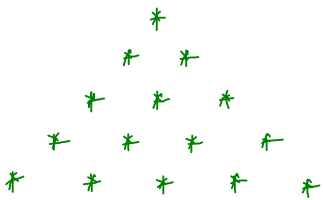
⑤ $r = 4,\ 4 < 4$ ✗

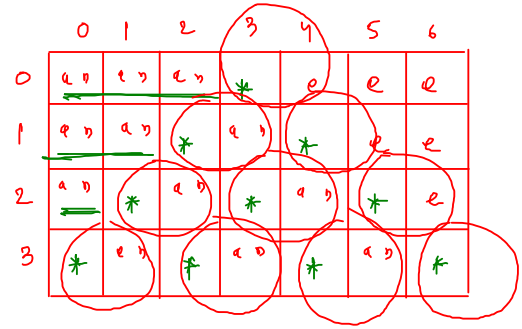
★ Star pyramid :

Eg: n = 3



Eg: n = 5





★ Same as previous problem only change,
" " → " * "

★ After character → empty
before character → " "

Col:

- Spaces
- Character ? (" ") (" * ")

	Spaces	Character
r = 0	3	1
r = 1	2	2
r = 2	1	3
r = 3	0	4

★ 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

psw(1234) → error

psw(String(1234))

⇒ psw("1234") ✓

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

ROWS = 4 = N

cols = # row + 1

r = 0, num = 1 to 1

r = 1, num = 2 to 1

r = 2, num = 3 to 1

r = 3, num = 4 to 1

★ start_num = row + 1

★ 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

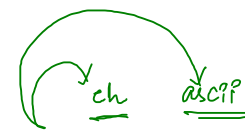
Rows = 4 = N

Cols = row + 1

ascii = 65 + row

String.fromCharCode(ascii)

(65 + row)



row = 0 A ← 65 (65+0)

row = 1 B ← 66 (66+1)

row = 2 C ← 67 (67+2)

row = 3 D ← 68 (65+3)

★ Armstrong Number :

Eg: n = 153

sum of (digit)$^{\text{numDigits}}$ = number

⇒ $1^3 + 5^3 + 3^3 = 1 + 125 + 27$

$= 153$

Eg: n = 371

⇒ $3^3 + 7^3 + 1^3 = 27 + 343 + 1$

$= 371$

Eg: n = 1634

⇒ $1^4 + 6^4 + 3^4 + 4^4$

$= 1 + 1296 + 81 + 256$

$= 1634$

Eg: 9474

⇒ $9^4 + 4^4 + 7^4 + 4^4$

$= 6561 + 256 + 2401 + 256$

$= 9474$

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

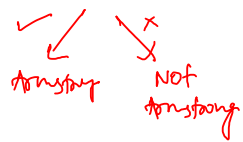
(digit Extraction)

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

1634 ⟶ 163 ⟶ 16 ⟶ 1 ⟶ 0

cnt = 0 1 2 3 4

3. Sum == n



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

1634, $1^4 + 6^4 + 3^4 + 4^4$

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

1634 ⟶ 163 ⟶ 16 ⟶ 1 ⟶ 0

Sum = 0

$+4^4 + 3^4 + 6^4 + 1^4$

* Print Armstrong B/w [m, n] :

m = 20, n = 10,000

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

    /* Check Armstrong
       Code
    */

}
```

num

20 → check → yes → print(20)

21 → check → yes → print(21)

22 → check → yes → print(22)

⋮

10,000 → check → yes → (print 23)

```
514  for (let num = m; num <= n; num++) {
515    // 1. Find no.of digits
516    let cnt = 0;
517    while (num != 0) {
518      num = parseInt(num / 10);
519      cnt++;
520    }
521
522    // 2. Find sum of (digit) ^ numDigits
523    let sum = 0;
524    while (num != 0) {
525      const lastDigit = num % 10;
526      num = parseInt(num / 10);
527      sum += lastDigit ** cnt;
528    }
529
530    // 3. check armstrong
531    if (sum == num) {
532      console.log(num);
533    }
534  }
```

let temp = num

num → temp

temp = num

num → temp

m = 1634, n = 1634 [1634, 1634]

num = 1634

① 1634 → 163 → 16 → 1 → 0 *

cnt = 0 1 2 3 4

② while (num != 0) → 0 != 0 → false

here we need num = 1634

↳ This loop will not run because num already became '0' due to the 1st loop.

* If in any problem you need any original variable multiple do not change it directly instead store it in other variable (temp) and use that, do not touch original (num).