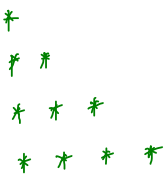
Q2: Right Angle Triangle Stars :

eg: n = 4

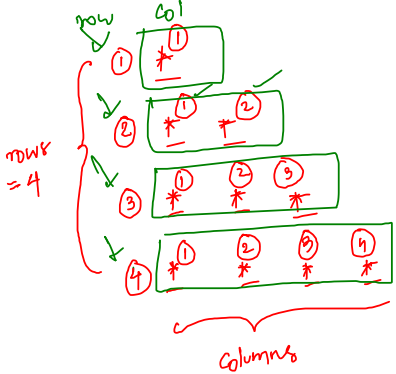
op:



n = 1

n = 2

n = 3



row = 1 ⇒ cols = 1
2 ⇒ cols = 2
3 ⇒ cols = 3
4 ⇒ cols = 4

* cols = row number
rows = n

n = 10 ⇒ 10 loops
n = 1000 ⇒ 1000 loops

→ i <= 1

→ i <= 2

→

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

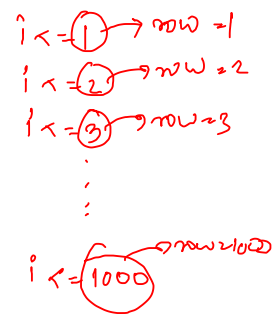
→

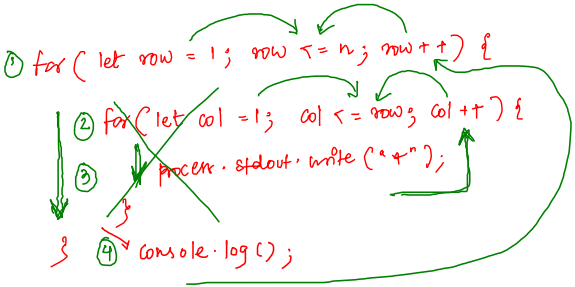
```
for (let i = 1; i <= 4; i++) {
    process.stdout.write("*")
}
```

* coding 1000 loops is impossible,
if you observe every loop has same task that is printing "*" based on no. of cols.

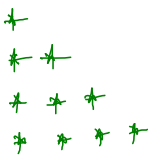
```
for( let row = 1; row <= n; row++) {
    for(let col = 1; col <= row; col++) {
        process.stdout.write("*");
    }
}
```

our repetitive task is for loop itself





op:



n = 4

① row = 1

row <= n ⇒ 1 <= 4

→ col = 1

col <= row ⇒ 1 <= 1

col++ ⇒ col = 2

→ col = 2

col <= row ⇒ 2 <= 1 ✗

row++ ⇒ row = 2

② row <= n ⇒ 2 <= 4

→ col = 1

col <= row ⇒ 1 <= 2

col++ ⇒ col = 2

→ col <= row ⇒ 2 <= 2

col++ ⇒ col = 3

→ col <= row ⇒ 3 <= 2 ✗

row++ ⇒ row = 3

③ row <= n

⇒ 3 <= 4

→ col = 1

col <= row

⇒ 1 <= 3

col++ ⇒ 2

→ col <= row ⇒ 2 <= 3

col++ ⇒ 3

→ col <= row ⇒ 3 <= 3

col++ ⇒ 4

→ col <= row ⇒ 4 <= 3 ✗

row++ ⇒ 4

Q: Staircase :

eg: n = 4

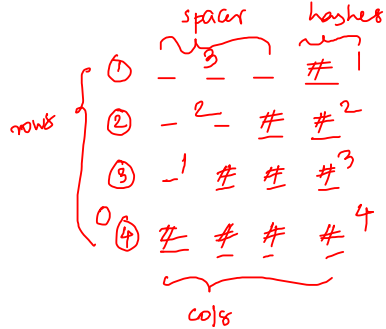
Op:

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

eg: n = 2

Op:

```
 #
# #
```



rows = n

cols = n

row = 1 → 3 spaces, 1 hash

2 → 2 spaces, 2 hashes

3 → 1 space, 3 hashes

4 → 0 space, 4 hashes

★ hashes = row

★ spaces = n − row

row = 1 → n − row

4 − 1 = 3

2 → n − row

4 − 2 = 2

3 → n − row

4 − 3 = 1

4 → n − row

4 − 4 = 0

* go to every row,
form the pattern for that row

```
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();
}
```

Reverse staircase:

n = 4 :

# # # #

# # #

# #

#

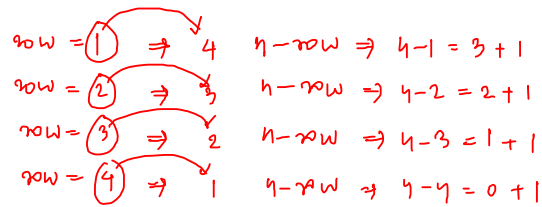
n = 3 :

# # #

# #

#

rows = n



* hashes = n − row + 1

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

row = n row >= 1 row--;

for (let ~~row = 1;~~ ~~row <= n;~~ ~~row++~~) {

    for (let col = 1; col <= row; col++) {

        process.stdout.write("*");

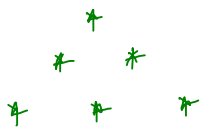
    }

}

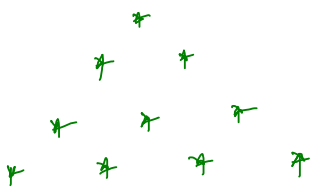
* run outer loop in backward direction for reversed patterns.

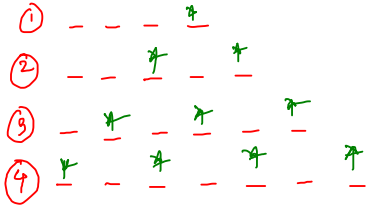
Q: star pyramid:

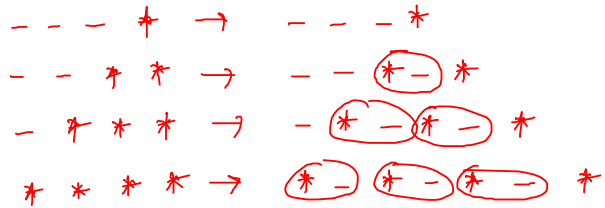
eg: n = 3



eg: n = 4







* Instead of "*" → "*_"

Q: Number pattern :

Eg: n = 5

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

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

* startNum = row

→ process.stdout.write( ) accept strings

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

due to type coercion this will be converted to string.

Q: Armstrong Numbers :

n = 153

* Sum of (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 $\Rightarrow 1^4 + 6^4 + 3^4 + 4^4$

$\Rightarrow (1 + 1296 + 81 + 256)$

$\Rightarrow 1634$

9474

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

$\Rightarrow 6561 + 256 + 2401 + 256$

$\Rightarrow 9474$

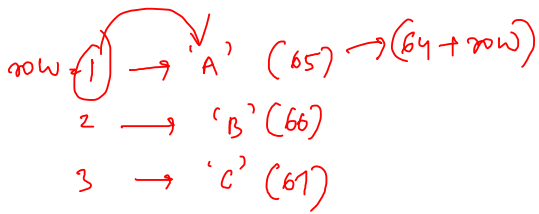
Sum of (digit)$^{\text{num of digits}}$

①
```
cnt = 0
while (n != 0) {
    n = n/10;
    cnt++;
}
```

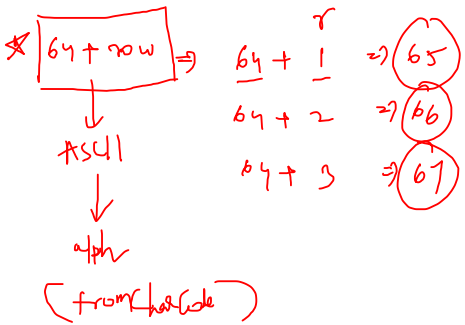
```
Sum = 0
while (n != 0) {
    digit = n % 10;
    sum = sum + (digit ** cnt);
    n = n/10;
}
```

```
if (givenNumber == sum) {
        Console.log (given Number);
}
```

Character pattern :



① A
② B B
③ C C C
④ D D D D



rows = n

cols = row

alph = String.fromCharCode (64 + row);

* ASCII :

for every letter on your keyboard there is some code number associated with it → ASCII value.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

'A' → 65

'B' → 66

'C' → 67

⋮

'Z' → 90

(65 → 90)

'a' → 97

'b' → 98

'c' → 99

⋮

'z' → 122

(97 → 122)

→ given ASCII how to get alphabet → string.fromCharCode(ascii)

given alphabet how to get ASCII → "A".charCodeAt(0);

"B".charCodeAt(0);