★ Functions :

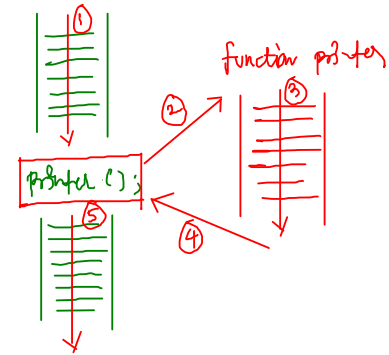
→ DRY : Do not repeat yourself
write code once and use it multiple times.

```
423  function printer() {
424      console.log("Hi I am Anurag");
425      console.log("I work at AccioJob");
426      console.log("I love teaching");
427  }
428
429  printer(); // calling a function
```

① functions can take inputs in form of parameters / arguments.

② functions can provide response / output back to the caller using a return statement.

★ when the function is called all the code inside func is executed after that only remaining code followed by function call is executed.

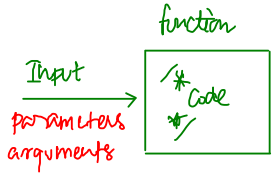


★ function are like variables, functions store code but variables store values. (Data)

type of printer → function

printer → all code

1



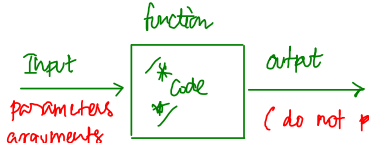
eg: Adding two numbers

```
436  function add(a, b) {
437    console.log(a + b);
438  }
439
440  add(2, 3);
441  add(10, 20);
442  add(100, 150);
```

(a, b) are variables created in the function. You can any choice of parameters.

440. add(2,3)
436. a=2, b=3
437. a+b => 5
441. add(10,20)
436. a=10, b=20
437. a+b => 30
442. add(100,150)
436. a=100, b=150
437. 100+150 => 250

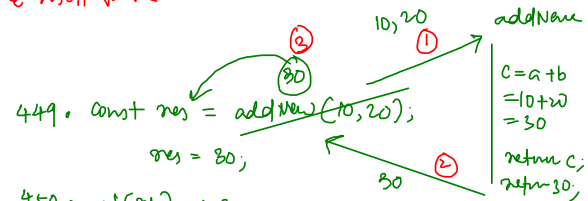
2



(do not print inside function but send the result to the caller)

★ return statement terminates the function. That means all the code is ignored and move back to the caller.

```
444  function addNew(a, b) {
445    const c = a + b;
446    return c;
447  }
448
449  const res = addNew(10, 20);
450  console.log(res);
451
452  addNew(100, 150);
```



(you are not storing the result)

```
454  function printer() {
455    console.log("Hi I am Anurag");
456    console.log("I work at Acciojob");
457    console.log("I love teaching");
458  }
459
460  const ans = printer();
461  console.log(ans);
```

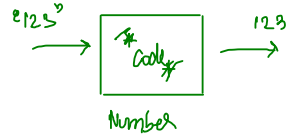
undefined

undefined

return;

★ JS automatically inserts a return; statement at the end of function.

→ const a = Number("123");



we know what ip to give and what op to expect. But we have no idea on the code inside Number();

(ABSTRACTION)

→ hiding implementation details.

* TV Remote,

① vol ⇒ volume increases

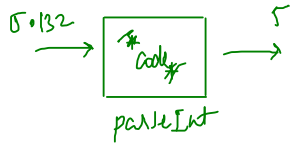
do you know how sensors b/w TV and remote work?

(physics) ↘ NO

ip: click
op: vol ↑

But implementation is abstracted

→ const b = parseInt("5.132")



→ Console.log("...")

Math.max(1,2,3,4) → 4

Math.min(1,2,3,4) → 1

#1 :

```
readline.question("", (line) => {
  let [m, n] = line.split(" ").map(Number);

  // Write the code here
  for (let num = m; num <= n; num++) {
    // 1. Find no.of digits
    let temp = num;
    let cnt = 0;
    while (temp != 0) {
      temp = parseInt(temp / 10);
      cnt++;
    }

    // 2. sum of (digit ^ numDigits)
    temp = num;
    let sum = 0;
    while (temp != 0) {
      const lastDigit = temp % 10;
      temp = parseInt(temp / 10);
      sum += lastDigit ** cnt;
    }

    // 3. If armstrong print that number
    if (sum == num) {
      console.log(num);
    }
  }

  readline.close();
});
```

extract to separate function

Modular way of writing the code.

```
function cntDigits(num) {
  let temp = num;
  let cnt = 0;
  while (temp != 0) {
    temp = parseInt(temp / 10);
    cnt++;
  }
  return cnt;
}

function checkArmstrong(num) {
  const cnt = cntDigits(num);
  let temp = num;
  let sum = 0;
  while (temp != 0) {
    const lastDigit = temp % 10;
    temp = parseInt(temp / 10);
    sum += lastDigit ** cnt;
  }

  if (sum == num) {
    return true;
  }

  return false;
}
```

```
readline.question("", (line) => {
  let [m, n] = line.split(" ").map(Number);

  // Write the code here
  for (let num = m; num <= n; num++) {
    const check = checkArmstrong(num);
    if (check == true) {
      console.log(num);
    }
  }

  readline.close();
});
```

#2 :

```
// DRY not followed
function calculate_nCr(n, r) {
  // write code here
  let nFact = 1;
  for (let i = 1; i <= n; i++) {
    nFact = nFact * i;
  }

  let rFact = 1;
  for (let i = 1; i <= r; i++) {
    rFact = rFact * i;
  }

  let nMinusRFact = 1;
  for (let i = 1; i <= n - r; i++) {
    nMinusRFact = nMinusRFact * i;
  }

  const nCr = nFact / (rFact * nMinusRFact);
  return nCr;
}
```

= DRY,
= Abstraction
= modular way

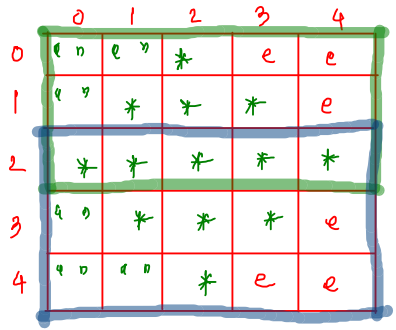
⇓

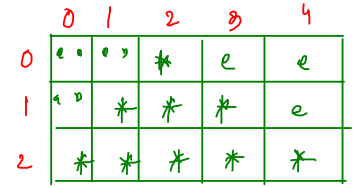
Good code

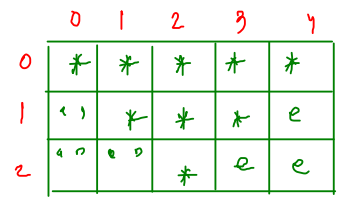
```
// Good code
function factorial(num) {
  let fact = 1;
  for (let i = 1; i <= num; i++) {
    fact = fact * i;
  }
  return fact;
}

function calculate_nCr(n, r) {
  // write code here
  const nFact = factorial(n);
  const rFact = factorial(r);
  const nMinusRFact = factorial(n - r);
  const nCr = nFact / (rFact * nMinusRFact);
  return nCr;
}
```

★ Diamond pattern :

Eg: n = 5







print upper pyramid :

eg: n = 3

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

No. of Rows = 4 = N

No. of spaces = N - r - 1

No. of stars = (2 * r) + 1

2 * 0 + 1 = 1 2 * 2 + 1 = 5

2 * 1 + 1 = 3 2 * 3 + 1 = 7

eg: n = 4

	0	1	2	3	4	5	6
0				*	e	e	e
1			*	*	*	e	e
2		*	*	*	*	*	e
3	*	*	*	*	*	*	*

	No. of spaces	No. of stars
r = 0	3	1
r = 1	2	3
r = 2	1	5
r = 3	0	7

print lower pyramid :

eg: n = 4

3	*	*	*	*	*	*	*
2		*	*	*	*	*	e
1			*	*	*	e	e
0				*		e	e

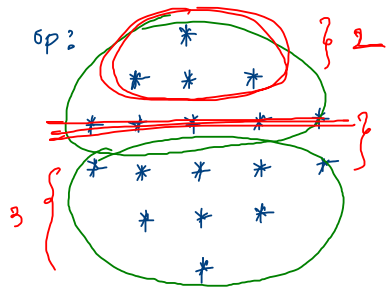
Simply run from

r = n - 1 to r = 0

(Reuse the loop of upper pyramid)

```
464  function upperPyramid(n) {
465    for (let r = 0; r < n; r++) {
466      const spaces = n - r - 1;
467      const stars = 2 * r + 1;
468      for (let sp = 0; sp < spaces; sp++) {
469        process.stdout.write(" ");
470      }
471      for (let st = 0; st < stars; st++) {
472        process.stdout.write("*");
473      }
474      console.log();
475    }
476  }
477
478  function lowerPyramid(n) {
479    for (let r = n - 1; r >= 0; r--) {
480      const spaces = n - r - 1;
481      const stars = 2 * r + 1;
482      for (let sp = 0; sp < spaces; sp++) {
483        process.stdout.write(" ");
484      }
485      for (let st = 0; st < stars; st++) {
486        process.stdout.write("*");
487      }
488      console.log();
489    }
490  }
491
492  upperPyramid(3);
493  lowerPyramid(3);
```

r < n-1



2+1=3

(5/2)+1

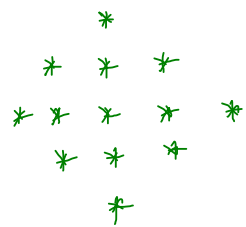
upperPyramid(3)

printDiamond(5)

2+1=3

(5/2)+1

lowerPyramid(3)



* In the upperPyramid function run the loop only n-1 times.

upper(3) ⇒ r=0, r=1, r=2

lower(3) ⇒ r=0, r=1 (one lesser row)