Comparison Operators

- `>` greater than
- `>=` greater than equal to
- `<`
- `<=`

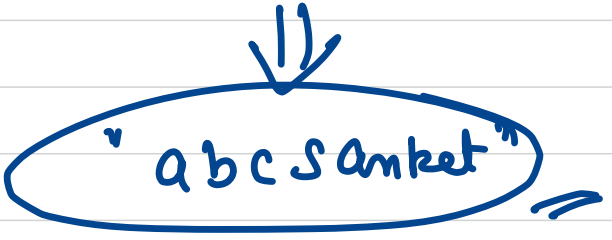
- `!=` → abstract not equals to
- `!==` → strict not equal to
- `==` → abstract equality
- `===` strict equality

↳ (coercion)

↳ worthy

String based operator

"abc" + "sanket"



Bitwise Operators

These operators doesn't directly operate on numbers, but instead operate on binary representation of number.

bitwise and → &
bitwise or → |
bitwise xor → ^
bitwise not → ~

left shift → <<
right shift → >>

5 & 3 = 1

101 & 011

101
011
001 → 1

5 | 4 → 5

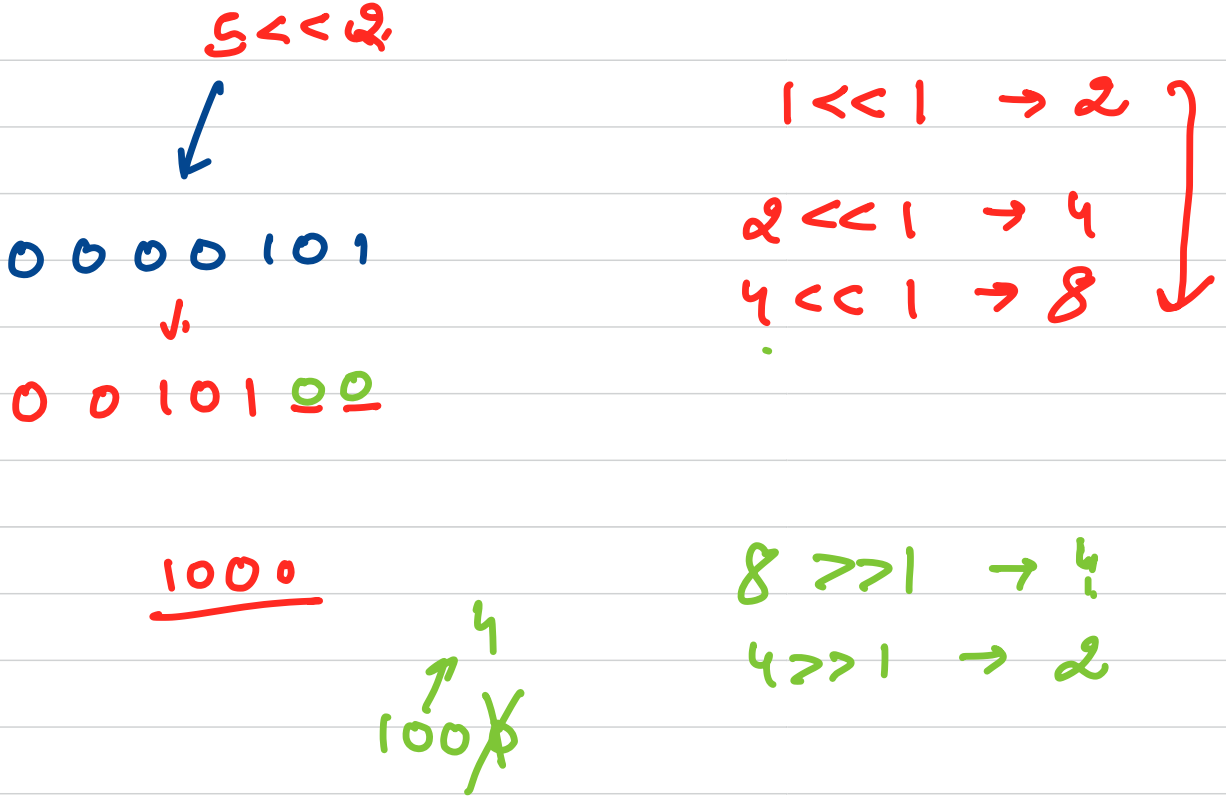
101 & 100

101
100
101

2 << 1

100

000010
000010

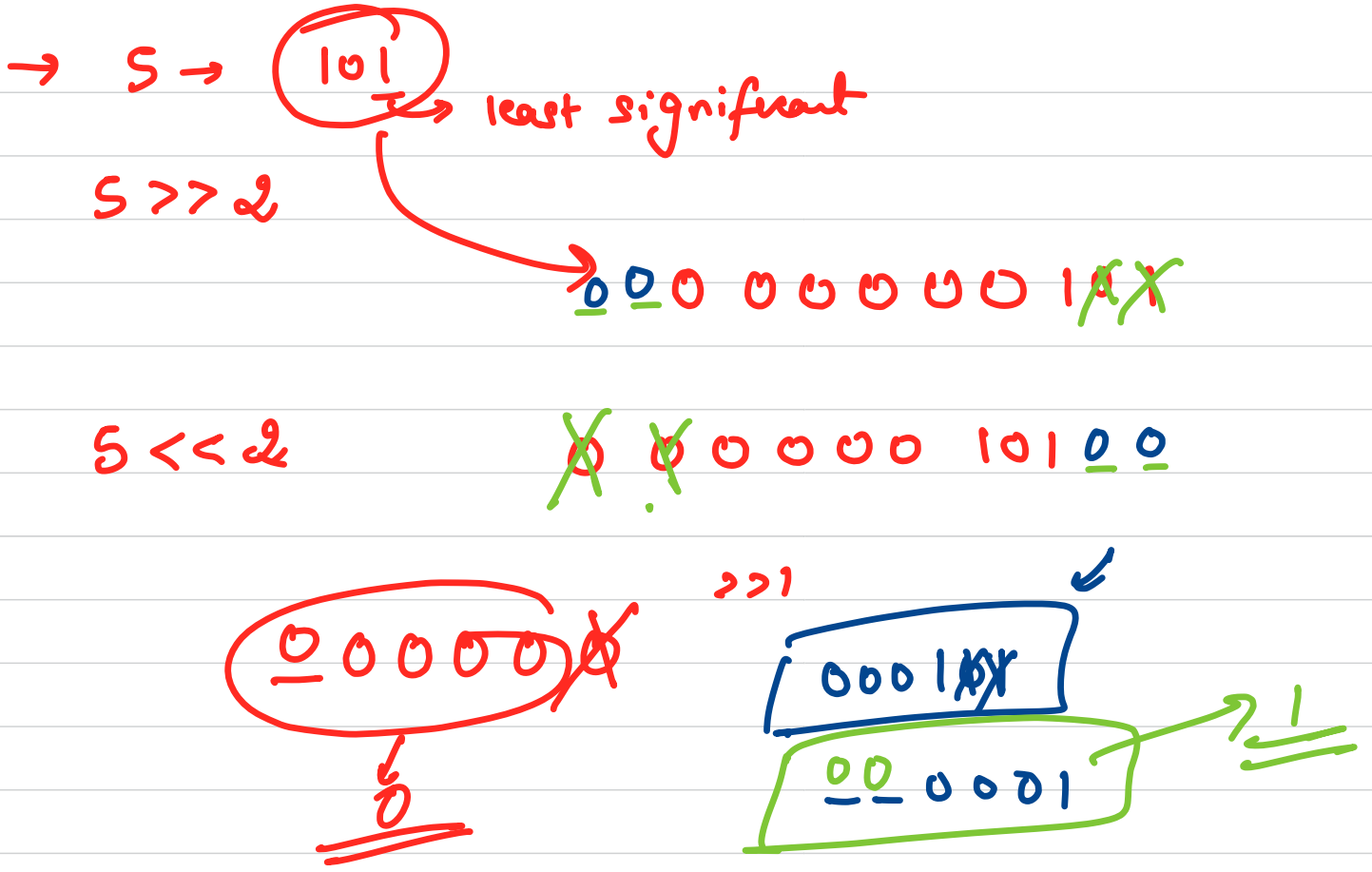

5<<2
0 0 0 0 1 0 1
0 0 1 0 1 0 0
1000
100
1<<1 → 2
2<<1 → 4
4<<1 → 8
8>>1 → 4
4>>1 → 2

5 >> 1

-000101 → 000010

8 >> 3 → 1

↳ 1


→ S →
101
least significant
S >> 2
S << 2
>>1
1
0

 → ternary operator

type of

(condition) ? (expression 1) : (expression 2)

↑ colon

$2^0 \leftarrow 1 \rightarrow 1$

$2 \rightarrow 10$

$4 \rightarrow 100$

$8 \rightarrow 1000$

$16 \rightarrow 10000$

$1 << 1 \rightarrow 2$

$2 << 1 \rightarrow 4$

$2 << 2 \rightarrow 8$

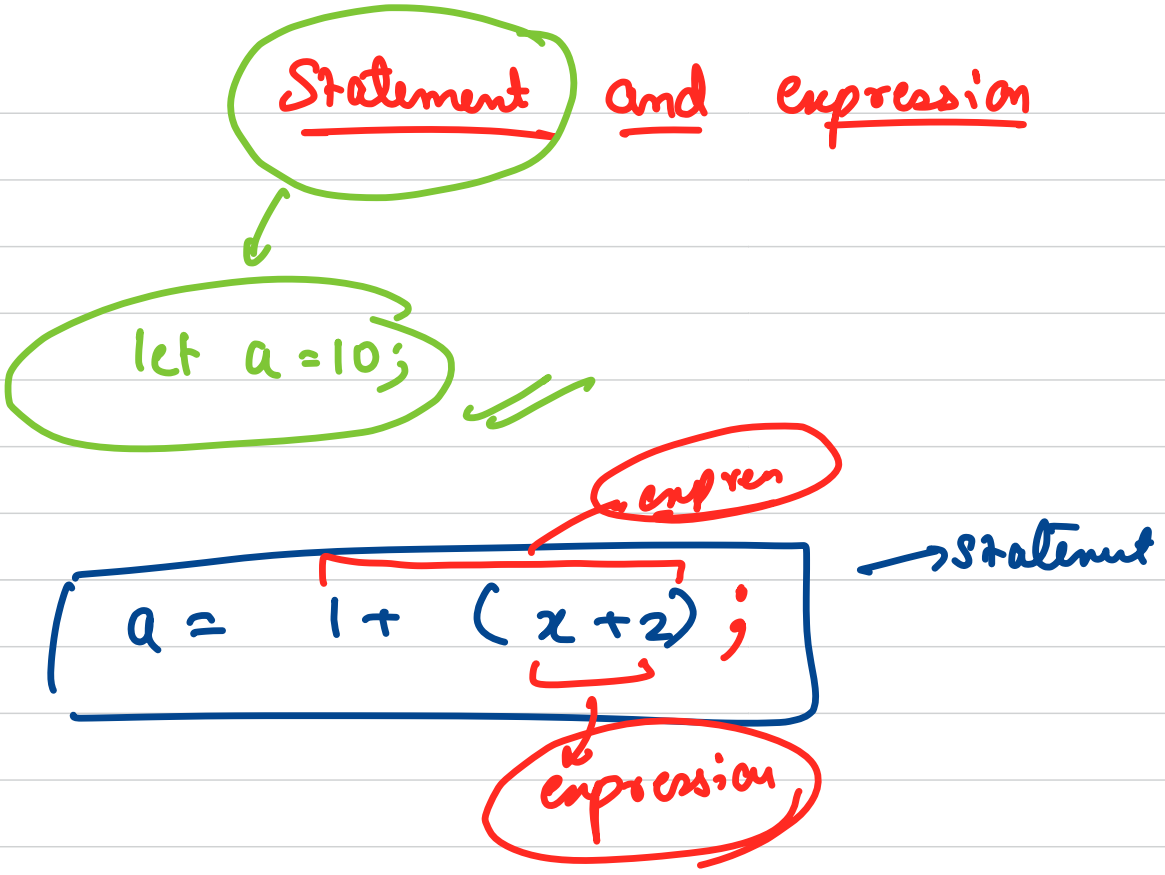
$7 \rightarrow 111$

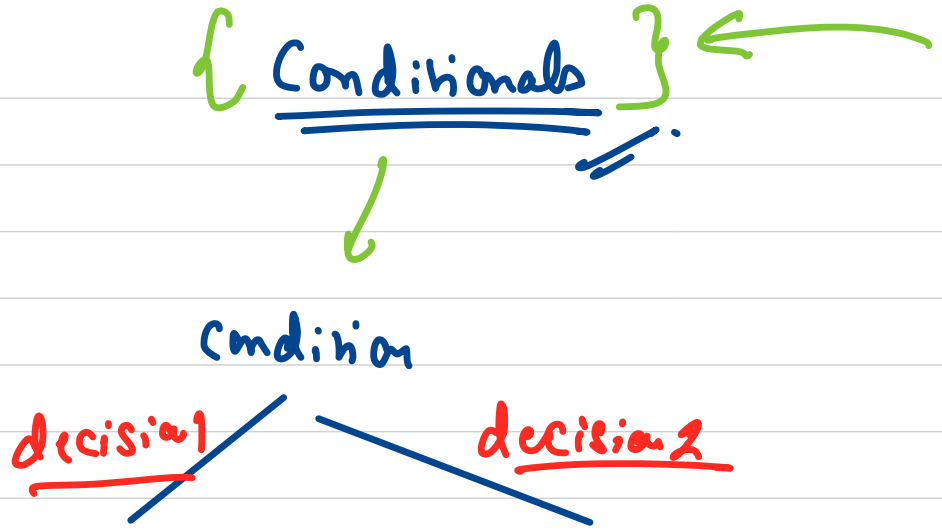
$7 >> 1$

00000111

↳ (3)

Statement and expression



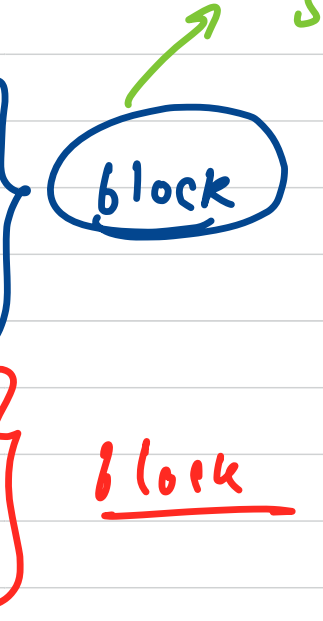

{ Conditionals }
Condition
decision1
decision2

if - else - else if — keywords

```
if (condition) {

} else {

}
```



a bunch of statements is called block

```
if (condition1) {

} else if (condition 2) {

} else if (condition 3) {

} else {

}
```

>
<
>=
<=
==
===
!=
!==

any thing can be used to prepare cond

a **a → a^b

↓

if ((2>1) && "Sanket")

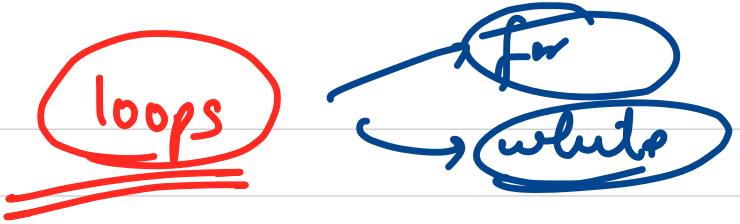
→ true

console.log ((2>1) && "Sanket")

↓

Sanket

→ condition inside if, else if always gets converted to boolean




```
let i =0;

while(i<10) {
    console.log(i)
    i+=1
}
```

```
for(let i=0; i<10; i++)
    console.log(i)
```

↳ break; → whenever we put break, we come out of the nearest loop.

↳ continue;

↳ when we put continue, we again move to the nearest loop for execution.

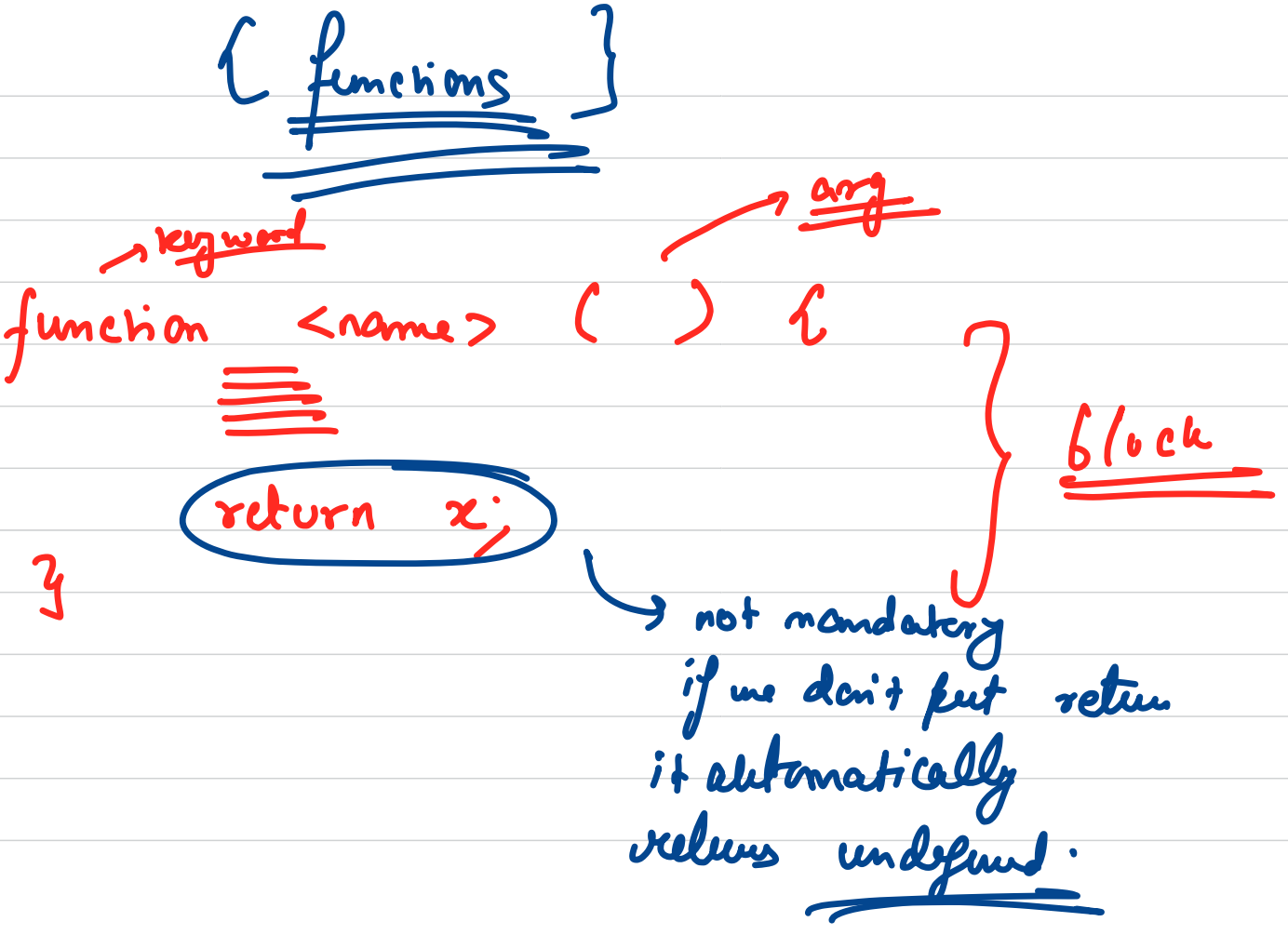
```
for(let i=0; i<5; i++){
    for(let j=0; true; j++){
        if(j==i){
            break;
        }
    }
}
```

```
for(let i = 0; i < 5; i++) {
    let str = "";
    for(let j = 0; true; j++ ) {
        str += "*";
        if(j == i) {
            break;
        }
    }
    console.log(str);
}
```

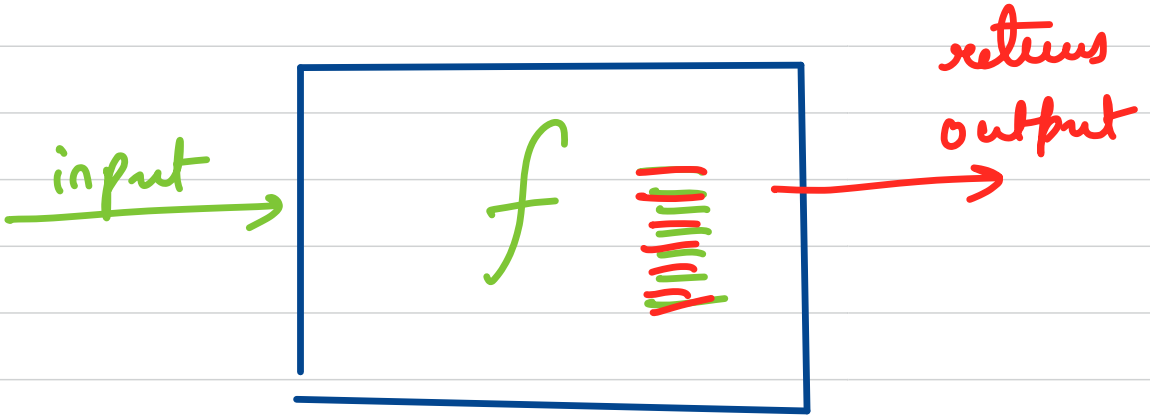
i:0 1

j = 0 1

Str =


{ functions }
keyword
arg
function <name> ( ) {
return x;
}
block
not mandatory
if we don't put return
it automatically
returns undefined.

Scopes, closures, hoisting - - - -



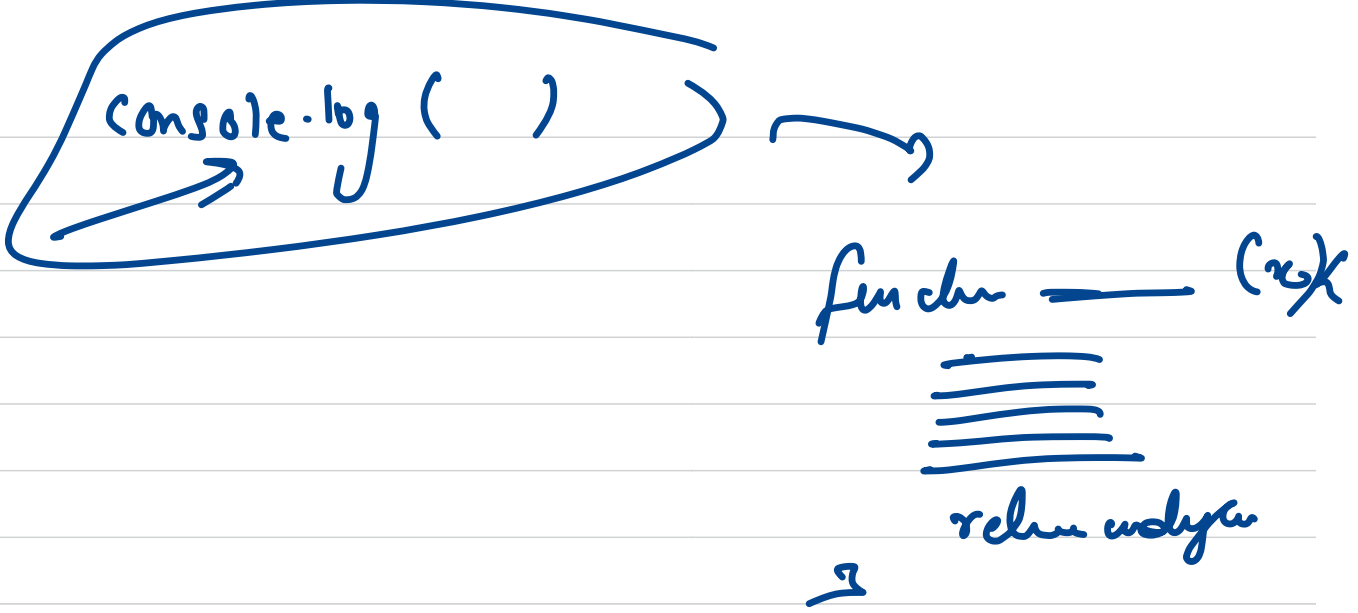

```
function print(x) {
    x += 10
    return x;
}

let result = print(10)
```

→ return keyword returns an output out of the function & immediately stops the func^ execution:

↳ Console.log(10)

undefined


console.log ( )
function (x){
return x+y
}

undefined

```
console.log( console.log(10) );
```

//

```
console.log(undefined);
```

10

undefined

unary operator

x++

++x

```
let x =10
let c = x++;
let y = x++;
console.log (x, y, c)
```

```
let x=10;
let c= ++x;
let y = ++x;
```