

① Bitwise and → &

$a \& b$

HW

$5 \& 6$

↓

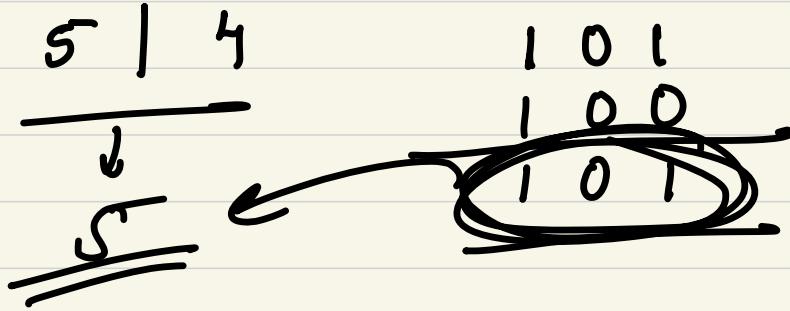
4

$101$

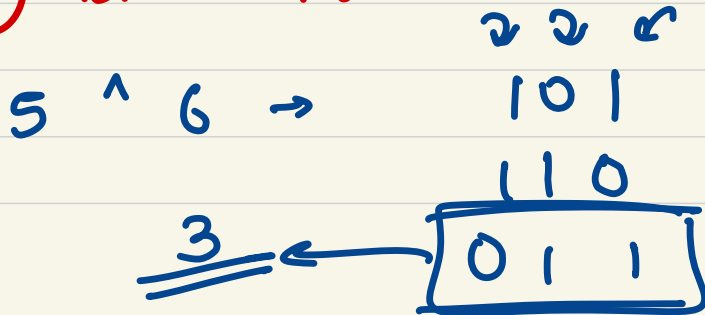
$\& 110$

~~$100$~~

(2) Bitwise OR  $\rightarrow$  |   
↘ single pipe



(3) Bitwise XOR  $\rightarrow$  ^




a	xor	b	
0		0	$\rightarrow$ 0
0		1	$\rightarrow$ 1
1		0	$\rightarrow$ 1
1		1	$\rightarrow$ 0

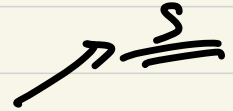
left Shift ( $\ll$ )

Suppose we have an  
8 bit machine


2  
 $5 \ll 3$



8

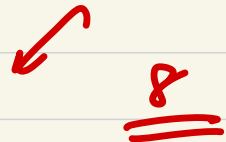


0 0 1 0 1 0 0 0



$1 \ll 3$

0 0 0 0 1 0 0 0



$$1 << \textcircled{x} \rightarrow \underline{\underline{2^x}}$$

↓  
any no

$$1 << 4 \rightarrow \underline{\underline{2^4}}$$

Right shift ( $>>$ )

$$\underline{\underline{27}} \rightarrow 27 >> 2$$

↙ .b.

0   0   0   0   0   1   1   0

$$x \gg 1 \quad \rightarrow \quad \left\lfloor \frac{x}{2} \right\rfloor \rightarrow \underline{\underline{\text{floor}}}$$

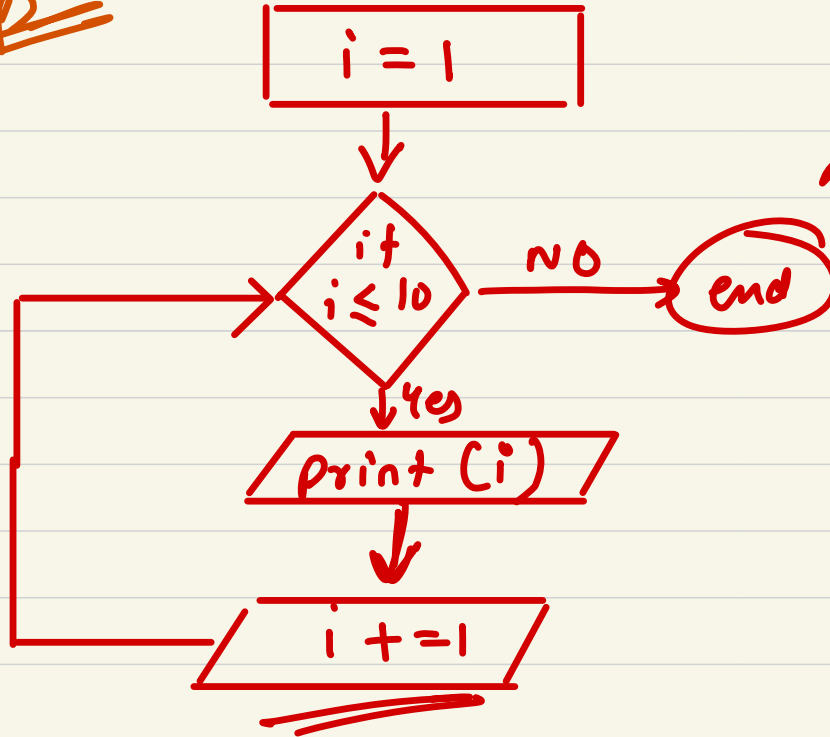
In programming we might have to do a task again & again.

(Loops)

These help us to do a task multiple times

print number from 1 to 10

loops



$i = 1, 2, 3, \dots, 9, 10$

1  
2  
3  
⋮  
8  
9  
10





Stay here.

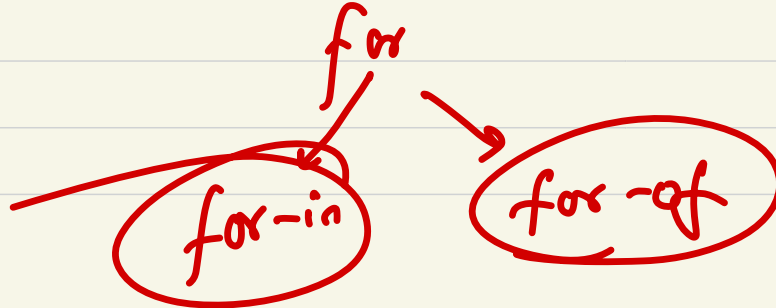
→ A loop which has the condition always true is infinite loop.

for loop

keyword

for(let i=1 ; i ≤ 10 ; i+=1) {  
 console.log (i);

}



do-while

→ exit controlled

keyword

do

{

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

} do-while  
block

} while(condition);

↳ unary operator

++

→ increment by 1

— —

---

—————

—————

let  $a = 1;$

```
let a=1;
console.log(a++);
console.log(a)
```

$$\frac{2}{7} \cdot a$$

2

let  $a=1$

```
console.log(++a);  
console.log(a)
```

2  
2

$$\left[ \frac{2}{1} \right]_a$$

break ;



break terminates the  
nearest loop immediately

continue ;



Brings you back to  
the nearest loop

~~i = 7~~

```
while ( i <= 10) {  
    if ( i % 7 == 0) {  
        ..... Continue;  
    }  
    console.log(i)  
    i++;  
}
```

function

why do we need functions ? !

→ In soft dev, a lot of times we have to reuse  
our logics.

# DRY → don't repeat yourself



if we repeat our common code impl then  
(maybe by copying) there will be issues.

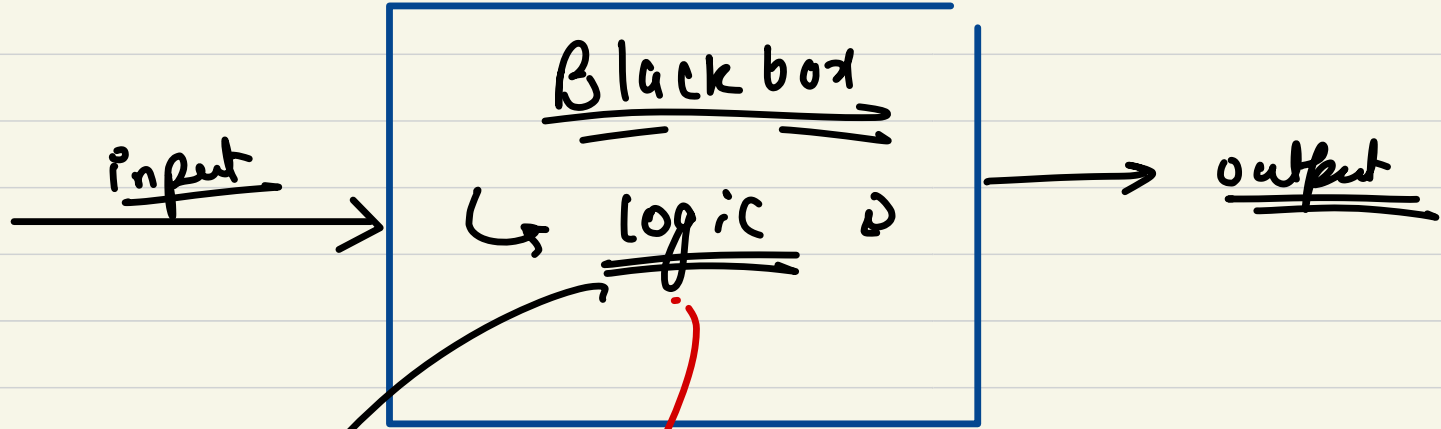
→ lot of code duplication

→ we have to maintain code at multiple places

→ we need to debug the same logic at multiple places

→ if modification is reqd we have to do it

at multiple places.



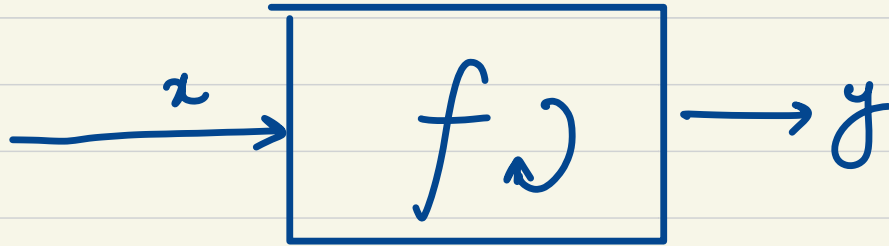
Reusable

we write our logic once &  
wrap it in the blackbox

result input

$$y = f(x)$$

function



In JS we can make our own func<sup>n</sup> &

apart from those we have a lot of inbuilt  
functions as well.

console.log ( )

↓  
object

↘  
function

console = {  
 log : function  
}

How to make your own function??

keyword function nameOfFunction (.....) {

inputs

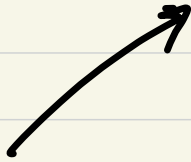
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

}

↑  
function  
definition

# How to use / call a function ?

→ name of function (.....);



↑  
give the inputs

Math.sqrt(100);  
objct      ↘ funcn

defn

function isEvenOrOdd ( x ) {

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

}

parameter

becomes an input

Kind of a placeholder

input

isEvenOrOdd ( 97 ) → function call

↳ argument

arguments are actual values that you provide as input to the func while calling the func

parameter is the placeholder variable that we mention during function definition. This has no value until we call a function and pass argument to it.