Q ⇒ Given a number x, write a function to determine whether the number is a prime number or not ??

```
function isPrime (x) {
    // logic
        :
        :
}
```

Ex → i/p → x = 13
o/p → true

i/p → x = 54
o/p → false

Let's decipher the logic

How to check if a no. is prime or not ??

$\downarrow$

what type of no.'s are prime ??

$\boxed{x = 5}$ $\boxed{x = 11}$

primes $\rightarrow$ are only divisible by 1 or the no. itself

$$\boxed{x = 12}$$

if there is atleast one more no apart from 1 and x that divides x completely then x is non-prime.

$$\begin{tikzcd} 2 \arrow[r, "yes"] & 12 \arrow[r, bend right] & \text{non prime} \end{tikzcd}$$

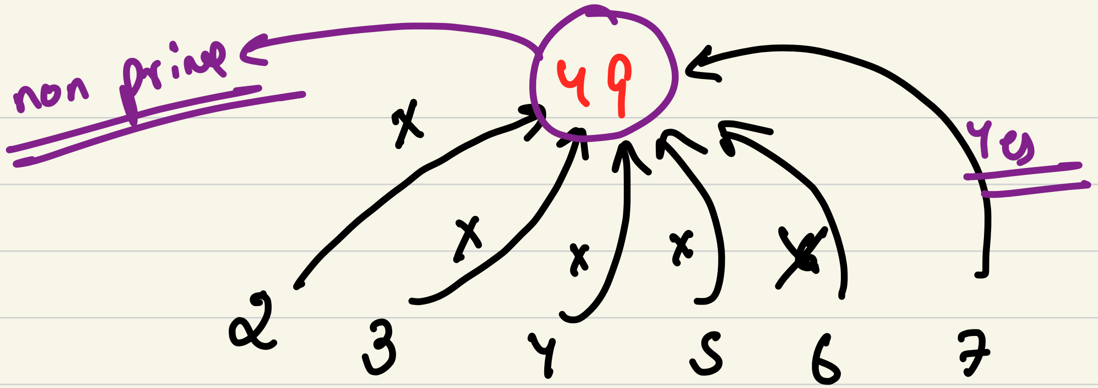
$\boxed{12 \% 2} \rightarrow == 0$??

$$\begin{tikzcd} & 9 \\ 2 \arrow[ur, "x"] & 3 \arrow[u, "\checkmark"'] \end{tikzcd}$$

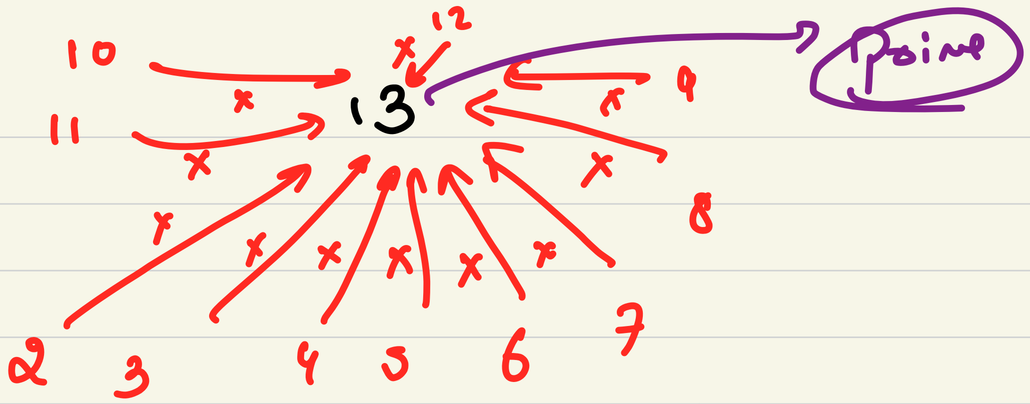
$[2, 8]$

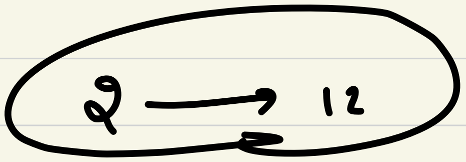
$9 \% 2 \rightarrow 1$ ✗

$9 \% 3 \rightarrow \underline{\underline{0}}$

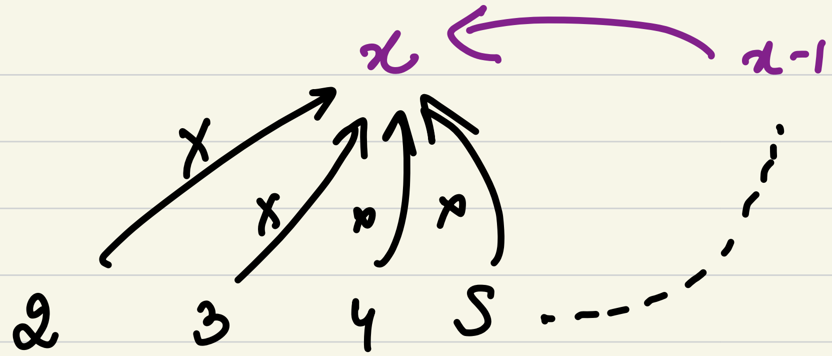


49 % 7 == 0

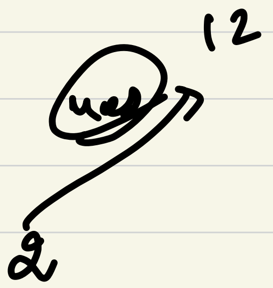




No no.
was able to completely
divide $\boxed{13}$



repition



$[2, x-1]$

first no. we found in this range that can divide x completely we will immediately will return false

$x = 7$

```
for (let i=2 ; i<= x-1 ; i+=1){
        if ( x % i == 0) {
            // non prime
            return false;
        }
}

return true;
```

$i = \cancel{2}\ \cancel{3}\ \cancel{4}\ \cancel{5}\ \cancel{6}$

if we exit the loop without returning false, means no number in range [2, x-1] can divide x, hence x is prime.

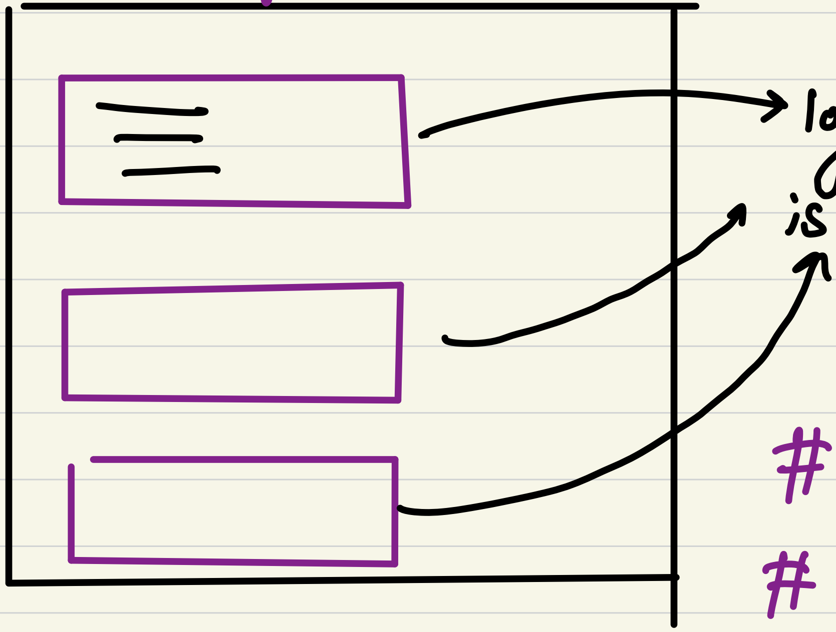
```
function isPrime (x) {
    for (let i=2 ; i<= x-1 ; i+=1) {
            if ( x % i = = 0) {
                    // non prime
                    return false;
            }
    }
    return true ;
}
```

How about a mechanism way which we can avoid this ??

Functions → DRY Principle

Don't Repeat Yourself

index.js



logic to check if the number is +ve or not.

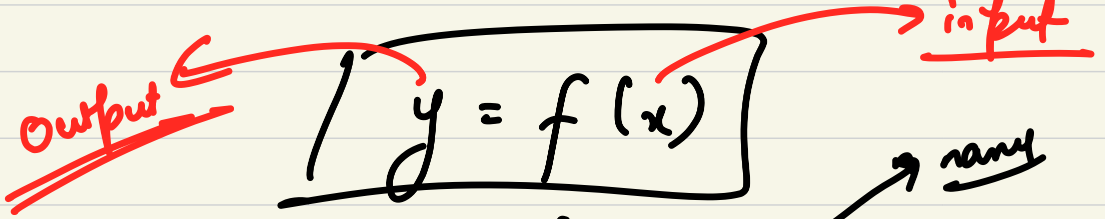
if (x>1) +ve
else -ve

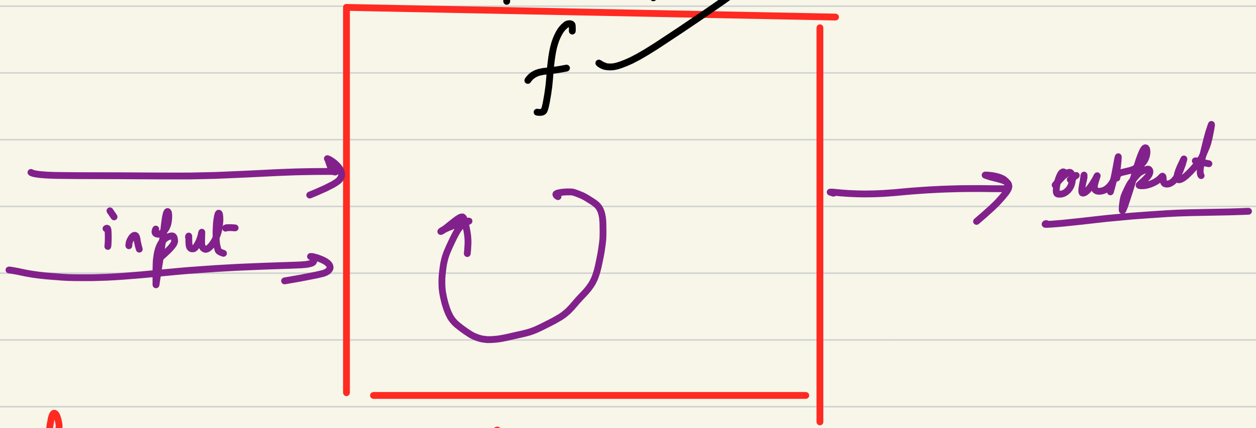
Situation 1 : mistake

situation 2: req changed

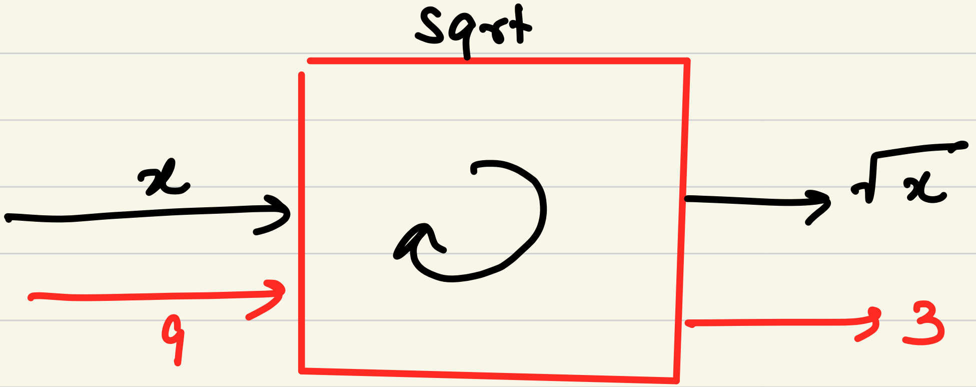
to rectify this we have <u>Functions</u>

$$\boxed{y = f(x)}$$





a function is a black box


sqrt
x
9
$\sqrt{x}$
3

We have 2 choices →

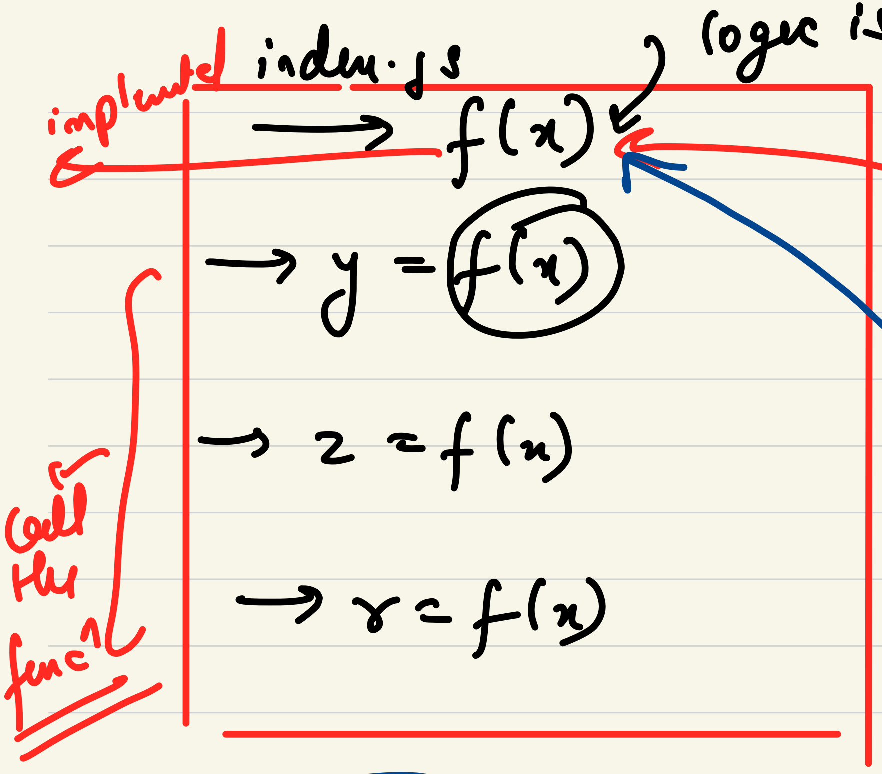
1) We can create our own functions

2) We can consume others functions. (i.e built function) → Ex → Math.sqrt

↳ y = Math.sqrt(9)

$\boxed{3}$

Now, if we use func^n, then we can store our logic inside a function, and doesn't matter how many times you want to use it, You will just call the function.



if there is a mistake then we need to rectify only once.

if we need to change the logic, we need to change it only once.

DRY

functions in JS

this is how we create a new function.

```
function myNewFunction (input1, input2, input3) {
    // → logic

    return "Sanket";
}
```

```
//create a function to check if the no. is even or odd

function isEven(num ) {
        if (num % 2 == 0) {
            return true;
        } else {
            return false;
        }
}
let x = 10;
if( isEven(x)) {
        console.log("Even");
}
```

→ Why we are doing return ?? Can we not use console.log ??

→ What if I don't return anything from func ??

↳ In JS, if you don't manually return something, it automatically returns undefined.

key

What is console.log(' ') ? → does it return something ??

Object

function

Yes → undefined

```
console = {
    log: function () {...}
}
```

x = console.log ("Sanket")

undefined

```
function add(x, y) {
    let c = x+y;
    return c;
}

let a = 10;
let b = 20;
let result = add(a, 30);
Console.log(result);
```

parameters

place where we defend func^n

place where we call the func^n

arguments