Recursion

→ Recursion is not just a computer science topic
It corelates with maths as well.

Mathematically, recursion means a function calling itself.

$$f(x) = f(f(x'))$$

$f(a,b) = a \times f(a,b-1)$

↳ func^n definition

This function calculates a^b

$$a^b = a \times a^{b-1}$$

Ex $f(2,3) \to 2^3 \to 8$

$f(2,4) \to 2^4 \to 16$

$2^4 = 2 \times 2^3$

In computer science, recursion means function calling itself.

```
function fun(x) {
    ...
    ...
    ...
    fun(x-1);
    ...
    ...
}
```

```
function gun(x) {
    console.log(x);
}

function fun(x) {
    ...
    gun(x-1)
    fun(x-1);
    ...
}
```

gun(x-1) → This is not recursion

fun(x-1); → This is recursion

Example-1

Factorial

factorial of any value n ⇒ $n \times (n-1) \times (n-2) \cdots \times 2 \times 1$

↳ n!

$5! = 5 \times 4 \times 3 \times 2 \times 1 \rightarrow 120$

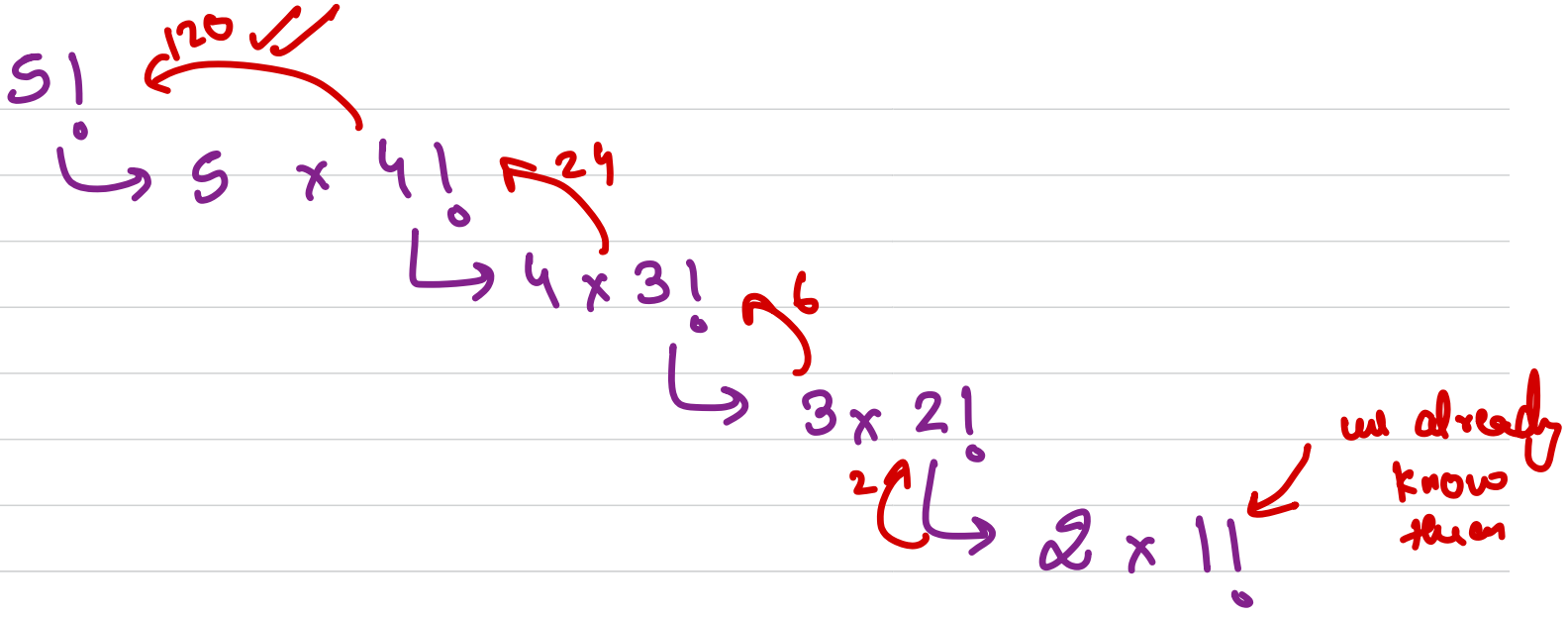
$4! = 4 \times 3 \times 2 \times 1 \rightarrow 24$

$3! = 3 \times 2 \times 1 \rightarrow 6$

$f(n) = n \times f(n-1)$

this function returns $n!$

The mathematical representation of a recursive function is called Recurrence Relation



What is $1!$? $\Rightarrow 1$

Recursion → In recursion, a function calls itself, where it tries to solve a smaller problem and then calc value of a bigger problem

PMI (principal of Mathematical Induction)

Q → What is the Sum of first N natural numbers ??

Ans → $\frac{n \times (n+1)}{2}$

3 step process

1) Base Case → the smallest value for which we can directly verify the ans.

2) Assumption

3) Verification / self task

for $n=1$ $\longrightarrow$ $\frac{1\times(1+1)}{2}$ $\longrightarrow$ is correct

↳ Base case

for some $n=k$ $\longrightarrow$ the formula is correct

i.e. $\frac{k\times(k+1)}{2}$

Let's prove ourselves that formula is correct for $n = k+1$

$$1 + 2 + 3 \text{ ------- } k + k+1$$

$$\rightarrow \frac{k \times (k+1)}{2} + (k+1)$$

$$\frac{k(k+1) + 2(k+1)}{2} \rightarrow \frac{(k+1)(k+2)}{2}$$

By formula → $n=(k+1)$

$$\frac{n\times(n+1)}{2} \rightarrow \frac{(k+1)(k+1+1)}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

Components Of Recursion

1) Base Case
2) Assumption
3) Self work

Write a recursive sol^n for factorial of n

We will write a func^n

$f(n)$

↓

returns n!

Base Case → $(n==1)$ $f(1) \rightarrow 1$

```
if (n==1) return 1
```

Assumption → assume the function works fine for n-1

f(n-1) correctly calcs (n-1)!

Selfwork → f(n) = n x f(n-1)

Power function a^b

$f(a, b)$

↓

returns a^b

Base Case → if $(b == 0)$ then ans is 1

Assumption → assum func^n works fine for $b-1$

i.e. $f(a, b-1)$ is correct.

Self work → $a^b = a \times a^{b-1}$

$f(a,b) \longleftarrow a \times f(a,b-1)$

(correct)

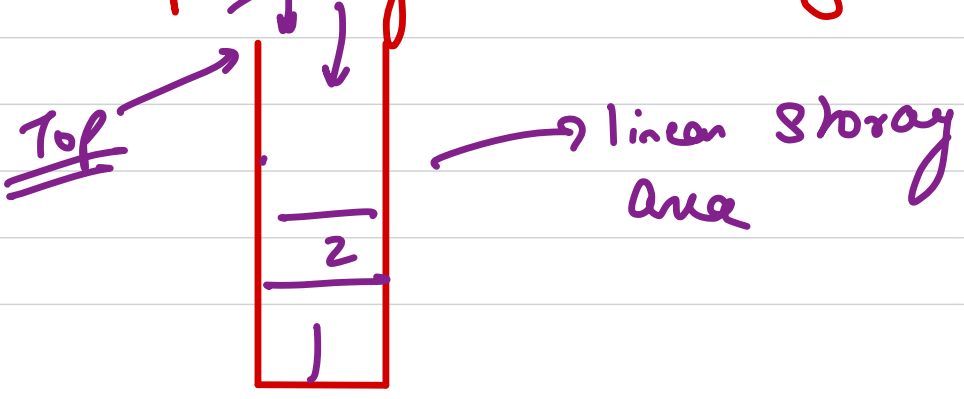
27^{51} $\quad 27 \times (27^{50}) =$

$27^{50} = y$

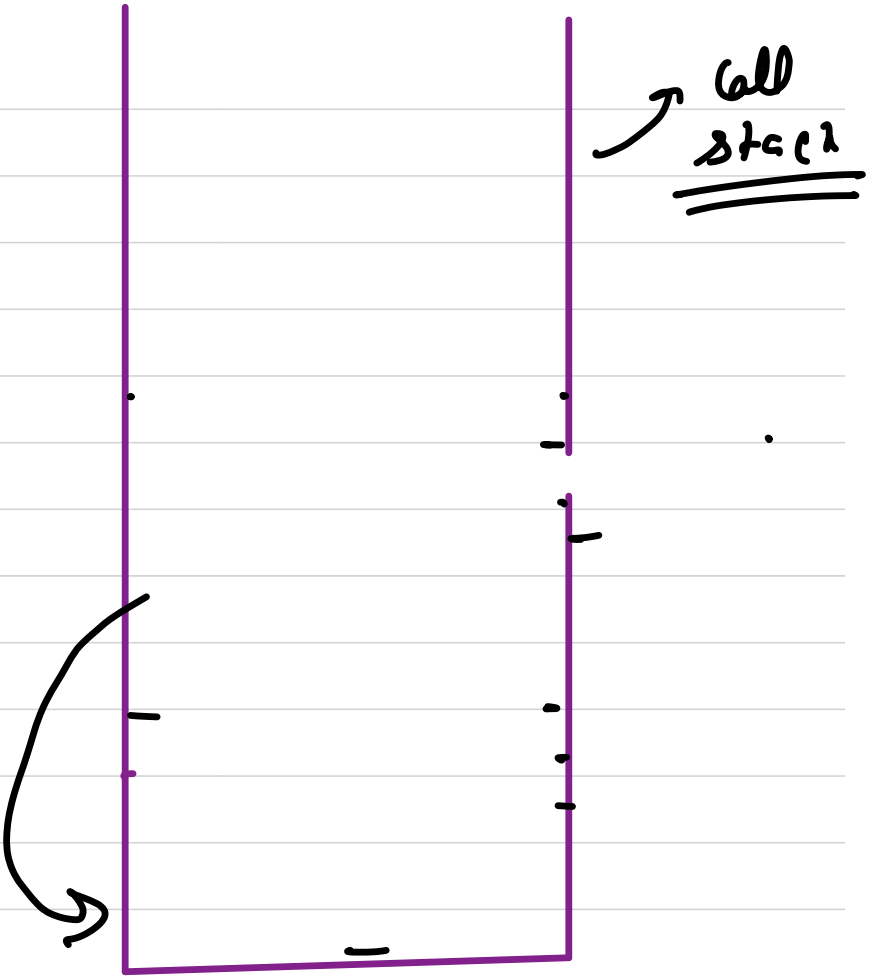
Q: What happens when we call a function??

In our memory, we have a lot of things going on.

One part of the memory is **call stack**



```
1 function fun(x)
2     let a = x + 10;
3     return a;
4 }
5 function gen(y)
6     let b = y + 20
7     let z = fun(b);
8     return z;
9 }
10 console.log(gen(10))
```



Whenever we call a new function it adds a new entry in the stack.

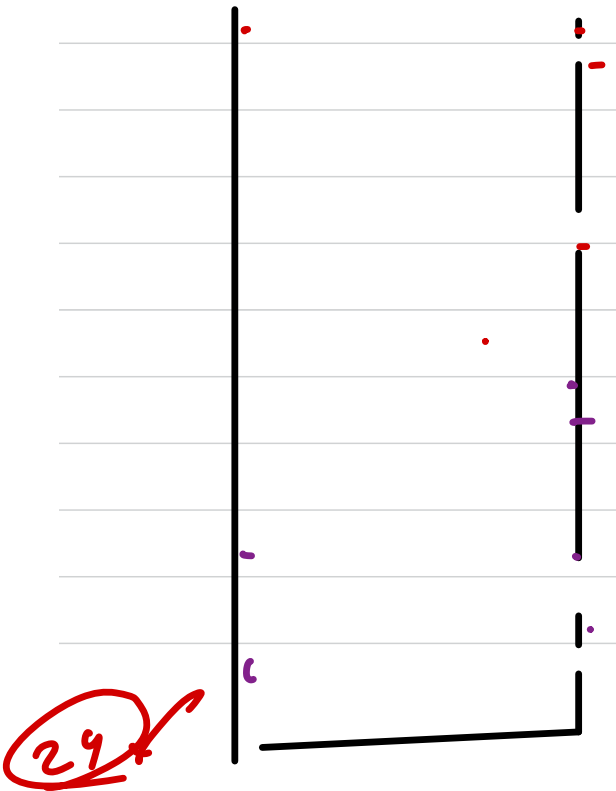
This new entry is called stack trace

Inside a trace we store all the variables and more data like current line of execution of function.

When we hit a return, the stack trace is removed & value is given back to the caller.

```
function fact(n) { // this function should calculate n!
    // Base case
    if(n == 1) return 1;
    let assume = fact(n-1); // that fact of n-1 is correct
    return n*assume;; // self work
}
```

fact (4)



Q. Given a value n, calc n^{th} fibonacci.

→ 0, 1, 1, 2, 3, 5, 8, 13, 21 ....

0^{th}	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}	.....
fib	fib	fib	fib	fib	fib	fib	

n = 7

Ans = 13

Solve it recursively

n^{th} fib

$f(n)$

↓

returns n^{th} fib

Base Cases

$f(0) \rightarrow 0$

$f(1) \rightarrow 1$

} → stacks

Assumption → we assume $f(n-1)$ works

for & $f(n-2)$ works for

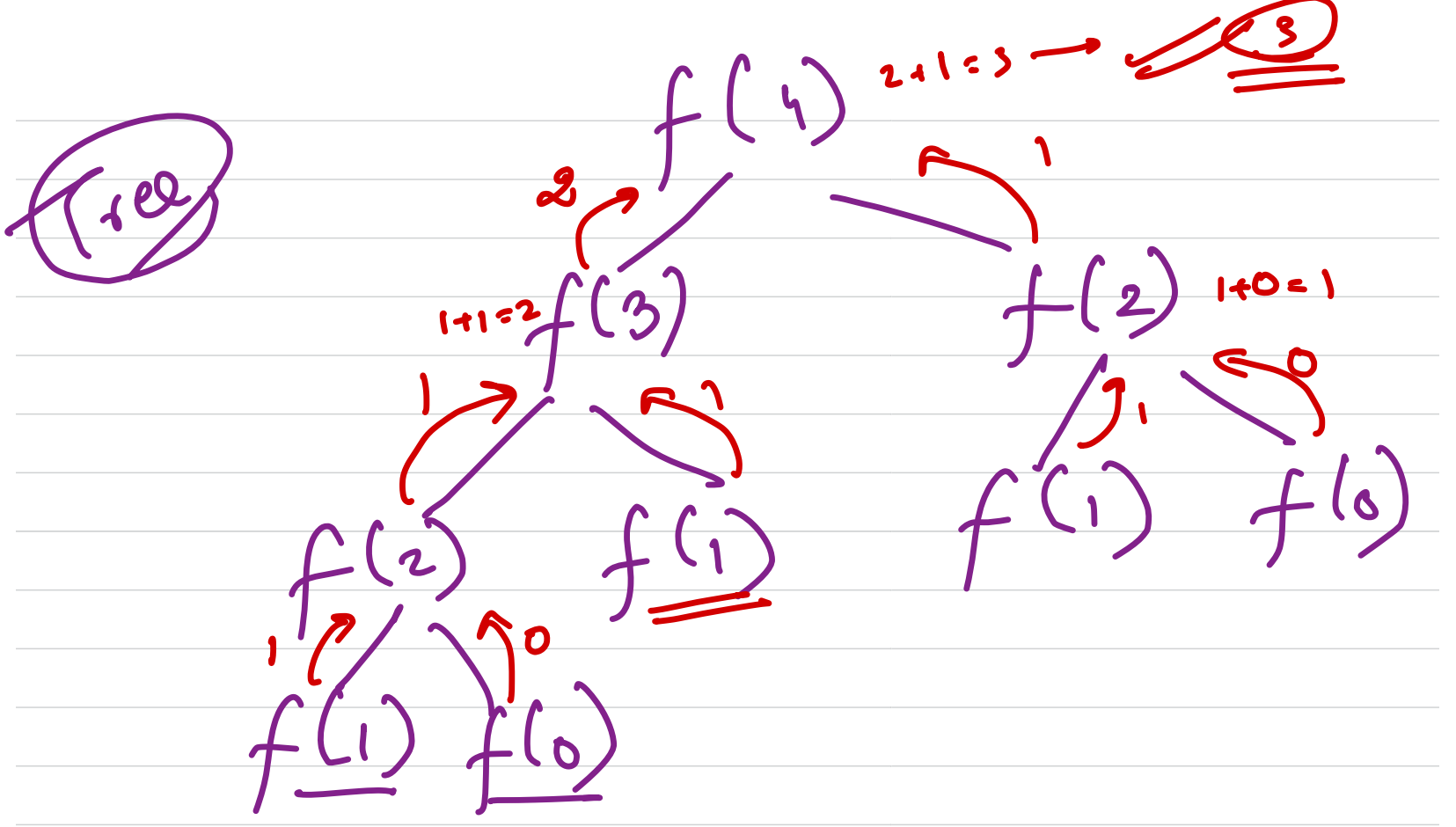
Self work → $f(n-1) + f(n-2)$

$f(n) = f(n-1) + f(n-2)$

```
function fib(n) {
    // base case
    if(n == 0) return 0;
    if(n == 1) return 1;
    // assumption
    let a = fib(n-1);
    let b = fib(n-2);
    // self work
    return a+b;
}

console.log(fib(3))
```

Recursion takes Extra space in memory


f(4)
2+1=3
3
2
1
f(3)
1+1=2
f(2)
1+0=1
1
1
1
0
f(2)
f(1)
f(1)
f(1)
f(0)
1
0
f(1)
f(0)

$$f(n) + f(n+1) = f(n+2)$$

$$f(n-1) + f(n) = f(n+1)$$

$$f(n-2) + f(n-1) = f(n)$$

Say $n+2 = x$

$$f(x-2) + f(x-1) = f(x)$$