JavaScript - Day 2

Execution context :-

```
function getMessage ()
{
    return 'Hi Rahul Aher';
}
function sayHello ()
{
    return getMessage ();
}
sayHello ();
```

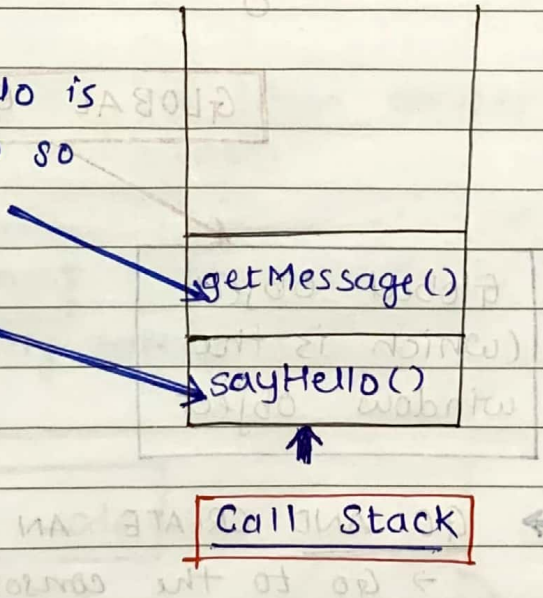
⬇

Javascript Engine (JE) sees these brackets and says 'OH... I need to execute this function, let me create an Execution context for it.

Okay.... But **WHAT IS EXECUTION CONTEXT???**

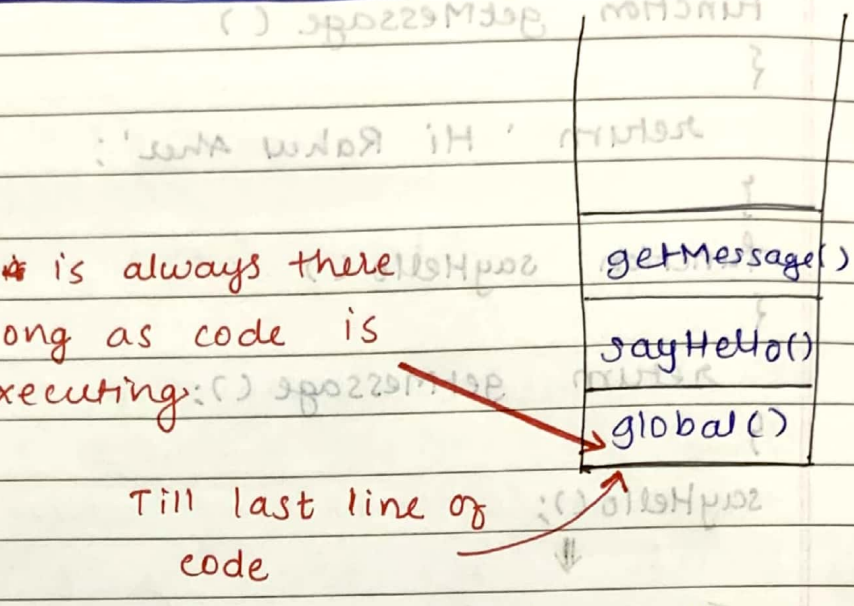
② JE then sees sayHello is calling getMessage() so creates another E.C.

① This is basically the Execution context



* BUT there is a base execution context (GLOBAL CONTEXT

→ So Basically the JE will always first creates the GLOBAL EXECUTION CONTEXT.



→ This means every line of code is part of an execution context. (e.g. global() or a function)

→ So what do I do with this global Execution context?

⇒ It gives us 2 things

GLOBAL EXECUTION CONTEXT

- Global Object (which is the window object)
- "this" keyword in JS (The one that everyone loves;)

→ GO AND CREATE AN EMPTY JS FILE

→ Go to the console and type

> this

→ window object ⇐ output

So this prints the global object. In a browser global object is nothing but window object.

SO CHECK

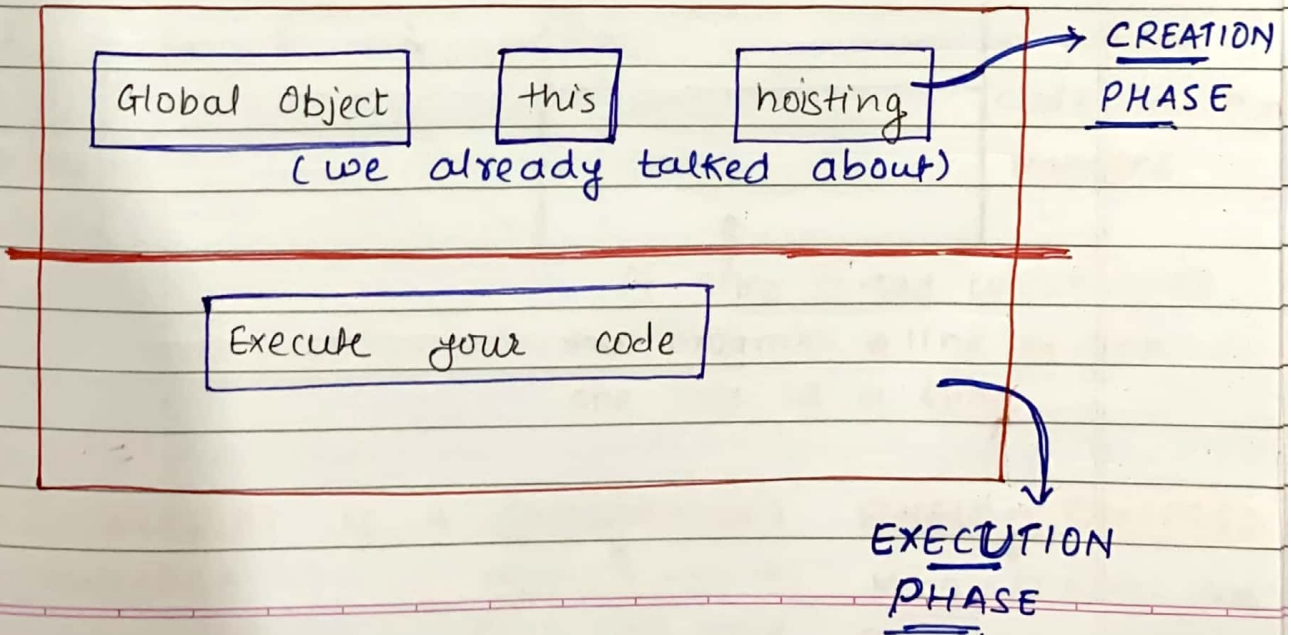
```
this === window    // true
```

* Are they always going to be same?
→ NO !!! we have a whole new section to understand this keyword. Hold on :D

* GO TO CONSOLE AND DECLARE A VARIABLE

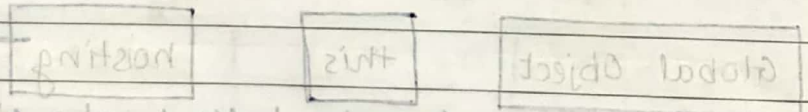
```
> var name = "Rahul Aher"
> window                    → Try to see window
  {
    :
    name: 'Rahul Aher"      [window object will have name variable]
    :
  }
```

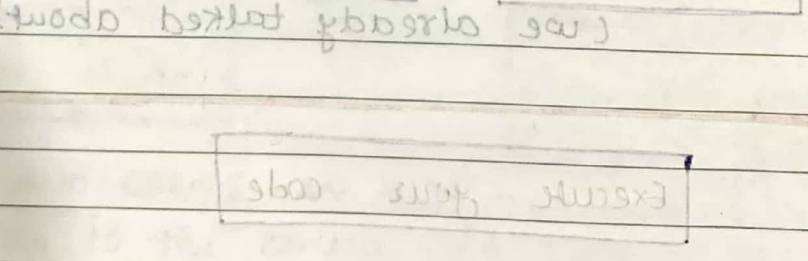
→ There are 2 phase of an execution context.



CREATION PHASE ⇒ Global Obj (window)
(happens first) this
hoisting

EXECUTION PHASE ⇒ Run Code
(happens next)





Extras in JavaScript

* How JavaScript Works?
  OR
* Is JavaScript is Single Threaded or MultiThreaded?
  OR
* Is JavaScript is Synchronous or Asynchronous?

"Everything in JavaScript happens inside an EXECUTION CONTEXT"

→ Assume execution context is like a big box in which whole JavaScript code executed.

- **EXECUTION CONTEXT** :- Has two components
  ① Memory ② Code

Also known as VARIABLE ENVIROMENT

MEMORY	CODE	Also known as THREAD OF EXECUTION
Key : value	0 ——	
	0 ——	
Place where → a : 10	0	← place where a actual code Execution Happens.
variable & fn : {.... }		
functions	0 ——	
stored in the form of key & values.		

it like thread which runs the program • line by line. one line at a time.

→ JAVASCRIPT IS A **SYNCHRONOUS** LANGUAGE
Means it goes to next line once current line & code

SINGLE - THREADED
Means. it runs one command at a time

- What happens when you run JavaScript code?

→ Everything in JavaScript happens inside an Execution context.

→ So when you run JS code an Execution context get created.

→ Let's see how this Execution context created & JS code run inside browser. with the help of program.

```
1 var n = 2;
2 function square(num) {
3     var ans = num * num;
4     return ans;
5 }
6
7 var square2 = square(n);
8 var square4 = square(4);
```

This code coppied a

① So when JavaScript code run's it creates an global execution context.

Memory	Code

② So, the execution context is created in two phases.
- (A) CREATION (Memory Creation phase).
- (B) EXECUTION (code Execution phase).

(A) Creation phase :- In this phase JS allocates memory to all the variable & functions.

Memory	code
n : undefined	
square : fn () {...}	
square2 : undefined	
square4 : undefind	

get there

→ variable stores a special value "undefind"
→ function's are stored as it is in function variable.

(B) CODE EXECUTION PHASE :-
→ In this phase, again JS run's the program line by line & start executing the code.

① → As soon as JS encountered line 1 it replace .value of n from undefined, to "2"

Memory
n: undefined → 2

→ At line 2 JS have nothing to do so it moves further.

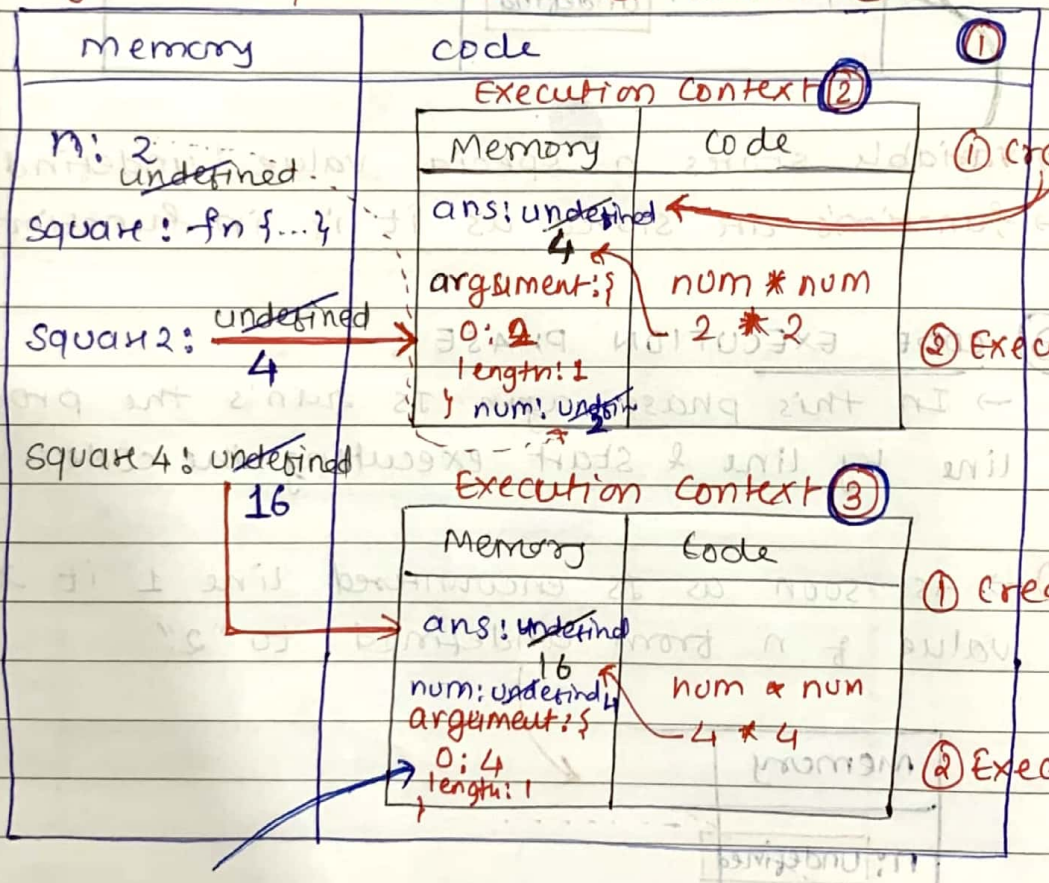
② → So when JS reach to line 6, it actually invoke a function & function Invocation happens. But How...?.

When JS sees square() it's get that it's function invocation.

Function's are the heart of Javascript ❤. They Behaves very diff as compared with any other language.

③ function is like a mini program, & you know for every program to get executed JS creates an Execution context. Therefore for each ~~execut~~ function call an new execution context is created

④ & again two phases comes ① creation ② Execution



We can talk on this in next incoming topics

(5) * return keyword :- It will return the control back to the line where function invokation taken place.

(6) → Once function Execution completed then, whole Execution context for that function is deleted. (i.e. as soon as function return value back to invocation of that function). & then further execution of function takes place.

(7) → on line no. 7 we are again invoking the function & again entire process happens.

(8) At the End of the program Global Execution context is also get's deleted.