Namaste Javascript Ep-2

# How Javascript code is executed & Call Stack

---

When a JS program is run, a global execution context is created.
The execution context is created in two phases.

- Memory creation phase - JS will allocate memory to variables and functions.
- Code execution phase - Execute code one line at a time.

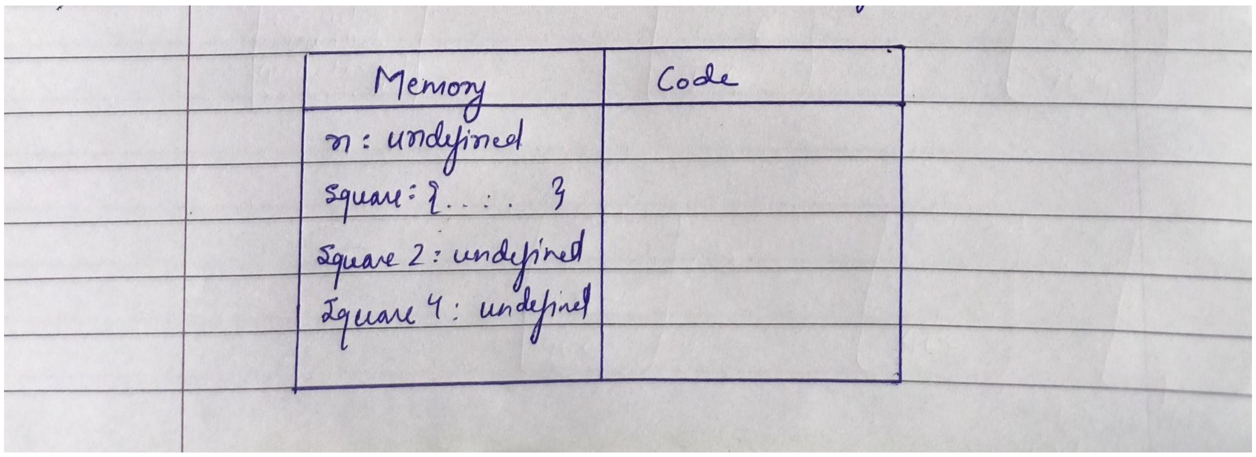
Let's consider the below example and its code execution steps:

```
var n = 2;
function square(num) {
 var ans = num * num;
 return ans;
}
var square2 = square(n);
var square4 = square(4);
```

At first, a global execution context is created then it proceeds for memory creation.

**First Phase i.e. Memory Creation**

For line 1, it allocates memory space for variable '**n**' and stores '**undefined**' , a special value for n.
For line 2, it allocates a memory space for the function '**square**'.
**(it stores the whole code of the function inside its memory space)**
Then, as square2 and square4 are variables as well, it allocates memory and stores 'undefined' for them, and this is the end of the first phase i.e. memory creation phase.



| Memory | Code |
|---|---|
| n : undefined | |
| square : { . . : . } | |
| square 2 : undefined | |
| square 4 : undefined | |

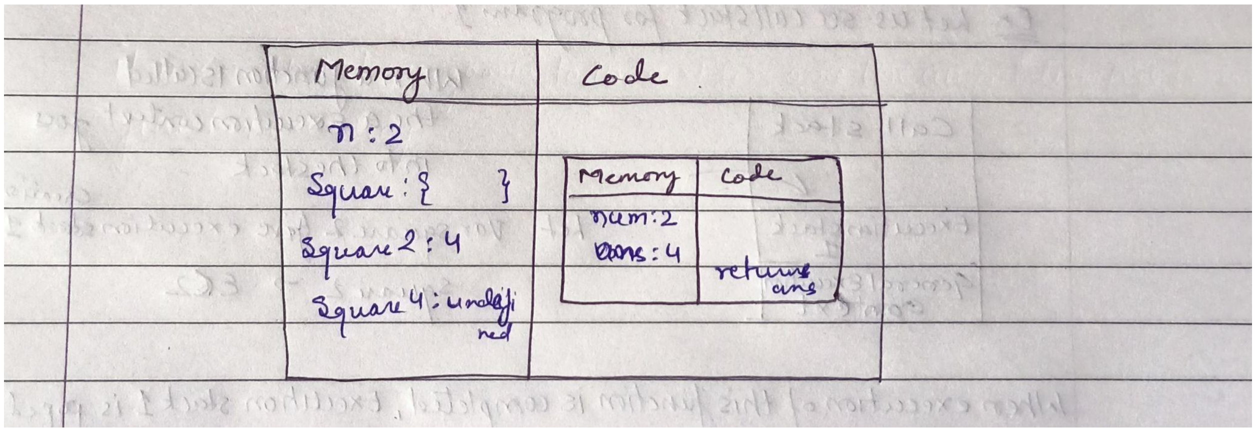
**Second Phase i.e. Code Execution**
Its starts going through the code line by line
For line 1, As it encounters var n = 2, it assigns 2 to 'n' (Until now, the value of 'n' was undefined).
For function, there is nothing to execute. As these lines were already dealt with in the memory creation phase.

Coming to line 6 i.e. var square2 = square(n), here functions are a bit different than any other language. **A new execution context is created altogether**.

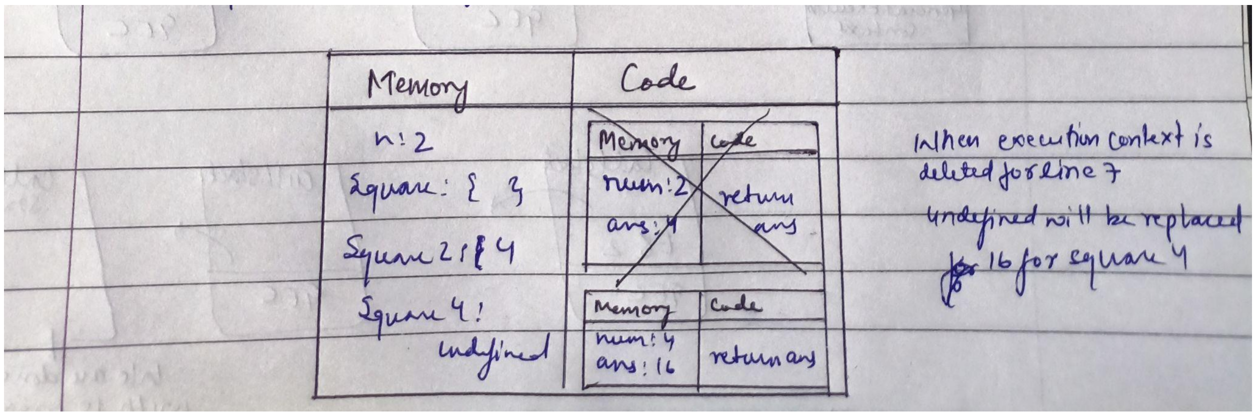
## In new execution context :

First in the memory creation phase, memory is allocated to variables 'num' and 'ans', stores 'undefined' value.
In Code execution phase

- 2 is assigned to variable 'num'
- Variable 'ans' store 4 (i.e. num*num = 2*2)
- Return ans gives the control back to where this function was invoked from i.e. form line 6
- Now 4 will replace undefined in global execution context for square2 function.



When the return keyword is encountered , it returns the control back to the line from where function was called and the new execution context is deleted.



Same steps will be repeated for line 7 i.e. function will be called a new execution context will be created and so on. Once it returns the value in the global execution context, this new execution context will be deleted.

After the whole code is executed i.e whole javascript program then the global execution context will also be deleted.

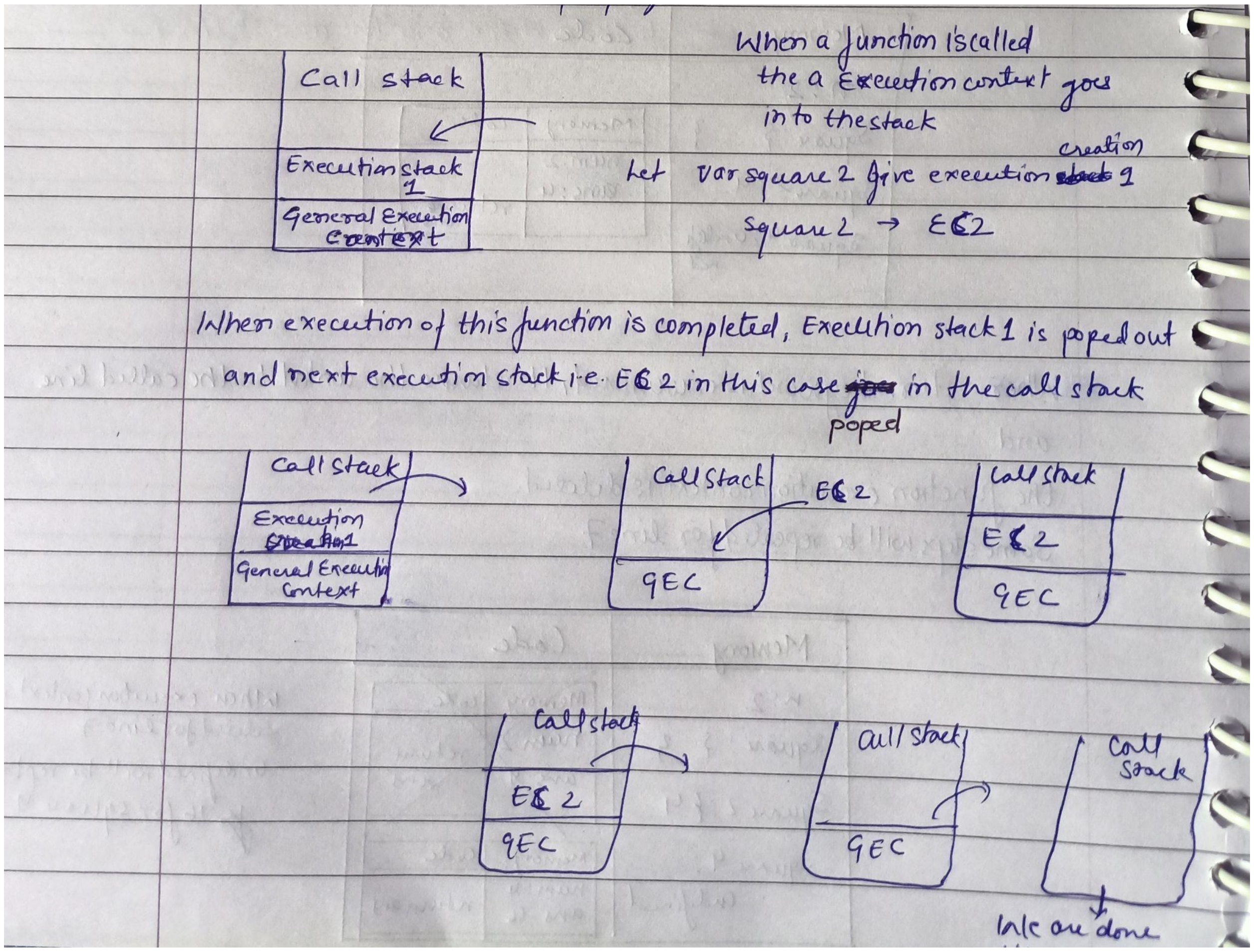
## <u>Call Stack</u>

### Why do we need call stack?

If in any javascript there are functions inside the function and again function inside that function it will become difficult to handle and keep track of but Javascript does it beautifully using call stack.

- Javascript manages code execution context creation and deletion with the help of Call Stack.
- Call Stack is a mechanism to keep track of its place in script that calls multiple functions.
- Call Stack maintains the order of execution of execution contexts. It is also known as Program Stack, Control Stack, Runtime stack, Machine Stack, Execution context stack.

When a javascript program is run, a global execution context is popped into the call stack.
Again when a function is encountered, a new execution context is popped into the call stack. After the value of the function is returned, the popped execution context is popped out of the call stack.
If we are done with the Javascript i.e. whole program is executed the global execution context is also popped out of the call stack and now the call stack is empty.



When a function is called the a Execution context goes into the stack

let var square 2 give execution ~~stack~~ creation 1

square 2 → EC2

When execution of this function is completed, Execution stack 1 is poped out and next execution stack i.e. EC 2 in this case ~~goes~~ poped in the call stack

Watch video on youtube :
How JavaScript Code is executed? ❤️& Call Stack | Namaste JavaScript Ep. 2