

HOW JS code is executed & call stack.

```

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

Parameter

argument

After phase 1 of EC

An Execution Context is created.

→ A global execution context will be created for the code is run.

Execution context is created in 2 phases.

Memory creation

code execution

→ Memory is allocated to variables & functions in memory component of EC. Every calculation & logical operation happens

Variable: A specific keyword which known as undefined (short a placeholder)

→ Actual value to variable is assigned.

function: Exact code of function is copied to memory component.

→ In 6th line of code square function is invoked.

→ function are hurt of JS. They act as mini program.

Hence a execution context is also created. It will again go through 2 phase of EC creation.

After phase 1 of EC

memory creation	Code Execution
n: undefined	
Square: { ... }	
Square2: undefined	
Square4: undefined	

After phase 2 of EC

memory creation	Code Execution
n = 2	
Square = { ... }	

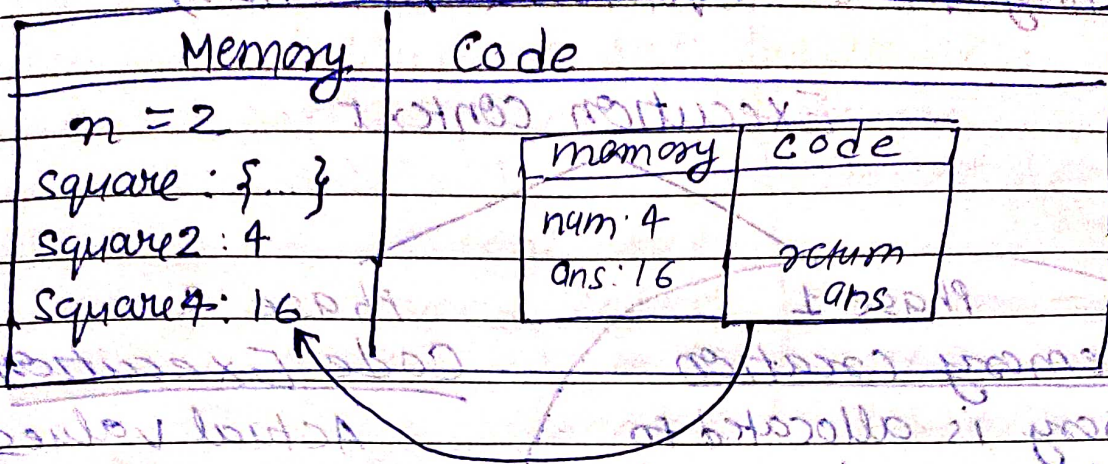
	local memory	memory	code execution
		num: 2	
		ans: 4	

Return will return value & control to execution context where the function was invoked

After phase 2 of execution context of square() function

For line 7 a brand new execution context will be created and similar process will happen

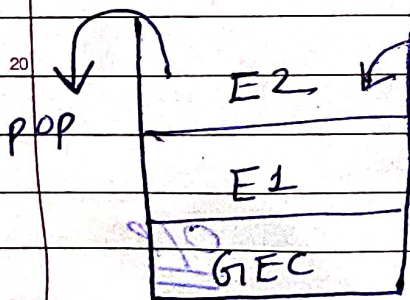
After phase 2 of Global EC.



Call stack maintains the order of execution of execution contexts

→ Call stack is also known as following name:-

1. Execution Context Stack
2. Program Stack
3. Control stack
4. Runtime stack
5. Machine stack: ("2T. 4120. 1000")



← JS call stack