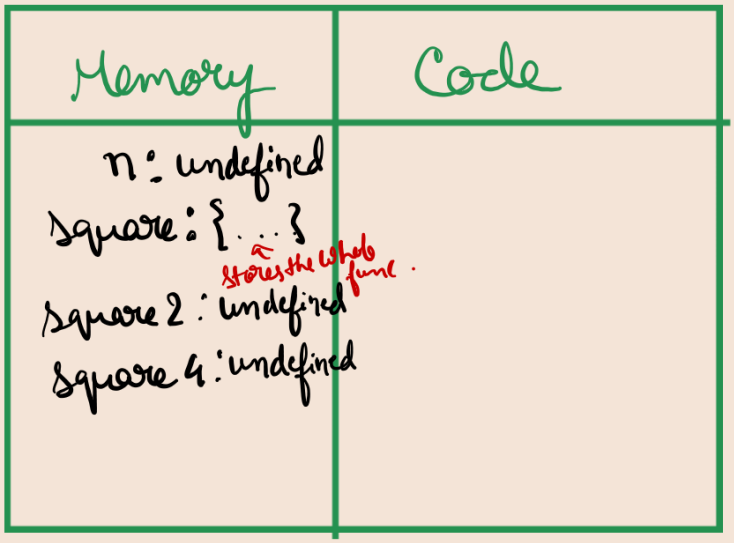
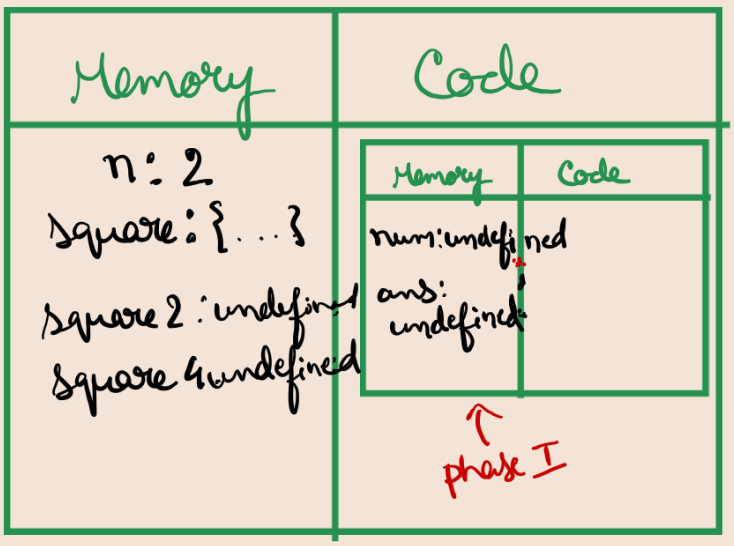
Code:-

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

1st phase



2nd phase

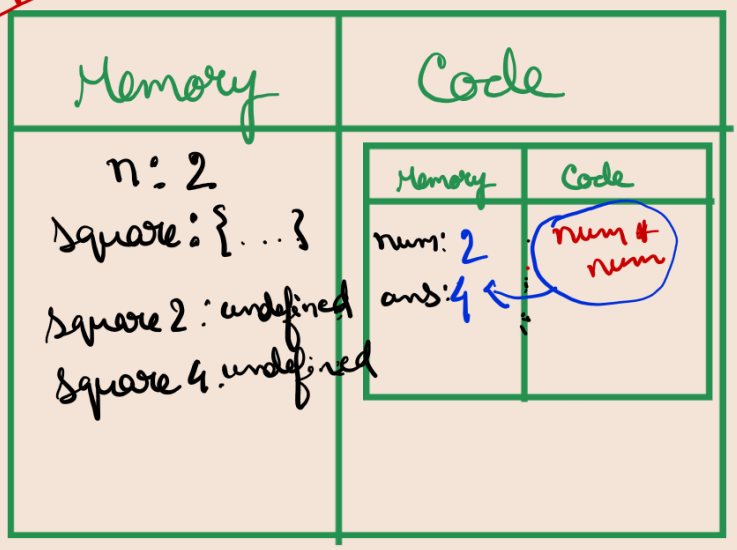


* After allocating value to n i.e 2, there is nothing to allocate in the function, so it goes after that which is function invocation → this creates an execution context in code

* Then again phase I starts that is memory creation

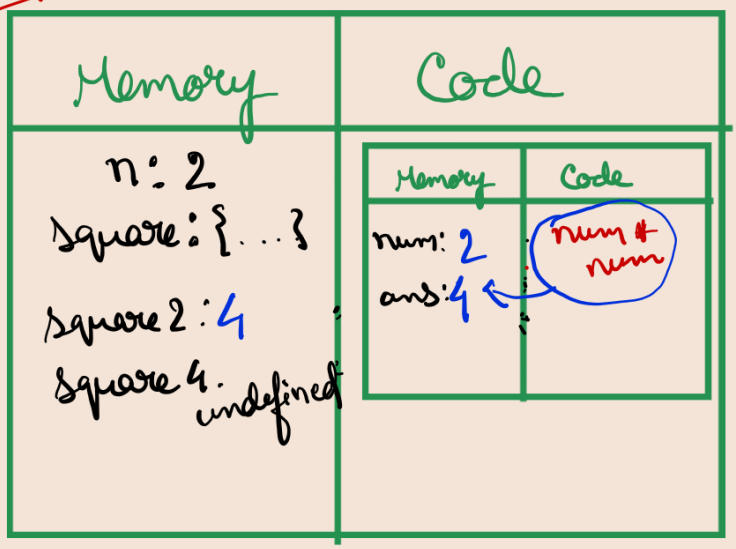
II phase → Runs code and allocates memory in undefined space.

(code execution phase



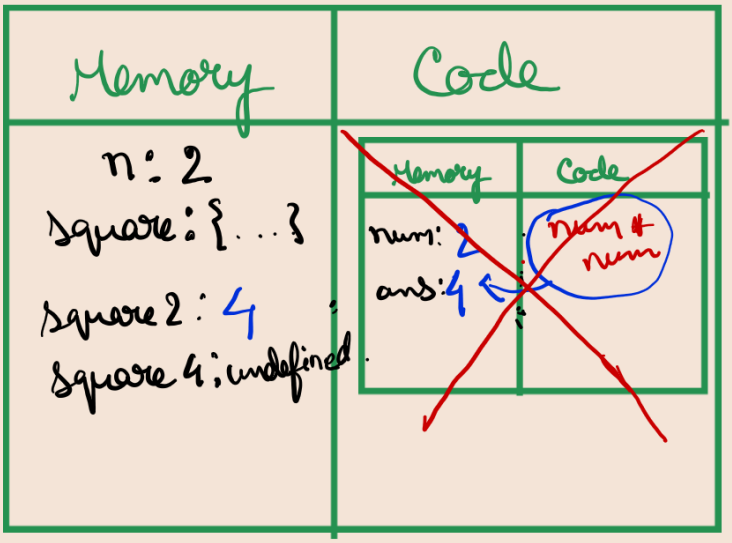
* Then it will encounter return and returns the value to square 2.

3rd phase



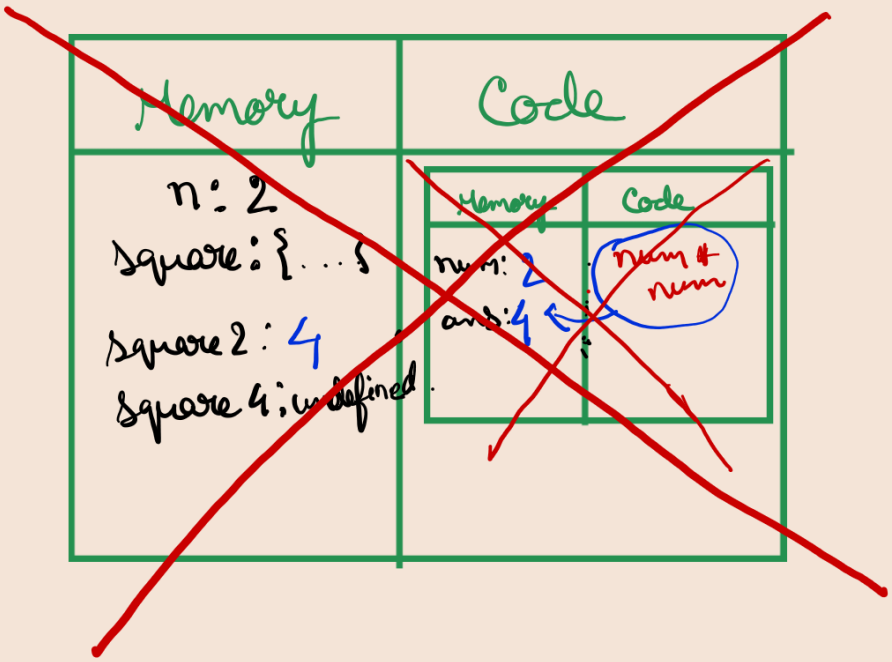
* After the value is returned, the whole execution context for that instance of the function will be deleted.

4th phase



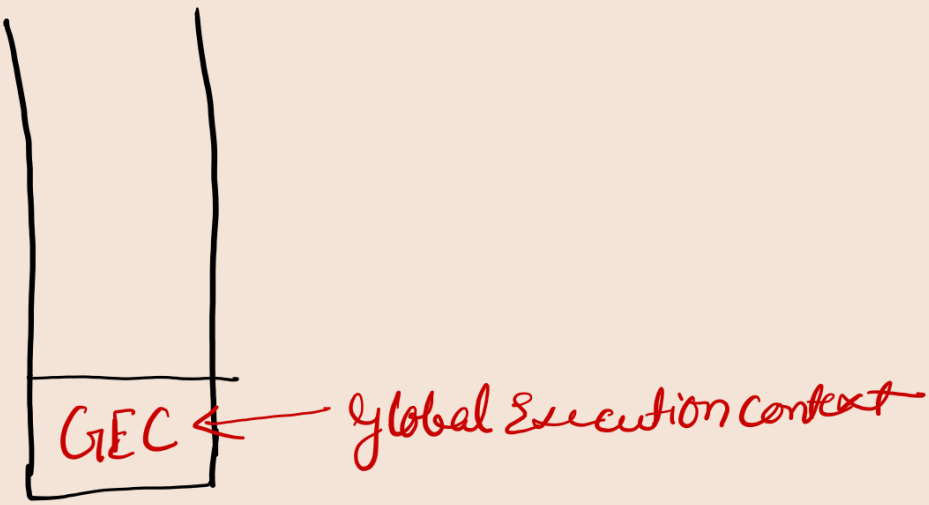
* Some steps happens when it encounters square4.

* When the whole program is finished, the whole global execution context is deleted.

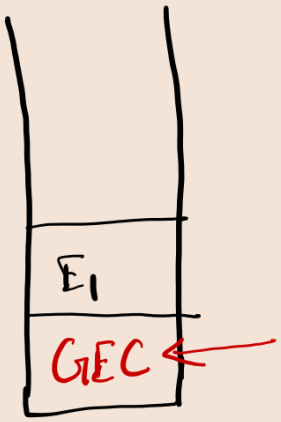


* The whole thing gets executed in call stack.

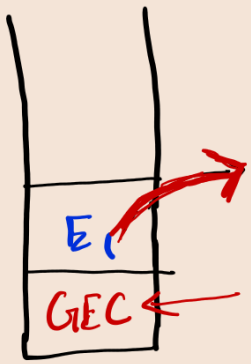
Call stack.



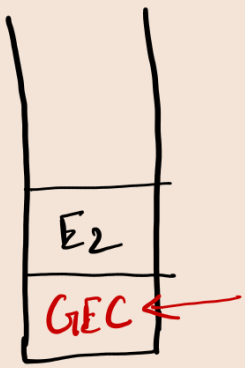
* Whenever a function is encountered, a new execution context is created and put inside the stack.



* Once we are done with the execution of the function E1 is moved out and control goes back to GEC.



* Similarly when it will encounter new function, new execution context is put in stack.



* The some steps gets repeated

* Finally when whole thing gets executed the call stack gets empty.

"Call stack maintains the order of execution of execution contexts"

Call Stack is also known by various names.

- Execution Context Stack
- Program Stack.
- Control Stack.
- Runtime Stack.
- Machine Stack.