JAVASCRIPT

"Everything in Javascript happens inside an Execution Context"

like

Execution Context – is a container & it has 2 components in it.

This is the place where all the variables & functions are stored as a key : value pairs

Memory aka { Variable Environment }	Code Component aka { Thread of Execution }
key : value a : 10 fn : {....}	○ —— ○ —— ○ —— ○ ——

This is the place where code is executed one line at a time.

* Javascript is a Synchronous single-threaded language.

It means JS can execute one cmd at a time.

→ When we say Synchronous Single-threaded language means

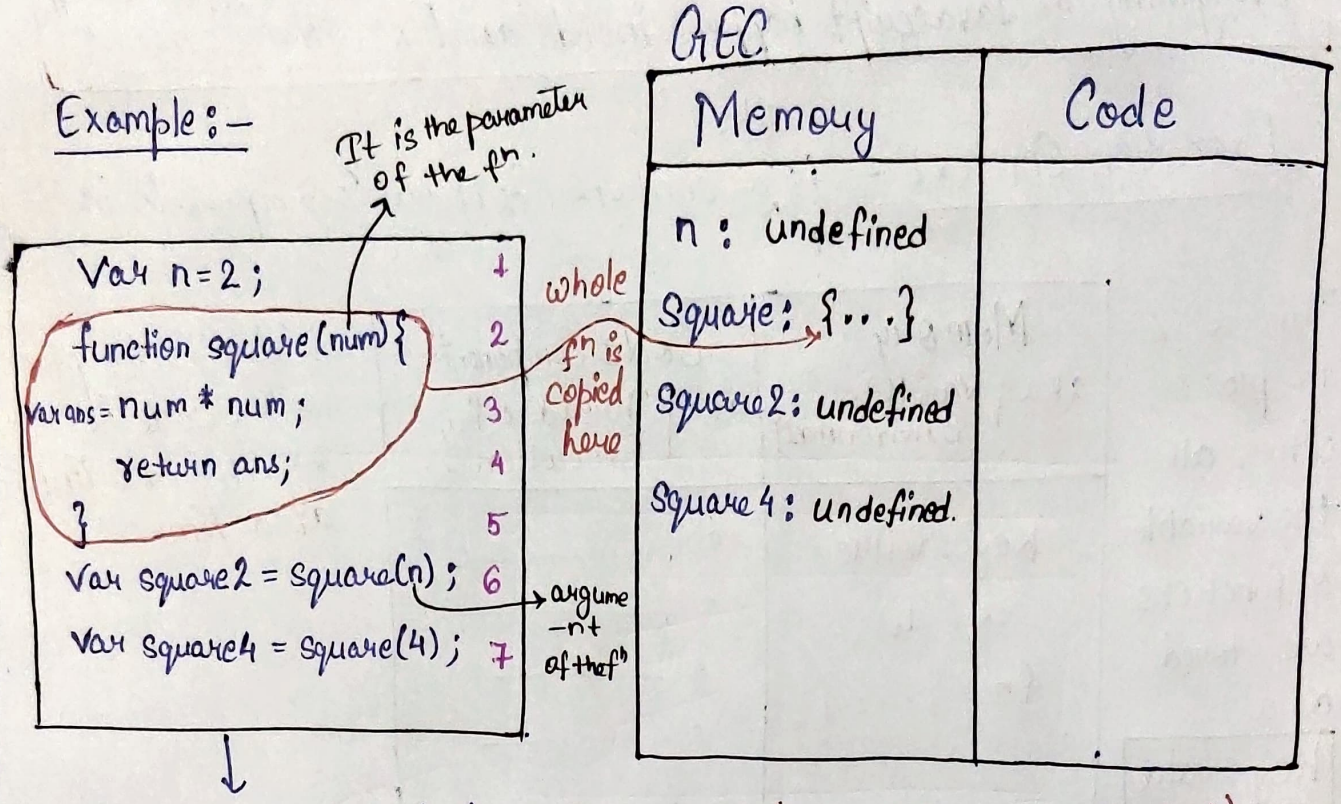
It means JS can only execute one cmd at a time. & in a specific order.

i.e. It can only go to the next line once the current line has been finished executing

Que: What happens when JS code is run?

→ An Execution context is created.

Example:-



When we run this hole code a GEC (Global Execution Content) is created

→ This Execution content is created in 2 phases.

[1st] CREATION PHASE (aka Memory Creation Phase)

→ It is a very Critical phase.

→ In the 1st phase JS (skims through program &) will allocate the memory to all the variables and functions.

- In case of variable — It allocates a special value "undefined" to it.
- In case of fn — The whole fn code is copied in the memory. ~~Space~~ ~~phase~~.

[2nd] Code Execution Phase :-

→ Now JS once again runs through the hole JS program line by line & it executes the code now.

→ As soon as it encounter `var n = 2;` it places the value 2. till now the value of n is undefined. In 2nd phase the value of 2 is assigned to n (i.e. placeholder)

→ After finishing line 1 it moves to 2 & see from 2-5 their is nothing to do.

→ Then line 6 is executed
In line 6 we do function invocation.

→ Now, we go to line 7. Here again we are invoking a function:

GEC

Memory	Code
n : 2 square : {...} square2 : ~~undefined~~ 4 square4 : undefined	Memory: num : ~~undefined~~ 2, ans : ~~undefined~~ 4 Code: num * num, return ans

Note:- When the hole fn is executed the hole E.C for that instance of fn. is deleted.
That means as soon as we return the value the whole E.C is deleted.

Q. What is function Invocation.

→ Whenever we see a fn name with "()" parenthesis. {round ~~bow~~ brackets}
i.e. Square (n);

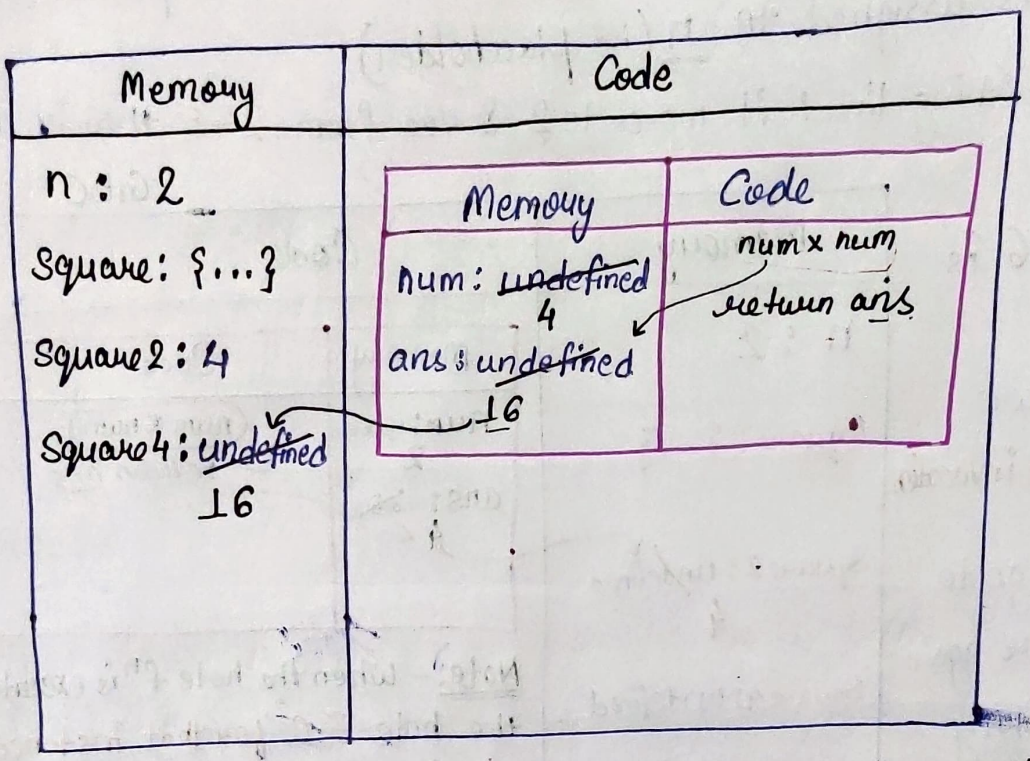
→ It means the fn is now been executed.

→ functions are the heart of JS.

→ fn over here are like mini program. So whenever a function is invoked a mini program is invoked & all together a new EC is created.

Return keyword — Tells the function you are done with your work now just return the hole control back to the execution context where the fn was invoked.

→ Now, it will go to line 7 & again function is invoked.

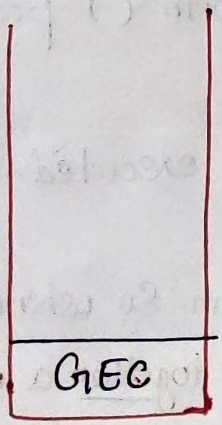


→ Once the whole program is executed, now the GEC is deleted.

→ JS, has its own Call Stack

- Every time in the bottom of this stack we have GEC

Call Stack:-



That means whenever any JS program runs, this call stack is populated with GEC.

The's hole execution context is pushed inside this stack.

→ Whenever a new execution context is created is put inside the stack.

↳ This call stack is only for managing this Execution Context.

* So, whenever a E.C is created is pushed into the stack.
& whenever a E.C is deleted is poped/move out of the stack.

→ And finally the whole program is executed, our call stack will be empty now.

* Call stack maintains the order of execution of execution contexts.

↳ Call stack is also known as:-

- ↳ Execution Content Stack
- ↳ Program stack.
- ↳ Control stack.
- ↳ Runtime Stack
- ↳ Machine Stack.

→ These are nothing but another name of "Call Stack"