How JS works behind the scene?

JavaScript Execution Context

* means jo bhi file banayi hai, us JS run kaise karegi.
* run karne ke liye JS, programs ko do phase me run karti hai.
* Sabse pehle Global Execution Context banta hai.

{} → GEC
↳ jaha bhi ye banega use refer kar diya jata hai this ko.

* Browser me GEC ki value Window aati hai.

NOTE => javascript single threaded hoti hai, esme sab kuch process hota hai.

↳ Global EC
↳ Function EC
↳ Eval Execution Context → more info lele

→ Cycle 1

{} → memory Creation Phase {esme jo bhi variable use kare hai, unke liye memory allocate hoti hai, use execute nahi kiya jata hai.

Only memory alloration for only & only vars

→ Execution Phase

→ Cycle 2

Pgm:

```
Let val1 = 10
Let val2 = 5
function addNum (num1, num2) {
    Let total = num1 + num2
    return total
}
Let result1 = addNum (val1, val2)
Let result2 = addNum (10, 2)
```

Step ① → Global Execution
↓
this

Step ② → Memory Phase — sabhi variable ko es phase me undefined assign hota hai

- val1 → undefined
- val2 → undefined
- addnum → defination → { Let total... return... }
- result1 → undefined
- result2 → undefined

Step ③ → Execution Phase

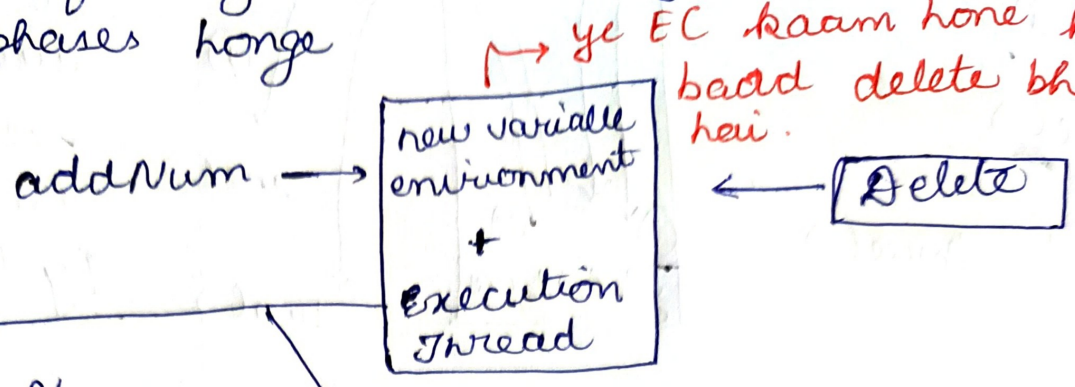
- val1 ← 10
- val2 ← 5

addNum {fn defn vala} yaha kuch nahi hoga, kyuki yaha to bass cheeze define ki hai.
• ab result1 vali line pe aaega.

addNum → new variable environment + Execution Thread

* jitni baar ye fn execute hote hai, utne baar ye naya box create hota hai.
called as → New Executional Context

* Then first es box ke liye first do phases honge. (2)
* ab ye dono phases addNum ke liye execute honge
* jitne baar fn aaega utne baar ye box create hoga, and ye phases honge



Memory Phase

(es memory phase e ko ab defn vale line ke context e se dekhenge, naaki result vali line se)

val1 → undefined
val2 → undefined
total → undefined

Execution Phase

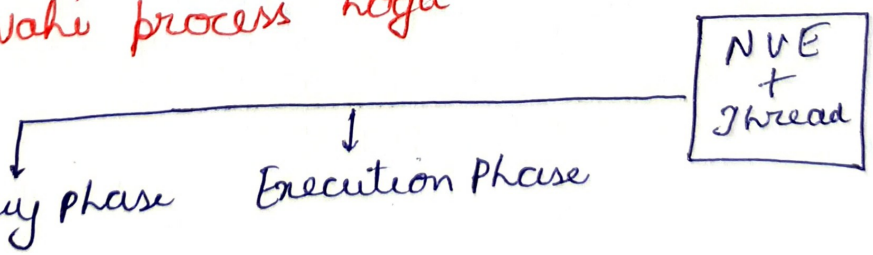
num1 → 10
num2 → 5 ← execute bhi hoga
total → 15

* ye jo return karega, vo parent ke executional context me hoga.

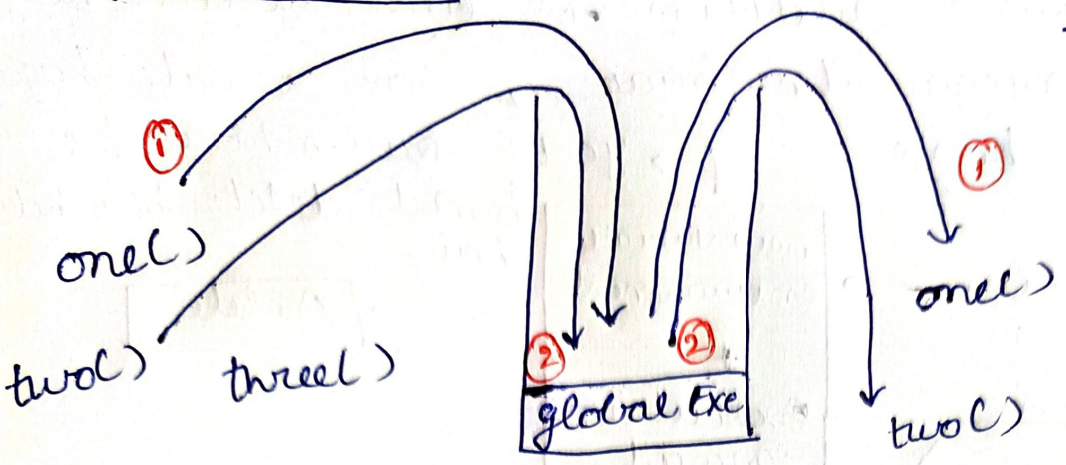
ye return hoga Global EC me

- result1 = 15
- result 2 =

vapas vahi process hoga.



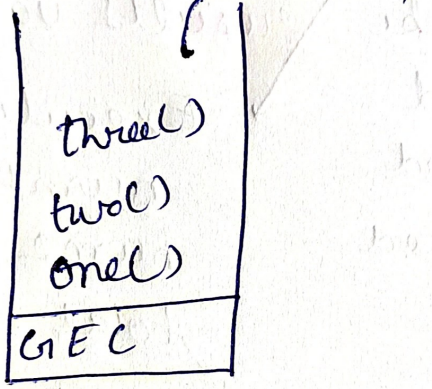
* CALLSTACK



LIFO

* jo fn pehle aaega vo execute hoke last me niklega by LIFO.

```
one()
{ two()
  {
    three(){}
  }
}
```



pehle three()
then two()
then one()

Acc. to LIFO.