JavaScript Execution Context

* In JS, programs are executed in two phases :
  i) Memory Creation Phase
  ii) Execution Phase

* While we are executing program Global Execution Context must be created automatically. It depend on the environment (Browser/Node/Bun/Dino)

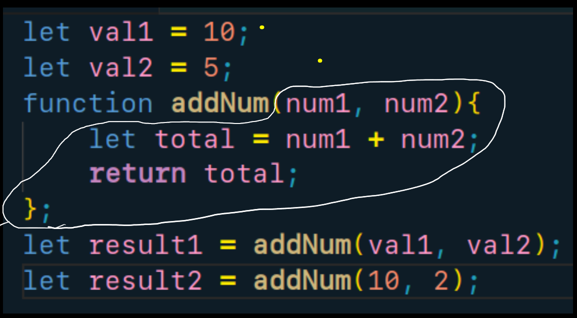
{}

this (Window object)

Execution Context
- ⤷ Global Execution Context
- ⤷ Function Excution Context
- ⤷ Eval Executon Context

---

* How this program will execute ?



```js
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);
```

1 – Global Execution
any program runs through the Global Execution and allocated to this

2 – Memory Phase
all variables are collected and stored

```
val1 = undefined
val2 = undefined
addNum = definition
result1 = undefined
result2 = undefined
```

⇒ not assigned any value.

3 – Execution Phase

val1 ← 10
val2 ← 5
→ nothing to do as definition already given
→ result1 = addNum()

New Executional Context (created)

$\boxed{\text{new variable environment + execution thread}}$

for addNum() again Memory Phase and Execution Phase will create

result2 = addNum() → (same thing will repeat)

Memory Phase

```
val1 ← undefined
val2 ← undefined
total ← undefined
```

Execution Phase

```
num1 ← 10
num2 ← 5
total ← 15
```

*

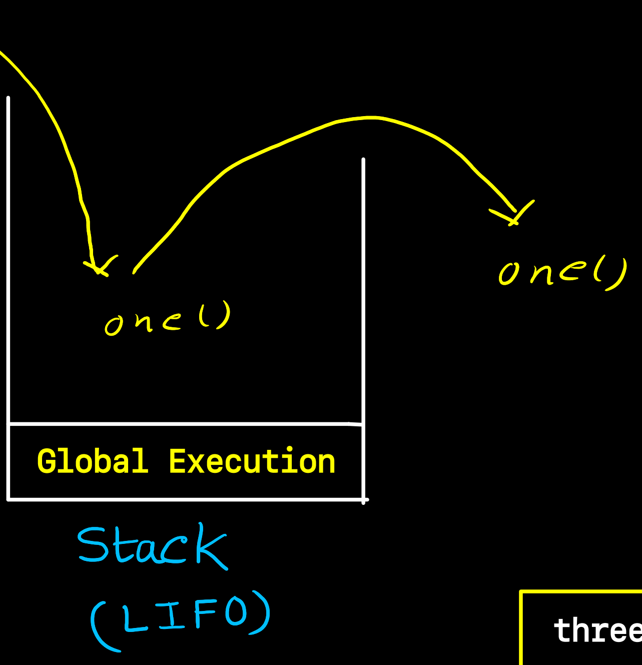
return to the global context and the created new executional context will be deleted after returning.

In this way programs are executed in JS

Call Stack

Example 1: Parallel execution

```
one()
two()
three()
```

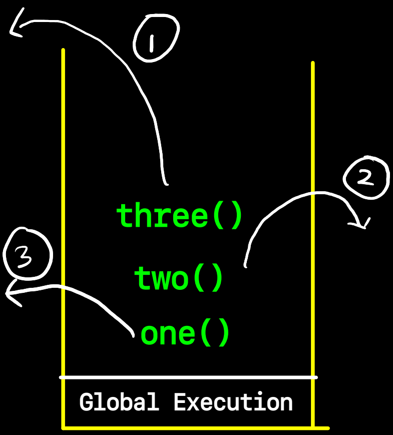


here parallel excution is happening. when one() execution it goes to the call stack after completing execution it is removed from the stack. Then two() will execute in the same way

Example 2:

calling ⇒ one()

after calling one(), two() is called, and two() is calling three(), after termination of three(), two() will terminate the one() will terminate.



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