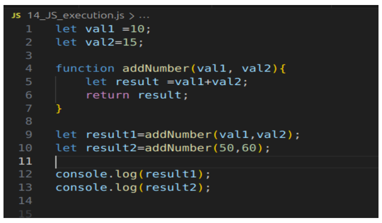
Everything that happens inside javascript , happens inside "Execution context".



14_JS_execution.js > ...

```javascript
let val1 =10;
let val2=15;

function addNumber(val1, val2){
    let result =val1+val2;
    return result;
}

let result1=addNumber(val1,val2);
let result2=addNumber(50,60);

console.log(result1);
console.log(result2);
```

As soon as the program is written , global execution context is created and "this" is assigned to global.

After which code is executed in two phases :
1. Memory Phase
2. Code execution Phase

① Memory phase

val 1 → undefined
val 2 → undefined
addNum → function definition
result1 → ~~undefined~~ 25
result2 → ~~undefined~~ 110

② Code Exec. Phase

val 1 → 10
val 2 → 15
result1 → 25
result2 → 110

new Exec. context

addNum → { new variable env + execution thread }

2a Memory Phase
val1 → undefined
val 2 → undefined
result → undefined

2b Code Exec. Phase
val1 → 10
val2 → 15
result → 25
this value is passed to global exec. context

addNum → { new variable environment + Execution thread }

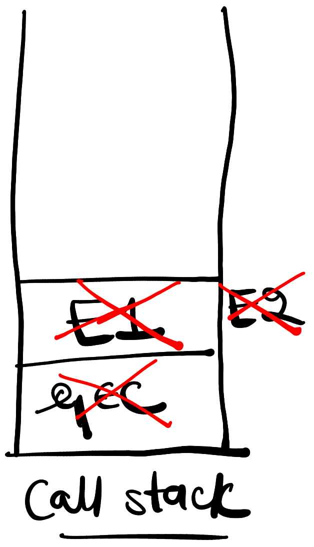
See from function's perspective (see fn. body)

memory phase →
val1 → undefined
val2 → undefined
result → undefined

Code Exec. Phase →
val 1 → 50
val 2 → 60
result → 110
this value is passed to global exec. context.

Note :— After returning value to global execution context, new execution context gets deleted from "call stack".

How the things are working inside Call stack ?



Inside the call stack is where the execution context are maintained , Firstly the global execution context is created , and after that vexecution context 1 is pushed, after the execution of addNumber(val1,val2) , it is deleted from callstack and execution context 2 is pushed into the stack and after its completion , it is also deleted from stack and after all the lines are executed , global execution context also gets removed .