Javascript Execution context

This simply mean how javascript executes a program. JS runs a program in two phases.

When ever a code block is given there will always be a Global Execution context.
The values of this context can be fetch using console.log(this) as discussed before.
The Global Execution context vary for different code Enviornment.
There is one more Execution context that is function Execution context.

One more is there Eval Execution context which is related when we are working with mongoose.

The two phases in which JS execute a program is →

(i) Memory Creation phase or creation phase →
In this phase only memory is allocated

(ii) Execution phase → your code get executed.

Understanding whole concept using example →

```
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 → this
② Memory Phase →
In this phase all the variables are collected and memory is allocated

val1 → undefined
val2 → undefined
addNum → defination
result1 → undefined
result2 → undefined

This is known as first cycle.

Second cycle is execution phase.

③ Execution Phase →
val1 ← 10
val2 ← 5

In case of result1 addNum will make it's own execution context and this happen every -time function is called.

addNum → new variable enviornment + Execution Thread

Every time a new Execution context is created the process will repeat for that context as in above case

Memory Phase of addNum →

val1 → undefined
val2 → undefine
total → undefined

This context is where you define a function, not where you call it

Execution context of addNum →
num1 ← 10
num2 ← 5
total ← 15

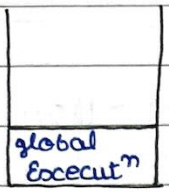
now as we see that we are returning a total, it will get return to global execution of parent enviornment.

now after all this is done the new execution context that was created get deleted as well.

now in parent Execution the result1 will hold value 15.

Same will happen with result2.

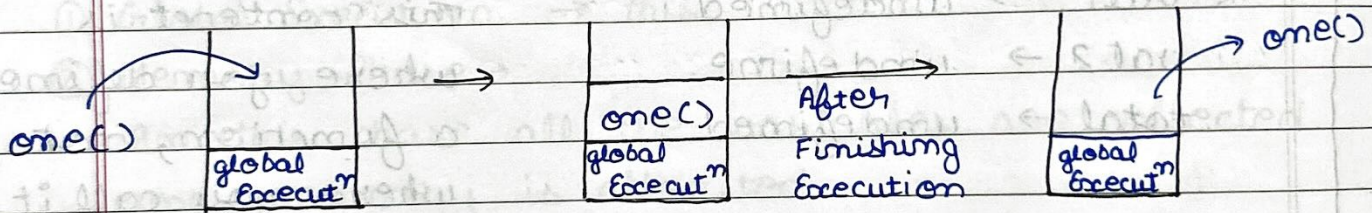
Understanding Execution stack



The first thing will be global execution.

now suppose there is a function one and you call that function. The function will first go in stack and then come out.

```
function one() {
}
```



now suppose there is function inside a function

```
function one() {
    function two() {
    }
}
```

First one() will go in the stack and as one is still in execution the two will be called, and after two() finish it's execution

and go out of stack, then one will finish execution and go out of stack