episode on

How javascript code is executed, call stack

phase (I) Memory creation phase

variable → n : undefined

function → square : {....} ← whole code is stored

square2 : undefined

phase (II) code execution phase

[nothing to do with 2nd line]

n : 2

Again execution context will be created due to function invoke

Memory component

code component

Again two phases

(I) Memory creation phase

num : undefined

ans : undefined

(II) code execution phase

num : 2

ans : calculation i.e. num×num = 2×2 = 4

ans : 4

[we get this value from argument n i.e. 2]

Global execution context creates → Inner exec context created ⟨.....⟩ inner exec. context deleted → Global execution context deleted

example which we consider in code execution :-

```
var n=2;                          // parameter
function square(num) {
  var ans = num * num;
  return ans;
}
var square2 = square(n);          // argument
```

Here when ans gets returned then inner execution context will be deleted

- call stack [all above critical things are managed by callstack]

execution context 2
execution context 1
[when a new function is invoked exec. context is created]

[everything will be deleted now] ← again poped out ← Global execution context ← push at bottom side
at very first

- at end call stack will be empty.
- call stack maintains the "order of execution" of execution contexts.

- call stack fancy names :-
  1. execution context stack
  2. program stack
  3. control stack
  4. runtime stack
  5. machine stack

All names are call stack itself.