Page No.
Date

Lecture-2

How JS is executed & Call Stack

When a JS program is executed, a global execution context is created

- **Memory creation phase** → allocating memory to variable & functions
- **Code execution phase** → execute code one line at a time.

Ex

```
var n=2;
function square(num) {
    var ans = num * num;
    return ans;
}
var square2 = square(n);
var square4 = square(4);
```

Let's see how this program is executed in JS

1st Phase:

In line 1, it allocates memory space for n, then (stores undefined)

2, for function (stores whole code of function)

6,7, it allocates memory variable square2, square4 and stores undefined.

Memory	Code
n: undefined	
square: {. . . .}	
Square 2: undefined	
Square 4: undefined	

Page No.
Date

2nd phase : it starts going through line by line.

line 1, assigns 2 to n.
For functions, there are nothing to execute as these lines were already dealt in memory creation phase.
line 6, When a function is called a new execution context is created altogether.

In this new execution context in mcp, allocated memory to num and ans placing undefined.
In code execution phase, 2 is assigned to num.
Var ans = num * num store 4
"return ans" gives the control back to where this function was invoked from i.e. line6.
Now '4' will replace undefined in Main memory block of execution

Memory	Code
n : 2 Square : { } Square 2 : 4 Square 4 : undefined	Memory: num : 2, ans : 4 — Code: return ans

When return keyword is encountered, It returns the control to the called line and
the function execution context is deleted.
Same steps will be repeated for line 7

Memory	Code
n : 2 Square : { } Square 2 : 4 Square 4 : undefined	Memory: num : 2, ans : 4 — Code: return ans Memory: num : 4, ans : 16 — Code: return ans

When execution context is deleted for line 7
undefined will be replaced for 16 for square 4

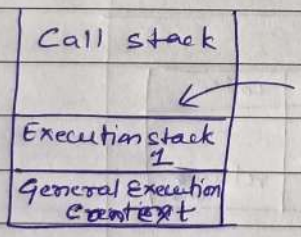
Page No.
Date

If there function inside function and another function. It will become difficult to handle and keep track of.
But JS does it beautifully using call stack.

CALL STACK

* JS manages code execution context and deletion with the help of Call stack.
* mechanism to keep track of its place in script that calls multiple function.
* maintains order of execution of execution context.
* also known as Program stack, Control Stack, Runtime stack, Machine Stack, Execution conter stack

Ex let us see callstack for program 1.



When a function is called the a Execution context goes in to the stack

let var square2 give execution creation ~~stack~~ 1

square2 → EC2

When execution of this function is completed, Execution stack 1 is poped out and next execution stack i.e. EC2 in this case ~~is~~ in the call stack poped