Closures

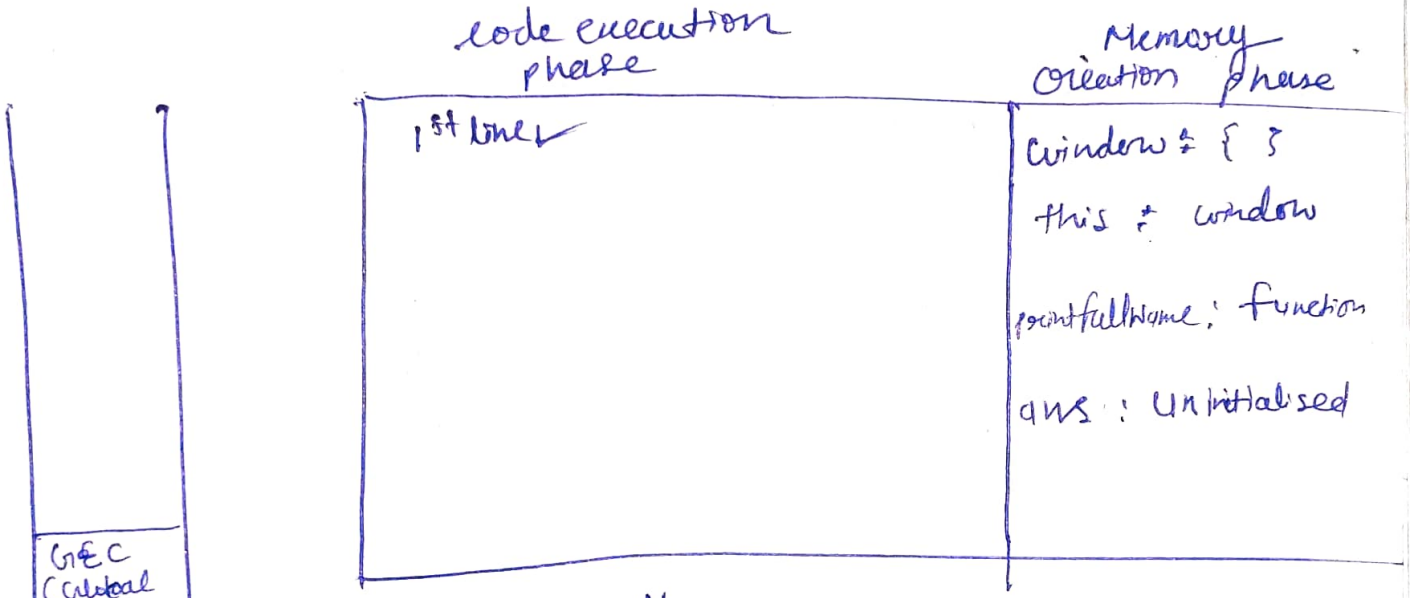
function can return function - also → Console: Harshit Sharma

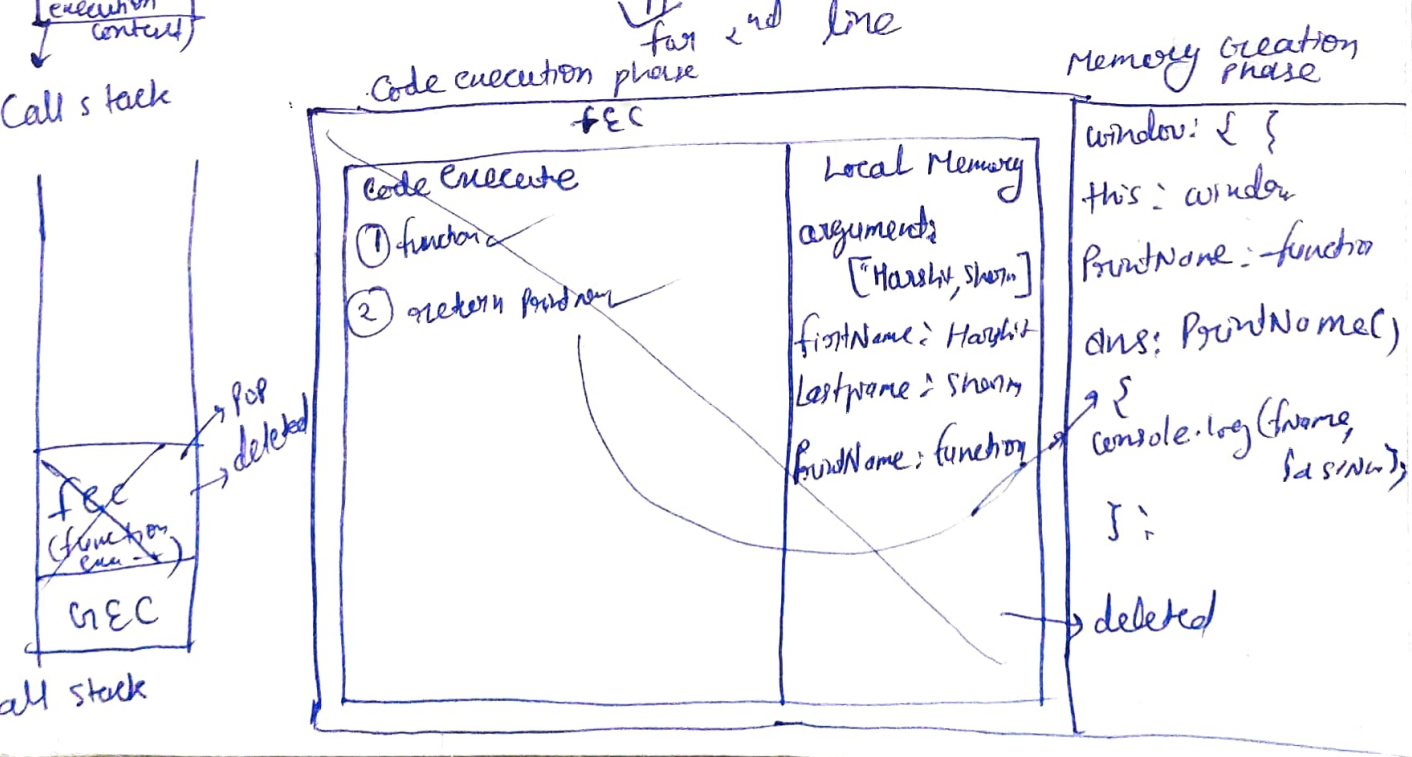
Example - function

```
function printfullName (firstName, lastName){
    function printName() {
        console.log (firstName, lastName);
    }
    return printName;
}
const ans = printfullName ("harshit", "sharma");
ans();
```

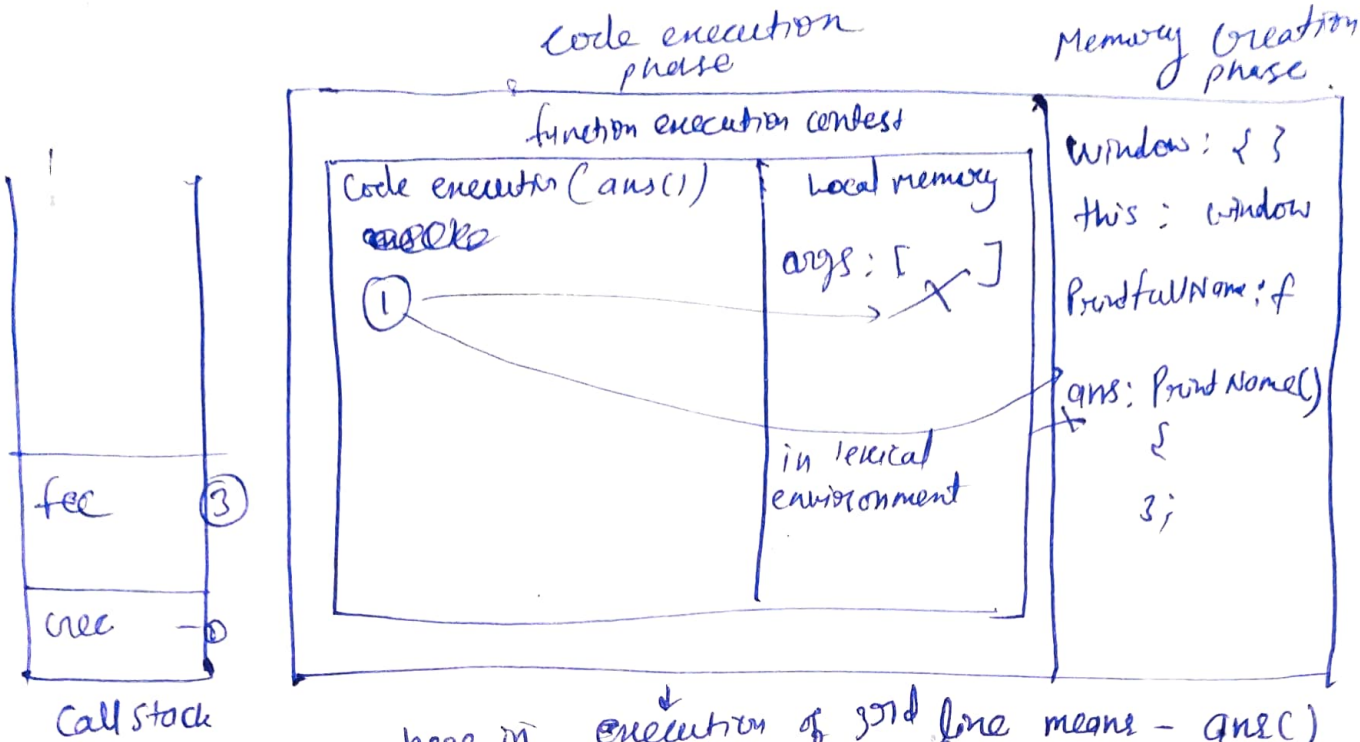
① ② ③

Analysis





3rd line execution



here in execution of 3rd line means - ans()
there is no arguments for function in local memory and also in lexical environment

so how console is giving output - Harshit Sharma

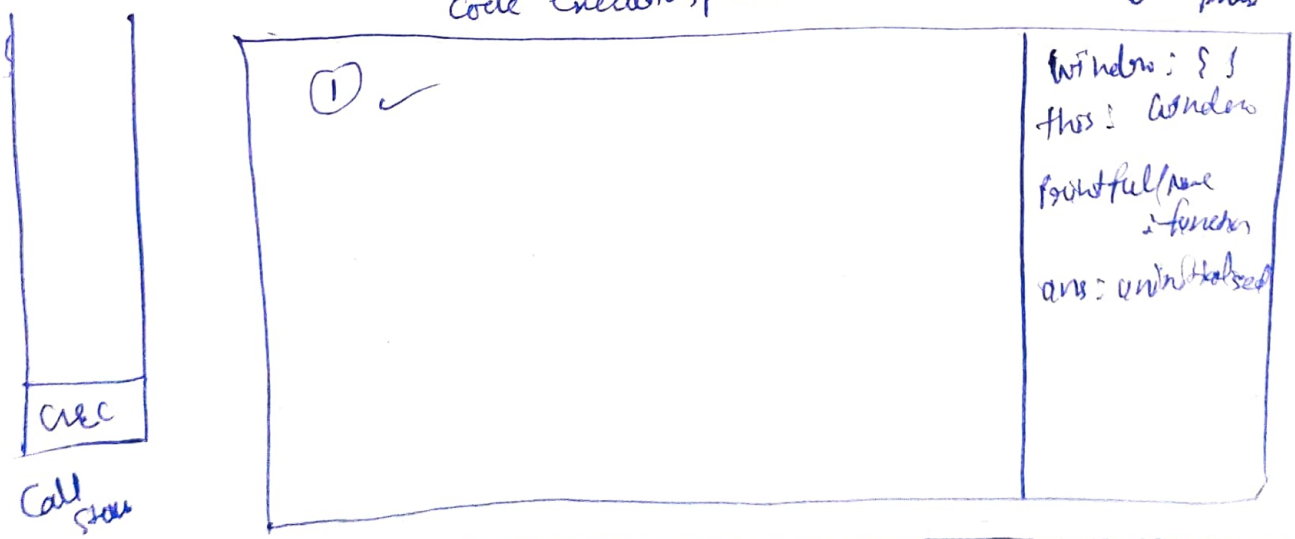
* जब भी कोई function दूसरी function से return होता है तो अपने साथ उस function की local memory को लेकर Return होगा which is called closures

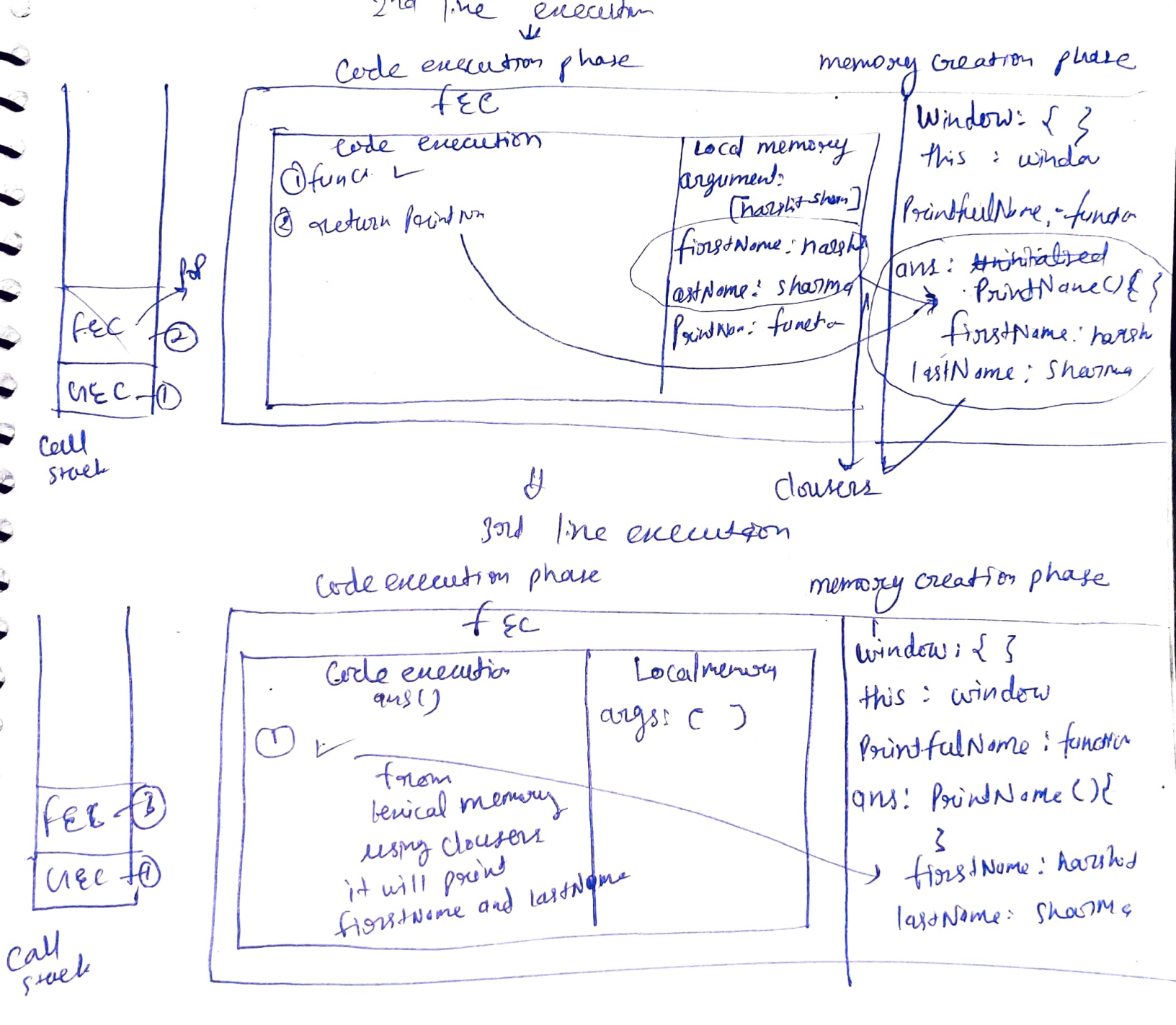
in this example when PrintName will be return it will come with local memory of PrintfullName

this is the reason for output Harshit Sharma

(i.e printName अपने साथ firstName और lastName dono को लेकर Return होता है)

New analysis from beginning



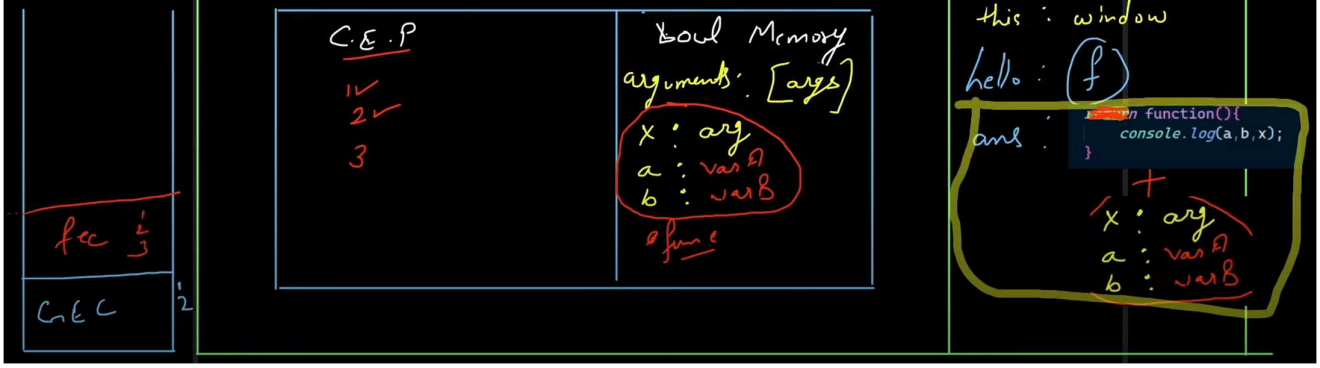

2nd line execution
Code execution phase
memory creation phase
fEC
Code execution
Local memory
argument: [harshit sharma]
① funcA
② return printNm
firstName: harsh
lastName: sharma
printNm: funcA
Window: { }
this: window
printfulName: funcA
ans: printName(){ }
firstName: harsh
lastName: Sharma
fEC
GEC
Call stack
Closures
3rd line execution
Code execution phase
memory creation phase
fEC
Code execution
ans()
Local memory
args: ( )
from lexical memory using closures it will print firstName and lastName
Window: { }
this: window
printfulName: funcA
ans: printName(){ }
firstName: harshit
lastName: Sharma
fEC
GEC
Call stack

```
function hello(x){
    const a  = "varA";
    const b = "varB";
    return function(){
        console.log(a,b,x);
    }
}

const ans = hello("arg");
ans();
```

Code Execution Phase GEC

Memory Creation Phase



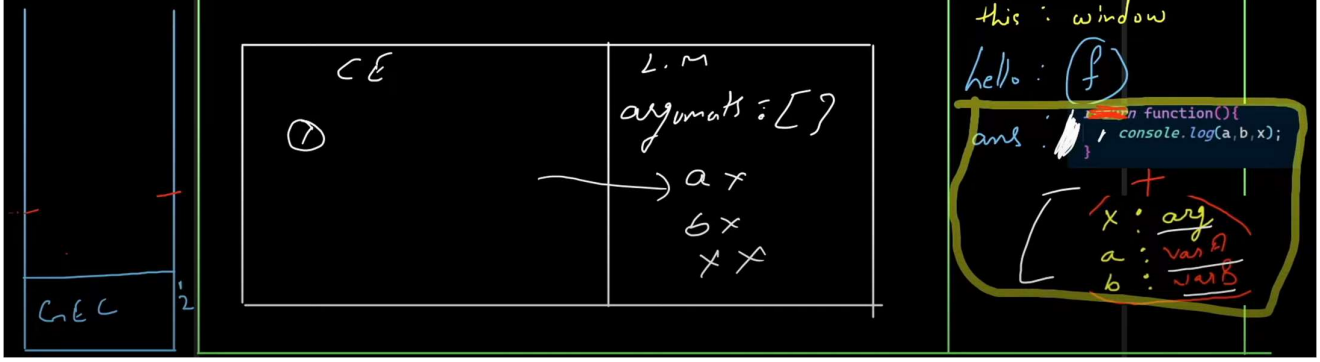

```
function hello(x){
    const a  = "varA";
    const b = "varB";
    return function(){
        console.log(a,b,x);
    }
}

const ans = hello("arg");
ans();
```

(arg, varA, varB) → output

Code Execution Phase GEC

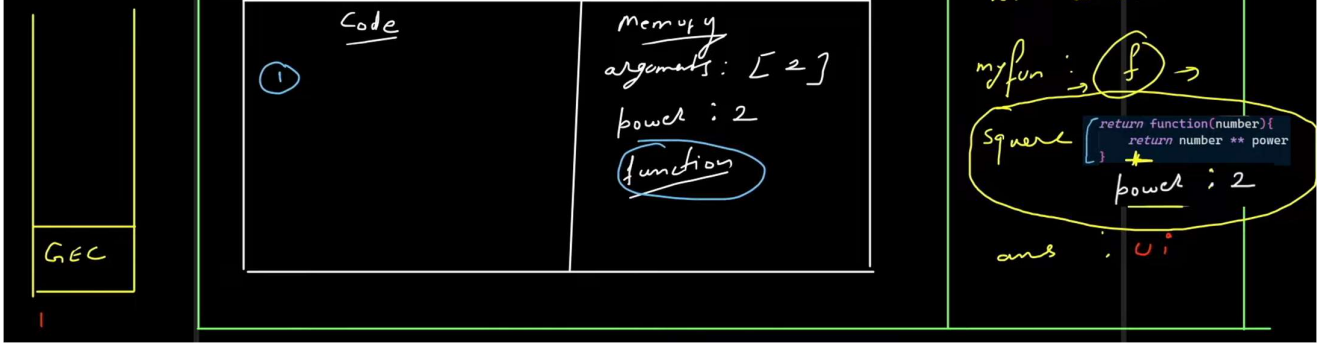
Memory Creation Phase

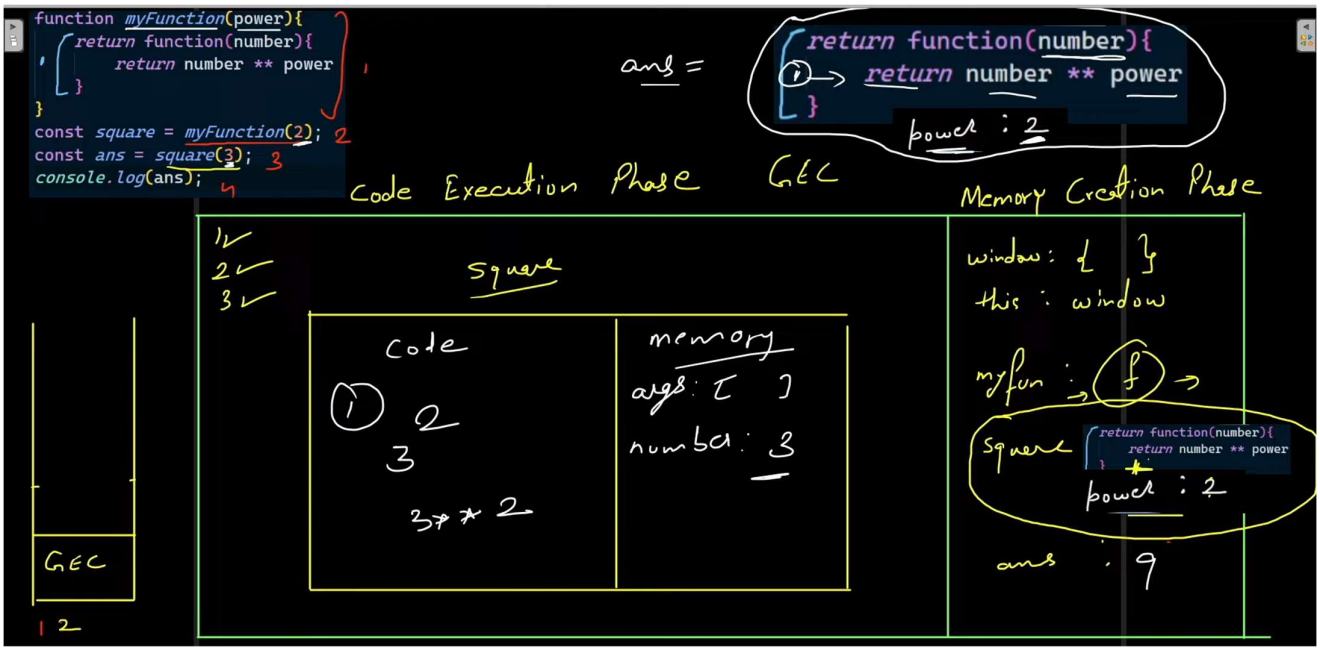


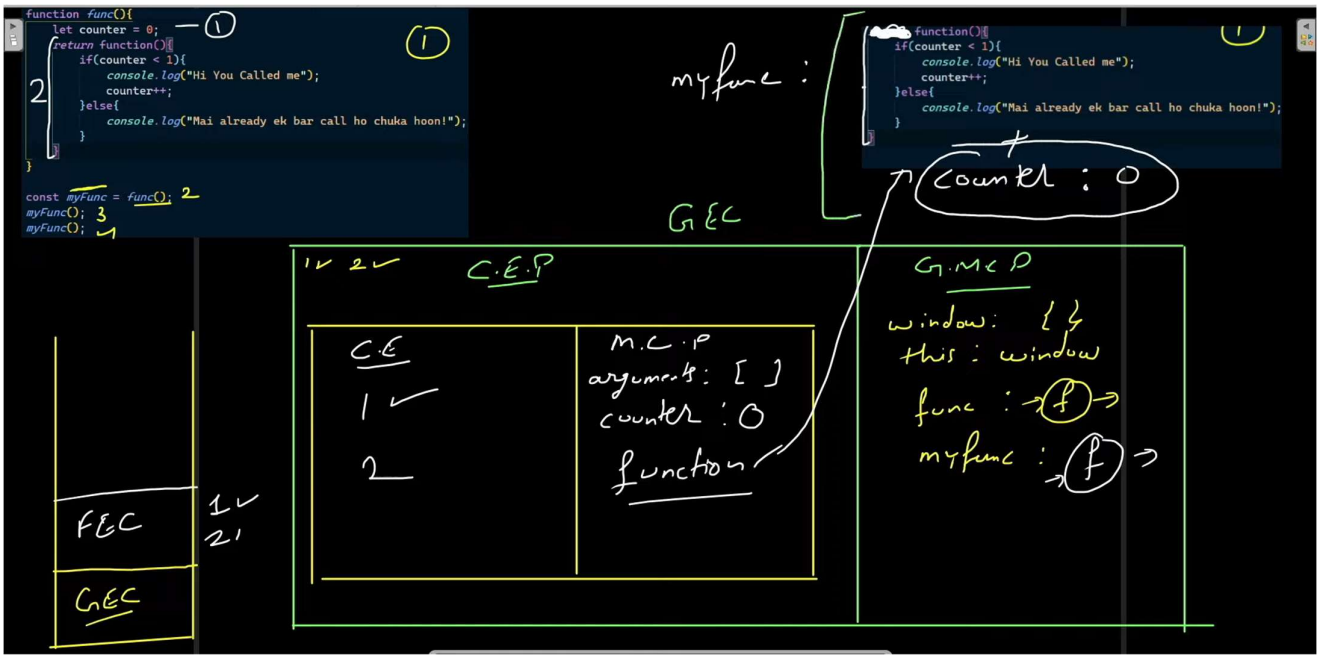

```
function myFunction(power){
    return function(number){
        return number ** power
    }
}
const square = myFunction(2);
const ans = square(3);
console.log(ans);
```

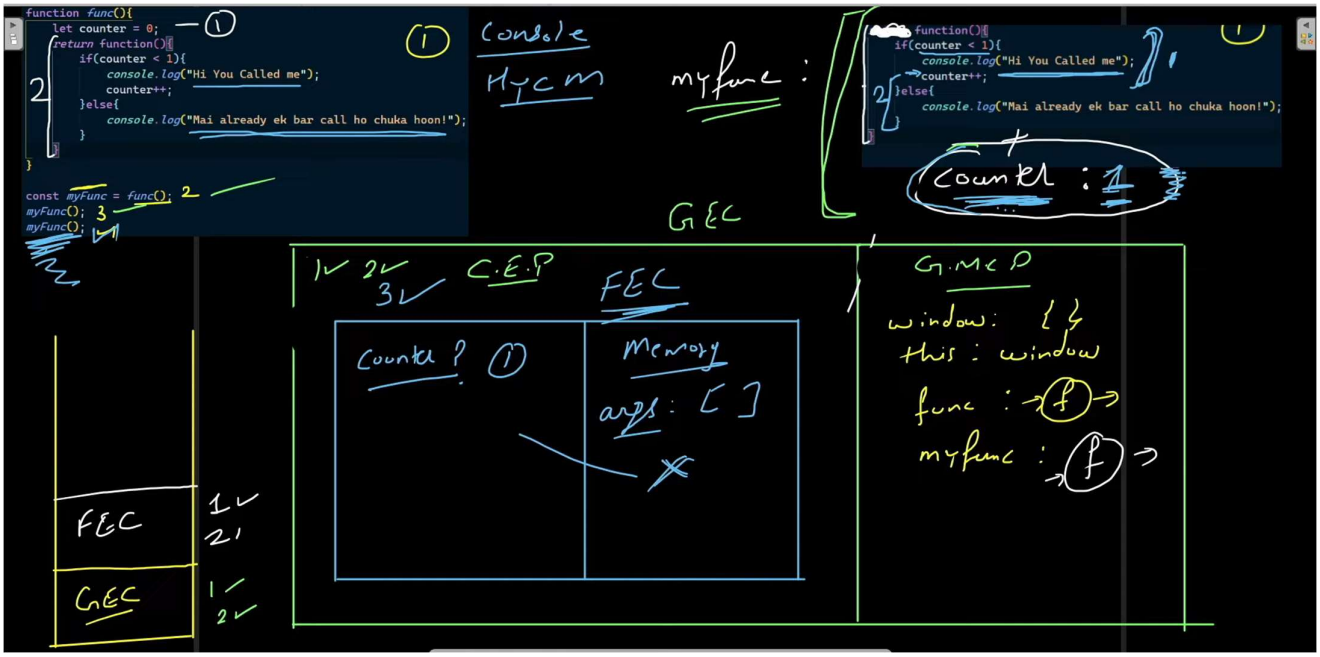
Code Execution Phase GEC

Memory Creation Phase




function myFunction(power){
return function(number){
return number ** power
}
}
const square = myFunction(2);
const ans = square(3);
console.log(ans);
ans =
return function(number){
return number ** power
}
power : 2
Code Execution Phase GEC
Memory Creation Phase
square
Code
memory
args: [ ]
number: 3
window: { }
this : window
myfun :
square
power : 2
ans : 9
GEC


function func(){
let counter = 0;
return function(){
if(counter < 1){
console.log("Hi You Called me");
counter++;
}else{
console.log("Mai already ek bar call ho chuka hoon!");
}
}
}
const myFunc = func();
myFunc();
myFunc();
myfunc :
counter : 0
GEC
C.E.P
C.E
M.C.P
arguments: [ ]
counter : 0
function
G.M.C.P
window: { }
this : window
func :
myfunc :
FEC
GEC


function func(){
let counter = 0;
return function(){
if(counter < 1){
console.log("Hi You Called me");
counter++;
}else{
console.log("Mai already ek bar call ho chuka hoon!");
}
}
}
const myFunc = func();
myFunc();
myFunc();
Console
HTCM
myfunc :
counter : 1
GEC
C.E.P
FEC
counter ? 1
Memory
args: [ ]
G.M.C.P
window: { }
this : window
func :
myfunc :
FEC
GEC