Pg No :
Date: / /20

- In the previous code, to access variables / functions in the global space, we can do:

```
console.log (window.a)
console.log (a)
console.log (this.a)   (this points to window in global level)
```

Output: All 3 is same, 10

- If we do,
  console.log (x)

⟹ Error: x is not defined (x is not in global level)

Episode-6

★ undefined vs not defined in JS

→ "undefined" keyword indicates the absence of a value or the uninitialized state of a variable.

→ In Memory Creation Phase, all variables are assigned to "undefined" before the code execution starts.

→ e.g.
```
var a
console.log(a)
```
⟹ undefined.

→ "not defined" refers to a situation where a variable has not been declared or is not accessible in the current scope.

→ e.g.
```
var a
console.log(a)
console.log(x)
```
⟹ undefined
Error: x is not defined.

→ JS is a loosely typed language or a weakly typed language. i.e. it does not depend on data type declarations.

→ e.g.
```
var a
c.l(a)
a = 10
c.l(a)
a = "hello world"
c.l(a)
```

⟹ undefined
10
hello world

i.e. we can assign it to a number, string etc.

→ a = undefined

It is not a good practise to assign a variable undefined.

The purpose of undefined is that it shows that the variable has not been assigned a value.

☆ Extra (undefined vs. null)

→ Undefined means a variable has been declared but has ~~not~~ yet not been assigned a value.

→ It is a global property

→ Null is an assignment value. It can be assigned to a variable as a representation of no value.

→ null == undefined ⟹ true
null === undefined ⟹ false

→ e.g. var a = null (we can do this)

typeof a ⟹ Object

→ var a
typeof a ⟹ undefined

Episode - 7

* The Scope Chain, Scope & Lexical Environment

→ Scope : Where you can access a specific variable or a function in our code.

→ Lexical Environment : Whenever a EC is created, a lexical Env is created

Lexical Env is the local memory along with the Lexical Env. of its parent.

→ lexical means in a hierarchy or in a sequence.

e.g.

```
function a() {
    var b = 10
    function c() {
       :
    }
}
a()
console.log(..)
```

c lexical parent is a
a lexical parent is global EC.

Global EC parent is null.

It makes a chain and is known as scope chain.

→ The outer reference checks the parents lexical environment and make it accessible for its own scope.

→ The scope of a code block can reach all its parents lexical environment.

→ In this way, we have a chain actually, chain of all parent's scope, known as scope chain.

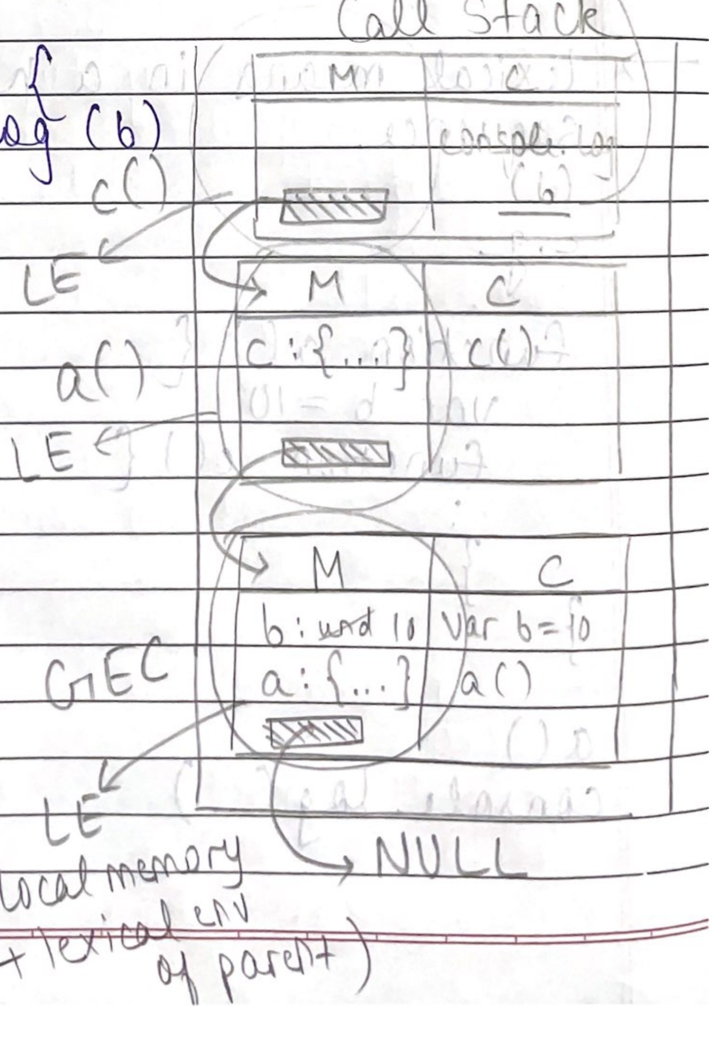
e.g. Code:

Tries to find b in its on LE. If not found, it searches it in parent, then its parent and so on, until it is found or reaches null.

```
var b = 10
function a() {
    function c() {
        console.log(b)
    }

    c()
}
a()
```

⇒ Output : 10


Call Stack
M
C
console.log(b)
c()
LE
M
C
c:{...}
c()
a()
LE
M
C
b: und 10
var b = 10
a:{...}
a()
GEC
LE
(Local memory + lexical env of parent)
NULL