Namaste Javascript

Javascript

1. "Everything in Javascript happens inside an Execution Context"

Exeution content

Memory (var. environ)	Code (Thread of exe)
key : Value.	o
a : 10	o
fn : {....}	o
	o

Javascript is a sychronous single-threaded language.

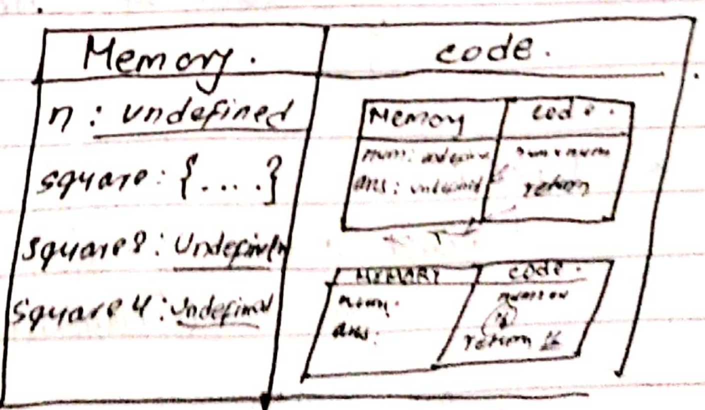
$\underbrace{\text{sychronous}}$ ↓ Specific order

$\underbrace{\text{single-threaded}}$ ↓ Only execute one command

* What happens when Javascript code is exeuted?

⇒ Exeution Content is created.

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



{ 1st phase : Memory allocation/creation.
{ 2nd phase : code execution. & now value in undefined }

"Call stack maintoine the order of execution contents"

Call stack.
- Execution content stack.
- Program stack
- Control stack.
- Runtime stack.
- Machine stack.

3.

Hosting in Javascript
(variables & functions).

Hosting : access anywhere below or after,

console.log( ) ← prints on consolo.

* Arrow function:

```
var getName = () => {
        console.log("Namaste Javascript")
    };
```

* Function.

```
function getName() {
        conso.log("Namaste Javascript")
    }.
```

another method to define function

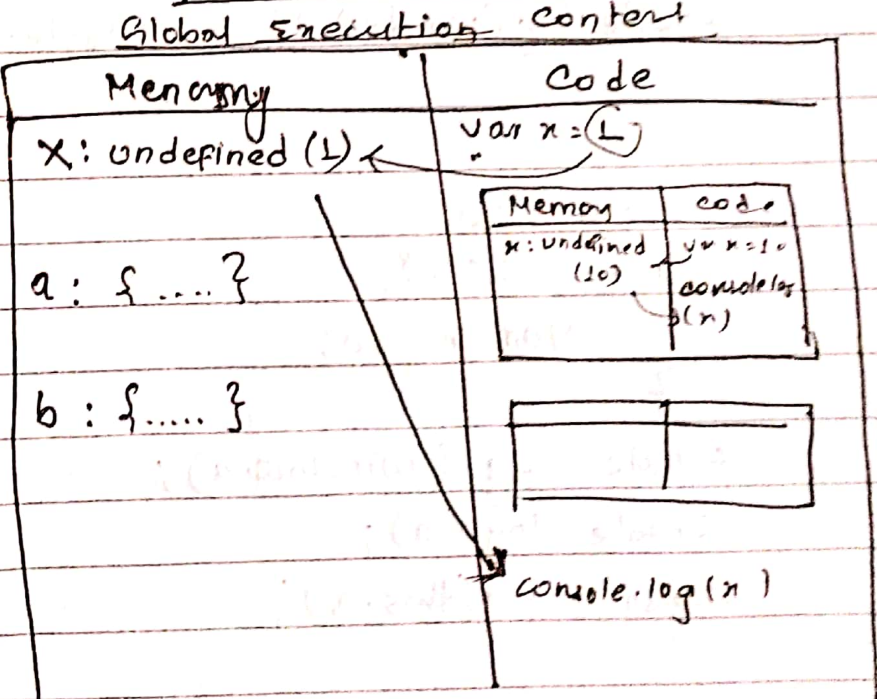
```
    var getName = function() {
          ____________
          }
```

⇒ call stack demo:

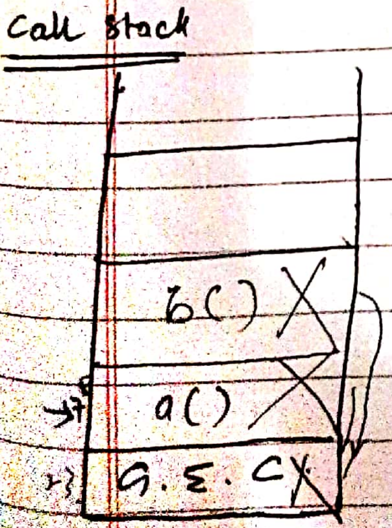
4 * How function work in JS.

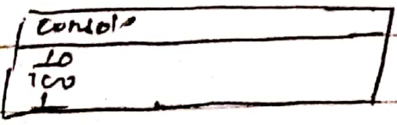
```
var x = 1;
a(); b();  ──→ console.log(x);
function a() {
    var x = 10;
    console.log(x);
}

function b(){
    var x = 100;
    console.log(x);
}
```

```
output : 10
         100
         1
```



Call Stack





Date:
Page:

* Shortest JS Program .

Empty file

# window
# This ← It points to window .

$$\boxed{\begin{array}{c}\underbrace{\text{this} === \text{window}}\\ \downarrow \\ \text{True}\end{array}}$$

Global Space : Any variable declared outside or not inside a
function.

console.log(window.a) = console.log (a) = console.log (this.a),

```
Var a = 10
function b() {
        var x = 10;
  }
console . log (window. a);
Console . log (a);
console. log(this. a);
```

Scanned with CamScanner

Date:
Page:

6. Undefined and Not defined

```
console.log(a);        // undefined
var  a = 7;
console.log(a);        // Defind and print as 7
console.log(x);        // not defined.
```

undefined ≠ Empty = but as called as placeholder.

```
a = 10;
if (a === undefined) {
        console.log("a is undefined")
}
else {
    console.log("a is not ~~do~~ undefined");
```

```
var a ;
console.log(a);
 a = 10;
console.log(a);
a = "hello world"
console.log(a);
```

⇐ Javascript : loosely type language.
∴ weakly language

$\underbrace{\hspace{6em}}$

⇩

variable can store anything

Scanned with CamScanner

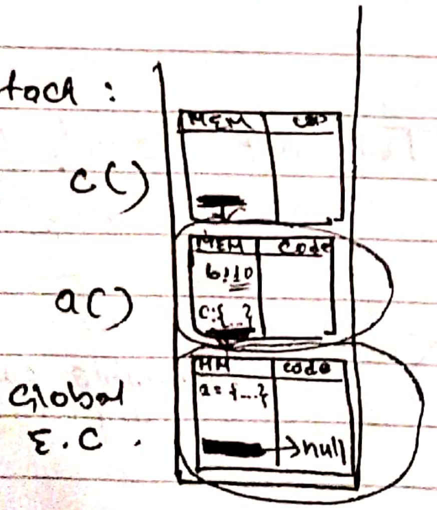
Date:
Page:

8. Scope & Lexical Environment, Scope chain

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

Scope: means where you can access a specific variable or function in our code.

Call stack :



Lexical Environment is the local memory along with the lexical environment of its parent.

Lexical : heirchial

Here # c function is lexically on a function
# a is lexically present inside Global E.C.

Chain of Lexical Environment ⇒ Scope chain.

Date:

Page:

1. Temporal Dead zone / let & const in JS.

"let & consts declarations are Hoisted".

```
console.log(b);          // undefined.
var a = 10;
var b = 100;
```

```
console.log(a);          // error. cannot access before.
let a = 10;                 or Reference error.
var b = 100;
```

```
let a =10;
console.log(a);          // prints 10.
var b = 100;
```

"Temporal dead zone" is the time since then when let variable is hoisted and till it is initialize some value. The time betwn them is temporal dead zone.

```
Let a = 10;              // Syntax error.
Let a = 100;
```

```
var b = 10;              // No error.
var b = 100;
```

Date:
Page:

```
Let a =10;
const b = 100;
```
} separate memory space.

```
let a;
const b = 1000;
a = 10;
console.log(a);
```
// fine and prints 10.

```
let a;
const b;
b = 1020;
a = 10 ;
```
// syntax error (missing initialization) ;
// missing initialization in const declaration

"const is more strict than let".
Const should be initialized and declared together.
let and const don't allow duplicate declaration."

1st place or constant	: Use const	①
	: Use let	②
	: later var	③.

Avoid temporal dead zone i.e move initializations error.

Scanned with CamScanner

Date:
Page:

```
let a = 1000;
const b = 1000;
 b = 1000;           // Type error. (Assignment to constant variable)
 a = 10;
let a = 100;         // Syntax error.
```

Reference error : when program tries to find a specific variable inside the memory space and you cannot access it

```
console.log(a);        ( Reference error) or when not defined
let a = 1900;
```

Three ways to declare a variable : const ①
let ②
var. ③.