How JavaScript Works?

- Everything in JavaScript happens inside an "Execution Context."

We can assume JavaScript Execution Context as a big box / Container in which whole J.S code is Executed.

Execution Context (looks like)

variable environment

Memory Component	Code
key: value	o———
x: 10	o———
fn: {...}	o———

thread of Execution

Execution Context Contains two Components first is Memory Component and Second is Code Component.

- Memory Component: "In memory Component where whole all variable and function are stored and which also have a named called "variable Environment." In this memory component all the variables and functions are stored in key value pairs.

In nutshell a memory component is a variable environment where all the variables function are stored in key: value pairs.

* Code Compouht: This is the place where code is actually executed. one line at a time.

It also known as Thread of Execution.
Thread of Execution is like a thread in which code is executed one line at a time.

* JavaScript is a Synchronous Single-threaded language.

Single threaded means it can only execute one Command at a time and Synchronous mean in Synchronous way means it execute next line when current line of Code has been executed already.

* What happens when we run JavaScript Code:

* The Execution is created in two phases.

1) Memory Creation Phase.
2) Code Execution phase.

Memory Creation phase:

Memory	Code

Simple J.S program

```
var n = 2;

function square (num) {
    var ans = num * num;
}

var square2 = Square(n);
var Square4 = Square(4);
```

So what is happening to the program in first phase

Memory Component	Code.
n: undefined.	
Square: {....}	
Square2: undefined	
Square 4: Undefined.	

So in phase 1 as we know J.S is a asynchronous single threaded language So first.

first line executed and in first phase the var n allocated a memory and set as undefined

* In case of variable it set as undefined in first phase

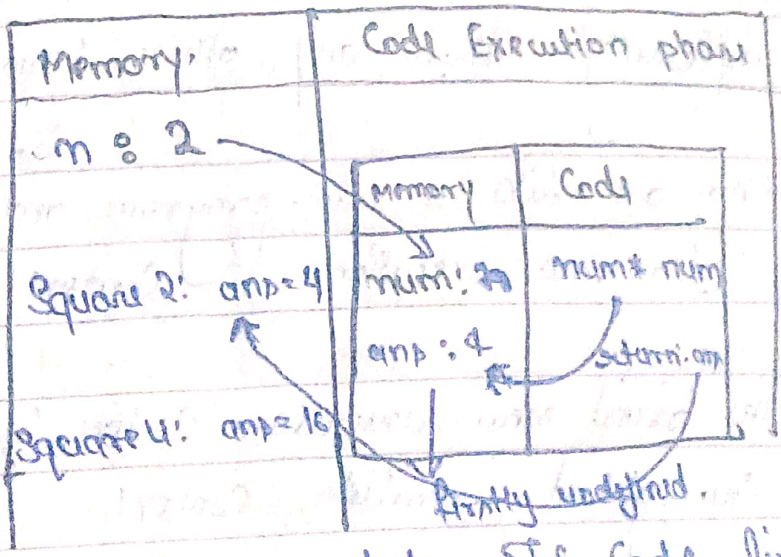
Then after that second line start executing.

* In Case of function the memory space store the whole code of function.

Then third line allocate memory to the last two variables Square2 and Square 4 as undefined as they

as variable.
Undefined in J.S is a special kind of keyword.

* Code Execution phase:



- Now J.S runs again the whole J.S Code line by line and it executes the whole Code and now the J.S execute the code and done all the calculation and function Execution.

- first line var n=2 it actually places 2 in the place of undefined.

- line no. 2 to 5 there is nothing to Execute now so it directly jumps to line 6. Firstly. So, here in this line function invocation is happening.

So what happen in functio invocation.

Whenever a function is invocated it invocate by using ().

In J.S functions are heart It behaves very differently than any other language.

* They are like a mini. program and for them a new execution Context has been created

The brand new execution Context is created inside the global execution Context.

- Then again code execution context own in two different phases.

- Now memory is allocated to the functions variables.

so now memory is allocated to the any and num variable in function.

- firstly stored undefined in phase 1 and then in phase 2 the num is allocated n which is passed "2" from above and

thats how num allocated as 2.

num: "Num" is parameter of the function which used in actual function.

n: n is an argument which has been passed during the function call. And value of n is setted to 2 in code phase and then passed to actual function.

- After first line execution then comes second line in function.

- In second line of function firstly ans is setted/allocated undefined and the calculation began in code execution box num * num which is 2 * 2 = 4 after calculation the memory setted for ans is replaced with 4.

- Now in line 4 the code/compiler see return keyword.

The compiler will say return the control of program from where the actual function was invoked.

Now control goes to line no. 6 again where we

calling function.

* After the Execution of function code the whole Execution Context for that code will be deleted.

So there is no execution context as soon as value is returned from function.

* After line 6 Execution

The Control / Execution goes to line 7 in line 7 we again Invoking function but this time the only different is we now only passing 4 directly as argument.

So In this again brand new executed Context Created and Some process is implemented as happen before.

After Execution "16" will be returned and Control back goes to line 7.

After the function returned the whole Execution Context will deleted after the Successful execution.

After the 7 line now there is nothing to Execute.
So after that the whole global Executing Context will be default.

That is how whole Code Execution happen.

Call Stack :

So We know that it is very hectic to manage the Execution Context.
what if the function call have another function call / Invocation so in this case the overall management become very clumsy so for proper management J.S use stack Implementation

Th last firstly the Global execution will be pushed to the Stack then the another local Execution will pushed when a function is Invoked after the Successful function Invocation the local Execution Context will be poped from function and lastly after whole Execution performed the last Global Execution Context will be poped

So the Stack is only for managing the Executing Context Creation and Deletion managed by the LIFO approach.

The stack is known as "Call Stack"

After whole Execution the call stack get empty and then we are done.

"Call Stack" maintains the order of Execution of execution Contexts.

There are many names for Call Stack.

1. Execution Context Stack.
2. Program Stack.
3. Control Stack.
4. Runtime Stack.
5. Machine Stack.

Hoisting JavaScript:

Hoisting is a behaviour in JavaScript where variable and function declarations are moved to the top of their Containing Scope during the Compiling phase. This means we can use a variable or function in our code before it's declared in the source code, and javascript will still be able to execute the code without errors, However it's essential to understand how hoisting works to avoid unexpected behaviour.

Both function and variable hoisting behaves differently

Variable Hoisting:

- When we declare a variable using 'var', 'let' or 'const' the declaration moved to the top of the current function or global scope.

- Proper Definition: As we know that when we run our js programm a global Execution Context is created which is divided into two phase 1. Memory Allocation phase 2. Code Execution phase Even before the code execution memory is allocated to variable which are present

in the Scope which is known as "undefined" So that's why when we console the variable before initialize it. The JS Engine will not show error it will print undefined it is known as hoisting in J.S.

- However only the declaration is hoisted, not the initialization. This means variable is used but it's not assigned a value until it's encountered in the code.

Ex: Console.log(x);
var x = 5;
Console.log(x);

undefined
5

2) function hoisting:

- function Execution/declarations are also hoisted to the top of their containing Scope, which means we can call a function before even defining it.

Proper ① Defination: Like in variable hoisting & global execution Context is created but incase of function js engine not assign undefine to a function, It actually assign the whole function body to the function during memory allocate and if we want to access before assign it will print whole function in the Console.

Ex: sayHello();
Console.log(Say Hello);

function sayHello() {
Console.log("Hello World");
}

Hello World
function SayHello()

In Case of arrow function J.S engine assign them undefined keyword because it is also like variable

var getName = () => {
Console.log("Hello")
}

~~Reading And writing to a file~~

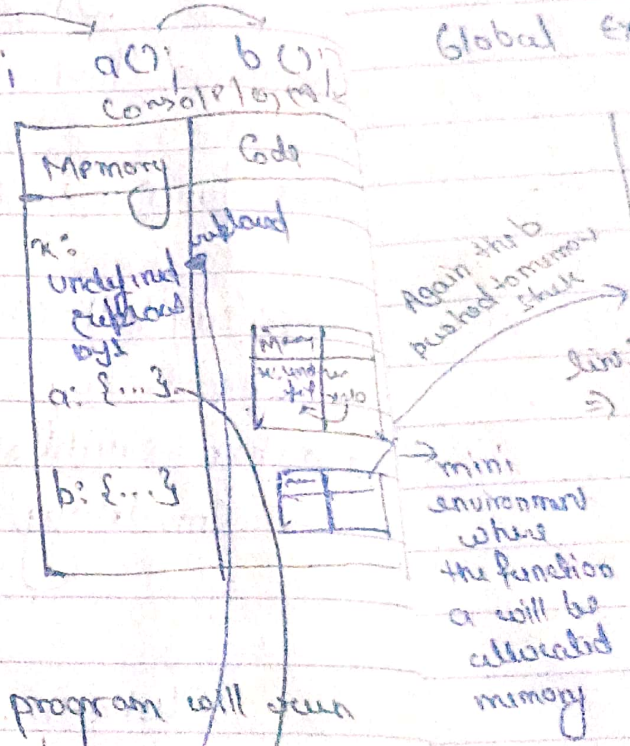
~~Reading and writing in a file~~

~~Const readline~~

How functions works In JS.

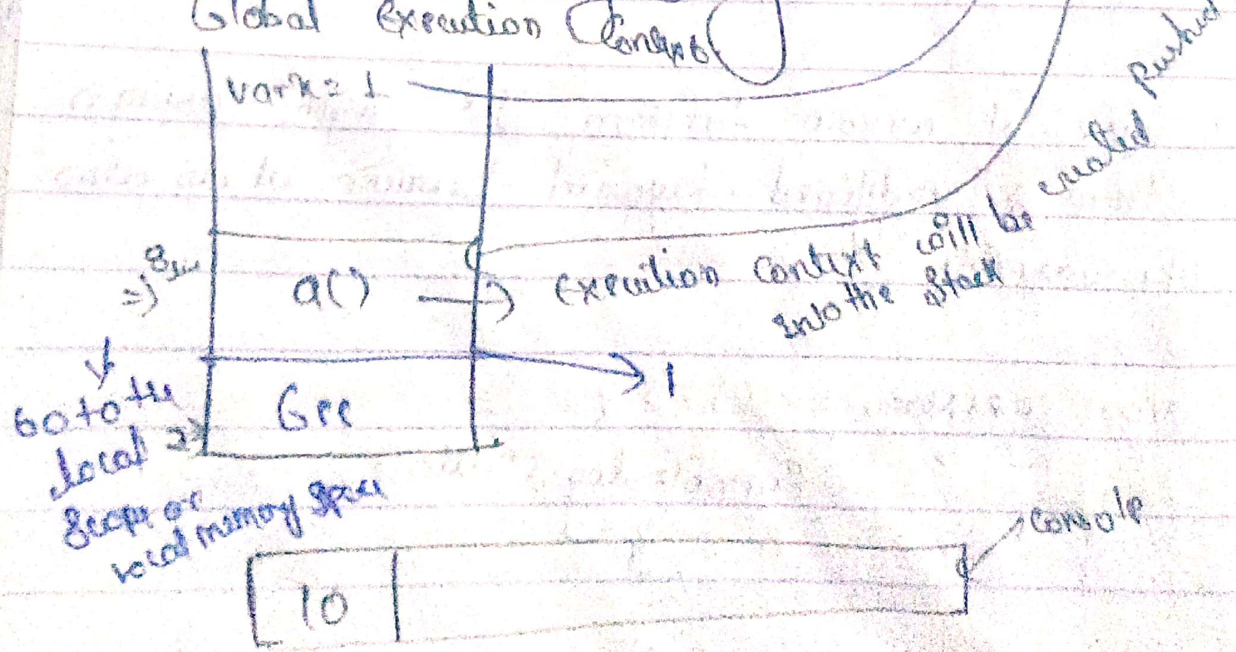
for examples →

```
function a(){
    var x=10;
    console.log(x);
}
function b(){
    var x=100;
    console.log(x);
}
var x=1;   a();   b();   console.log(x)
```



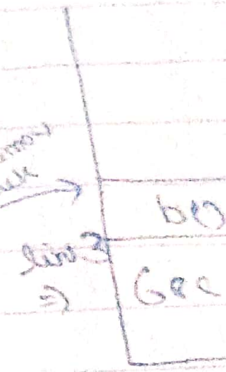
So because of hoisting this program will run and not give any error.

Global Execution Context



After Successful Exe...

The Execution Context ... Call Stack ... Global Execution ...

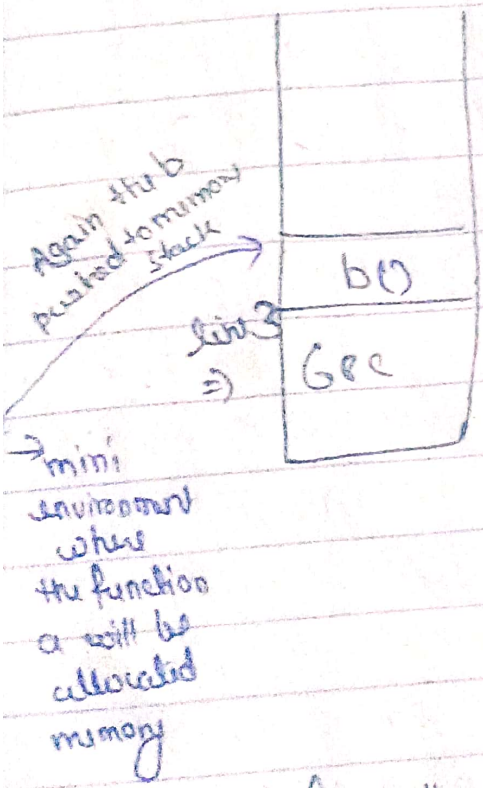


Now at ... after check... Execution...

After Con... ... not ... -cution ...

After Successful Execution of the function then

- The Execution Context will pop out from the Global Execution Call Stack and Control will go to Global Execution Context line 3



Again a local Execution will created and Control goes to that after Compiling then the b() will be poped out also

- Now at last the Control goes to the Gec and after checking Control goes to line 4 of Global Execution Context and execute

After Compiling x check it in local Context and print in Console and whole code is run and now all the progrm executed and the Global Execution Context will pop out from Stack.