For everything that looks easy, there lies an underlying force doing the hard work for you.

- *Dr. Homi Bhabha*

Advanced Javascript

Day 2- How Javascript works

What is your name?

你叫什麼名字？

¿Cuál es tu nombre?

Behind every language, there lies a processing engine to understand it.

For humans, its Brain.

For computers, it’s Compiler.

Js Engines

Wherever Js is used, there exist a JS engine helping computer to understand the code you wrote.

Chakra

SpiderMonkey

v8

If there are different Js engines understanding my code differently, do I have to write different Js code for each one of them?

Next time you get an error while writing Js, just know that Js engine is doing it’s job of helping you and the computer communicate in a proper way.

How does that process looks like?

Hoisting

Hoist

To Raise

Hoisting helps keep the attendance sheet of a Javascript Program.

var name = “Raj” //Declaration and Assignment

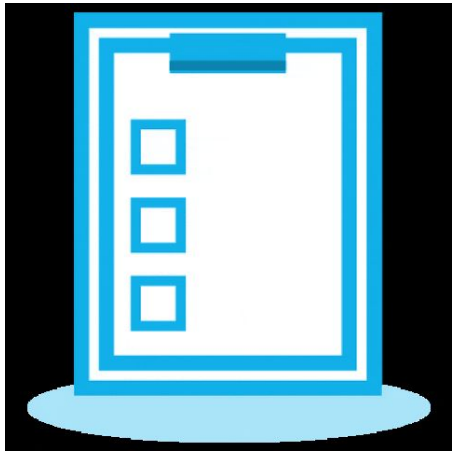
Becomes

var name; // Declaration
name=”Raj” // Assignment

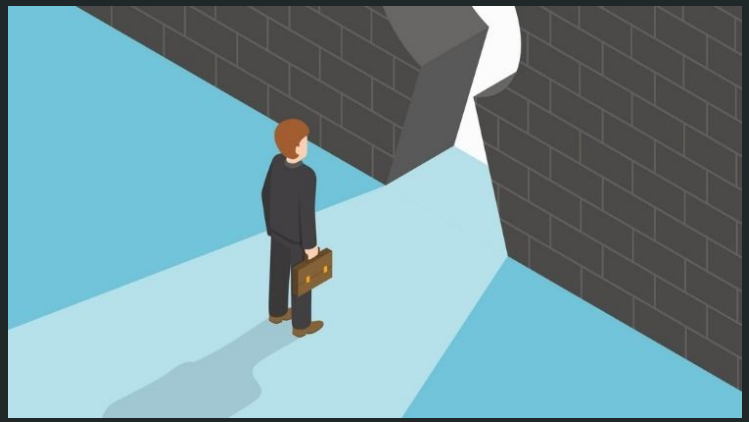
Hoisting only applies to Variables and functions.

Also, it only considers declaration part and not the actual assignment.

All hoisting cares about is what variables and functions are present in a program.

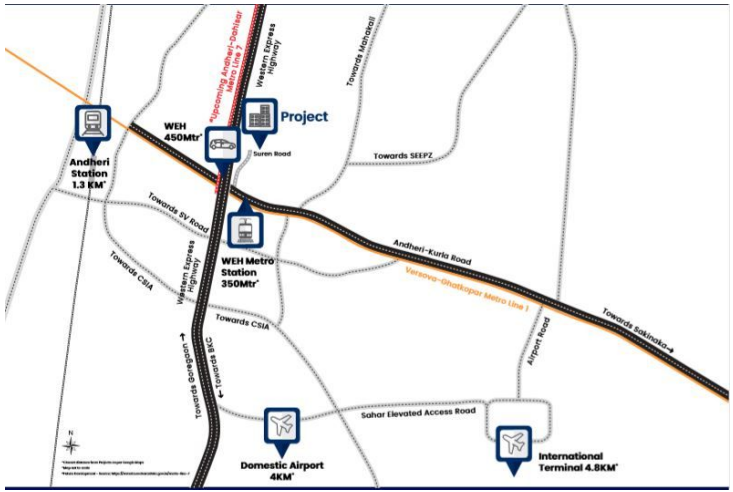


Execution Context



Context

background, environment, or surroundings of events or occurrences.



Execution Context

Environment in which the JavaScript code is executed.

By environment, I mean the value of this, variables, objects, and functions JavaScript code has access to at a particular time.

Depending upon which part of code Js is running, execution context changes.

It might seem complex but it changes only when it is running a code inside of a function.

Types of Execution Context:

Global Execution Context

Functional Execution Context

Global Execution Context

This is the default execution context in which JS code start its execution when the file first loads in the browser.

Functional Execution Context

Functional execution context is defined as the context created by the JS engine whenever it finds any function call.

When a JavaScript engine executes a script, it creates execution contexts. Each execution context has two phases:

1. Creation phase

2. Execution phase.

During this creation phase, it performs the following tasks:

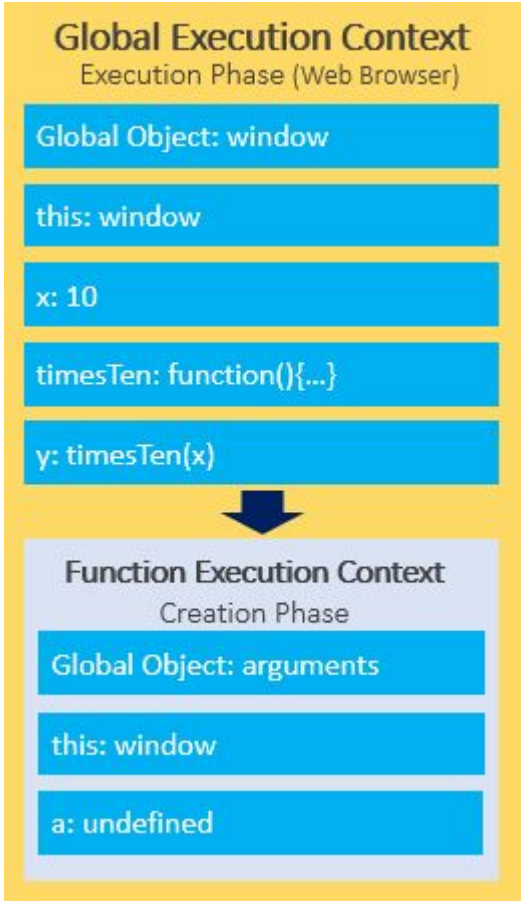
1. Create a global object i.e., window in the web browser.
2. Create a ‘this’ object which points to the global object above.
3. Setup a memory for storing variables, objects etc.
4. Hoisting

```
var x = 10;

function timesTen(a){
    return a * 10;
}

var y = timesTen(x);

console.log(y); // 100
```

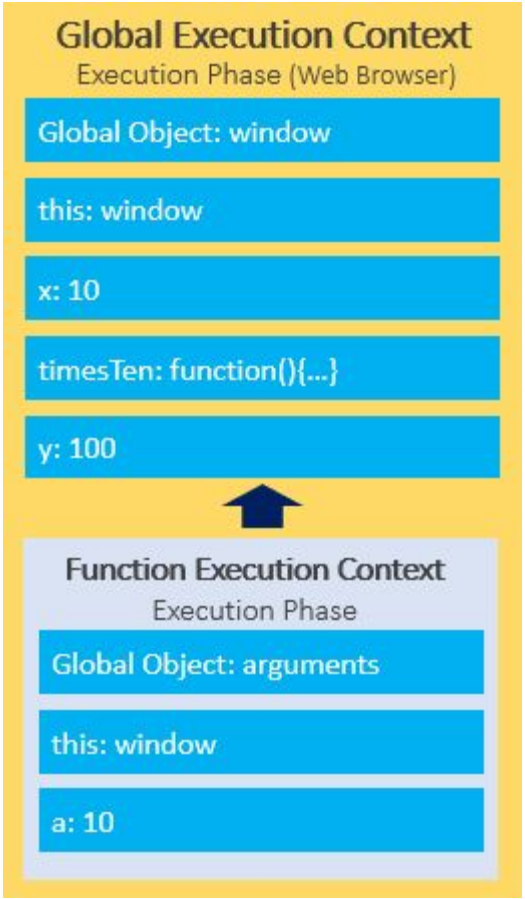
During the execution phase, the JavaScript engine executes the code line by line. In this phase,

1. It assigns values to variables
2. Executes the function calls.

For every function call, the JavaScript engine creates a new Function Execution Context.

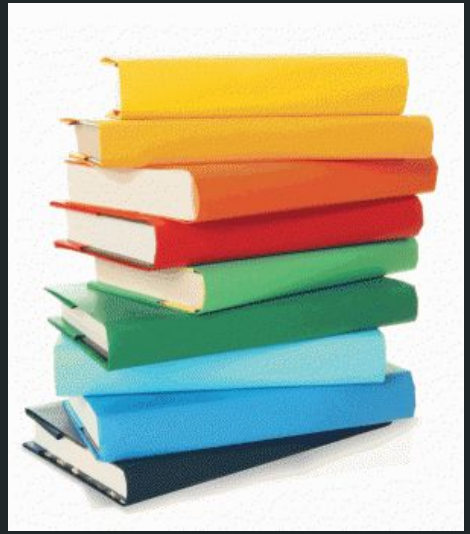


Communication between Global and Functional execution context.



To keep track of all the execution contexts including the Global Execution Context and Function Execution Contexts, the JavaScript engine uses a data structure named Execution stack.

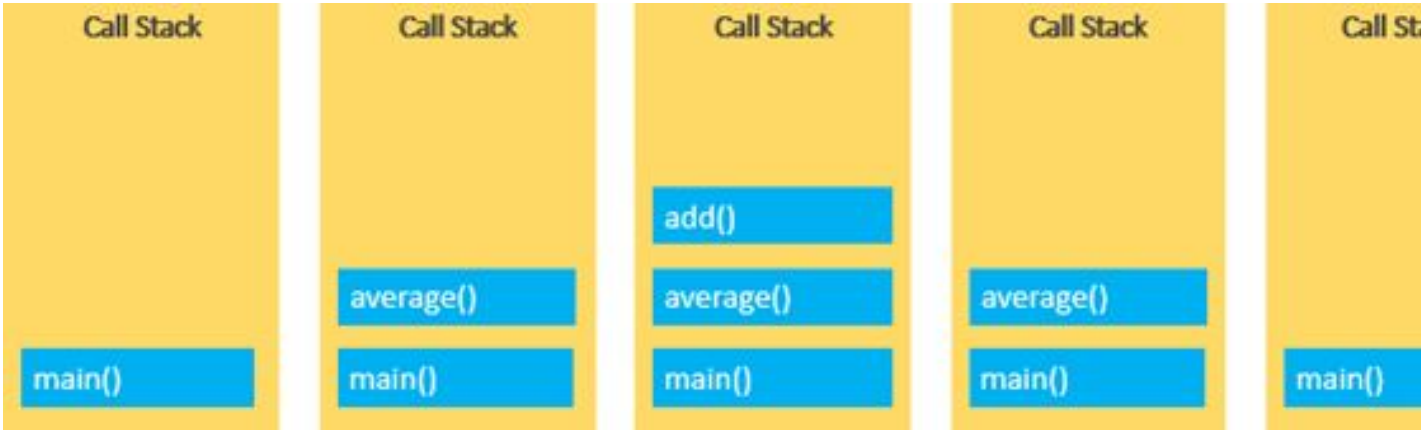
Execution Stack



JavaScript engine uses a call stack to manage execution contexts: the Global Execution Context and Function Execution Contexts.

The call stack works based on the LIFO principle i.e., last-in-first-out.

```
function main()
{
average()
}
function add()
{
}
function average()
{
add()
}

main()
```



Variable Scope

Scope

Range or Coverage.

What is Variable Scope in Js?

Scope determines the visibility and accessibility of a variable. JavaScript has three scopes:

1. Global scope
2. Local scope
3. Block scope.

Global Scope

When you execute a script, the JavaScript engine creates a global execution context.

It also assigns variables that you declare outside of functions to the global execution context. These variables are in the global scope.

They are also known as global variables.

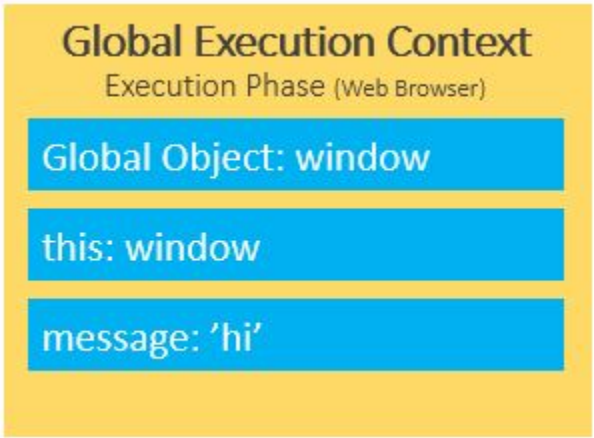
Global Scope

When you execute a script, the JavaScript engine creates a global execution context.

It also assigns variables that you declare outside of functions to the global execution context. These variables are in the global scope.

They are also known as global variables.

```
var message = 'Hi';
```

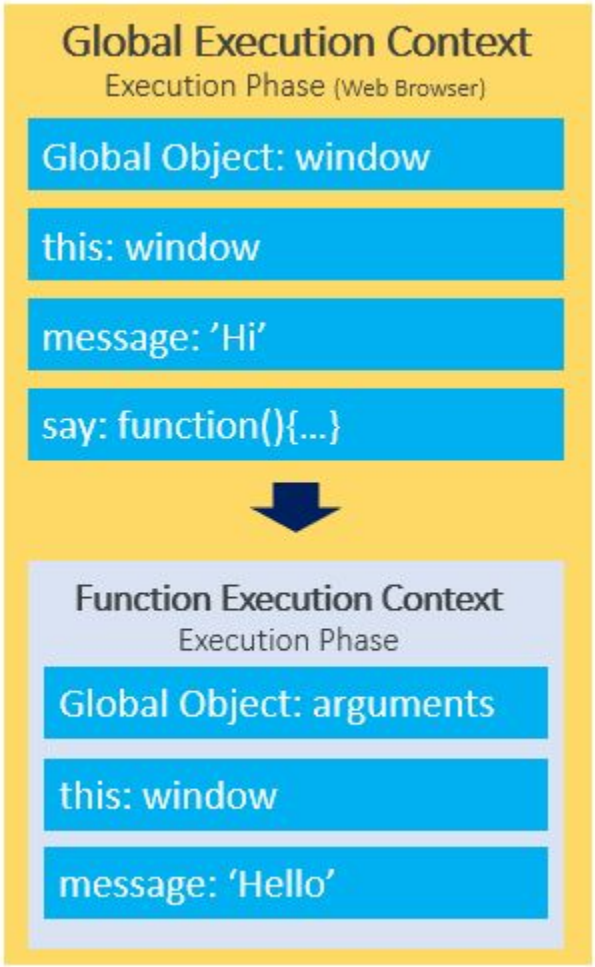


Local Scope

Variables that you declare inside a function are local to that function.

They are called Local Variables and can’t be accessed in Global Scope.

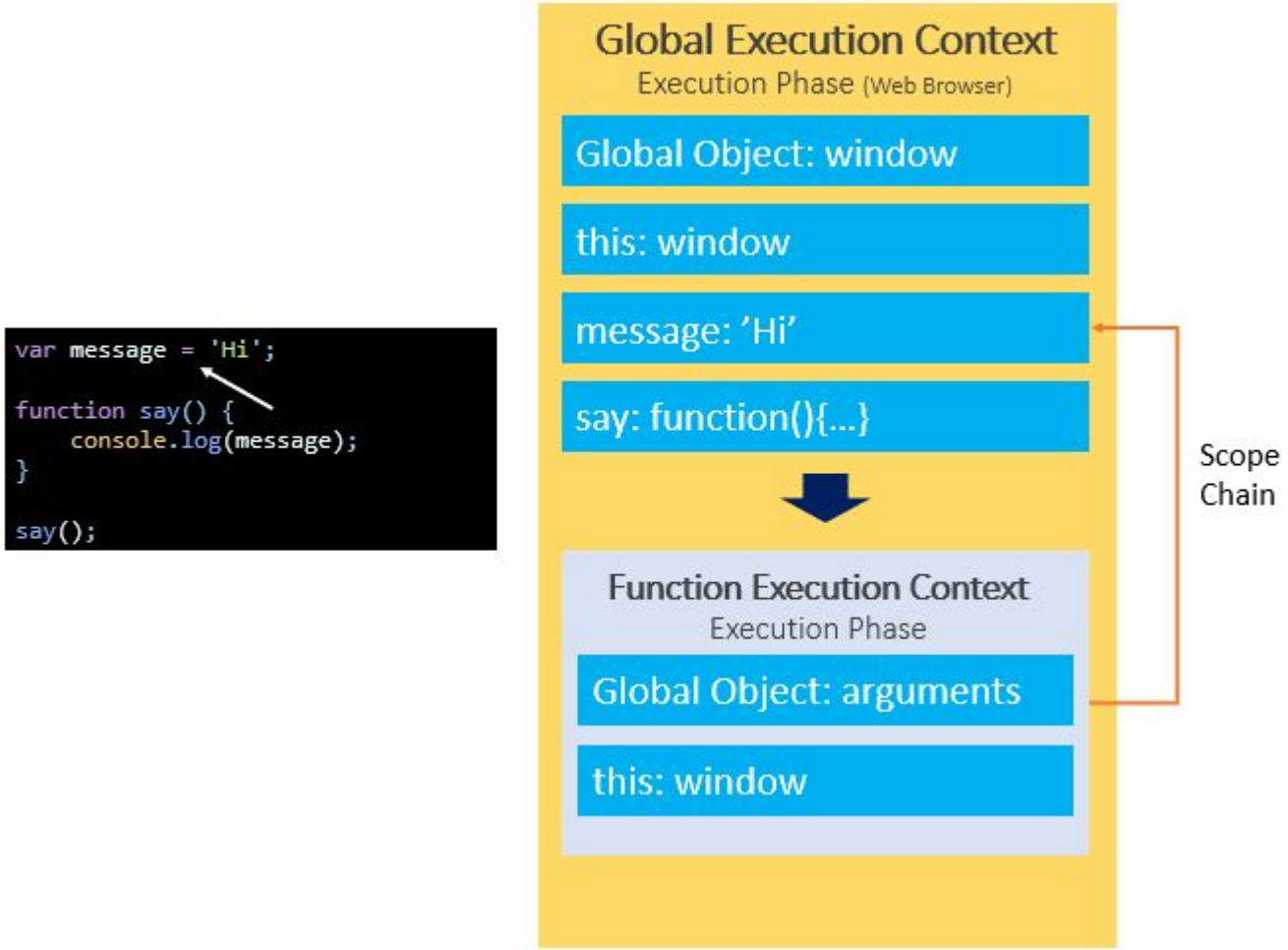
```
var message = 'Hi';

function say() {
    var message = 'Hello';
    console.log('message');
}

say();
console.log(message);
```



Scope Chain

Global Scope

JavaScript resolves a variable by looking it in its current scope, if it cannot find the variable, it goes up to the outer scope, is called the scope chain.


var message = 'Hi';
function say() {
console.log(message);
}
say();
Global Execution Context
Execution Phase (Web Browser)
Global Object: window
this: window
message: 'Hi'
say: function(){...}
Function Execution Context
Execution Phase
Global Object: arguments
this: window
Scope
Chain

Let’s Recap

Hoisting -

Execution Context -

Execution Stack

Scope chain