**Global Execution Context Details:**

Global Execution Context is a wrapper to help manage the code that is running In your code you will see lot's of lexical environments means areas of the code between { } but which one is currently running is managed via execution contexts.it can contains your code and it also can contains beyond what you've written in your code.

Execution Context runs in 2 phases - the first phase involves:

1. Global Object
2. This
3. Outer Environment

In this phase the parser runs through your code and set up what you have written for translation. It recognizes where you have created variables and where you have created functions. So it sets up in the creation phase that memory space for the variables & functions.

This step is confusingly called “Hoisting”. All it means is that before your code begins to be executed line by line. The javascript engines has already set aside a memory space for the variables that you have created in the entire code also for the functions. So those functions & variables exist in the memory.

So when the code begins to execute line by line it can access them.

However, when it comes to function, its entirety is placed into memory space i.e. function name and other details.

But when it comes to variables in the execution phase that where these kind of assignments are set i.e.

var a = “Hello world”; If you console.log (a); before the value has been assigned then it will always return undefined because in the execution phase it sets the value against the variable. If you console.log the value of a then it literally means “I don’t know what the value of this is as yet” but because of the first phase it knows that in the memory space variable a in defined but not its value i.e. var a; till the parser reached the line where the assignment value is being assigned.

All variables in JavaScript are initially set to “Undefined”. And function in its entirety is sitting in its memory. In other words, it sets up the memory space for all the variables & functions that are going to be used.

That’s why it’s a bad idea to rely on hoisting. In other words you’re allowing JavaScript code to take your code and make decisions instead you should code in a manner to avoid such situations.

**An example of Javascript Hoisting:**

b(); // this will return the function and its value

console.log(a); // this will return undefined

var a = "Hello World!";

function b() {

    console.log('called b!');

};

**Ideal code example to prevent hoisting:**

var a = “Hello World”;

console.log(a);

function b() {

console.log(‘’called b!");

}

console.log(b);