Table of Contents

[1. Chapter Name 2](#_Toc128213771)

[1.1 How JavaScript Works 2](#_Toc128213772)

[1.2 Execution Context 3](#_Toc128213773)

[1.2.1 Variable Object 4](#_Toc128213774)

[Hoisting 4](#_Toc128213775)

[Function hoisting 4](#_Toc128213776)

[Variable hoisting 4](#_Toc128213777)

[1.2.2 Scope chain 5](#_Toc128213778)

[Lexical scoping 5](#_Toc128213779)

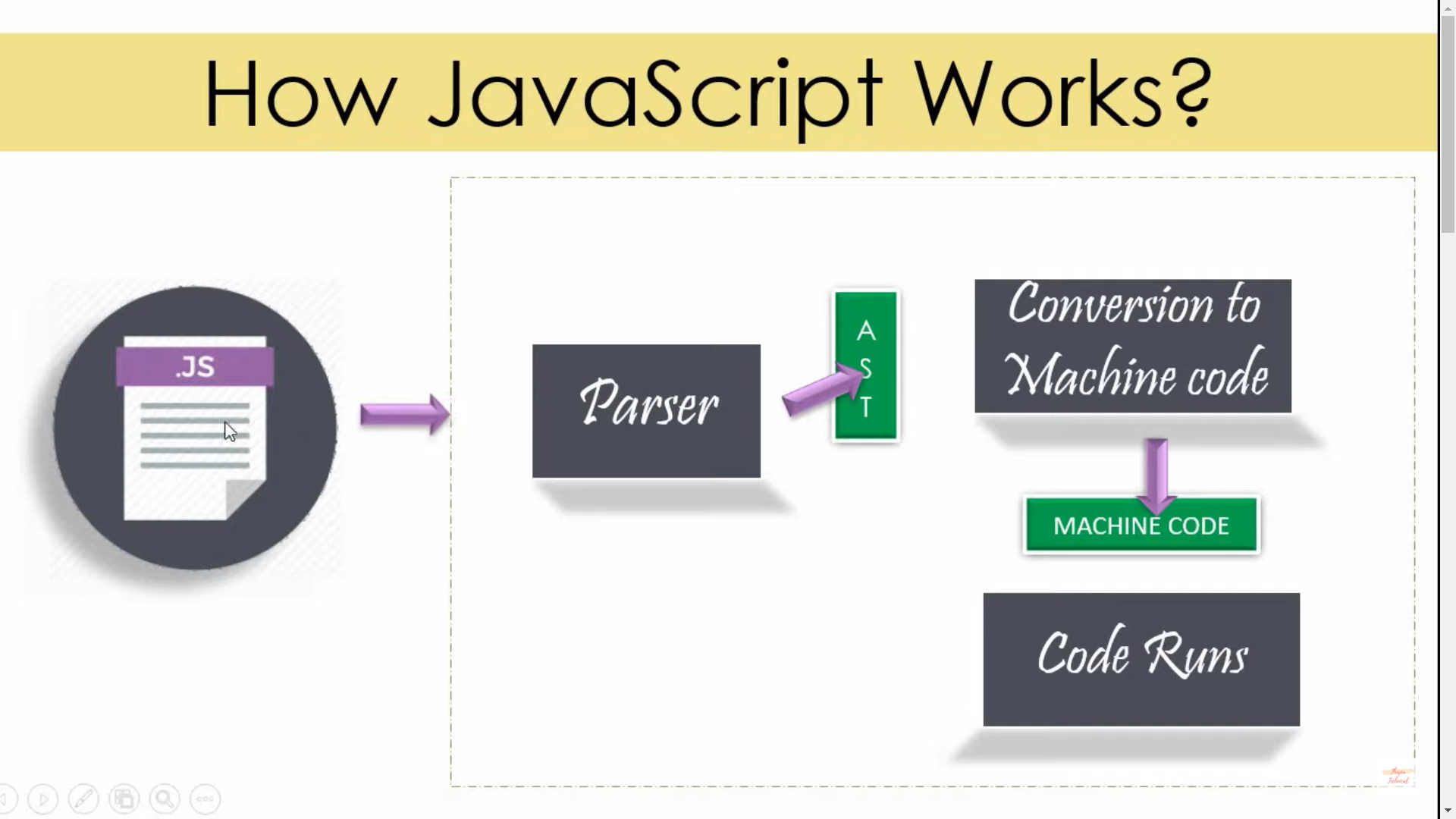
[1.3 Execution Stack 6](#_Toc128213780)

[1.4 How To Check Execution Context or Stack for Code. 7](#_Toc128213781)

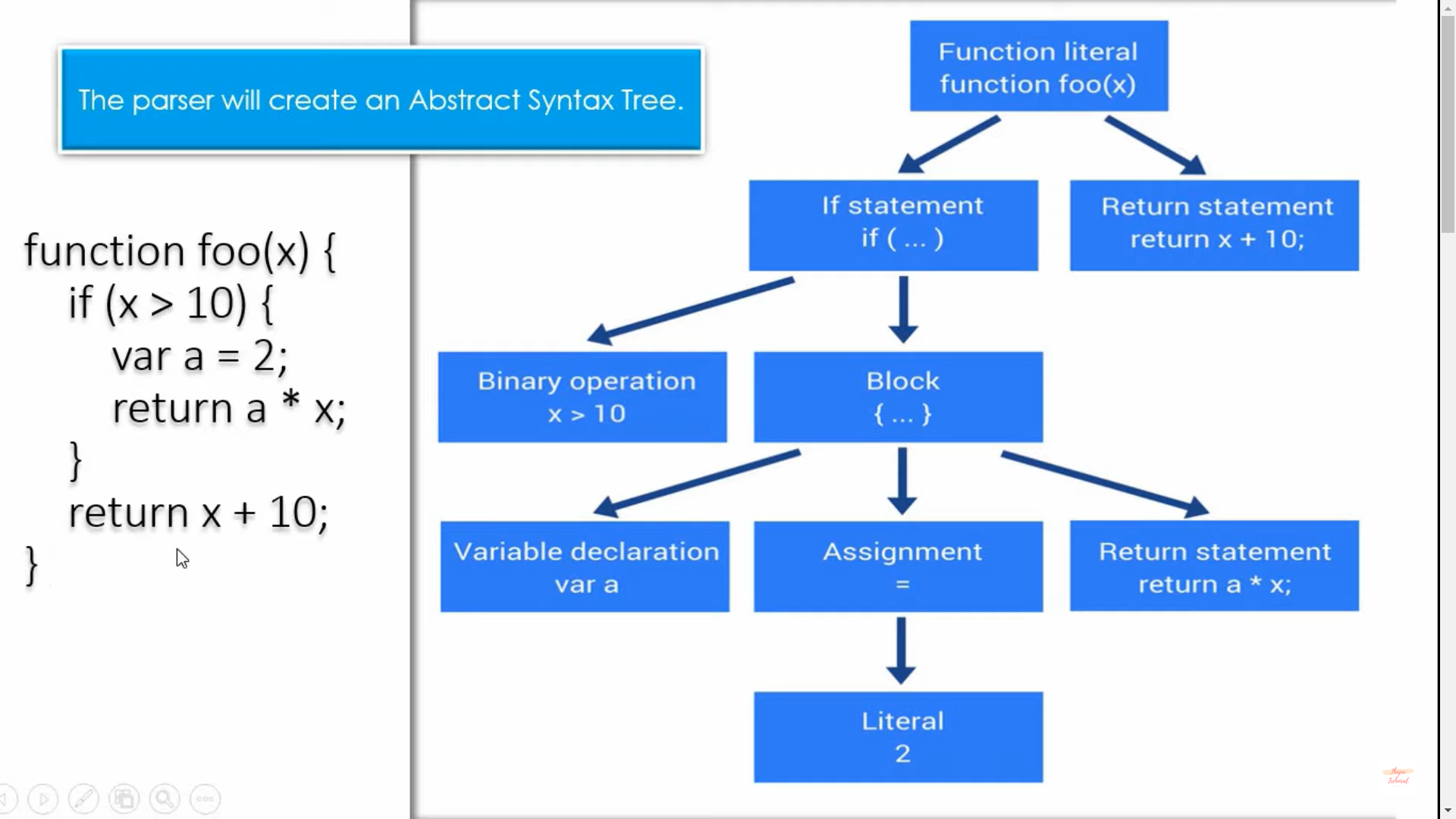
1. Chapter Name

# How JavaScript Works

Java script is interpreted language. Client-side scripting language. Each browser has JavaScript engine. Chrome have v8, safari have safari, Mozilla have spider monkey,



Engine have parser which parse the code. Means its check code line by line, code haven’t an error then it will produce data structure(AST-Abstract Syntax Tree). Of your code.



Then code convert into machine code. Once code convert into machine code then produced file run and get output

**Note :** You can produce AST for your code From <https://astexplorer.net/>

# 1.2 Execution Context

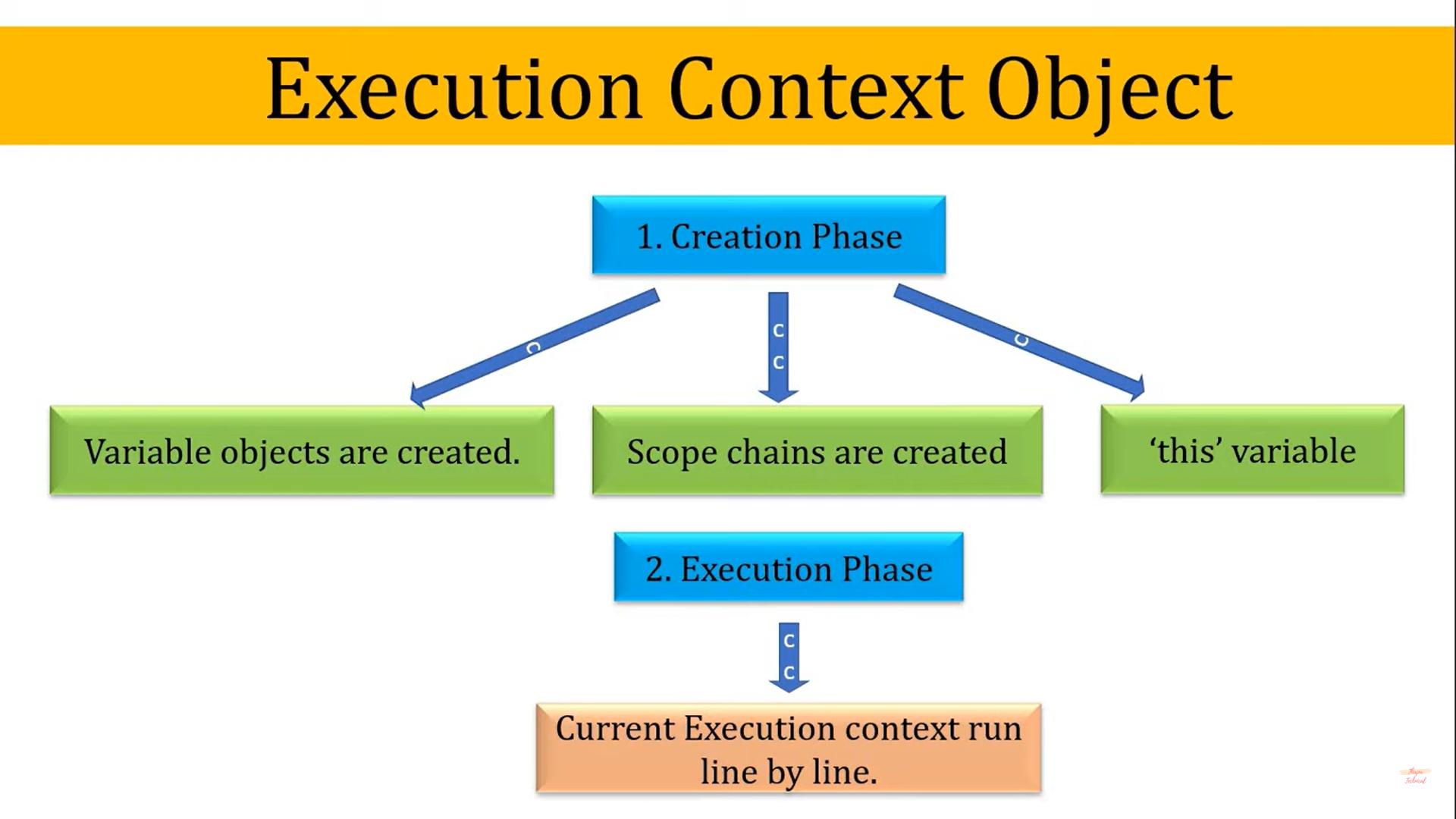
Browser followed his process to run the JavaScript code. But how actually it run, for this execution and evaluation we need an environment which is called execution context in execution context we have default global execution context

Javascript engine create the Global execution context before it starts to execute any code. Variable and functions that is not inside any function its stored in global execution context.

global execution context is associated with global object. Ex this === window it will give true value. you can run this in console you will get window object where your all data will be stored.

When we call function then each function creates new execution context for itself automatically

Execution context have three properties **variable object, scope chain, this variable** and have two phase **creation phase** and **execution phase**



For each function (function declaration) a property is created in this variable object which is pointing to that function. If argument passed in function argument object are created that were passed into the number.

Now for each variable (variable declarations) a property is created in the variable object. Which is then set to undefined.

Variable declaration and function declaration are move to the top of their scope before the code execution. It’s called **Hoisting**

## 1.2.1 Variable Object

### Hoisting

### Function hoisting

Sum(5,10);// call first and the declare

Function sum(a,b)

{

add = a+ b;

Console.log(add);  
}

Now the output is possible because variable and function declaration are moved to the top of their scope before the code execution. That means above code execute like that

Function sum(a,b)

{

add = a+ b;

Console.log(add);  
}

Sum(5,10);// call first and the declare

You can use function expression then hoisting is not work. Means function store in variable. In Modern JavaScript function hoisting not work .

### Variable hoisting

For let keyword javascript hoisting not work

console.log("I am a " + job);

var job = "web developer";// move top

console.log("I am a " + job);

function whoAmI(){  
 console.log("I am a " + job);

var job = "web developer";

console.log("I am a " + job);

}

whoAmI();

this code executes =>

var job = undefined;

console.log("I am a " + job);

var job = "web developer";

console.log("I am a " + job);

function whoAmI(){

var job = undefined;  
 console.log("I am a " + job);

var job = "web developer";

console.log("I am a " + job);

}

whoAmI();

this code executes

## 1.2.2 Scope chain

The scope chain is used to resolved the value of variable name in javascript. Scope chain in javascript lexically(where function write) defined, which means that we can see what the scope chain will be by at the code.

Lexical scoping **:** a function that is lexically within another function get access to the scope of the outer function (inner function can get access to their parent functions variable but vice versa is not true)

var a = "hello guys";

function first() {

var b = " How Are You ?";

second();

function second() {

var c = " My Self Parmar Aayush";

console.log(a + b + c);

}

thired(); // give error a not find

}

first();

function thired()

{

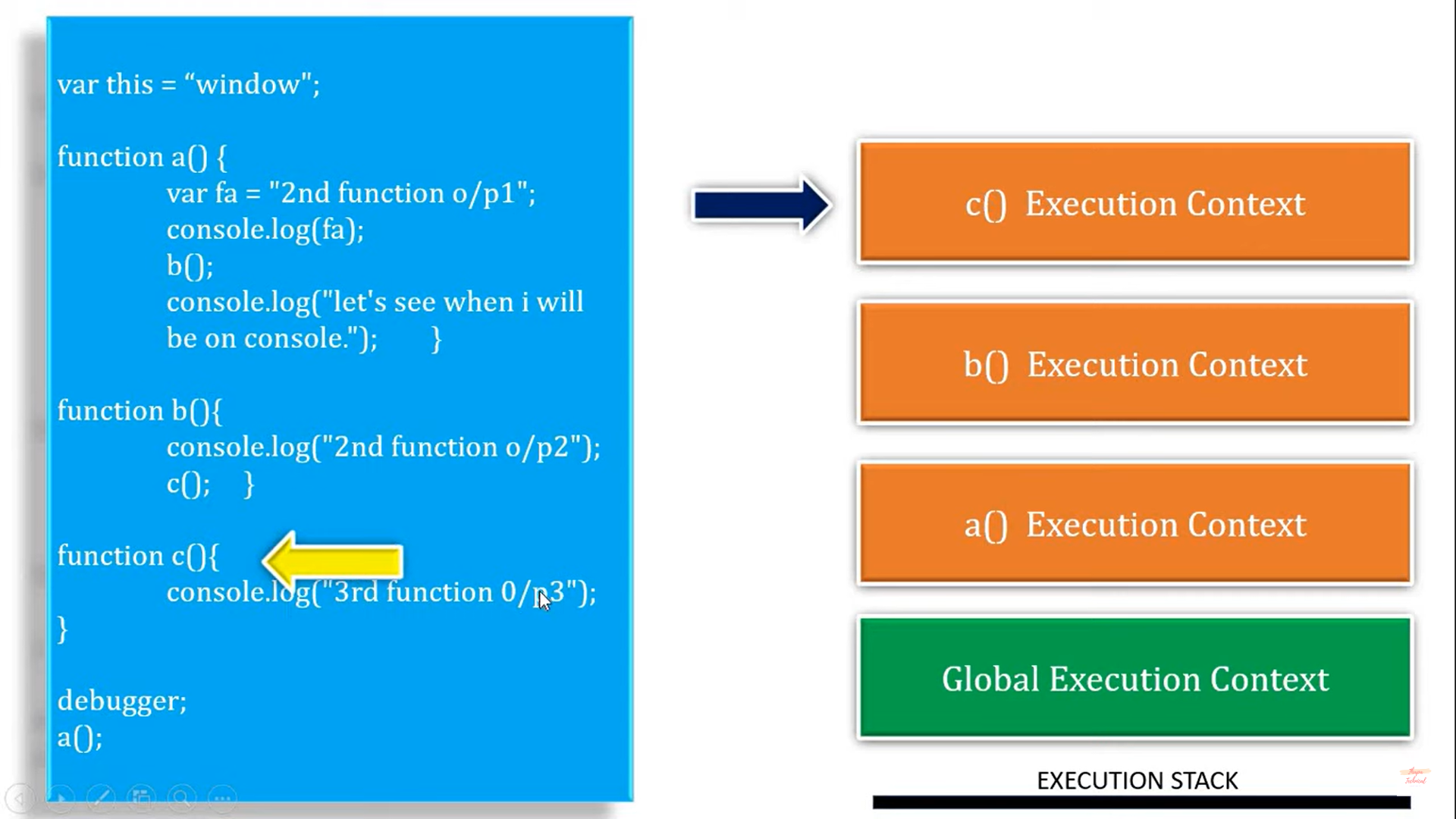
var d = "it will Give error";

console.log(d+a+b+c);

}

# 1.3 Execution Stack

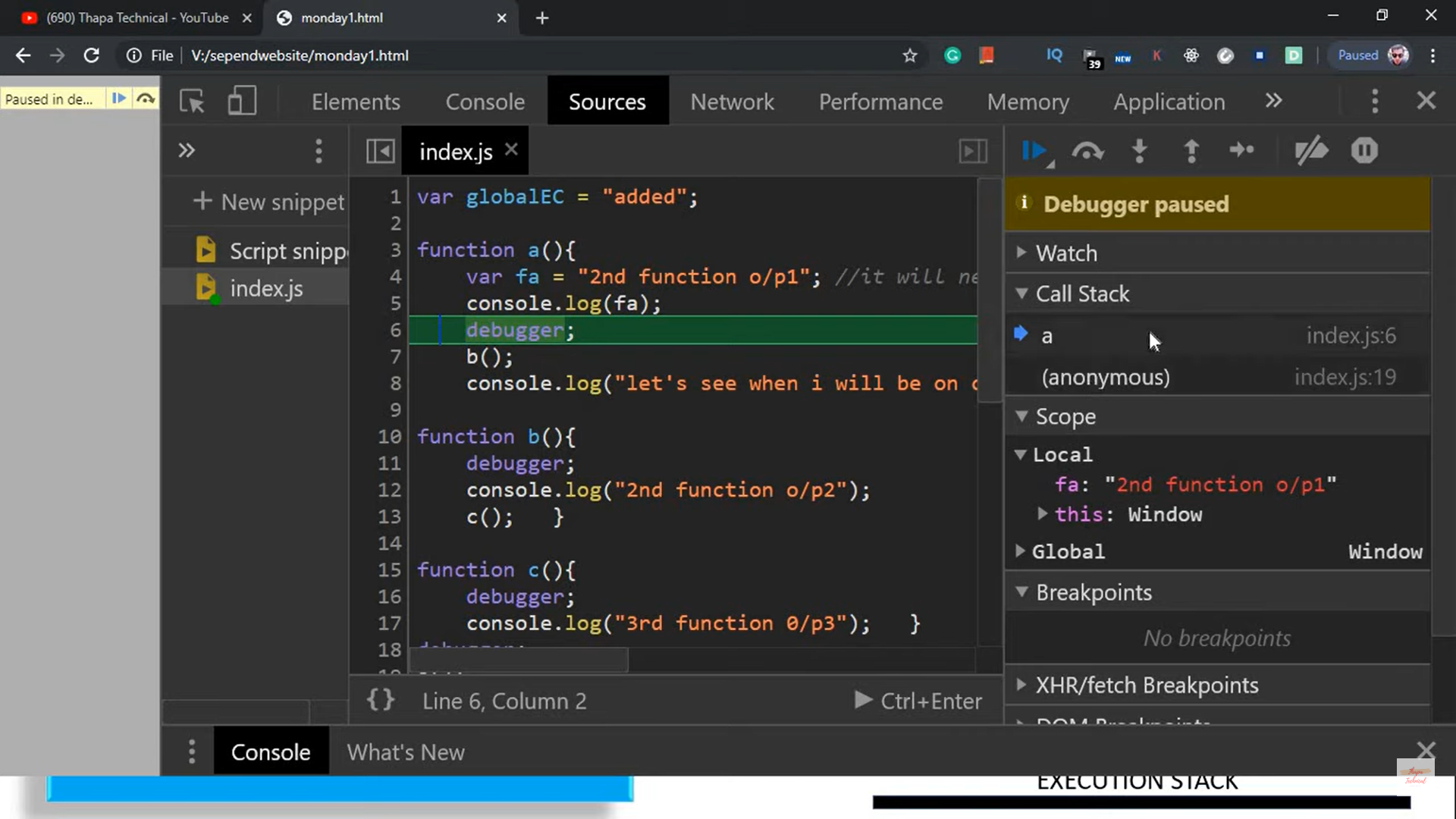
Execution stack also known as calling stack is a stack with a LIFO structure which is execution context created during the code execution.



After pointer reached at c() Execution context and it will POP out in LIFO Manner

# 1.4 How to Check Execution Context or Stack for Code.

Write code and in Source option in developer mode and create new file and write your code inside this file. Now you can see right side and click on next section and tap one by one see execution of code



Anonymous is global execution context. After run whole code pointer will get back to the anonymous block which is global execution context.

# 1.5 This keyword

This keyword refer to an object in which objects its depends on where to invoked from function, from object , alone, from object method binding

|  |  |
| --- | --- |
| **Invoked from** | **Response** |
| Alone | Window object |
| From object method binding | Parent object |
| From events | HTML Element |
| Regular function | Global object or Window |

**Example**

const profile = {

name: "aayush",

qualification: "Diploma in CSE",

sum: function () {

console.log(this.name);

var add = 2 + 2;

console.log("Sum Of Two Number is" + add);

console.log(this); // it refer to profile object

function childMethod() {

var name = "parmar";

console.log("Child Method Name : " + name);

console.log(this); // it refer to window object or global object

}

childMethod();

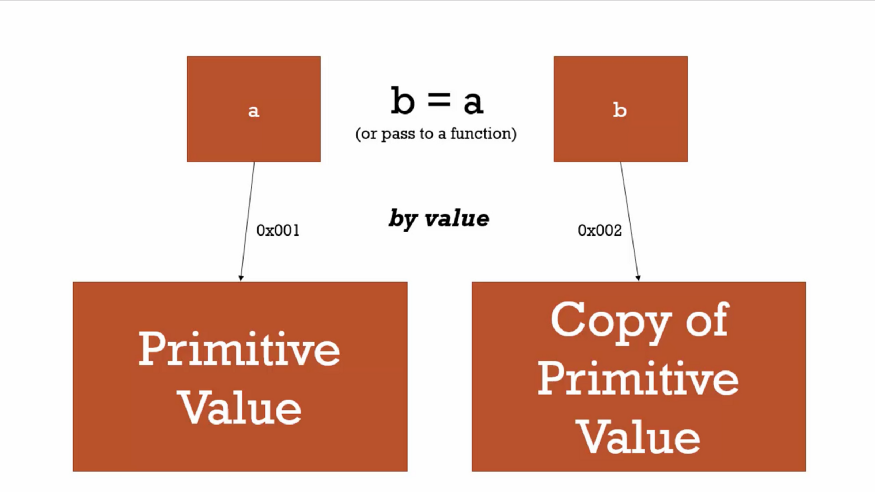
}

}

profile.sum();

# 1.6 Pass By Value And Pass By Reference

**[1] Pass By Value**



When you assign value of variable “b” from variable “a” and print value of “a” and “b” both are same. You perform any operation on variable “b” or a its totally independent from each other. Only change performed on operand “b” value not affect to copied “a” variable value.

let a = 10;

let b = a; // pass by value

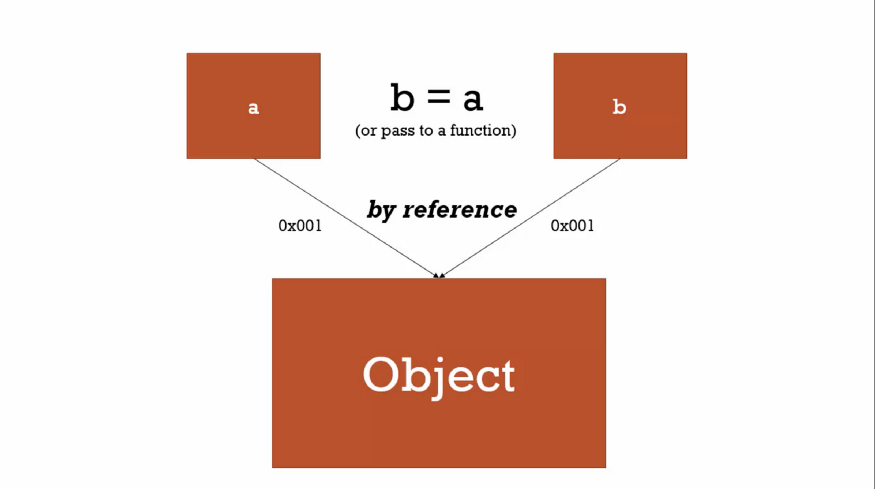
console.log("\n Value Of A : " + a + "\t\t Value Of B : "+ b); // 10 10

a++

console.log("Value Of A after increment : " + a); // 11

console.log("Value Of B after increment : " + b); // 10

**[2] Pass By Reference**



When you assign the value of **“object a”** from **“object b”** and print value of a and b both are same. You perform any operation on object “b” or “a” its totally dependent from each other. Change performed on object “b” values it is affect to object “b” as well as “a” both.

const obj1 = {

userName : "Aayush",

password : "Aayush@123",

}

const obj2 = obj1;

obj2.password = "Change@123";

console.log(obj1); // change both object password

console.log(obj2); // change both object password