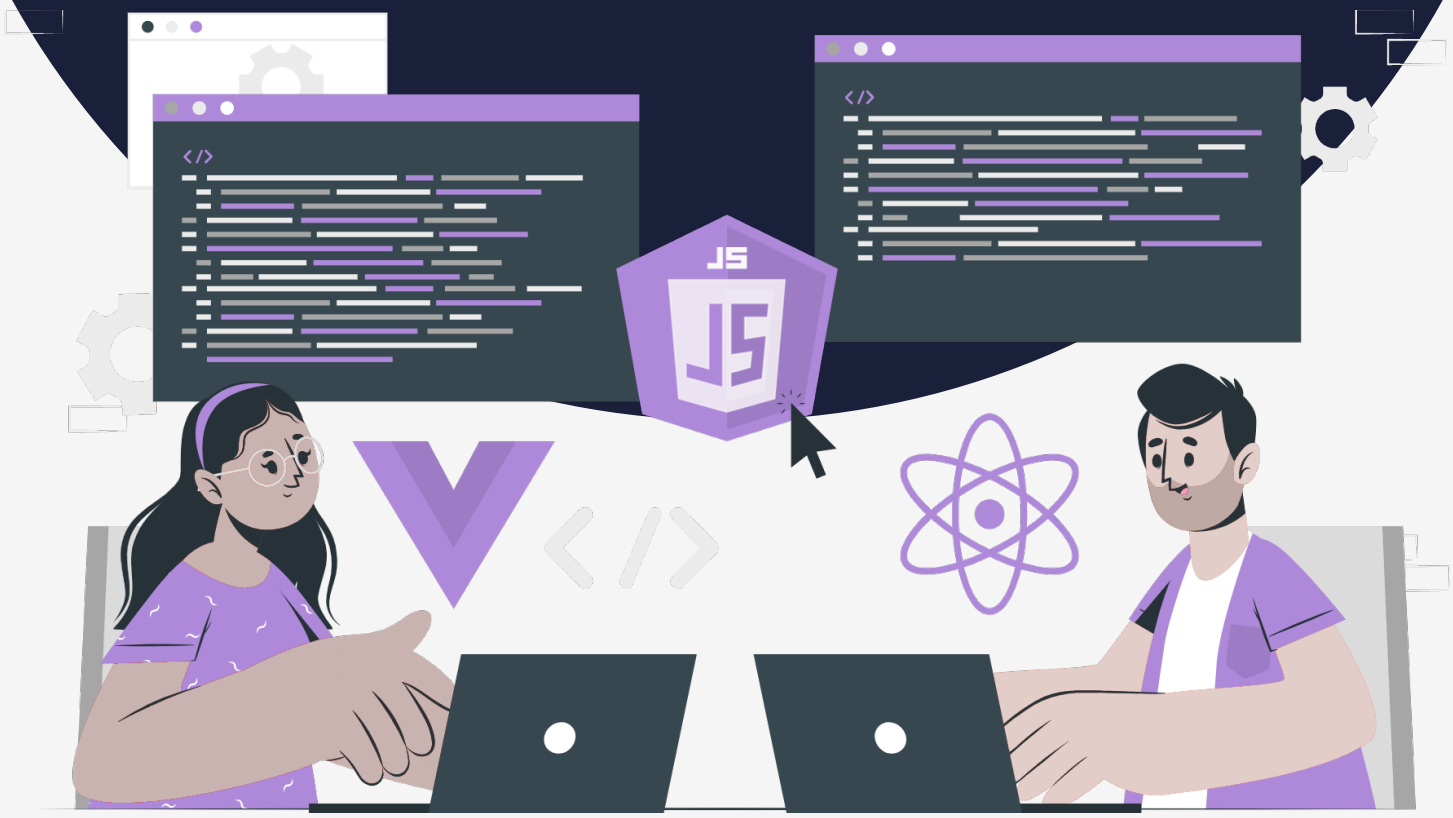
Lesson:

Execution Context



Topics Covered:

1. Introduction.
2. Execution Context.
3. Types of Execution Context.
4. Eval() function.
5. Creation of execution context.

Execution context is an important concept that helps us understand how code runs in JavaScript. Execution context refers to the environment in which a piece of code is executed. This environment includes information about variables, functions, objects, and whatever the js engine will execute. Understanding the execution context is essential because it helps us write code that runs efficiently and accurately. By managing the execution context, we can control the behavior of our code and ensure that it executes in the way we intend it to. We'll be exploring the execution context in more detail, in order to gain a better understanding of this fundamental concept in JavaScript.

Execution Context
Execution context is like a toolkit for a piece of code in JavaScript. Just like a toolkit has all the necessary tools for a builder to complete a project, the execution context has all the necessary information for a piece of code to execute. The toolkit contains tools like hammers, nails, and screwdrivers, while the execution context contains variables, functions, and objects.

The JavaScript engine acts like the builder, using the tools in the toolkit to complete the project. Similarly, the JavaScript engine uses the execution context to execute the code. Understanding the execution context is important because it allows us to control the behavior of their code and ensure that it runs smoothly and efficiently, just like how a builder can use the right tools to complete a project quickly and accurately.

To put it in simple terms, the Execution context is the environment in which a piece of code is executed in JavaScript. It includes all the information the code needs to run properly, such as variables, functions, and objects. By managing the execution context, we can control the behavior of their code and ensure that it executes correctly.

Types of the execution context
There are three types of execution context in JavaScript:

1. Global execution context.
2. Local/Function execution context.
3. Eval execution context.

Global Execution Context
Global execution context is the outermost and default execution context, and it is created when the JavaScript code is loaded. The global execution context contains all the globally defined variables, functions, and objects.

Local/Function Execution Context
When a function is called, a new execution context is created for that function. This execution context has its own local variables and functions, as well as access to the variables and functions in the outer scopes.

Eval Execution Context
Before understanding the eval execution context it is important to understand the eval() function in javascript.

Eval() Function
eval() is a built-in function in JavaScript that allows you to execute a string of code as if it were part of the current script. When the eval() function is called, it takes a string argument that contains the code to be executed.

```
let x = 10;
let y = 15;
let result = eval("x + y");

console.log(result);

// OUTPUT: 25
```

In the above code, the eval() function is used to evaluate the string "x + y", which is a simple arithmetic expression. The result of the expression (x + y = 10 + 15 = 25) is then stored in the variable "result" and printed to the console using console.log().

In modern JavaScript, the use of eval() is discouraged, and there are usually safer alternatives to achieve the same result. Instead of using eval(), it is generally recommended to use other JavaScript features like functions or object literals.

Eval Execution Context
The eval() function creates a new execution context whenever it is called. This is the Eval execution context.

Creation of Execution Context
Execution Context is created in 2 phases

1. Memory Creation Phase
2. Code Execution Phase

The memory creation phase is the first stage of the execution context creation process in JavaScript. During this phase, the JavaScript engine sets up the memory space for the execution context, including the variable environment and the scope chain.

The scope chain defines the order in which JavaScript looks for a variable or function when it is referenced in your code.

When you define a variable or function in your code, JavaScript creates a new scope that can access variables and functions defined in its outer (parent) scope. If the variable or function is not found in the current scope, JavaScript looks in the next outer scope, and so on until the global scope is reached.

It is essentially a list of nested scopes that the JavaScript engine uses to search for a variable or function when it is referenced in your code.

In the Memory Creation phase, all the variables are assigned memory and are initialized with a special value called undefined. All the function definitions are stored in memory.

The code execution phase is the second stage of the execution context creation process in JavaScript. During this phase, the JavaScript engine executes the code in the current execution context line by line. Variables are initialized with the given value.

Seeing all this we could think that we write so much complex code and how javascript keeps a track of it and executes it. To support execution context we also have a call stack to keep track of. We will be looking at the call stack in the next lecture.