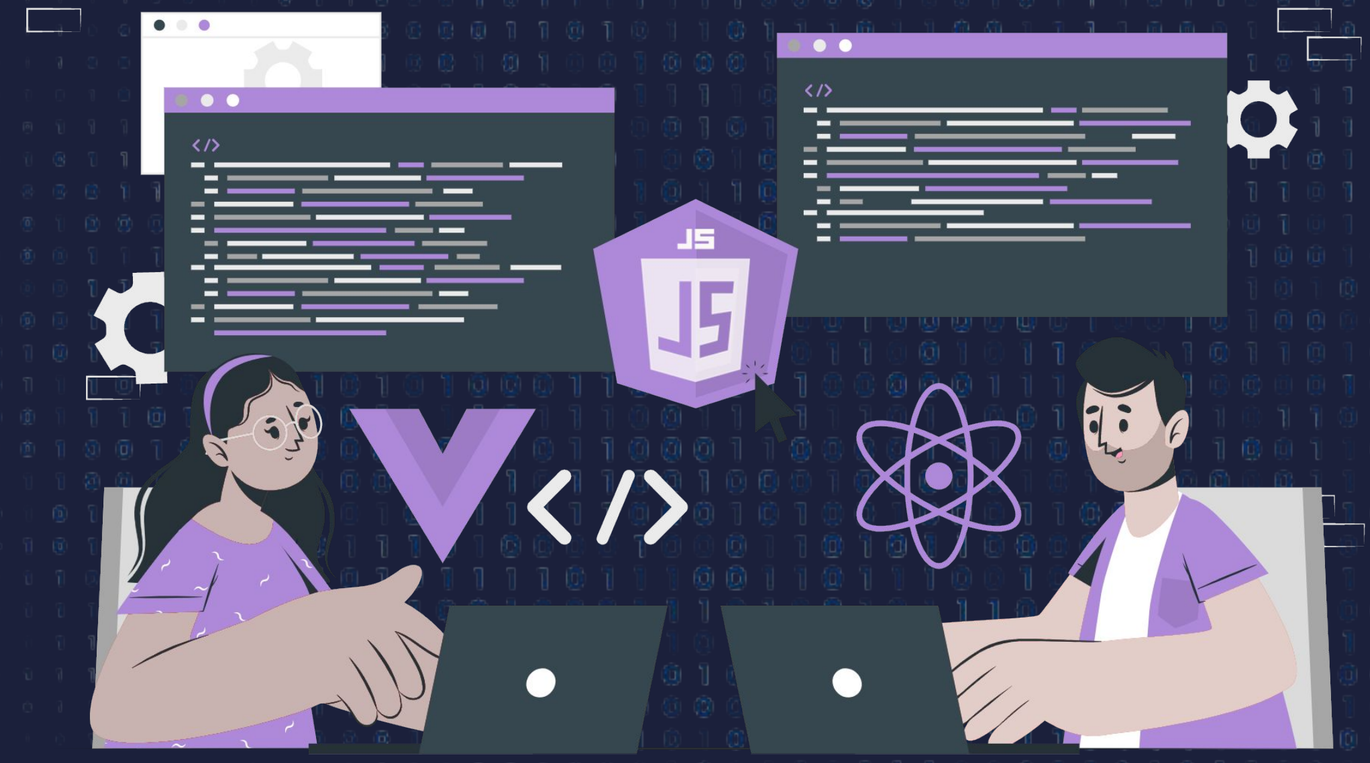
PW SKILLS

Call Stack



Lecture CheckList

1. Introduction.
2. What is call stack?
3. Call stack location.
4. Callstack: How it works?
5. Stack overflow and underflow.
6. Visualising Call Stack.

Introduction

The call stack is a fundamental concept in JavaScript and is crucial to understand how functions and code execution work. In JavaScript, every time a function is called, it is added to the call stack. When the function is finished executing, it is removed from the stack, and the execution continues from where it left off. A call stack is an essential tool for tracking the order in which functions are called and helps prevent stack overflow errors. Understanding how the call stack works is crucial for any developer to write efficient, bug-free code. We will be looking at call stack in depth in this lecture.

What is call stack?

The call stack in JavaScript is a data structure that helps manage the execution of functions. Whenever a function is called, it is added to the call stack. When the function completes its execution, it is removed from the stack. This process follows the Last-In-First-Out (LIFO) principle, meaning that the last function added to the stack is the first one to be executed and removed from the stack.

Call stack location

The JavaScript call stack is an internal data structure that is maintained by the JavaScript engine. It is present in the browser's runtime environment or on js runtime like Node.js. The call stack is a part of the JavaScript runtime environment and is not accessible directly.

Callstack: How it works?

1. Whenever a user executes a script, the JavaScript engine generates a Global execution context and places it at the top of the call stack for execution.
2. Whenever a function is called, the JavaScript engine generates a Function execution context and places it at the top of the call stack.
3. If a function calls another function, the JavaScript engine generates a Function execution context for the called function, adds it to the top of the call stack, and starts executing it.
4. After a function finishes executing, the JavaScript engine removes it from the call stack, allowing the execution of other functions stored in the stack to continue.

Stack overflow and underflow

The call stack in JavaScript has a limited amount of space available, and if this space is exceeded, it can lead to a "stack overflow" error. This error occurs when there are too many nested function calls, and the call stack becomes full, causing it to run out of memory. When this happens, the JavaScript engine cannot add any more function execution contexts to the stack, and the program terminates with an error message.

On the other hand, if there are no more function execution contexts left in the call stack, and we try to pop an execution context, it leads to a "stack underflow" error. This error occurs when the program tries to remove an execution context from an empty call stack. This can happen when we try to pop more functions than what is present in the stack.

Visualising Call Stack

Let’s understand the working of the call stack in depth by considering an example.

THANK YOU