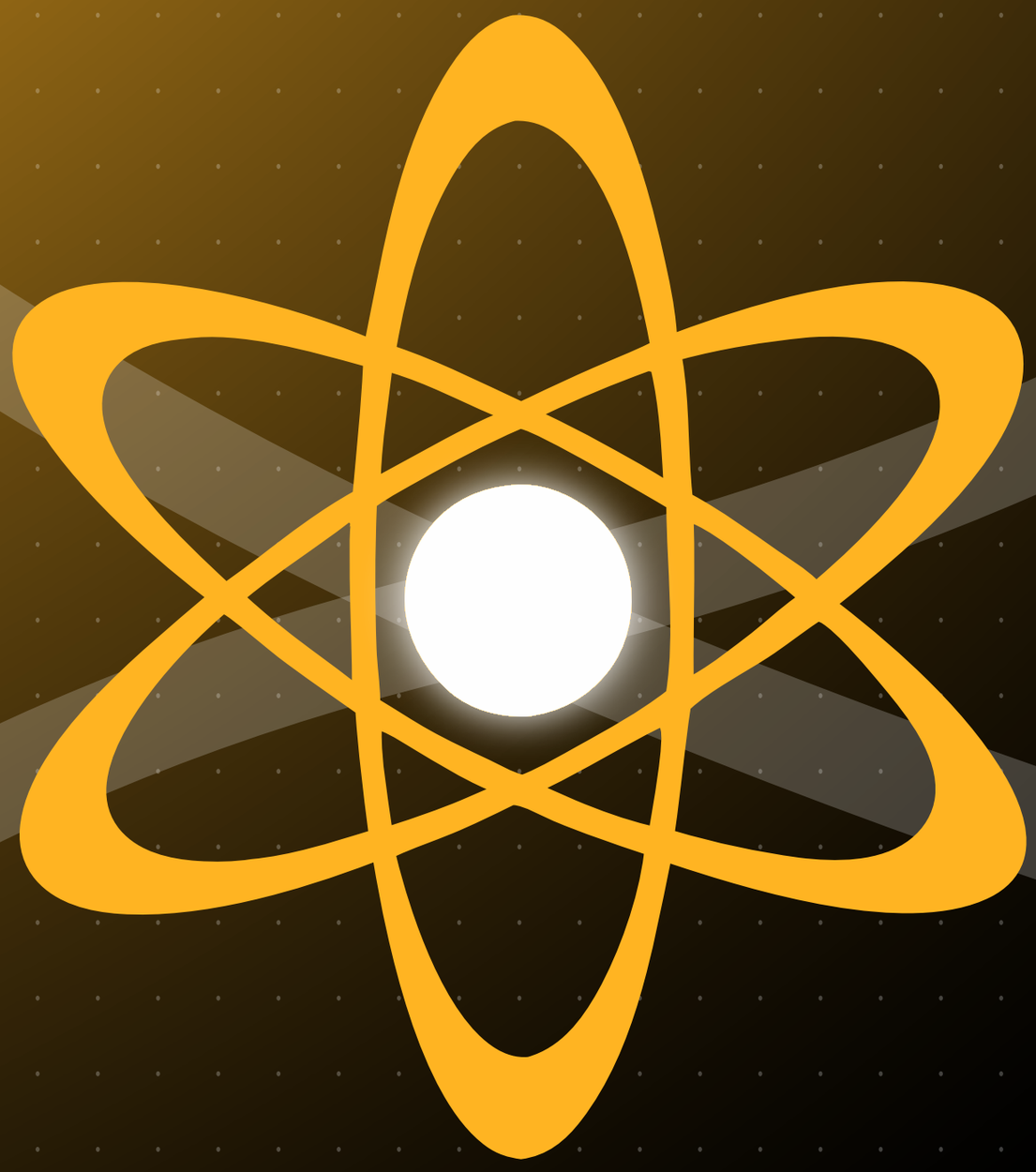
Understanding JavaScript Series Part 3: Hoisting & Execution Contexts



SUMANTH M
Frontend developer

1

What is Hoisting in JavaScript?

Definition: Hoisting is a behavior in JavaScript where variables and functions are accessible before they are declared in the code.

Key Insight: Hoisting doesn’t mean moving the code to the top; it’s a way JavaScript allocates memory before execution begins.

Why It Matters: Hoisting affects how and when variables and functions are initialized, which can impact the program flow.

SUMANTH M
Frontend developer

2

How Does Hoisting Work?

Memory Creation Phase: JavaScript allocates memory for all variables and functions. Variables are initialized with undefined.

Code Execution Phase: Variables are assigned their actual values, and functions are called.

```
console.log(x); // undefined
var x = 10;
console.log(x); // 10
```

Explanation: Even though x is declared after console.log(x), JavaScript allows access to it due to hoisting, initializing it with undefined

SUMANTH M
Frontend developer

React Js

3 Function Declarations vs Function Expressions

Function Declarations:

Fully Hoisted: The function code is hoisted and available before it’s declared.

```
greet(); // "Hello"
function greet() {
  console.log("Hello");
}
```

Function Expressions:

Hoisted as Variables: Only the variable declaration is hoisted, and the function itself is undefined until assigned.
it’s declared.

```
greet(); // Error: greet is not a function
var greet = function() {
  console.log("Hello");
};
```

Key Insight: Function declarations behave differently from function expressions in terms of hoisting. Understanding this distinction helps prevent errors in your code.

SUMANTH M
Frontend developer

React Js

4

undefined vs not defined

undefined: A variable is declared but hasn't been assigned a value yet.

```
console.log(a); // undefined
var a = 5;
```

not defined: A variable has not been declared in the scope and results in a ReferenceError.

```
console.log(b); // ReferenceError: b is not defined
```

Key Insight: undefined means the variable exists but has no value, while not defined means the variable doesn't exist in the scope.

SUMANTH M
Frontend developer

5

React Js

What is an Execution Context?

Definition: An Execution Context is a container where JavaScript code is evaluated and executed.

Why It Matters: Each function call creates a new local Execution Context, ensuring that variables and functions inside it don’t interfere with those outside.

Key Insight: Execution Contexts help manage scope, memory, and the flow of function execution.

SUMANTH M
Frontend developer

6

The Call Stack in Action

What is the Call Stack?: A LIFO (Last In, First Out) structure that keeps track of the function calls and their execution contexts.

How It Works:

Global Execution Context is created first.

Each function call pushes a new Execution Context onto the stack.

When a function finishes, its context is popped off the stack.

Why It Matters: The Call Stack ensures the correct order of function execution and helps manage function calls efficiently.

SUMANTH M
Frontend developer

7

React Js

Code Flow with Execution Context

Global Execution Context (GEC) is created when the program starts.

Memory Phase: Variables are initialized, and functions are hoisted.

Execution Phase: Function calls create new local execution contexts, pushing them onto the call stack.

After Execution: Once a function finishes, its context is popped off, and control returns to the previous execution context.

SUMANTH M
Frontend developer

8

Impact of Hoisting & Execution Contexts

redict Code Behavior: Hoisting helps you understand when variables and functions are available.

Scope Management: Execution Contexts ensure that variables inside functions are isolated and do not interfere with others.

Code Optimization: Understanding hoisting and execution contexts can help you optimize your code and avoid memory leaks or unnecessary function calls.

SUMANTH M
Frontend developer

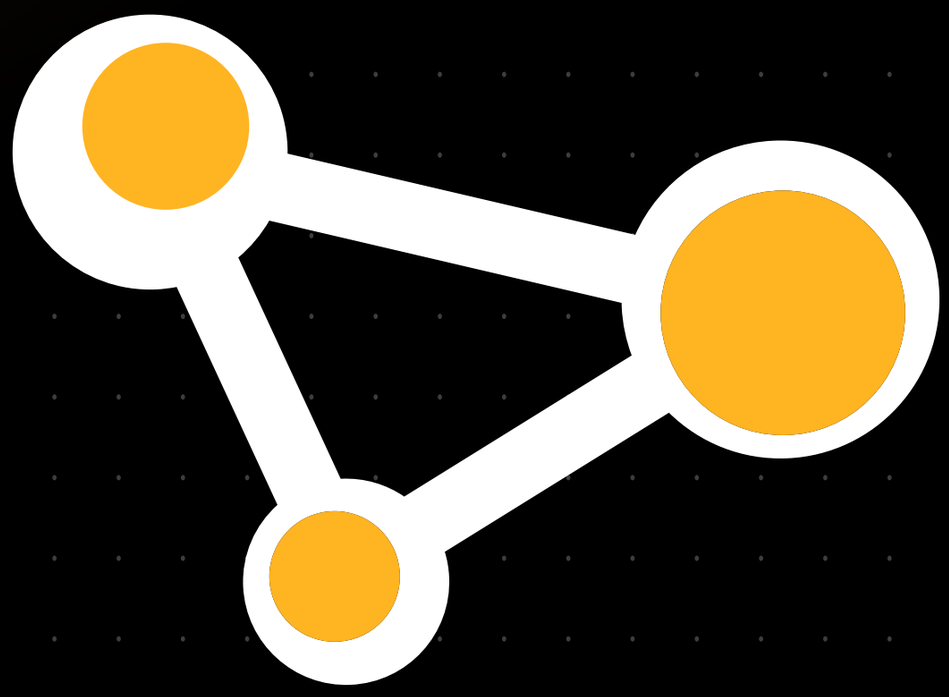
React Js

Conclusion

Why These Concepts Matter

Mastering hoisting, execution contexts, and the call stack is crucial for writing clean, maintainable JavaScript.

These concepts help you understand how JavaScript handles variables and function calls, allowing you to predict behavior, prevent bugs, and improve performance.



SUMANTH M
Frontend developer