How JavaScript Executes Code

Behind the Scenes

JavaScript is a single-threaded, synchronous language.

The Journey Begins – Global Execution Context

When JavaScript starts running, it creates the Global Execution Context – the control room for code!"
Two Phases:

1. **Creation Phase:** Prepares memory for variables and functions.
2. **Execution Phase:** Runs the code line by line.

Example code:

```
1   var n = 2;
2   function square (num){
3     var ans = num*num;
4     return ans;
5   }
6   var square2 = square(n);
7   var square4 = square(4);
```

Phase 1 – Memory Creation

JavaScript prepares memory for all variables and functions.

1. Variables are stored as key-value pairs and initialised with "**undefined**"
2. Functions are stored with their full definitions.

n = undefined
square = {...}
square2 = undefined
square4 = undefined

Phase 2 – Code Execution

- JavaScript starts executing code step by step.variables are assigned with some values and function execution takes place.
- When a function is invoked, a new execution context is created. This context is deleted after the function's execution is complete.

n = 2
square = {...}
square2 = square(n)
square4 = undefined

- When the function **square(n)** is invoked, a new execution context is created, and the function starts executing. Inside the function, we see a **return** statement that returns the value of **ans**. This returned value replaces the initial **undefined** assigned to the **square2** variable.Then this execution context is deleted.
- Similarly, the same process occurs for the **square4** variable. When the **square(4)** function is invoked, a new execution context is created, the function executes, and the value returned by the **return** statement replaces the initial **undefined** assigned to the **square4** variable.

Call Stack

Call stack maintains the order of execution of execution contexts.

All these execution contexts are managed by the Call Stack. The Call Stack initially contains the GEC. Whenever a new execution context is created, it is pushed onto the Call Stack. Once the function execution is complete, its context is popped off the stack. Finally, after all functions have executed and there is nothing left to process, the GEC itself is popped off the Call Stack, and the code execution is complete.

Thank you!

"JavaScript organizes and executes code step-by-step, ensuring everything works smoothly behind the scenes!"

"Did this explanation help you? Let me know your thoughts or share your own tips about JavaScript! 🚀"

Let’s keep learning and growing together! 🚀