How JavaScript is Executed?

How JavaScript is Executed?

JavaScript Execution Context & Call Stack

1. Execution Context

Definition: The environment where JavaScript code is evaluated and executed.

Types of Execution Contexts:

1. Global Execution Context (GEC): Created first when the script runs.
   - `this` refers to the `window` object in browsers.
2. Function Execution Context (FEC): Created each time a function is called.
3. Eval Execution Context: Created inside `eval()` (rarely used).

2. Phases of Execution

1. Memory Creation Phase (Compilation)

- Variables and functions are allocated memory.
- Variables are initialized to `undefined`.
- Functions store their full definition.

Example:

var a = 10; // a = undefined initially

function add() { ... } // add = function reference stored

2. Execution Phase

- Code is executed line-by-line.
- Variables get assigned actual values.
- Functions create new execution contexts when invoked.

Example:

a = 10; // Now a = 10

3. Example Walkthrough

Code Snippet:

var val1 = 10;

```
var val2 = 5;
function add(num1, num2) {
  var total = num1 + num2;
  return total;
}
var result1 = add(val1, val2);
var result2 = add(10, 2);
```

Execution Breakdown

1. Global Execution Context (GEC) Creation:
   - Memory Phase:
     - val1: undefined
     - val2: undefined
     - add: Function definition stored
     - result1: undefined
     - result2: undefined
   - Execution Phase:
     - val1 = 10, val2 = 5
     - result1 = add(val1, val2) triggers a new FEC

2. Function Execution Context (FEC) for add(val1, val2)
   - Memory Phase:
     - num1: undefined
     - num2: undefined
     - total: undefined
   - Execution Phase:
     - num1 = 10, num2 = 5
     - total = 15 (returned to result1)
   - FEC is destroyed after execution!

3. Repeat for result2 = add(10, 2)

4. Call Stack (Execution Stack)
A LIFO (Last-In-First-Out) structure that manages execution contexts.

Process:
- GEC is pushed first.

- Each function call creates an FEC and pushes it onto the stack.
- When a function completes, its FEC is popped off.
- The stack is empty when the script ends.

Example & Call Stack Order:

```
function one() { two(); }
function two() { three(); }
function three() { console.log("Done"); }
one();
```

Stack Flow:

1. GEC one() two() three()
2. After execution: three() two() one() GEC

5. Key Concepts

- Hoisting: Variables/functions are accessible in the memory phase before execution.
- Single-Threaded: JavaScript executes one command at a time.
- Lexical Environment: Each execution context maintains a reference to its outer scope.

6. Practical Debugging Tips

Use browser developer tools:

- Sources Tab Set breakpoints & step through code.
- Call Stack Panel Track execution order.
- Scope Panel Inspect variables in different contexts.

7. Interview Focus Areas

Be prepared to explain:

- Execution context phases and variable/function handling.
- The call stack and its LIFO behavior.
- How nested function execution works.