JavaScript Concurrency Model & Event Loop - Key Concepts

Core Concepts and Terminology

1. Single-threaded

- JavaScript executes one task at a time using a single thread.

- Code is processed sequentially.

2. Blocking vs Non-blocking Code

- Blocking: Synchronous operations (e.g., long for-loops) halt further code execution.

- Non-blocking: Asynchronous operations (e.g., setTimeout, fetch) allow the program to proceed while they wait in the background.

Event Loop System Components

3. Memory Heap

- Unordered memory space for storing objects and variables.

4. Call Stack

- Tracks currently executing functions in LIFO (Last-In, First-Out) order.

- A new function call adds a frame to the top of the stack.

5. Web APIs / Node APIs

- Browser or Node.js features (like setTimeout or fetch) that handle operations asynchronously.

6. Event Queue

- A FIFO (First-In, First-Out) queue holding callback functions waiting to run when the call stack is empty.

7. Event Loop

- Continuously checks if the call stack is empty.

- If empty, it moves the first function in the event queue to the stack for execution.

JavaScript Concurrency Model & Event Loop - Key Concepts

How Asynchronous Code Runs

Example:

```
console.log("Start");
setTimeout(() => console.log("Async"), 1000);
console.log("End");
```

Execution Order:

1. Logs "Start"
2. setTimeout is sent to Web API with a timer.
3. Logs "End" immediately.
4. After 1 second, the callback is moved to the event queue.
5. When the stack is clear, the event loop moves "Async" to the stack and it executes.

Key Takeaway

- JavaScript is single-threaded but emulates concurrency using the Event Loop.
- The Event Loop coordinates asynchronous execution, making JavaScript responsive and efficient.