# Event Loop in NodeJS

The event loop in NodeJS is a mechanism that allows asynchronous tasks to be handled efficiently without blocking the execution of other operations.

1. Executes JavaScript synchronously first and then processes asynchronous operations.

2. Delegates heavy tasks like I/O operations, timers, and network requests to the libuv library.

3. Ensues smooth execution of the multiple operations by queuing and scheduling callbacks efficiently.

NodeJS operates in a single thread.

The Event loop is essential in Node.js because it allows non-blocking,asynchronous operations to be handled efficiently,even through Node.js operates on a single thread.

1. Enables non-blocking execution despite Nodes.js being single-threaded.

2. Helps handle I/O tasks efficiently.

3. Makes NodeJS suitable for scalable applications like web servers.

console.log("This is the first statement");

setTimeout(function() {

console.log("This is the second statement");

}, 1000);

console.log("This is the third statement");

First statement is logged immediately.

setTimeout schedules the second statement to log after 1000 milliseconds.

Third statement is logged immediately after.

After 1000 milliseconds, the callback from setTimeout is executed, logging the Second statement.

# 

# How the Event Loop Works?

Synchronous code first and then moves to handle asynchronous tasks.

1. Initialization (When Node.js starts, it loads the script, executes synchronous code, and registers any asynchronous tasks)

2. Execution of Input Script (The call stack executes synchronous code first.)

3. Handling Asynchronous operations with libuv (Node.js uses a special C library called libuv to handle asynchronous operations. This library manages a thread pool that offloads heavy tasks (like file I/O, database operations, or network requests) that would otherwise block the event loop.)

4. Callback Execution (Once the thread pool completes its tasks, it sends callbacks to the event queue. Execute only When No synchronous code is currently executing)

5. Event Loop Phases (It checks for events, handles asynchronous callbacks, and executes tasks in the correct order.)

6. Callback Execution from Event Queue

### Phases of the Event Loop

#### 1. Timers Phase : This phase processes timers that have been set using setTimeout() and setInterval()

console.log('Start');

setTimeout(() => {

console.log('Timeout callback');

}, 2000);

console.log('End');

#### 2. Pending Callbacks: This phase executes I/O related callbacks that were deferred from the previous loop cycle.