Event Loop in Simple Language

What is the Event Loop?
The Event Loop is like a manager in JavaScript that keeps checking if there is any work to do and decides what should run now and what should wait.
JavaScript itself is single-threaded → it can do only one thing at a time. But still, it feels like it's doing many things at once (like handling clicks, timers, API calls). This is possible because of the Event Loop.

How it works (step by step):

1. Call Stack → Where JavaScript executes code line by line. Example: console.log('Hello') goes here first.
2. Web APIs / Background Tasks → Things like setTimeout, fetch, or event listeners don't block the main thread. They are handed over to the browser to handle.
3. Callback Queue (Task Queue) → Once the background task is finished (like timer completed or data fetched), the result (callback function) is placed in the queue.
4. Event Loop → Keeps watching: If Call Stack is empty, it takes the first task from the Queue and pushes it to the Call Stack. This cycle keeps going forever.

Example:

```
console.log('Start');

setTimeout(() => {
console.log('Inside Timeout');
}, 2000);

console.log('End');
```

Flow:

- 'Start' → goes to Call Stack → printed.
- setTimeout → sent to Web API → waits for 2 sec.
- 'End' → goes to Call Stack → printed.
- After 2 sec → callback goes into Queue.
- Event Loop checks → Call Stack empty → moves callback to stack → prints 'Inside Timeout'.

Output:

```
Start
End
Inside Timeout
```

In short: Event Loop = A cycle that keeps checking if JavaScript can run the next task from the queue. It makes async things (like timers, API calls) possible even though JS is single-threaded.