**Day 37**





**“Web Development + Security”**

**JavaScript Callbacks & Promises:**

**What is asynchronous nature of JavaScript?**

Some operations (like network requests, timers, reading files) don’t block the rest of the code. JavaScript can continue executing the next lines while waiting for the async operation to complete.

**Why is JS Asynchronous?**

* JavaScript is single-threaded (runs one command at a time).
* To avoid blocking the UI or main thread, JS handles time-consuming tasks asynchronously.
* This allows smooth user experience — the page doesn’t freeze while waiting for data.

**Common async operations:**

* setTimeout / setInterval
* Fetching data from APIs (fetch(), AJAX)
* Reading files (Node.js)
* Event listeners (click, scroll)

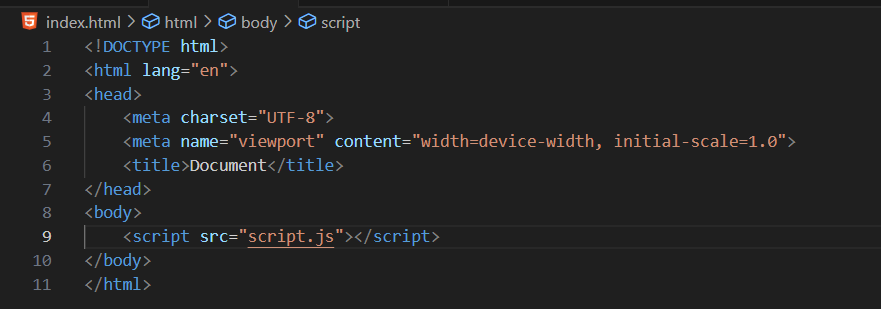
**Why Asynchronous is Important**

* Improves performance and responsiveness.
* Lets the UI remain interactive while waiting for long operations.
* Essential for modern web apps (AJAX, animations, event-driven programming).

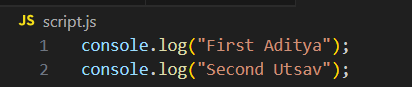
Example: basic example without any anyc operations

Code:

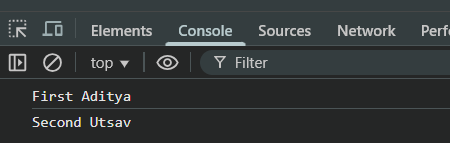
Index.html:



Script.js:



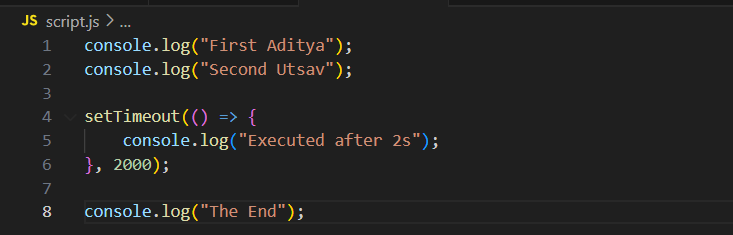
Output: in console. Clearly, the first one get printed first and then the print went down.



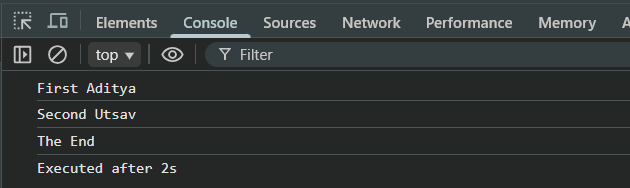
Now, example for asynchronous operations:

Index.html: same as above

Script.js: for this first the line “First Aditya” will get printed then “Second Utsav” and then instead of “Executed after 2s” it will print “The End” as the setTimeout is aynsc in nature

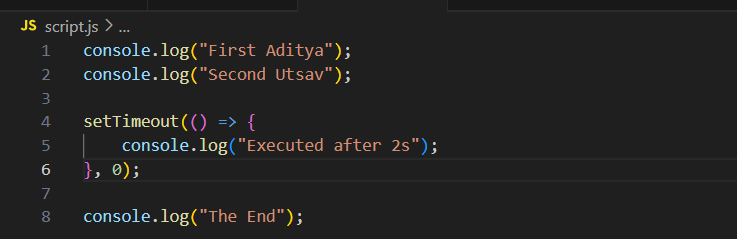


Output:

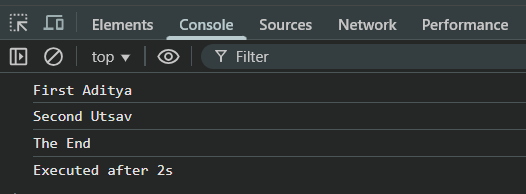


Even if the time interval be 0, if it is setTimeout then it will be treated as async only:  
index.html: same as above

Script.js:



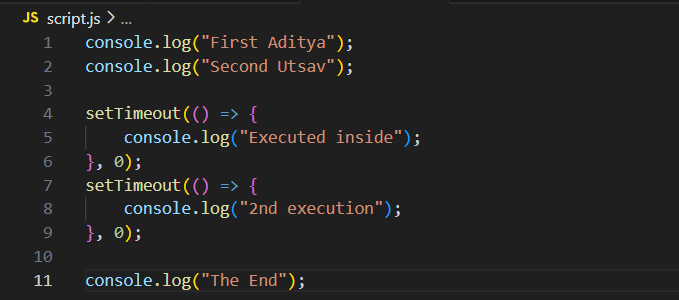
Output:



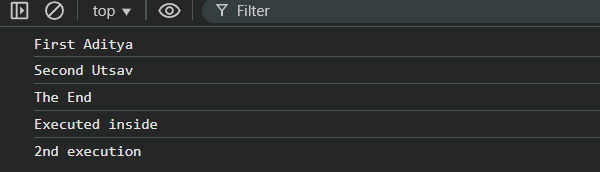
Now, adding one more async in the same script.js:

Index.html: same as above

Script.js:

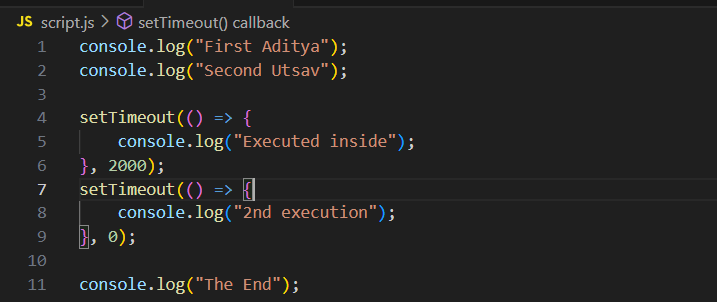


Output:

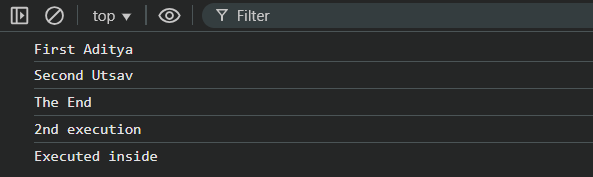


For the same example, If one of them has different interval, then the one who have less will get executed first:

Script.js:



Output:

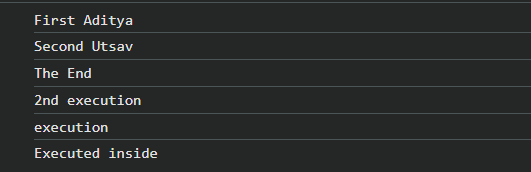


Similarly,

Script.js:



Output:



**What is a Callback Function?**

A callback function is a function passed as an argument to another function, and is executed later after some operation finishes.

**Why Callbacks Are Needed**

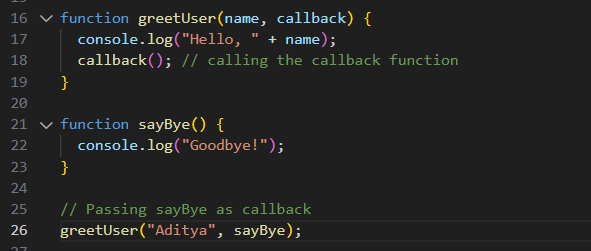
Some operations in JS are asynchronous (take time), like:

* Fetching data from a server
* Reading a file (Node.js)
* Timers (setTimeout, setInterval)
* User events (click, keypress)

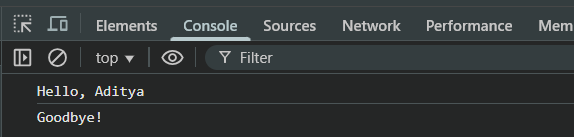
Without callbacks, JS would not know what to do after the operation completes.

Example: synchronous

Index.html: same as above

Script.js:  


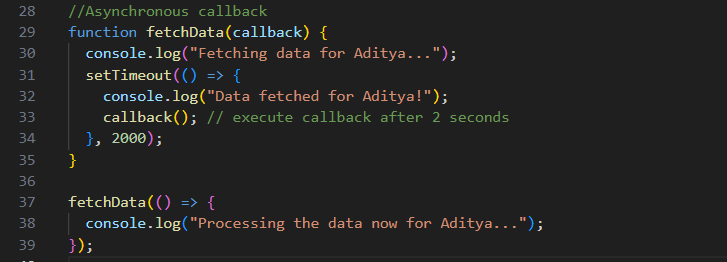
Output:



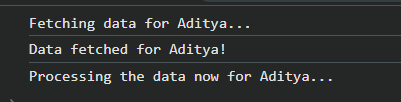
Example: asynchronous

Index.html: same as above

Script.js:



Output:



**What is a Promise?**

A Promise is a JavaScript object that represents the future result of an asynchronous operation.

It promises that it will eventually either:

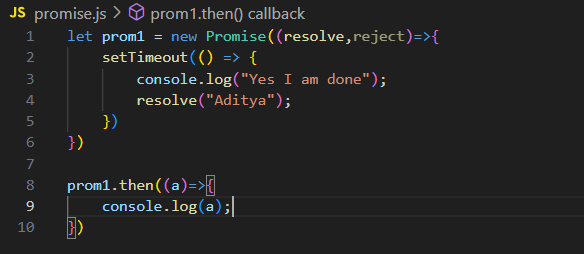
1. Resolve → the operation was successful
2. Reject → the operation failed

**Promise states:**

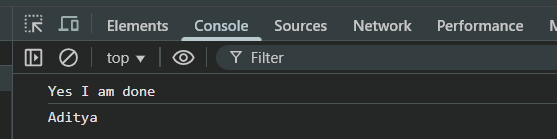
| **State** | **Description** |
| --- | --- |
| **Pending** | Initial state, operation not completed yet |
| **Fulfilled** | Operation completed successfully (resolve) |
| **Rejected** | Operation failed (reject) |

A very basic example: here the promise get resolved.

Promise.js:



Output:



--The End--