



## 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:

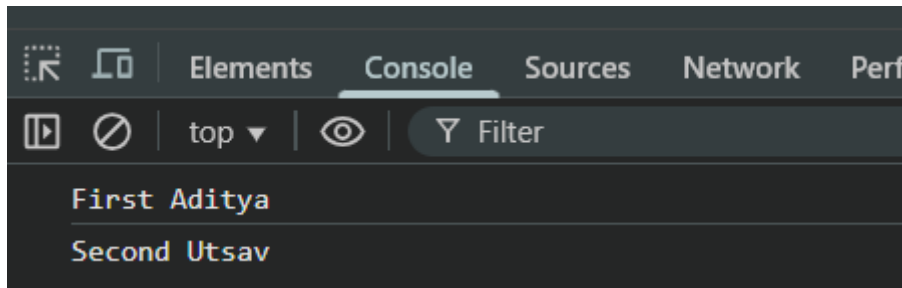
Index.html:

```
index.html > html > body > script
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  |   <title>Document</title>
7  </head>
8  <body>
9  |   <script src="script.js"></script>
10 </body>
11 </html>
```

Script.js:

```
JS script.js
1 console.log("First Aditya");
2 console.log("Second Utsav");
```

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



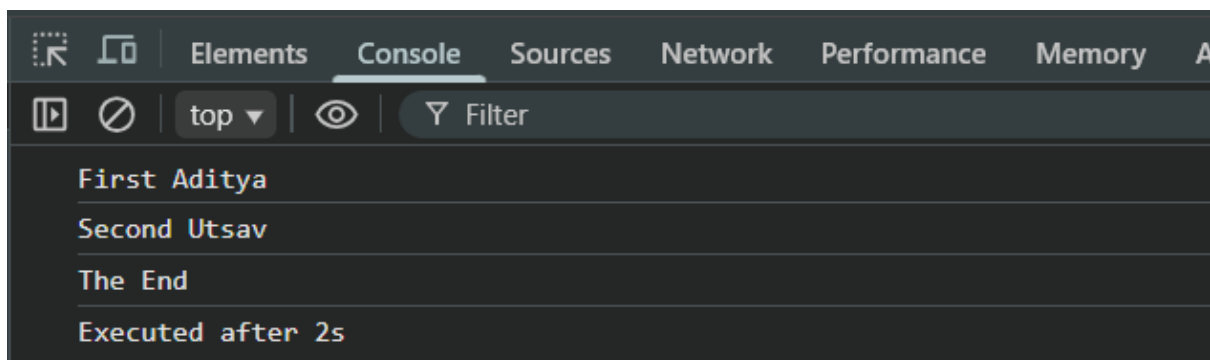
Now, example for asynchronous operations:

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 aync in nature

```
JS script.js > ...
1 console.log("First Aditya");
2 console.log("Second Utsav");
3
4 setTimeout(() => {
5   console.log("Executed after 2s");
6 }, 2000);
7
8 console.log("The End");
```

Output:



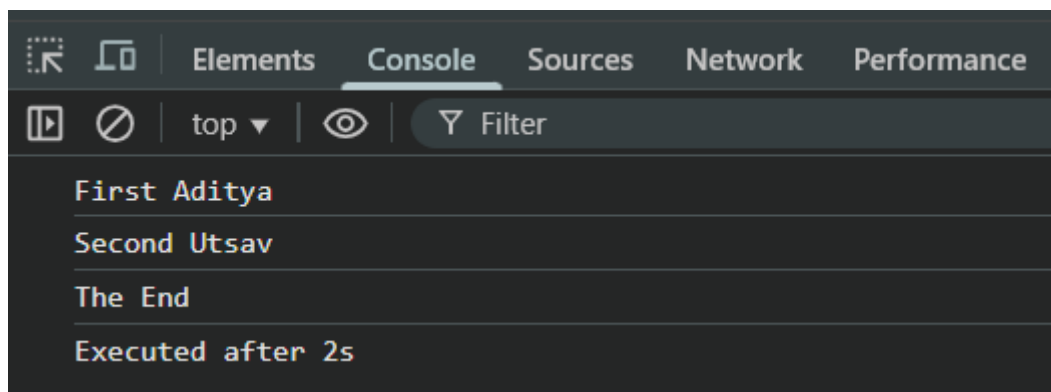
Even if the time interval be 0, if it is setTimeout then it will be treated as async only:

index.html: same as above

Script.js:

```
JS script.js > ...
1 console.log("First Aditya");
2 console.log("Second Utsav");
3
4 setTimeout(() => {
5   console.log("Executed after 2s");
6 }, 0);
7
8 console.log("The End");
```

Output:



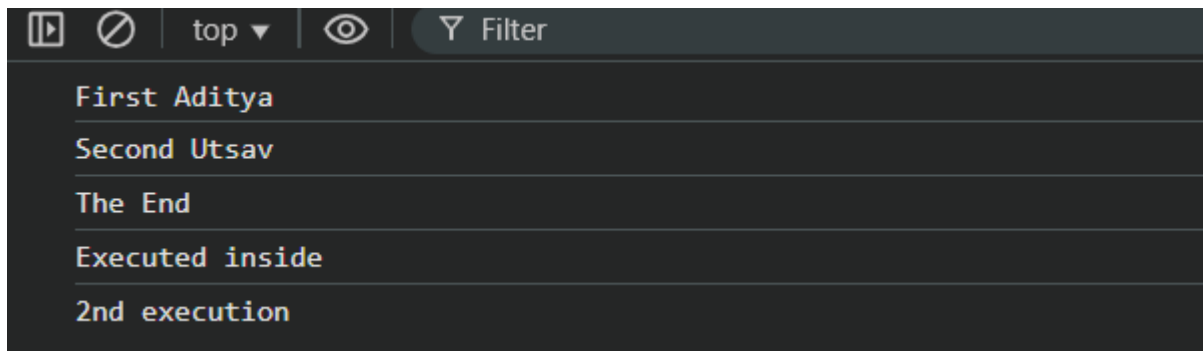
Now, adding one more async in the same script.js:

Index.html: same as above

Script.js:

```
JS script.js > ...
1 console.log("First Aditya");
2 console.log("Second Utsav");
3
4 setTimeout(() => {
5   console.log("Executed inside");
6 }, 0);
7 setTimeout(() => {
8   console.log("2nd execution");
9 }, 0);
10
11 console.log("The End");
```

Output:



```
First Aditya
Second Utsav
The End
Executed inside
2nd execution
```

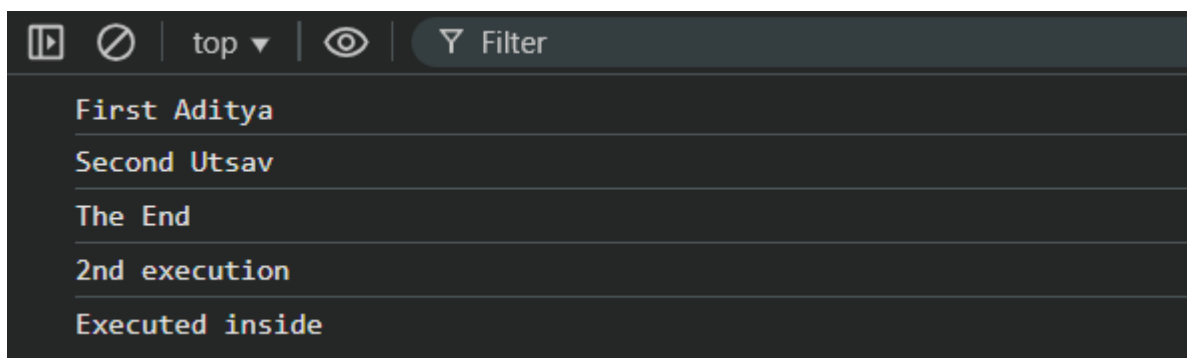
For the same example, If one of them has different interval, then the one who have less will get executed first:

Script.js:



```
JS script.js >  setTimeout() callback
1 console.log("First Aditya");
2 console.log("Second Utsav");
3
4 setTimeout(() => {
5   console.log("Executed inside");
6 }, 2000);
7 setTimeout(() => {
8   console.log("2nd execution");
9 }, 0);
10
11 console.log("The End");
```

Output:



```
First Aditya
Second Utsav
The End
2nd execution
Executed inside
```

Similarly,

Script.js:

```
JS script.js > ...
1 console.log("First Aditya");
2 console.log("Second Utsav");
3
4 setTimeout(() => {
5   console.log("Executed inside");
6 }, 2000);
7 setTimeout(() => {
8   console.log("2nd execution");
9 }, 0);
10
11 console.log("The End");
12 setTimeout(() => {
13   console.log("execution");
14 }, 1000);
```

Output:

```
First Aditya
Second Utsav
The End
2nd execution
execution
Executed inside
```

## 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.

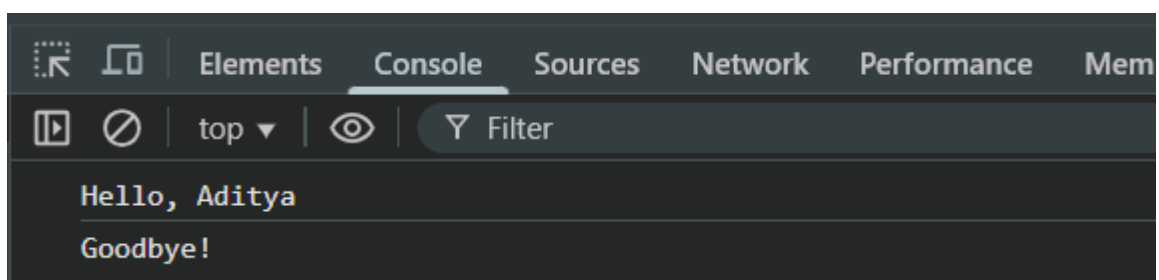
Example: synchronous

Index.html: same as above

Script.js:

```
16 function greetUser(name, callback) {  
17   console.log("Hello, " + name);  
18   callback(); // calling the callback function  
19 }  
20  
21 function sayBye() {  
22   console.log("Goodbye!");  
23 }  
24  
25 // Passing sayBye as callback  
26 greetUser("Aditya", sayBye);  
27
```

Output:



Example: asynchronous

Index.html: same as above

Script.js:

```
28 //Asynchronous callback
29 function fetchData(callback) {
30   console.log("Fetching data for Aditya...");
31   setTimeout(() => {
32     console.log("Data fetched for Aditya!");
33     callback(); // execute callback after 2 seconds
34   }, 2000);
35 }
36
37 fetchData(() => {
38   console.log("Processing the data now for Aditya...");
39 });
```

Output:

```
Fetching data for Aditya...
Data fetched for Aditya!
Processing the data now for Aditya...
```

## 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:

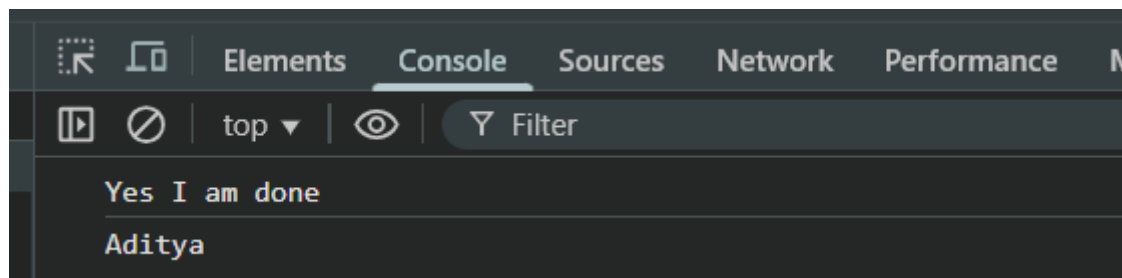
State	Description
Pending	Initial state, operation not completed yet
Fulfilled	Operation completed successfully (resolve)
Rejected	Operation failed (reject)

A very basic example: here the promise get resolved.

Promise.js:

```
JS promise.js > prom1.then() callback
1   let prom1 = new Promise((resolve,reject)=>{
2       setTimeout(() => {
3           console.log("Yes I am done");
4           resolve("Aditya");
5       })
6   })
7
8   prom1.then((a)=>{
9       console.log(a);
10  })
```

Output:



--The End--