

passing a function to another function is called **Callback function**.

It gives access to the whole asynchronous world in a synchronous single threaded language

-> Any function which blocks the "Call Stack" is known as **Blocking the main thread**.

10:36

The screenshot shows a web browser on the left with the title "Namaste JavaScript" and a "Click Me" button. The browser's developer tools are open, showing the "Sources" panel with a file named "index.js". The code in "index.js" is as follows:

```
1 // What is a Callback Function in JavaScript
2
3 setTimeout(function () {
4   console.log("timer");
5 }, 5000);
6
7 function x(y) {
8   console.log("x");
9   y();
10 }
11 x(function y() {
12   console.log("y");
13 });
14
15 // JavaScript is a synchronous and single-threaded language
16
17 // Blocking the main thread
18
19 // Power of Callbacks?
20
21 // Deep about Event listeners
22
23 // Closures Demo with Event Listeners
24
25 // Scope Demo with Event listeners
26
27 // Garbage Collection & removeEventListeners
28
```

The browser's console shows the output of the code: "x", "y", and "timer". The "Call Stack" panel is also visible, showing the current state of the execution stack.

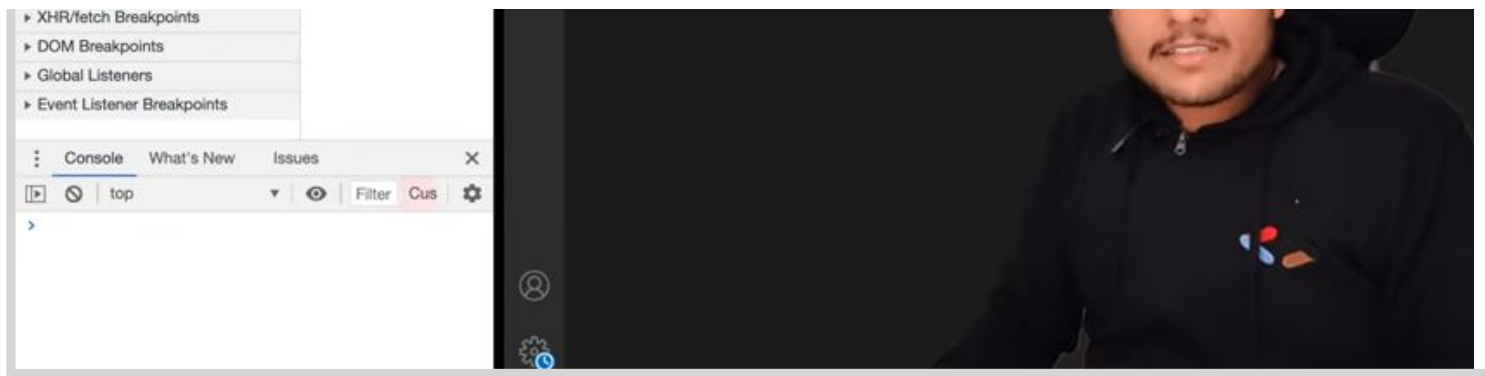
call back function FT.EventListener

11:40

The screenshot shows a web browser on the left with the title "Namaste JavaScript" and a "Click Me" button. The browser's developer tools are open, showing the "Sources" panel with a file named "index.js". The code in "index.js" is as follows:

```
1 // Deep about Event listeners
2
3 // Closures Demo with Event Listeners
4
5 // Scope Demo with Event listeners
6
7 // Garbage Collection & removeEventListeners
8
9
10 document.getElementById("clickMe")
11   .addEventListener("click", function() {
12
13   });
14
```

The browser's console shows the output of the code: "x", "y", and "timer". The "Call Stack" panel is also visible, showing the current state of the execution stack.



14:27

