

127.0.0.1:5500 Incognito

Namaste JavaScript

Paused in debugger

Elements Sources >> Settings

Page >> index.js x

- top
- 127.0.0.1:
 - js
 - (index)

```
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
```

4 lines, 59 characters selected

Scope Watch

Paused on breakpoint

Call Stack

- (anonymous) index.js:4
- setTimeout (async)
- (anonymous) index.js:3

Breakpoints

- XHR/fetch Breakpoints
- DOM Breakpoints
- Global Listeners

Local

- this: Window


Global

- Window

call stack will magically appear after 5 seconds

CALLBACK FUNCTIONS

```
js > js index.js
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
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 & removeEventListe
28
```



Callback Functions in JS ft. Event Listeners 🔥 | Namaste JavaScript Ep. 14



Namaste 🙏 JavaScript

[Click Me](#)

Elements Sources >> ⚙️ ⋮ ✕

Page >> ⋮ index.js ✕

▼ top
127.0.0.1:5500
js
(index)

9 document.getElementById("clickMe")
10 console.log("Button Clicked")
11 }
12 }

{ } Line 10, Column 3 Coverage: n/a

Scope Watch

▼ Call Stack
Not paused

► Breakpoints
► XHR/fetch Breakpoints
► DOM Breakpoints
► Global Listeners
► Event Listener Breakpoints

⋮ Console What's New Issues ✕

▶ ⓧ top ▼ 👁 F 1 hidden ⚙️

4 Button Clicked index.js:10
>

```
js > js index.js
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 document.getElementById("clickMe").addEventListener("click", function xyz() {
10 | console.log("Button Clicked");
11 | });
12
```

this will console the button clicked thing just once...



← → ↻ ⓘ 127.0.0.1:5500 ☆ Incognito

Namaste 🙏 JavaScript

Click Me

Elements Sources >> ⚙️ ⋮ ✕

Page >> ⋮ index.js ✕

▼ top
127.0.0.1: js
(index)

```
9 let count = 0;  
10 document.getElementById("clickMe")  
11 console.log("Button Clicked",  
12 {});
```

{ } Line 10, Column 1 Coverage: n/a

Scope Watch

▼ Call Stack
Not paused

► Breakpoints
► XHR/fetch Breakpoints
► DOM Breakpoints
► Global Listeners
► Event Listener Breakpoints

⋮ Console What's New Issues ✕

▶ ⏏ top ▼ 🔍 Filter Cus ⚙️

Button Clicked 1	index.js:11
Button Clicked 2	index.js:11
Button Clicked 3	index.js:11
Button Clicked 4	index.js:11
Button Clicked 5	index.js:11

>

```
js > JS index.js  
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 let count = 0;  
10 document.getElementById("clickMe").addEventListener("click", function xyz() {  
11   console.log("Button Clicked", ++count);  
12 });  
13
```

here we have counted number of time button is clicked

but making variable as a global
variable is not a good practise



← → ↻ ⓘ 127.0.0.1:5500 ☆ Incognito

Namaste 🙏 JavaScript

button#clickMe 67.11 x 21

Click Me

Elements Console Sources Network >> ⚙️ ⋮ ✕

Page >> ⋮

index.js ✕

```
9 function attachEventListeners() {
10   let count = 0;
11   document.getElementById("clickMe").addEventListener("
12   console.log("Button Clicked", ++count);
13   });
14 }
15 attachEventListeners();
16
```

{ } Line 12, Column 5 Coverage: n/a

Scope Watch

▼ Local

- ▶ this: button#clickMe

▼ Closure (attachEventListeners)

- count: 3

▶ Global Window

Paused on breakpoint

▼ Call Stack

- xyz index.js:12

▶ Breakpoints

Console What's New Issues ✕

top Filter Custom levels ⚙️

Button Clicked 1 index.js:12


Button Clicked 2 index.js:12

Button Clicked 3 index.js:12

>

js > JS index.js

```
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 function attachEventListeners() {
10   let count = 0;
11   document.getElementById("clickMe").addEventListener("click"
12   console.log("Button Clicked", ++count);
13   });
14 }
15 attachEventListeners();
16
```



127.0.0.1:5500 Incognito

Namaste 🙏 JavaScript

Click Me

Elements Console Sources Network >>

```
<html lang="en">
  <head>...</head>
  <body>
    <h1 id="heading">Namaste 🙏 JavaScript </h1>
    <button id="clickMe">Click Me</button> == $0
    <script src="js/index.js"></script>
  </body>
</html>
```

html body button#clickMe

Styles Computed Layout Event Listeners DOM Breakpoints Properties >>

✓ Ancestors All Framework listeners

click

button#clickMe Remove index.js:11

useCapture: false

passive: false

once: false

handler: f xyz()

[[Scopes]]: Scopes[2]

- 1: Global {window: Window, self: Window, document: document, name: }
- 0: Closure (attachEventListeners) {count: 0}

[[FunctionLocation]]: <unknown>

__proto__: f ()

prototype: {constructor: f}

name: "xyz"


Console What's New Issues

top Filter Custom levels >>

```
js > JS index.js
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 function attachEventListeners() {
10   let count = 0;
11   document.getElementById("clickMe").addEventListener("click"
12     | console.log("Button Clicked", ++count);
13   });
14 }
15 attachEventListeners();
16
```

go to element tab
go to even listner
select button and it will show its
listener

and it will remember its lexical
scope even if call stack is empty



CALLBACK functions in js...

What are callback functions in js?

as functions are first class citizens.

So a function pass on to another function is a callback function.

Gives us the access to whole Async world in a synchronous single threaded language.
(to achieve async nature)

Example: Async Task: \rightarrow callback
`setTimeout(function() { ... }, 3000);`
`function x(y) { ... }`

`function y(x) { function y() { ... } }`
 \rightarrow callback

Gain async nature

After 3 second, call stack appears maguallly.
 JS only has 1 call stack, known as the main thread.

If any operation blocks the call stack,
 it means blocking the main thread.

* If we don't have callback functions, we
 wouldn't have achieved async nature
 execution in js.

Event listeners: locate a button with id → click me
`document.getElementById("clickme").addEventListener("click", function xyz() {
 console.log("button clicked");
});`

// So whenever the button is clicked, the callback function is called & xyz will be pushed into call stack & then executed.

Closure demo along with event - listeners :-
 ~ SS (Screenshot)

Garbage collection and remove Event listeners.
 It's a good practice to remove event listeners. They are heavy because they carry closures and takes memory & even when call stack is empty, then also program don't free up memory, cause we don't know when action listener is called.

(That's why event listeners are heavy)
 1000 of event listeners can make our site lag.

If we remove this event listener, all the closure under it / scope would be garbage collected.