

Topics Covered

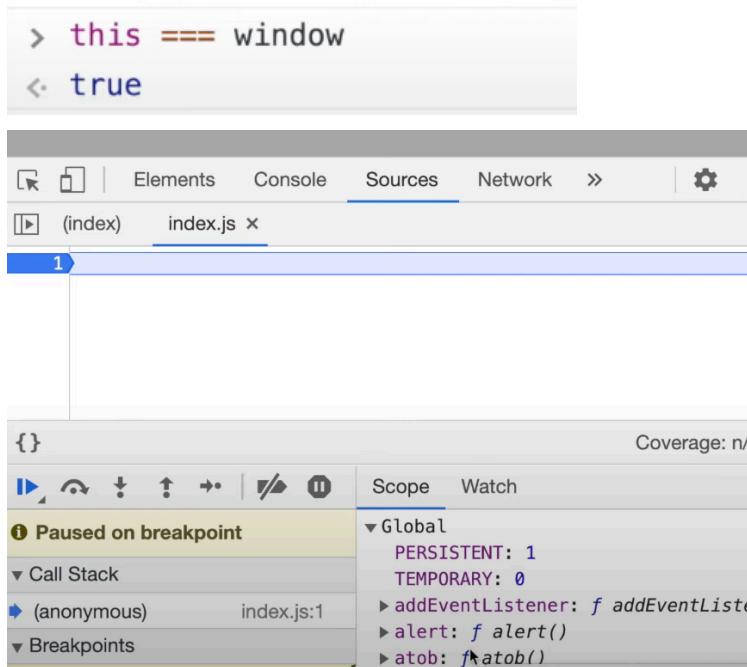
- Window & this keyword
- Hoisting in Javascript
- Functions in Javascript
- Undefined/Not defined in JS
- Scope chain, Scope and Lexical Environment

Window & This

Window is a global object which is created along with the execution context . Whenever any javascript code runs global object is created , global execution context is created and along with the execution context 'this' variable is created.

Javascript will run on browsers, servers etc.Every browser has its own javascript engine. Chrome has v8 engine, safari has its own, firefox, microsoft have its own js engine. All these js engines have responsibility to create this global object . In case of browsers it is known as window, in case of node it would be something else. Even the js file is empty js engine will create this global object,GEC and call stack. This will be created for both functional and global execution context

At global level, this === window in case of browsers



The screenshot shows the Chrome DevTools interface. The top navigation bar has tabs for Elements, Console, Sources, Network, and more. The Sources tab is active. Below the tabs, there's a list of files: '(index)' and 'index.js x'. A blue arrow icon indicates a breakpoint is set at the top of 'index.js'. The main pane shows the code of 'index.js': `1` (with a blue background). At the bottom, the 'Scope' tab of the sidebar is selected, showing the global scope with variables: PERSISTENT: 1, TEMPORARY: 0, addEventListener: f, alert: f, and atob: f.

```
> this === window
< true
```

{}

Coverage: n/

Paused on breakpoint

Call Stack

(anonymous) index.js:1

Breakpoints

Scope Watch

Global

PERSISTENT: 1

TEMPORARY: 0

addEventListener: f

alert: f

atob: f

Global Space

```
var a = 10;  
Function b()
```

Local Space

```
{  
    var x = 10;  
}
```

The variables and functions are attached to the global object window

```
> window  
< ▾ Window {window: Window, self: Window, document: document, name:  
  "", location: Location, ...} ⓘ  
  a: 10  
  ▷ alert: f alert()  
  ▷ atob: f atob()  
  ▷ b: f b()  
  ▷ blur: f blur()  
  ▷ btoa: f btoa()  
  ▷ caches: CacheStorage {}  
  ▷ cancelAnimationFrame: f cancelAnimationFrame()  
  ▷ cancelIdleCallback: f cancelIdleCallback()
```

Namaste 🌈 JavaScript

Elements Console Sources »
top Custom levels

10	index.js:6
10	index.js:7
✖ Uncaught ReferenceError: x is not defined at index.js:8	index.js:8

```
2 var a =10;  
3 function b() {  
4     var x =10;  
5 }  
6 console.log(window.a);  
7 console.log(a);  
8 console.log(x);
```

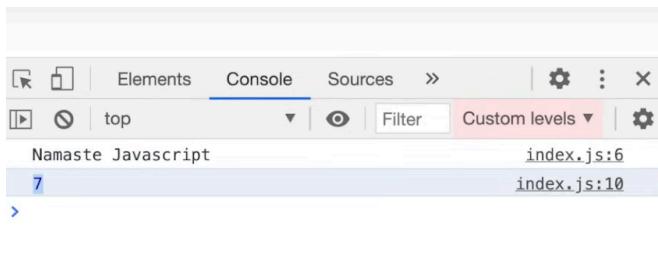
Namaste 🌈 JavaScript

Elements Console Sources Network »
top Custom levels

10	index.js:6
10	index.js:7
10	index.js:8

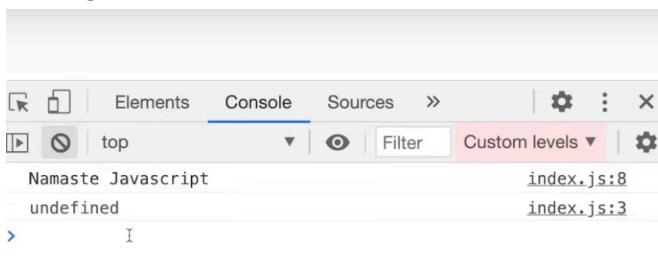
```
1 var a =10;  
2 function b() {  
3     var x =10;  
4 }  
5 console.log(window.a);  
6 console.log(a);  
7 console.log(this.a);
```

Hoisting in Javascript



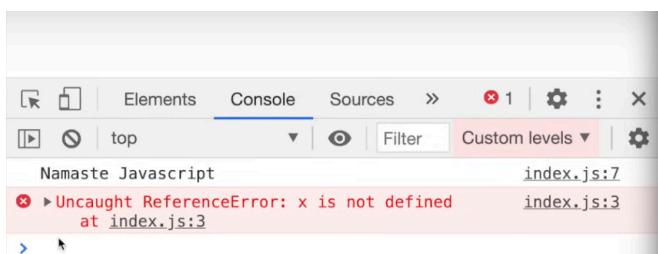
```
3 var x = 7;
4
5 function getName() {
6   console.log("Namaste Javascript");
7 }
8
9 getName();
10 console.log(x);
11
```

Calling function and variable before declaration



```
2 getName();
3 console.log(x);
4
5 var x = 7;
6
7 function getName() {
8   console.log("Namaste Javascript");
9 }
```

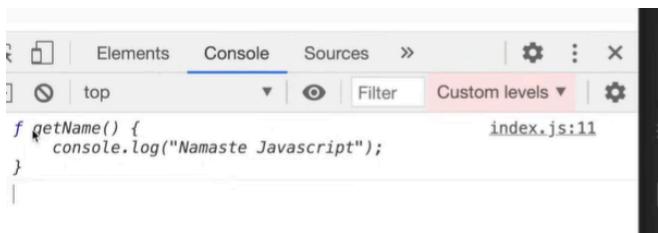
It would give error in other languages but not in javascript. U can call function and access the variable before its declaration



```
2 getName();
3 console.log(x);
4
5
6 function getName() {
7   console.log("Namaste Javascript");
8 }
9
```

Hoisting

is a phenomenon in Javascript by which you can access these variables and functions even before u r initializing it. In Javascript before code executes Global execution context will get created and in memory phase itself variables with value undefined and whole function code will be copied in memory. So we can access variables and functions before its declaration. This is called hoisting



```
3 // console.log(x);
4
5 var x = 7;
6
7 function getName() {
8   console.log("Namaste Javascript");
9 }
10
11 console.log(getName);
```

The screenshot shows the Chrome DevTools Sources tab for a file named index.js. The code is as follows:

```
// getName();
// console.log(x);
console.log(getName());
var x = 7;
function getName() {
    console.log("Namaste Javascript");
}
```

Even before the first line of execution, in memory creation phase x is undefined and for function whole function block code is copied in memory

The screenshot shows the Chrome DevTools Sources tab for index.js. A breakpoint is set at line 3, column 9, where the variable x is defined. The code is:

```
var x = 7;
```

The Scope tab in the bottom right shows the current state of variables:

- visualViewport: VisualViewport...
- webkitCancelAnimationFrame: f ...
- webkitRequestAnimationFrame: f...
- webkitRequestFileSystem: f web...
- webkitResolveLocalFileSystemUR...
- webkitStorageInfo: DeprecatedS...
- window: Window {window: Window...
- x: undefined

```
1
2
3 var x = 7;
4
5 function getName() {
6     console.log("Namaste Javascript");
7 }
8
9 getName();
10 console.log(x);
```

{ } Line 3, Column 9 Coverage: n/a

Scope Watch

- ▶ focus: f focus()
- frameElement: null
- ▶ frames: Window {window: Window...}
- ▶ getComputedStyle: f getCompute...
- ◀ getName: f getName()
- arguments: null
- caller: null
- length: 0
- name: "getName"

Arrow Function

Error on line 3 itself

```
1
2
3 getName();
4 console.log(x);
5 console.log(getName());
6
7 var x = 7;
8
9 var getName = () => [
10     console.log("Namaste Javascript");
11 ]
```

For arrow function, getName is treated as a variable not a function. Like for variable x, for variable getName also special value 'undefined' will be assigned

Elements Console Sources > index.js

```
3 getName();
4 console.log(x);
5 console.log(getName);
6
7 var x = 7;
8
9 var getName = () => {
  console.log("Namaste Javascript");
10}
11
12
```

{ } 7 characters selected Coverage: n/a

Scope Watch

on breakpoint

frameElement: null
frames: Window {window: Window...}
getComputedStyle: f getCompute...
getName: undefined
getSelection: f getSelection()
history: History {length: 2, s...
indexedDB: IDBFactory {}
innerHeight: 150

var getName2 = function () { } getName2 is a variable with value undefined

var getName = () => { ... } getName is a variable with value undefined

function getName3() { } whole function code will get copied

Elements Console Sources > index.js

```
1
2
3 getName();
4 console.log(x);
5 console.log(getName);
6
7 var x = 7;
8
9 var getName2 = function () {
10}
11
12
13
14
15
16
17
```

{ } Line 3, Column 1 Coverage: n/a

Scope Watch

on breakpoint

frames: Window {window: Window...}
getComputedStyle: f getCompute...
getName: undefined
getName2: undefined
getSelection: f getSelection()
history: History {length: 2, s...
indexedDB: IDBFactory {}



Functions in Javascript

The screenshot shows the Chrome DevTools interface. On the left, the 'Console' tab is selected, displaying the output of three console.log statements: '10', '100', and '1'. To the right, the 'Sources' tab shows the source code of 'index.js' with line numbers 1 through 14. The code defines variables 'x' (1, 10, 100) and functions 'a()' and 'b()'.

```
js > JS index.js
1 var x = 1;
2 a();
3 b();
4 console.log(x);
5
6 function a() {
7     var x = 10;
8     console.log(x);
9 }
10
11 function b(){
12     var x = 100;
13     console.log(x);
14 }
```

Local Memory, Global Memory, Global Execution context and a() inside call stack

The screenshot shows the Chrome DevTools Debugger. A breakpoint is set at line 7, column 13, where the code is 'var x = 10;'. The 'Call Stack' pane shows the current stack trace: 'a' at index.js:7 and '(anonymous)' at index.js:2. The 'Scope' pane shows the local scope with 'this: Window' and 'x: undefined', and the global scope with 'Window'. A checkbox for the breakpoint at line 7 is checked.

```
1 var x = 1;
2 a();
3 b();
4 console.log(x);
5
6 function a() {
7     var x = 10;
8     console.log(x);
9 }
10
11 function b(){
12     var x = 100;
13     console.log(x);
14 }
```

Undefined vs Not defined in Javascript

Undefined - special place holder value for variables before it is assigned

Not defined - variable not present in memory

Example

```
console.log(a);      //undefined
var a = 7;
console.log(a);      // 7
console.log(x);      // Reference Error - x is not defined
var b;
console.log(b);      // undefined
```

The screenshot shows two separate instances of the Chrome DevTools Console tab. In both cases, the code being run is:

```

2 var a;
3
4 console.log(a);
5
6 if(a === undefined) {
7   console.log("a is undefined")
8 }
9 else {
10   console.log("a is not undefined");
11 }

```

In the top instance, the output in the console is:

```
undefined index.js:4
a is undefined index.js:7
```

In the bottom instance, the output in the console is:

```
undefined index.js:4
a is not undefined index.js:11
```

Loosely/ weakly typed language

It does not attach variables to its datatype. You can put string in x and also you can put number in x in later point of time

The screenshot shows the Chrome DevTools Console tab with the following code:

```

var a;
console.log(a);
a=10;
console.log(a);
a = "hello world";
console.log(a);

```

The output in the console is:

```
undefined
10
hello world
```

```

var a;
a = undefined; // not error but not a good practice as undefined is a special value for variable
console.log(a) // output is undefined

```

The Scope Chain, Scope & Lexical Environment

Scope is related to lexical environment

```
function a() {
| console.log(b);
}
var b =10;
a();
```

Output is 10

```
function a() {
| c();
| function c(){
| | console.log(b);
| }
}
var b =10;
a();
```

Output is 10

```
function a() {
| var b =10;
| c();
| function c(){
| | console.log(b);
| }
}
a();
```

Output is 10

The screenshot shows a browser's developer tools with the "Console" tab selected. The title bar says "Namaste 🙏 JavaScript". The console output area shows the following:

```
Uncaught ReferenceError: b is not defined
at index.js:10
```

The source code pane on the right displays the following JavaScript code:

```
1 function a() {
2 | var b =10;
3 | c();
4 | function c(){
5 | | console.log(b);
6 | }
7 }
8 a();
9
10 console.log(b);
```

Scope directly depends on lexical environment

Lexical Environment

Whenever execution context is created lexical environment is also created

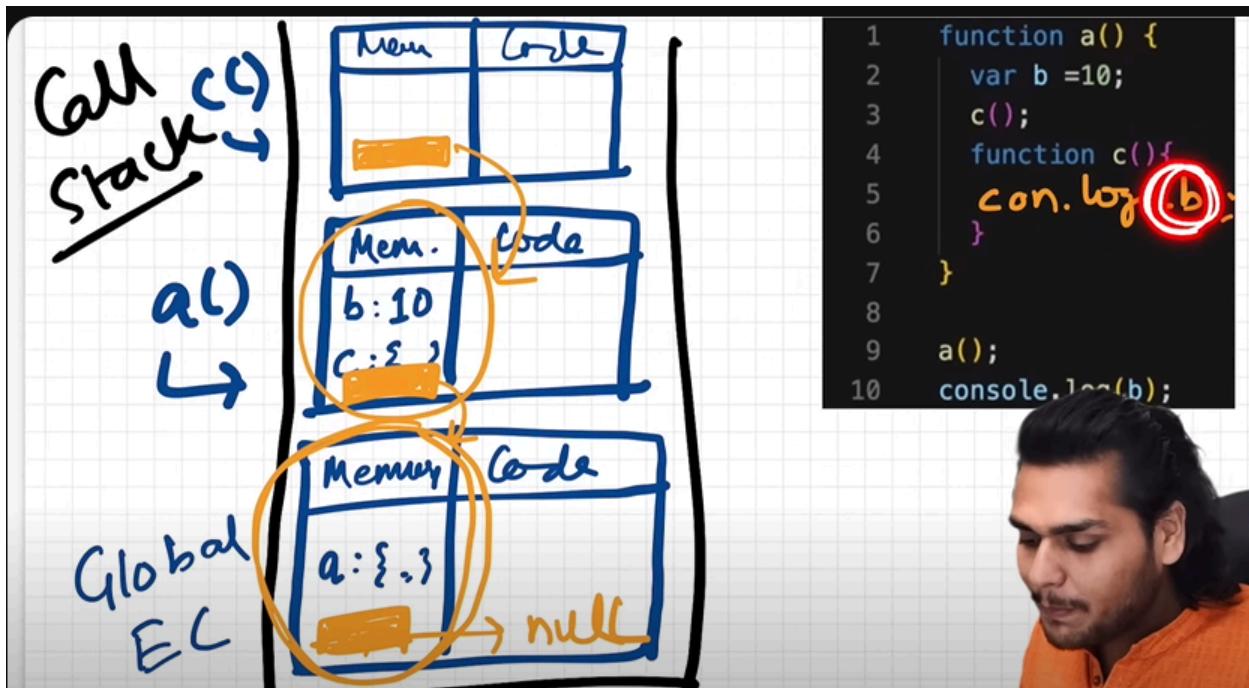
Lexical environment = local memory + reference to lexical environment of its parent

Lexical means hierarchy

Function c() is lexically inside function a() and function a() is lexically inside global environment

Scope chain

Chain of all the lexical environment and its parent reference



function a() {
 var b = 10;
 c();
 function c(){
 console.log(b);
 }
}
a();

{} Line 9, Column 1 Coverage: n/a

Paused on breakpoint

Call Stack

- c index.js:5
- a index.js:3
- (anonymous) index.js:9

Breakpoints

- index.js:5

Scope Watch

Global

- PERSISTENT: 1
- TEMPORARY: 0

- a: f a()
- addEventListener: f addEventListener()
- alert: f alert()
- atob: f atob()
- blur: f blur()
- btoa: f btoa()
- caches: CacheStorage {}

anonymous - global execution context in stack

a() function in stack

b() function in stack

U can find a() in global scope

Lexical environment of a() - Local memory + Global

Screenshot of the Chrome DevTools Sources tab showing a paused breakpoint at line 5 of index.js.

Code:

```
1 function a() {
2     var b = 10;
3     c();
4     function c(){
5         console.log(b);
6     }
7 }
8
9 a();
```

Breakpoint status: Paused on breakpoint (index.js:5)

Call Stack:

- c (index.js:5)
- a (index.js:3)
- (anonymous) (index.js:9)

Breakpoints:

- index.js:5 (checked)

Scope (Local):

- b: 10
- c: f c()
- this: Window

Watch (Global):

- Window

Lexical environment of c() - Local + Closure(a) + Global

Screenshot of the Chrome DevTools Sources tab showing a paused breakpoint at line 5 of index.js.

Code:

```
1 function a() {
2     var b = 10;
3     c();
4     function c(){
5         console.log(b);
6     }
7 }
8
9 a();
```

Breakpoint status: Paused on breakpoint (index.js:5)

Call Stack:

- c (index.js:5)
- a (index.js:3)
- (anonymous) (index.js:9)

Breakpoints:

- index.js:5 (checked)

Scope (Local):

- this: Window
- Closure (a)
- Global

```
▼ Local
  ► this: Window
▼ Closure (a)
  b: 10
▼ Global                               Window
  PERSISTENT: 1 ↗
  TEMPORARY: 0
  ► a: f a()
  ► addEventListener: f addEventListener()
  ► alert: f alert()
```