Programming

Namaste JavaScript Ep. 3

Hoisting in JavaScript 🔥(variables & functions)

JavaScript

Timestamp	27 December 2024
Resource	https://www.youtube.com/watch?v=Fnlnw8uY6jo&list=PLlasXeu85E9cQ32gLCvAvr9vNaUccPVNP&inde
Difficulty	Unraked

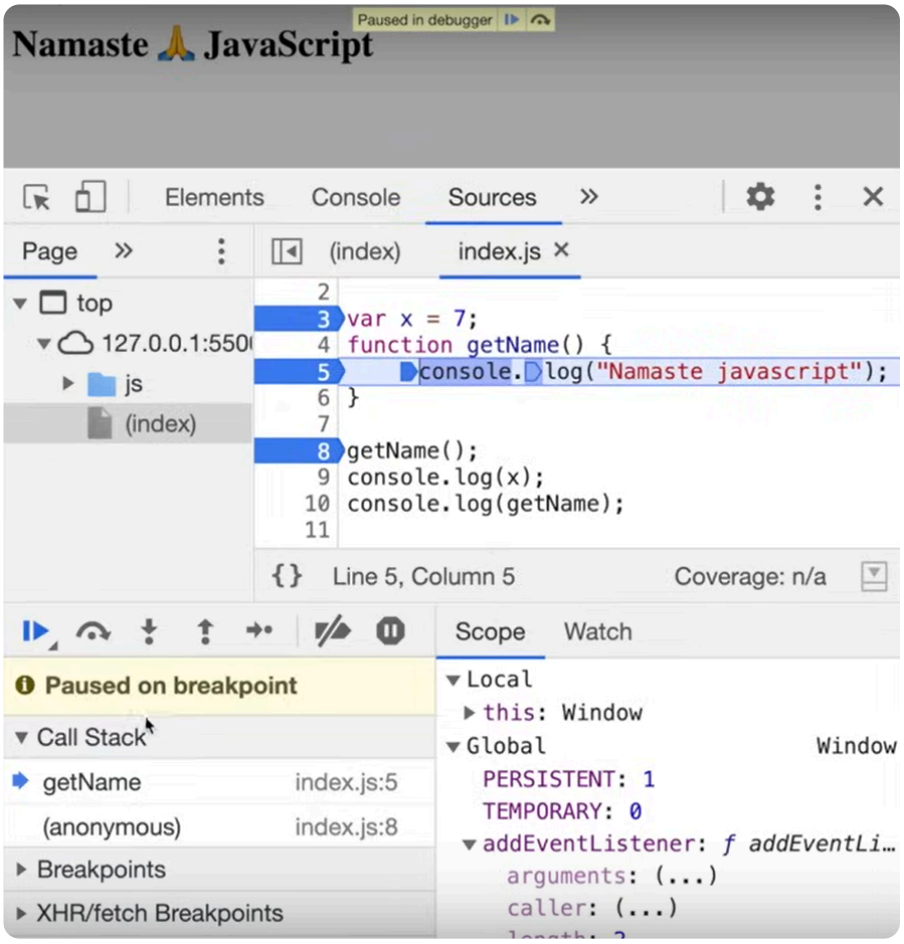
Lesson Main Points

Hoisting: Phenomenal in JavaScript by which you can access the variables and functions even before you have initialised it without any error.

Arrow Function: behaves like a variable in JavaScript.

Only a proper formal way of function initialisation will store the whole code in the exeContext.

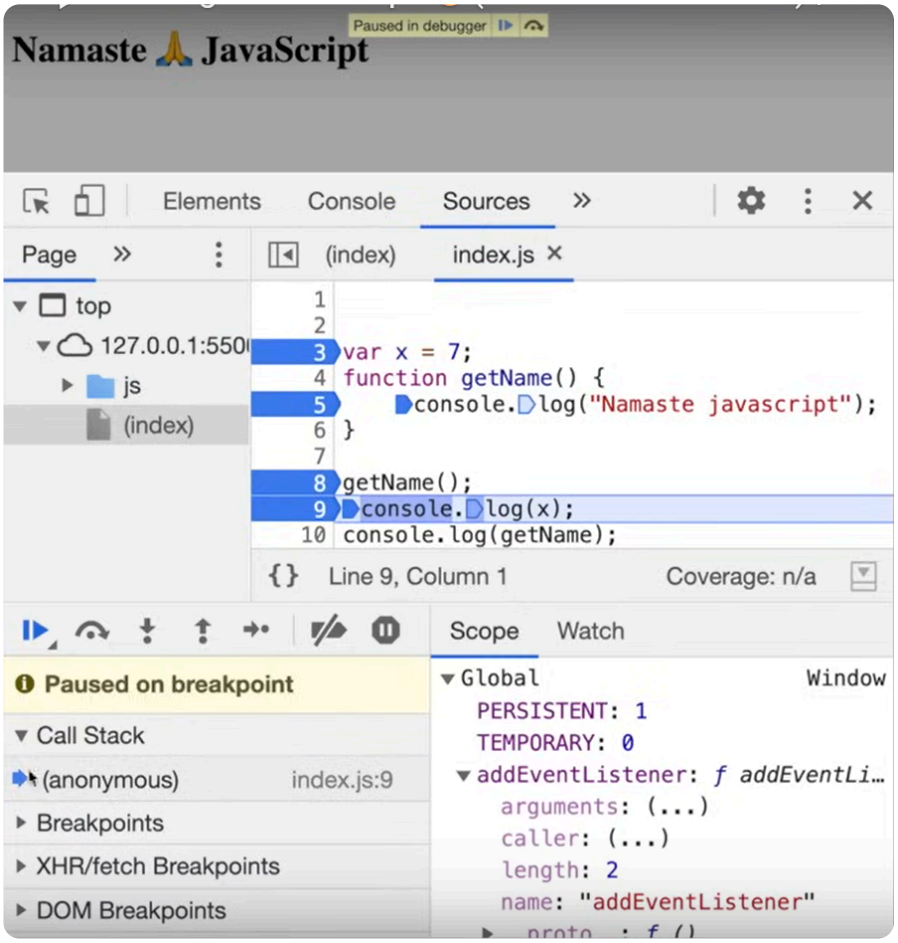
Demo of Call Stack in the Browser



Global ExeContext: (anonymous) currently at line 8 that leads to executing the getName()

Current function ExeContext at Line 5

Blue arrow indicates where the current control of the program (currenly at Line 5 for getName function operation)



Function Execution Context popped out of the stack and the control goes back to the global ExeContext and a following line has started its execution.

Short Notes

Case 1 | Call function and var after initialisation (Common)

```JavaScript
var x = 7;

function getName() {
  console.log("Namaste JavaScript");
}

getName();
console.log(x);
```

Output:

Namaste JavaScript

7

Case 2 | Call the functions and var before the initialisation

```
JavaScript
getName();
console.log(x);

var x = 7;

function getName() {
  console.log("Namaste JavaScript");
}
```

Output:

Namaste JavaScript

undefined

Case 3 | Call the function and the var that doesn't exist before the initialisation

```
JavaScript
getName();
console.log(x);

//var x = 7;

function getName() {
  console.log("Namaste JavaScript");
}
```

Output:

Namaste JavaScript

Uncaught ReferenceError: x is not defined at index.js:2

Case 4 | Console output for the function itself after initialisation

```
JavaScript
//getName();
//console.log(x);

var x = 7;

function getName() {
  console.log("Namaste JavaScript");
}

console.log(getName);
```

Output:

f getName() {
console.log("Namaste JavaScript");
}

Case 5 | Console output for the function itself before initialisation

JavaScript

```
//getName();
//console.log(x);
console.log(getName);

var x = 7;

function getName() {
  console.log("Namaste JavaScript");
}
```

Output:

f getName() {
console.log("Namaste JavaScript");
}

Case 6 | Calling function, var and function code before initialisation

JavaScript

```
getName();
console.log(x);
console.log(getName);

var x = 7;

function getName() {
  console.log("Namaste JavaScript");
}
```

Output:

Namaste JavaScript

undefined

f getName() {
console.log("Namaste JavaScript");
}

Case 7 | Call the **Arrow function** before initialisation

JavaScript

```
getName();
console.log(x);
console.log(getName);

var x = 7;

var getName = () => {
  console.log("Namaste JavaScript");
}

//another way of writing a function
but this will still count getName2 as
var
var getName2 = function () {
    console.log("Namaste
JavaScript");
}
```

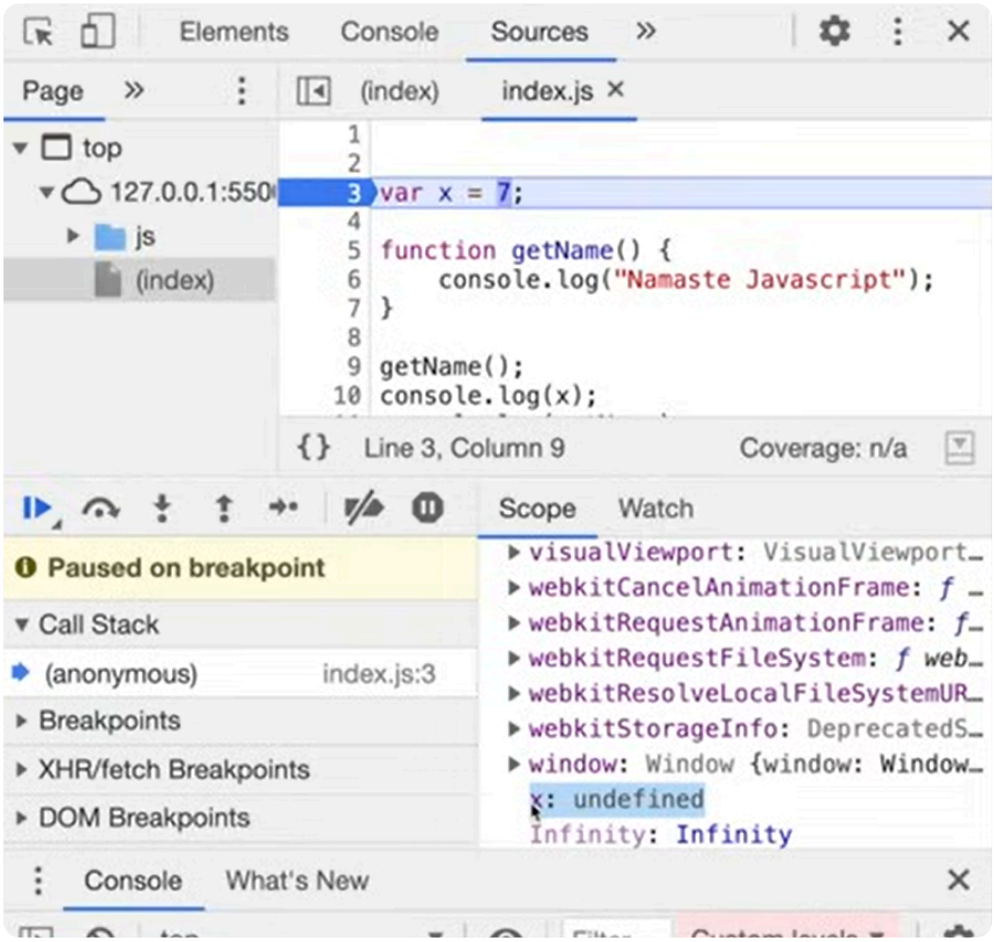
Output:

Uncaught RefereneceError: x is not defined at index.js:1

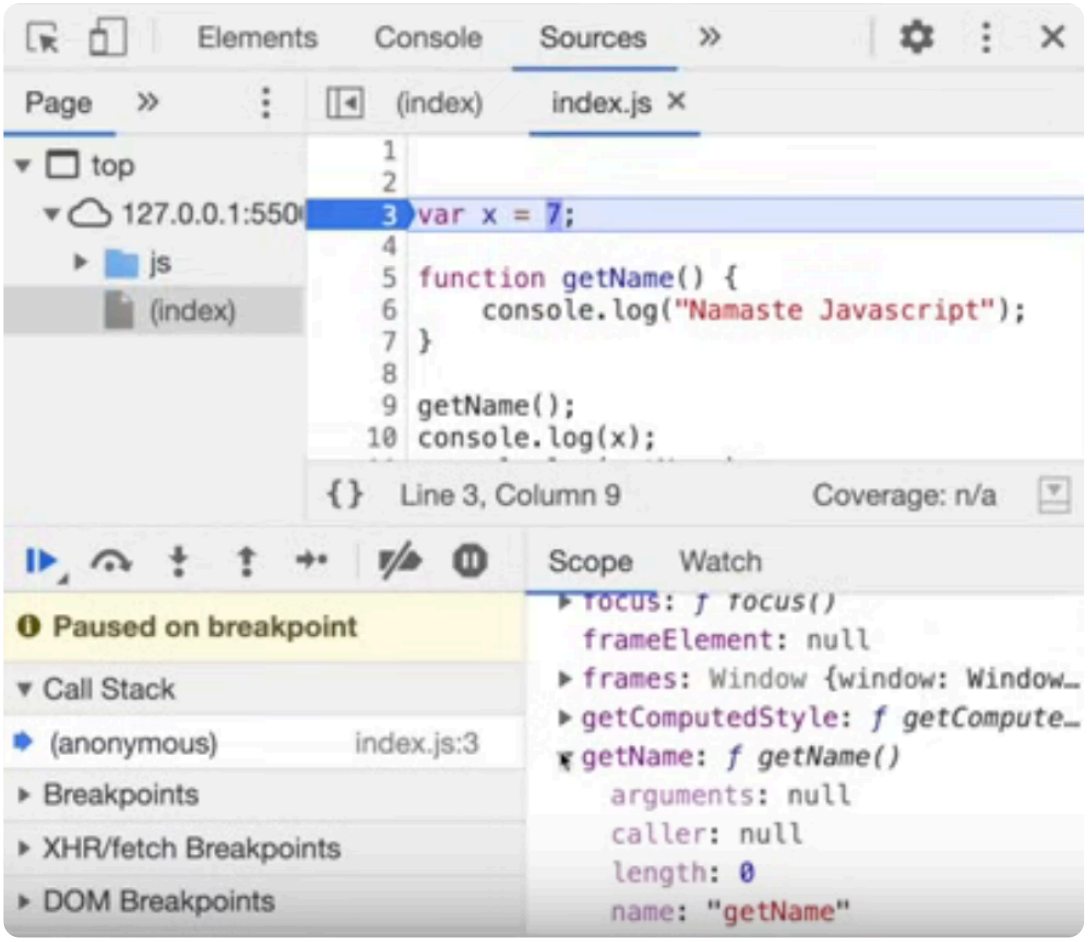
Takeaway

The whole concept of JavaScript Hoisting lies in the creation of Execution Context

- Even before the first line is executed JavaScript has reserved the memory for the variables.



- For the function, it stores the whole code (actual copy of the function) instead of reserving the memory space as undefined.


Elements
Console
Sources
Page
(index)
index.js
top
127.0.0.1:550
js
(index)
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
Paused on breakpoint
Call Stack
(anonymous) index.js:3
Breakpoints
XHR/fetch Breakpoints
DOM Breakpoints
focus: ƒ focus()
frameElement: null
frames: Window {window: Window…
getComputedStyle: ƒ getCompute…
getName: ƒ getName()
arguments: null
caller: null
length: 0
name: "getName"