Topics Covered

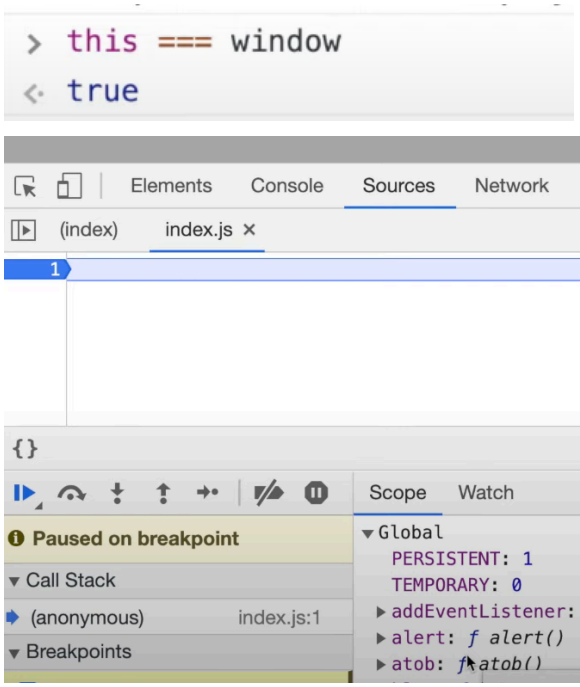
- Window & this keyword
- Hoisting in Javascript
- Functions in Javascript
- Undefined/Not defined in JS
- Scope chain, Scope and Lexical Environment

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

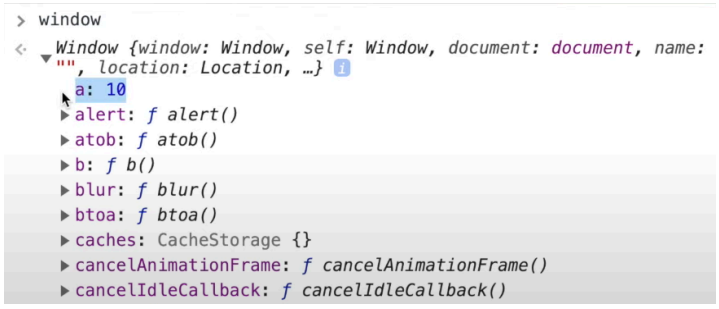


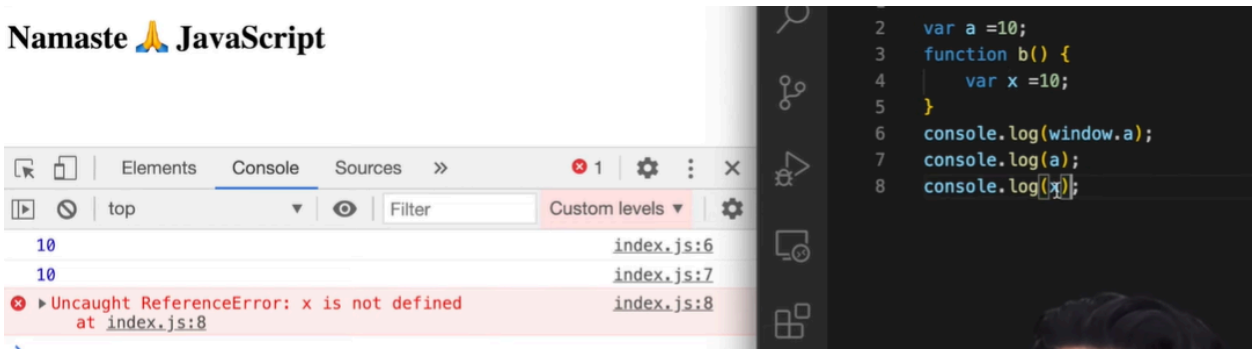
Global Space

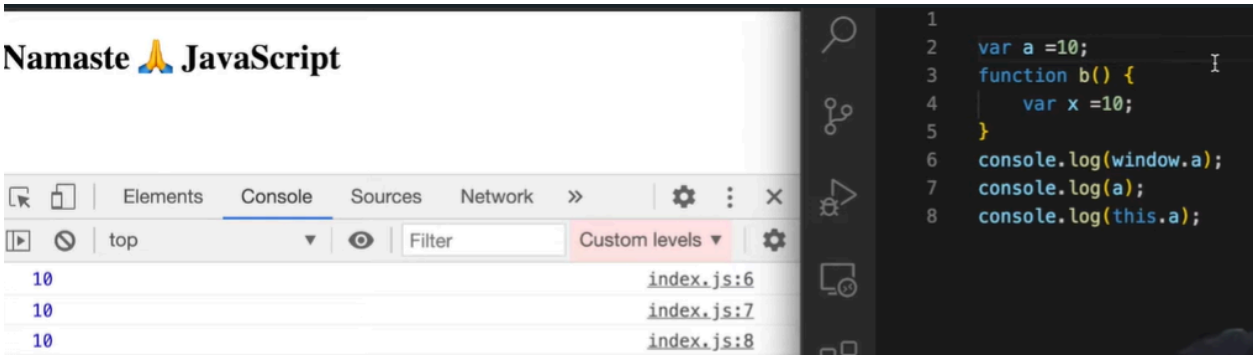
var a = 10;
Function b()

Local Space
{
var x = 10;
}

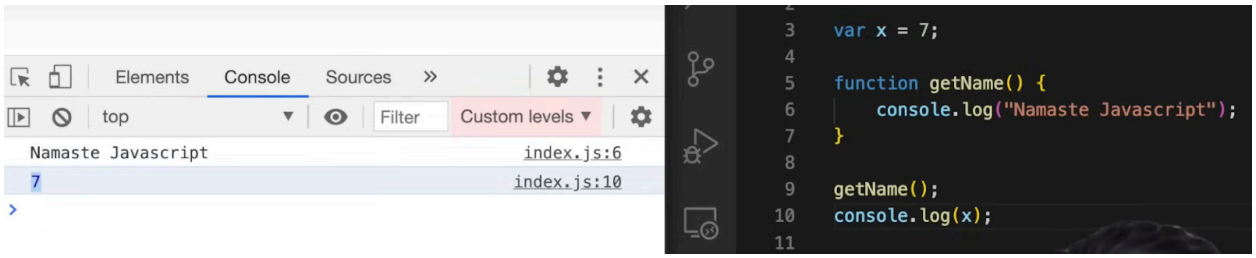
The variables and functions are attached to the global object window



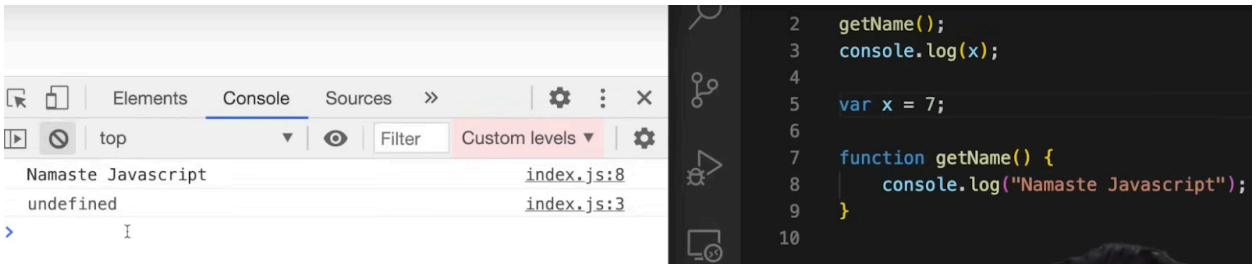




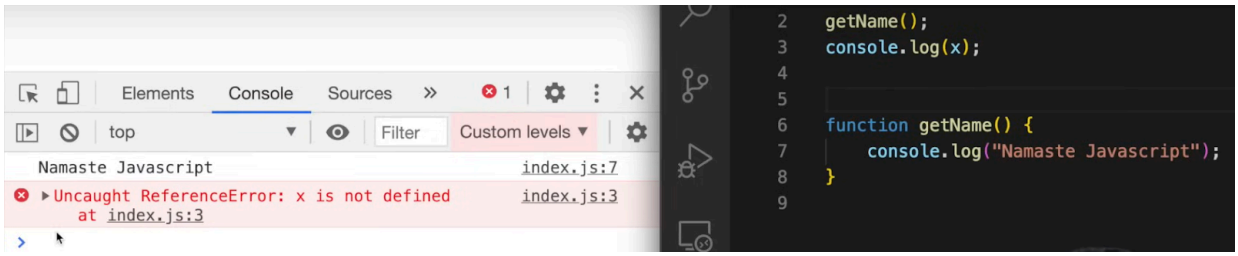
Hoisting in Javascript



Calling function and variable before declaration

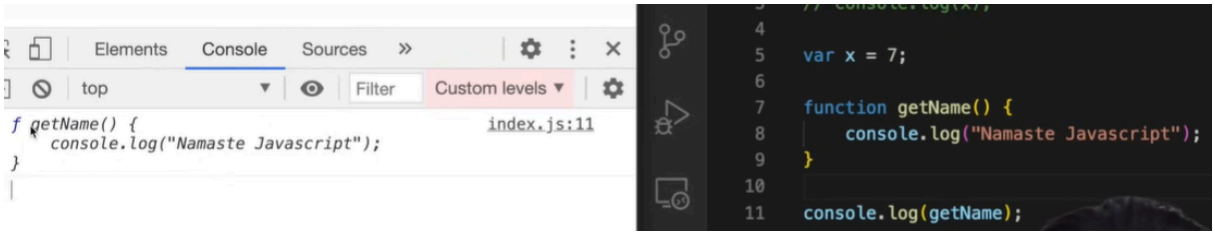


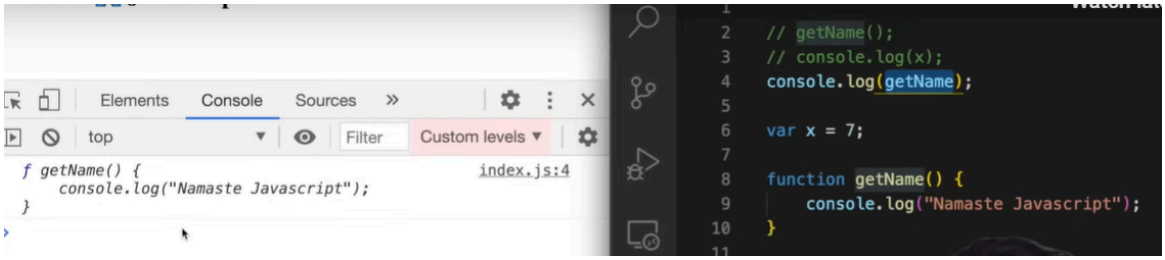
It would give error in other languages but not in javascript. U can call function and access the variable before its declaration



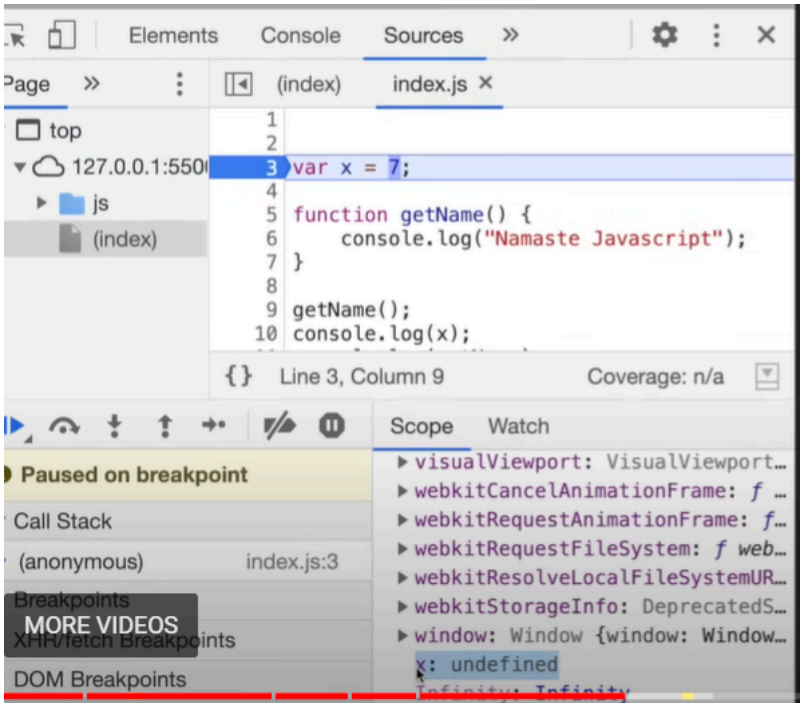
Hoisting

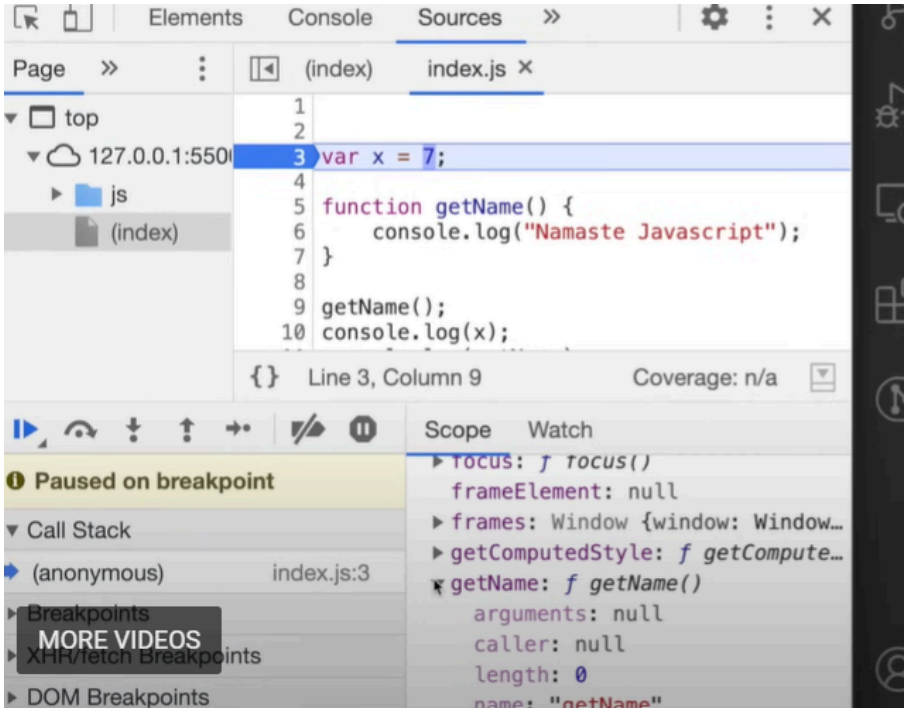
is a phenomenon in Javascript by which you can access these variables and functions even before u r initializing it. In Javascript before code executes Global execution context will get created and in memory phase itself variables with value undefined and wholefunction code will be copied in memory.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





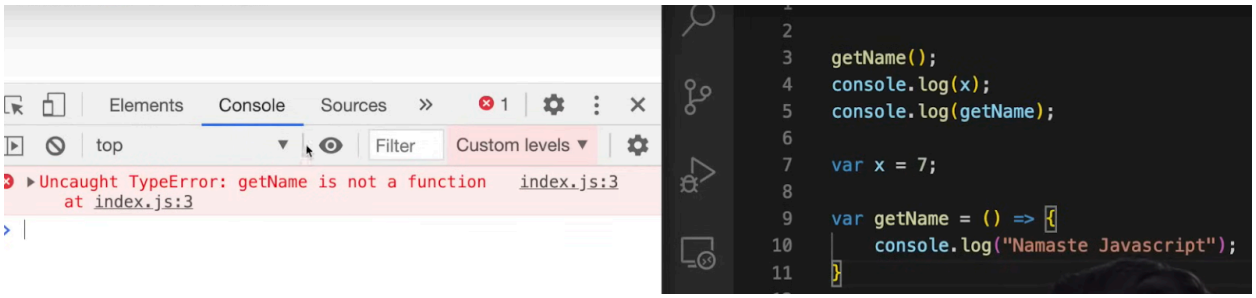
Even before the first line of execution, in memory creation phase x is undefined and for function whole function block code is copied in memory



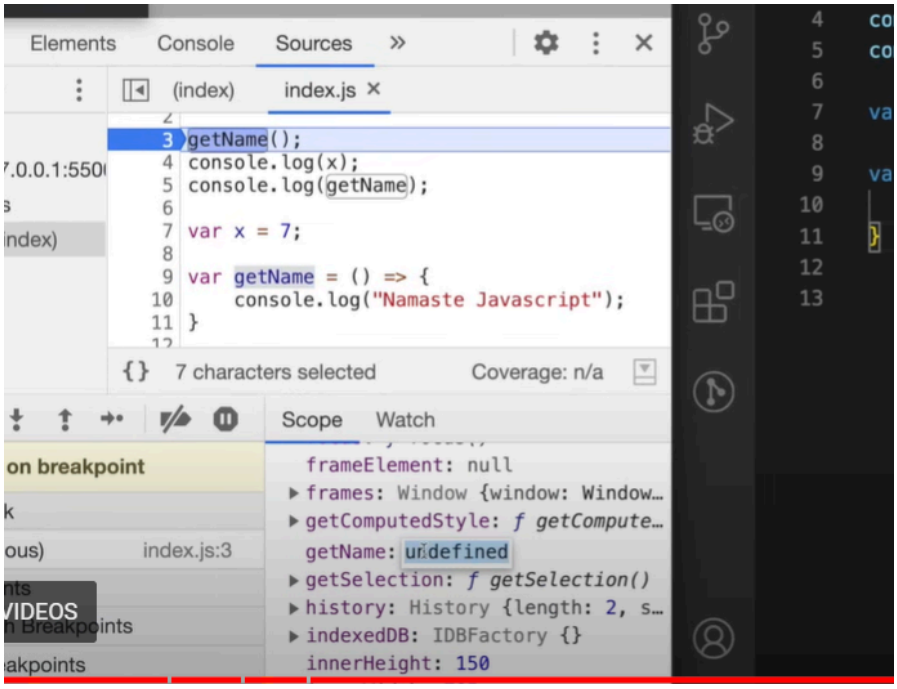


Arrow Function

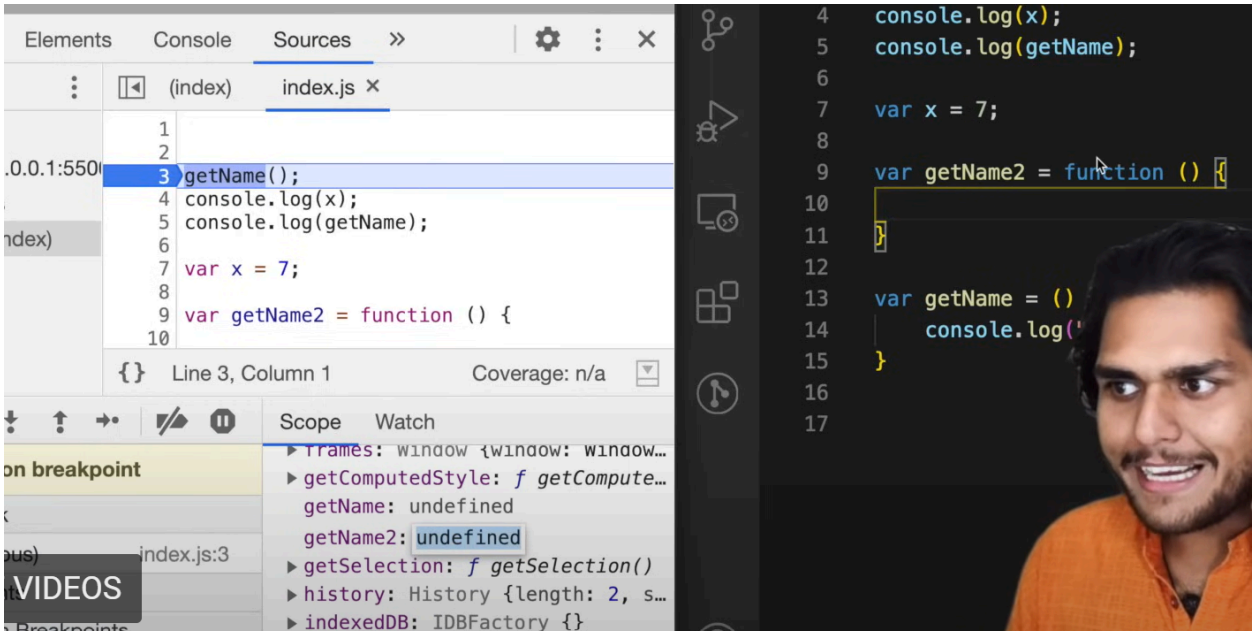
Error on line 3 itself



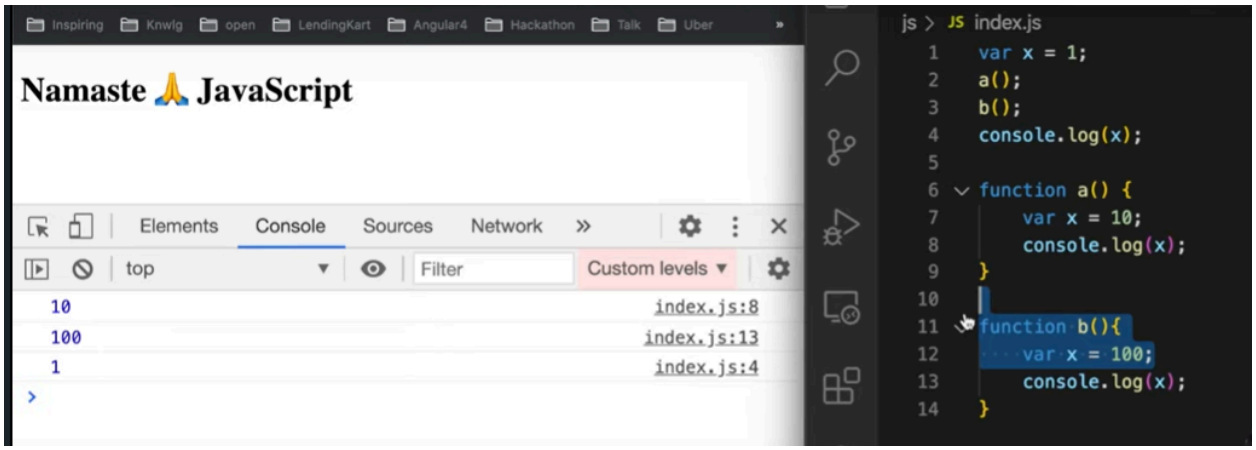
For arrow function, getName is treated as a variable not a function. Like for variable x, for variable getName also special value ‘undefined’ will be assigned



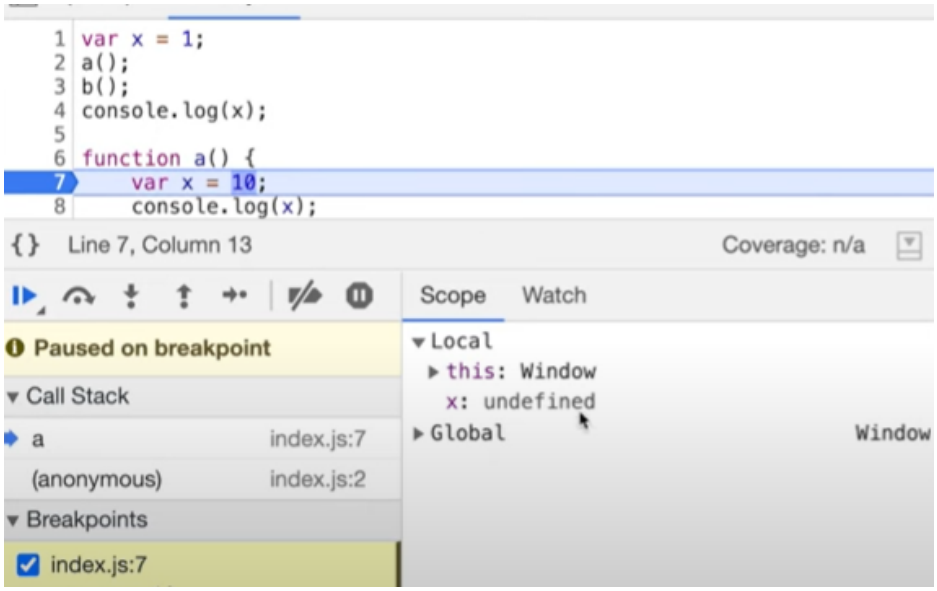
var getName2 = function () { …. } getName2 is a variable with value undefined
var getName = () => { … } getName is a variable with value undefined
function getName3() { …. } whole function code will get copied



Functions in Javascript



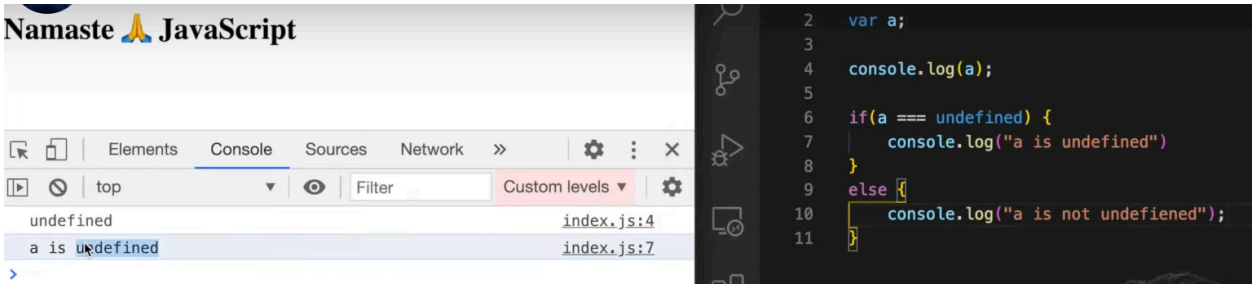
Local Memory,Global Memory, Global Execution context and a() inside call stack

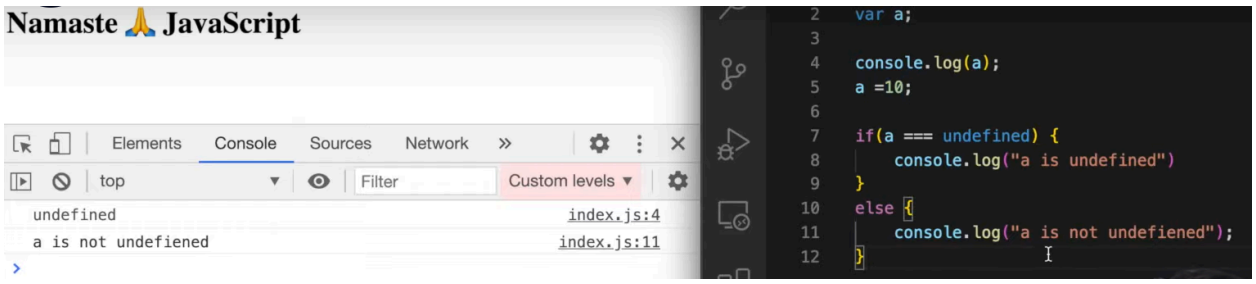


Undefined vs Not defined in Javascript

Undefined - special place holder value for variables before it is assigned
Not defined - variable not present in memory
Example

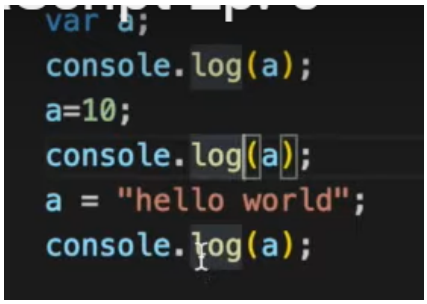
```
console.log(a);        //undefined
var a = 7;
console.log(a);        // 7
console.log(x);        // Reference Error - x is not defined
var b;
console.log(b);        // undefined
```





Loosely/ weakly typed language

It does not attaches variables to its datatype. U can put string in x and also u can put number in x in later point of time




```
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

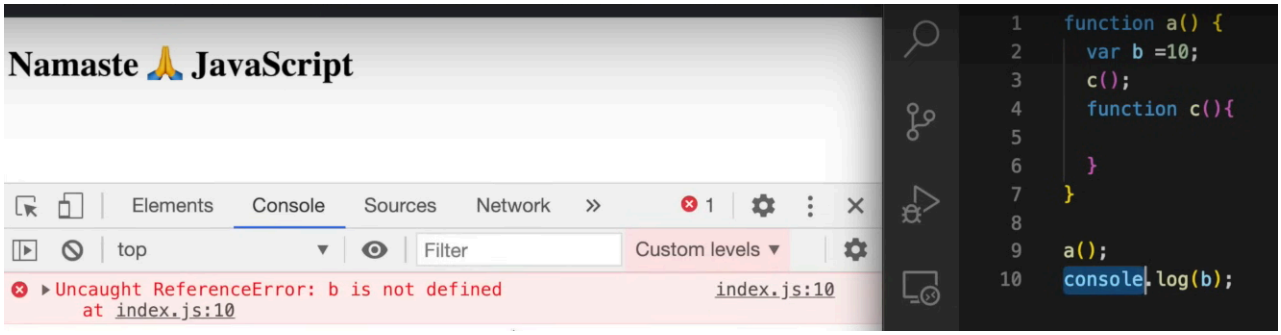
```
function a() {
  c();
  function c(){
    console.log(b);
  }
}
var b =10;
a();
```

Output is 10

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

a();
```

Output is 10



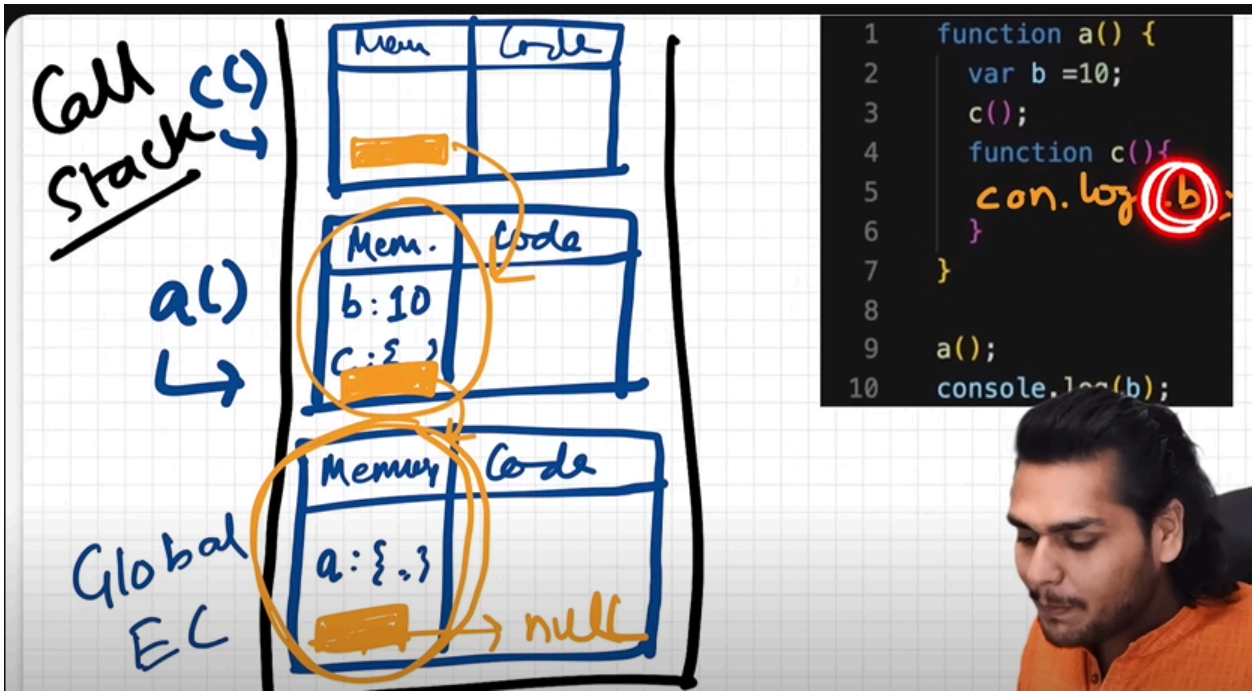
Scope directly depends on lexical environment

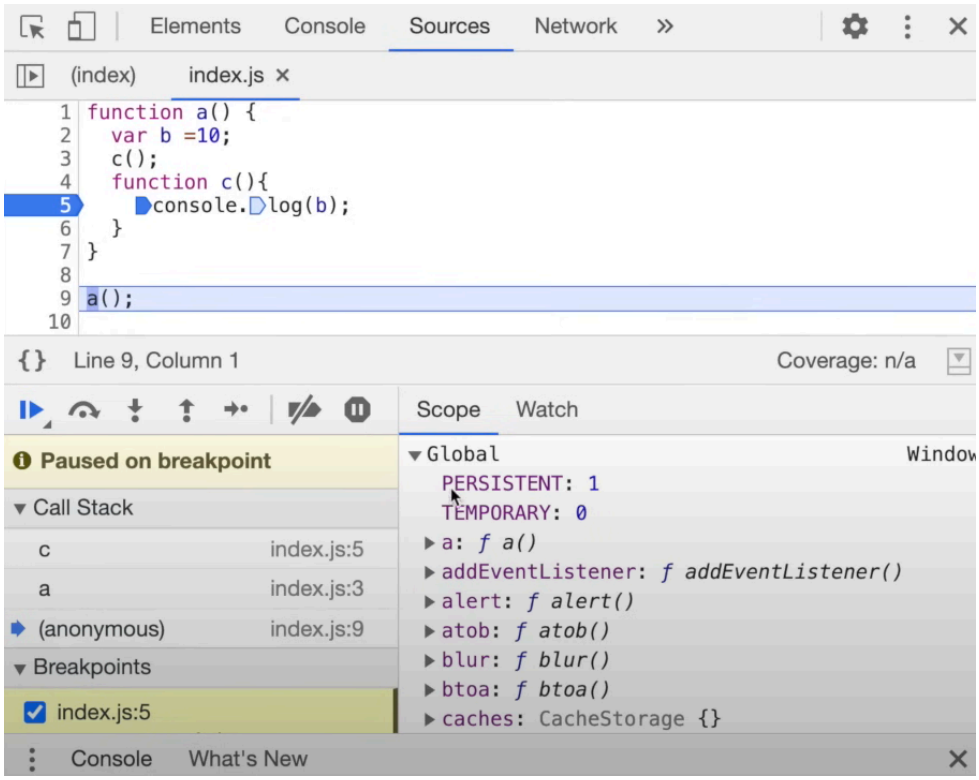
Lexical Environment

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

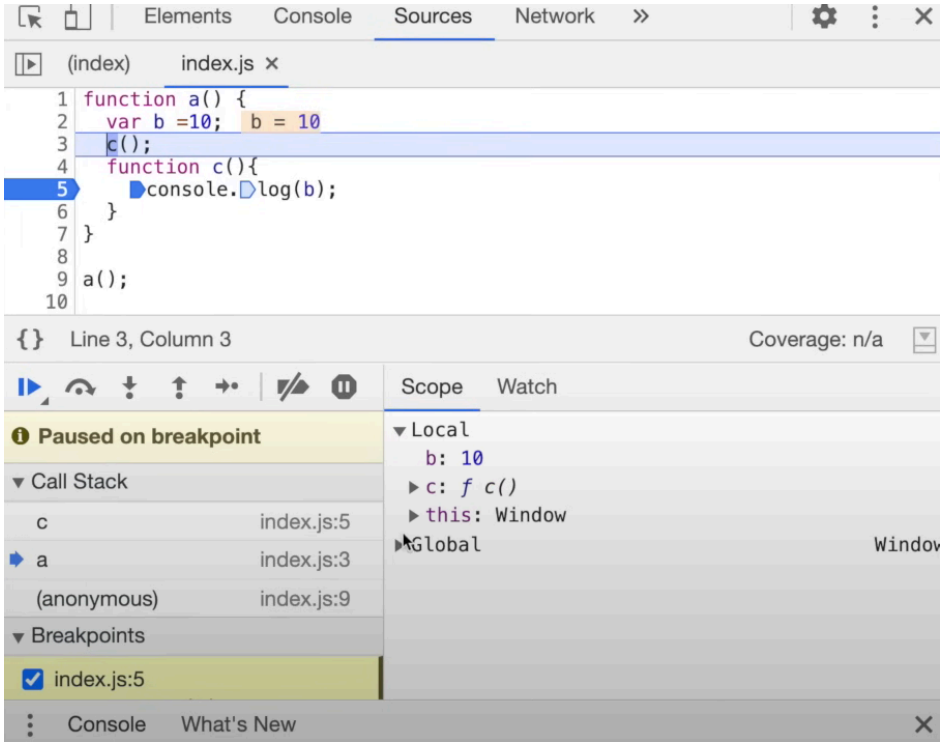




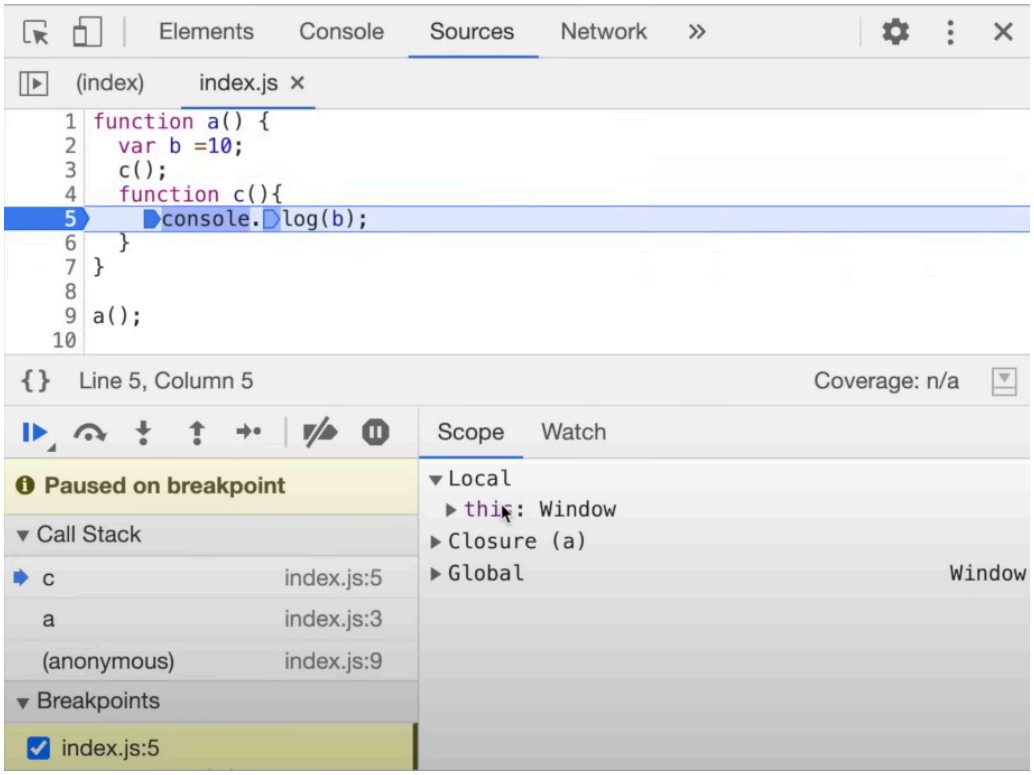
anonymous - global execution context in stack
a() function in stack
b() function in stack

U can find a() in global scope

Lexical environment of a() - Local memory + Global



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




Local
this: Window
Closure (a)
b: 10
Global
Window
PERSISTENT: 1
TEMPORARY: 0
a: ƒ a()
addEventListener: ƒ addEventListener()
alert: ƒ alert()