JavaScript Variable Scope and Hoisting Explained

JAN. 31 2013 102

In this post, we will learn JavaScript's variable scope and hoisting and all the idiosyncrasies of both.

We must understand how variable scope and variable hoisting work in JavaScript, if want to understand JavaScript well. These concepts may seem straightforward; they are not. Some important subtleties exist that we must understand, if we want to thrive and excel as JavaScrip developers.

menu

Receive Updates

Your Email: Go

Variable Scope

A variable's scope is the context in which the variable exists. The scope specifies from where you can access a variable and whether you have access to the variable in that context.

Variables have either a local scope or a global scope.

Local Variables (Function-level scope)
Unlike most programming languages, JavaScript does not have block-level scope (variables scoped to surrounding curly brackets); instead, JavaScript has function-level scope. Variables declared within a function are local variables and are only accessible within that function or by functions inside that function. See my post on Closures for more on accessing variables in outer functions from inner functions.

Demonstration of Function-Level Scope

```
1  var name = "Richard";
2
3  function showName () {
4        var name = "Jack"; // local variable; only accessible in this showName function
5        console.log (name); // Jack
6  }
7  console.log (name); // Richard: the global variable
```

No Block-Level Scope

```
1  var name = "Richard";
2  // the blocks in this if statement do not create a local context for the name variable
3  if (name) {
4        name = "Jack"; // this name is the global name variable and it is being changed to "Jack" here
5        console.log (name); // Jack: still the global variable
6  }
7
8  // Here, the name variable is the same global name variable, but it was changed in the if statement
9  console.log (name); // Jack
```

- **If You Don't Declare Your Local Variables, Trouble is Nigh**
  Always declare your local variables before you use them. In fact, you should use JSHint to check your code for syntax errors and style guides. Here is the trouble with not declaring

local variables:

```
// If you don't declare your local variables with the var keyword, they are part of the global scope
var name = "Michael Jackson";

function showCelebrityName () {
        console.log (name);
}

function showOrdinaryPersonName () {
        name = "Johnny Evers";
        console.log (name);
}
showCelebrityName (); // Michael Jackson

// name is not a local variable, it simply changes the global name variable
showOrdinaryPersonName (); // Johnny Evers

// The global variable is now Johnny Evers, not the celebrity name anymore
showCelebrityName (); // Johnny Evers

// The solution is to declare your local variable with the var keyword
function showOrdinaryPersonName () {
        var name = "Johnny Evers"; // Now name is always a local variable and it will not overwrite th
        console.log (name);
}
```

- **Local Variables Have Priority Over Global Variables in Functions**

If you declare a global variable and a local variable with the same name, the local variable will have priority when you attempt to use the variable inside a function (local scope):

```
var name = "Paul";

function users () {
        // Here, the name variable is local and it takes precedence over the same name variable in the
var name = "Jack";

```

```
// The search for name starts right here inside the function before it attempts to look outside the f
console.log (name);
}

users (); // Jack
```

Global Variables

All variables declared outside a function are in the global scope. In the browser, which is what we are concerned with as front-end developers, the global context or scope is the window object (or the entire HTML document).

- Any variable declared or initialized outside a function is a global variable, and it is therefore available to the entire application. For example:

```
// To declare a global variable, you could do any of the following:
var myName = "Richard";

// or even
firstName = "Richard";

// or
var name; //
name;
</pre>
```

- If a variable is initialized (assigned a value) without first being declared with the var keyword, it is automatically added to the global context and it is thus a global variable:

```
function showAge () {
        // Age is a global variable because it was not declared with the var keyword inside this functic
        age = 90;
        console.log(age);//
}

showAge (); // 90

// Age is in the global context, so it is available here, too
console.log(age); // 90
```

Demonstration of variables that are in the Global scope even as they seem otherwise:

```
// Both firstName variables are in the global scope, even though the second one is surrounded by
var firstName = "Richard";
{
var firstName = "Bob";
}

// To reiterate: JavaScript does not have block-level scope

// The second declaration of firstName simply re-declares and overwrites the first one
console.log (firstName); // Bob
```

Another example

```
for (var i = 1; i <= 10; i++) {
        console.log (i); // outputs 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
};

// The variable i is a global variable and it is accessible in the following function with the last value
function aNumber () {
console.log(i);
}

// The variable i in the aNumber function below is the global variable i that was changed in the for
aNumber ();  // 11

```

- **setTimeout Variables are Executed in the Global Scope**

Note that all functions in setTimeout are executed in the global scope. This is a tricky bit; consider this:

```
 // The use of the "this" object inside the setTimeout function refers to the Window object, not to

var highValue = 200;
```

```
var constantVal = 2;
var myObj = {
        highValue: 20,
        constantVal: 5,
        calculateIt: function () {
 setTimeout (function  () {
        console.log(this.constantVal * this.highValue);
}, 2000);
        }
}

// The "this" object in the setTimeout function used the global highValue and constantVal variables

myObj.calculateIt(); // 400
// This is an important point to remember.
```

Do not Pollute the Global Scope

If you want to become a JavaScript master, which you certainly want to do (otherwise you will be watching Honey Boo Boo right now), you have to know that it is important to avoid creating many variables in the global scope, such as this:

```
// These two variables are in the global scope and they shouldn't be here
var firstName, lastName;

function fullName () {
        console.log ("Full Name: " + firstName + " " + lastName );
}
```

This is the improved code and the proper way to avoid polluting the global scope

```
// Declare the variables inside the function where they are local variables

function fullName () {
        var firstName = "Michael", lastName = "Jackson";

        console.log ("Full Name: " + firstName + " " + lastName );
}
```

In this last example, the function fullName is also in the global scope.

Variable Hoisting

All variable declarations are hoisted (lifted and declared) to the top of the function, if defined in a function, or the top of the global context, if outside a function.

It is important to know that only variable declarations are hoisted to the top, not variable initialization or assignments (when the variable is assigned a value).

Variable Hoisting Example:

```
1  function showName () {
2  console.log ("First Name: " + name);
3  var name = "Ford";
4  console.log ("Last Name: " + name);
5  }
6
7  showName ();
8  // First Name: undefined
9  // Last Name: Ford
10
11 // The reason undefined prints first is because the local variable name was hoisted to the top of the fur
12 // Which means it is this local variable that get calls the first time.
13 // This is how the code is actually processed by the JavaScript engine:
14
15 function showName () {
16       var name; // name is hoisted (note that is undefined at this point, since the assignment happens b
17 console.log ("First Name: " + name); // First Name: undefined
18
19 name = "Ford"; // name is assigned a value
20
21 // now name is Ford
22 console.log ("Last Name: " + name); // Last Name: Ford
23 }
```

Function Declaration Overrides Variable Declaration When Hoisted

Both function declaration and variable declarations are hoisted to the top of the containi

scope. And function declaration takes precedence over variable declarations (but not over variable assignment). As is noted above, variable assignment is not hoisted, and neither is function assignment. As a reminder, this is a function assignment: var myFunction = function () {}.

Here is a basic example to demonstrate:

```
1  // Both the variable and the function are named myName
2  var myName;
3  function myName () {
4  console.log ("Rich");
5  }
6
7  // The function declaration overrides the variable name
8  console.log(typeof myName); // function
```

```
1  // But in this example, the variable assignment overrides the function declaration
2  var myName = "Richard"; // This is the variable assignment (initialization) that overrides the function dec
3
4  function myName () {
5  console.log ("Rich");
6  }
7
8  console.log(typeof myName); // string
```

It is important to note that function expressions, such as the example below, are not hoisted.

```
1  var myName = function () {
2  console.log ("Rich");
3  }
```

In strict mode, an error will occur if you assign a variable a value without first declaring the variable. Always declare your variables.

Be good. Sleep well. And enjoy coding.

Posted in: 16 Important JavaScript Concepts, JavaScript / Tagged: Learn JavaScript, Variable Hoisting, Variable Scope

Richard

Thanks for your time; please come back soon. Email me here: javascriptissexy at gmail email, or use the contact form.

102 Comments

Rameş Aliyev

February 1, 2013 at 2:52 am / Reply

In for loop, when i create “i” with “var”, why it becomes global? I try it and see it, its no make sense, is it a bug of js?

Here is live example
http://jsbin.com/azawaj/1/edit

Richard Bovell (Author)

February 3, 2013 at 11:59 am / Reply

Hello Rames,

If you are referring to the example below, i is a global variable because it was not defined in a function, and it is therefore part of the global scope. Even if the variable i were declared and initialized in the for loop, it would still be a global variable because JavaScript does not have block-level scope, so the block (curly brackets) from the for loop do not create a local scope.

Here is the code:

_____

for (var i = 1; i <= 10; i++) { console.log (i); // outputs 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; }; // i is a the global variable and it is accessible here with the last value it was assigned above function aNumber () { console.log(i); // 11 } aNumber (); // 10 _____ The code is valid, just copy and paste it and run it in your browser's console.

isabel marant sneakers

February 2, 2013 at 9:21 am / Reply

"Its always good to read tips as you share with regard to blog placing. As I just now started placing comments with regard to blog plus facing challenge of loads of rejections. I believe that your suggestion will be helpful personally. I will allow you to know if perhaps its improve me too. "

Vetri

February 13, 2013 at 4:55 am / Reply

What do you mean by this?

It is important to note that function expressions, such as the example below, are not hoisted.

var myName = function () {
console.log ("Rich");
}

but I can see this hosted.

Richard Bovell (Author)

February 13, 2013 at 2:29 pm / Reply

@Vetri The function expression, var myName = function () {console.log (“Rich”);}, is not actually hoisted. Instead, if you initialize it after the var myName = “Rich” initialization, the myName variable is assigned a new value, which is the function.

It looks like this when the code is executed:
var myName;
myName = “Rich”;

// Then it is assigned a new value, the function
myName = function () {console.log (“Rich”);}

Ramon Royo

February 15, 2013 at 1:51 pm / Reply

Hi Richard,

thank you for the article, I wasn’t aware of the scope for ‘this’ inside timeouts and intervals.
I just wanted to point out a small problem in one of the examples, the variable names in the following example, should be surrounded by double quotes, as now they’re checked as properties inside window:

console.log(myName in window);
console.log(firstName in window);

Should be:

console.log(“myName” in window); // true
console.log(“firstName” in window); // true

Richard Bovell (Author)

February 15, 2013 at 5:08 pm / Reply

Thanks for pointing out the typo, Ramon. I just fixed it.

Sean

March 6, 2013 at 10:29 pm / Reply

Great Post!! Quick Question Richard.. So basically theres no need to declare variables outside of a function if i can declare them within a function.. basically in what case would I need to declare variables outside a function when its easier to use them inside a function as below... it seems like local variables are easier to work with then global...

```
function myFunction () {
var a = “foo”
}

function newFunction () {
var a = “boo”
}
myfunction()
newFunction()
```

why would i want to declare a variable outside a function and I can make one inside a function via local scope and use them later???
Thanks!!!!

Richard Bovell (Author)

March 7, 2013 at 3:36 pm / Reply

> *So basically theres no need to declare variables outside of a function if i can declare them within a function.*

You are correct. It is best to just use local variables, unless you need a global variable.

> *basically in what case would I need to declare variables outside a function when its easier to use them inside a function as below*

There are times when you do need global variables, for example, when you want any function in your application to have access to the variable. Keep in mind your other functions wouldn't have access to your local variables, so if you want other functions to access a variable, you would have to declare it as a global variable or pass it via the parameter to a function.

There are many times when you are building a large application where you will have one or more global variables.

Sean

March 7, 2013 at 4:13 pm / Reply

Great!! thanks for that, closures and hoisting are a bit confusing but after reading this it totaly makes sense…im on my journey reading and learning from this site along with Nicholas Zakas Professional Book … back to reading & coding 🙂

whaleshark

May 8, 2013 at 12:49 pm / Reply

In your example for hoisting, you say that the variable assignment will override the function declaration — well, in the Firefox (20) console it does not. Is that a browser bug? I see it works in Chrome.

thanks

Richard Bovell (Author)

May 8, 2013 at 9:42 pm / Reply

Good question and good research, whaleshark.

You are correct that the overriding is working as expected in Chrome, while it is not in Firefox. The reason is that in Firefox, the second use of “myName” is a reassignment, so essentially, Firefox is assigning a new value to the myName variable. It is assigning the function to it.

This is how the code is executed in Firefox:

```
1
2  // A new myName global variable is created in Firefox and assigned the va
3  var myName = "Richard";
4
5  /* The same global myName variable is now assigned a new value. The firs
6  myName = function () {
7  console.log ("Rich");
8  }
9
```

Now, you might be wondering why it is not executed the same way by both browsers. Welcome to land of Browsers gone rogue. All the browsers have similar but different implementation for many os JavaScript’s API, so you have to watch out for these kinds of side effects, which is why using the same global variable in this manner is not a good idea to begin with.

A N Sinha

May 16, 2013 at 4:40 pm / Reply

Hi Richard,

Thanks your very much 🙂

JavaScript Variable Scope :
==========
for (var i = 1; i <= 10; i++) {
console.log (i); // outputs 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
};
function aNumber () {
console.log(i); // 11
}
aNumber ();
==========
I REALLY SURPRISE
var i is define inside the for loop it treated as a global variable.

I have test the code in firebug, its treated as a global variable.....

that's great...

thanks your very much 🙂

A.N. Sinha 🙂

Chachaching

November 19, 2013 at 1:32 pm / Reply

The behavior you are seeing right now on JavaScript is a new change in JavaScript. The variable scope used to be the same as other languages, but I also noticed the same thing a while back that there is no longer a variable scope WITHIN a function. In other words, once you start using/assigning a new variable inside a function, the variable will live through out the function scope regardless there are other sub scopes inside the function. I am still trying to find out whether this is a bug or intention of JavaScript (and it is definitely unconventional idea for all other computer languages).

Chachaching

November 19, 2013 at 1:38 pm / Reply

One more note, a variable you declared outside any function but inside the script tag will be GLOBAL regardless where you declare it and with or without 'var' keyword in front. To me, this idea is silly and it will break any variable scope idea.

Richard Of Stanley (Author)

November 22, 2013 at 7:31 am / Reply

Good observation, Chachaching.

Modern browsers add different implementations for how to hand variable scope and when to garbage collect said variables.

There was a vibrant discussion on this issue on Hacker News a few months back. If I find the link, I will post it.

Mike

May 24, 2013 at 11:38 am / Reply

Great site! Your articles are perfect for experienced Javascript developers who want to round out their knowledge and get into the fine details. Thanks!

Richard Bovell (Author)

May 27, 2013 at 10:06 am / Reply

Thanks very much, Mike.

Eddie Romeiro

July 10, 2013 at 5:29 pm / Reply

Hi there. Great post and site. In regards to hoisting, there was a little confusion (at least for me) in your explanation about function hoisting. It should be made clear that function declarations get fully hoisted to the top, while function expressions behave like variables. Maybe even add that function declarations get hoisted and turn into an expression at the top. Just my opinion.

Richard Bovell (Author)

July 11, 2013 at 7:50 pm / Reply

Hi Eddie,
Your opinion is actually the fact: function declarations are indeed hoisted while function expressions are not.

I did note this in the article, maybe you missed it. Here is the quote from the article:

> *Both function declaration and variable declarations are hoisted to the top of the containing scope. And function declaration takes precedence over variable declarations (but not over variable assignment). As is noted above, variable assignment is not hoisted, and neither is function assignment. As a reminder, this is a function assignment: var myFunction = function () {}.*

abeing

August 30, 2013 at 10:09 am / Reply

Hi, great article!
I did this:
function doSomething(){
function formatSomething(element){
return “”+element.text+””;
}

```
anObject({ format: formatSomething });
anotherObject({format: formatSomething });
}
```

a function within a function, which I find very useful, because it worked and I had the function at the place where I need it.
But is this good coding style?

Richard Bovell (Author)

August 30, 2013 at 4:17 pm / Reply

> *a function within a function, which I find very useful, because it worked and I had the function at the place where I need it.*
> *But is this good coding style?*

If your question is whether a function inside a function (better known as a closure) is a good coding style, then the answer is a definite yes.

Closures are used a lot by advanced JavaScript coders, and probably by most JavaScript coders in general. And sometimes it is even tough to avoid using a closure, so feel free to use closures liberally.

I actually wrote an article on just closures—see the archive.

Foysal Mamun

September 30, 2013 at 9:21 am / Reply

thanks nice post.

Deepak

October 3, 2013 at 7:58 am / Reply

what happens when i am not used var for delectation. just used name; instated of var name.?

Richard Of Stanley (Author)

October 11, 2013 at 2:41 pm / Reply

If you do not use var, the variable will be a global variable defined on the Window object. But this can cause trouble, especially when using strict mode.

Gulsen

October 6, 2013 at 1:31 am / Reply

Hi Richard,

Thanks for the great article.

In your example with the celebrity name and ordinary person name, the first call to showCelebrityName() function, it logs “Michael Jackson”, and the second time “Johnny Evers”. Since the ShowOrdinaryPersonName is defined before the first showCelebrityName() call, does that mean that the name variable is changed to “Johnny Evers” once the ShowOrdinaryPersonName() is called but not when it is defined? Just to make sure I understand it correctly…

Thanks,
Gulsen

Richard Of Stanley (Author)

October 11, 2013 at 2:56 pm / Reply

You are correct, Gulsen.

Vipul

October 6, 2013 at 11:54 am / Reply

Hi Richard,

Thanks for the great article.
I am new to Javascript, learnt new things from your posts. Actually i am focusing as a Javascript developer as my carrier path. Hope it will help me.

Thanks,
Vipul

Richard Of Stanley (Author)

October 11, 2013 at 2:25 pm / Reply

You are welcome, and much success to you in your JavaScript career.

Alisa

October 9, 2013 at 10:06 pm / Reply

Hello,
In gratitude for your informative article, I offer you some proofreading corrections…
“Jhonny Evers”
“global scape”
“Variable Hosting”

Cheers!

Richard Of Stanley (Author)

October 11, 2013 at 2:25 pm / Reply

You rock, Alisa. Thanks very much.

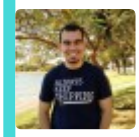
I care much about grammar mistakes: I don’t like them so I like to get rid of them promptly. The trouble is that it seems like you can never get rid of **all** grammar mistakes.

Richard Of Stanley (Author)

October 11, 2013 at 2:51 pm / Reply

Hi Alisa,

Those were some difficult to catch errors, IMO. Are you a professional proofreader or a grammar guru? I am very curious to know 🙂



Eric Oliveira

October 11, 2013 at 12:04 pm / Reply

You are a master of the didactics, your articles are simply amazing! Thank you for this excellent work!

Richard Of Stanley (Author)

October 11, 2013 at 2:28 pm / Reply

You made my day, Eric. Thanks very much. I am very happy that others can learn and benefit from y articles.

yougen

October 28, 2013 at 11:28 pm / Reply

Hi,
“the variable assignment overrides the function declaration” example is not exactly accurate.
function declaration is always precedence variable declaration. What happens is Execution Context has creation stage and code execution stage. Hoisting happens in creation stage. More info here: http://davidshariff.com/blog/what-is-the-execution-context-in-javascript/

I modify the code as following:

```
console.log(typeof myName); // function , NOT string
var myName = “Richard”;

function myName () {
console.log (“Rich”);
}
```

Richard Of Stanley (Author)

October 29, 2013 at 3:55 pm / Reply

Yougen,

The variable assignment always overrides the function declaration. The reason it shows as function in your example code is because first the variables are hoisted (both the function and variable). Since they haven’t been assigned a value yet, the function declaration takes precedence, as I noted in the article. Because function declaration takes precedence over variable declaration.

But then in the execution context when variable assignment occurs, the variable is assign the value “Richard” and it takes precedence over the function declaration. If the function were a function expression of this form: var myName = function () {}, then it would have had precedence over the variable assignment. But because only the variable is assigned a value, it takes precedence.

Here is proof:

```
1
2  console.log("Before: " + typeof myName); // Before: function
3  var myName = "Richard";
4
5  function myName () {
6  console.log ("Rich");
7  }
8
9  console.log("After: " + typeof myName); // After: string
```

```
10
```

Fon

November 5, 2013 at 2:21 am / Reply

Hey Richard, thanks for you blog! It's an amazing learning material.

I want to share with everybody here about my experience of running the example code. At the beginning, I try to run the code with jsFiddle, however, the result sometimes will different. And I am confused. But after I try to just create a html file and run the code within the script tag. The result is correct.

So, I think jsFiddle will change the behavior of scope for some reasons. Richard, do you know anything about this?
This's an example about "setTimeout Variables are Executed in the Global Scope", the output of "console.log(this.constantVal * this.highValue);" is NaN

http://jsfiddle.net/fangunxs/tUHPY/

Richard Of Stanley (Author)

November 10, 2013 at 2:06 am / Reply

The reason it did not work was because you use the "this" keyword inside the settimeout function, and the settimeout function changes the scope; it is run from the global scope.

Read on post on JavaScript's this for more:
Understand JavaScript's "this" With Clarity, and Master It

Here is the fix for the NaN issue with your code:

```
1
2  var highValue = 200;
3  var constantVal = 2;
4  var myObj = {
5          highValue: 20,
```

```
6         constantVal: 5,
7         calculateIt: function () {
8         var self = this;
9   setTimeout (function  () {
10
11        console.log(self.constantVal * self.highValue);
12  }, 2000);
13        }
14  }
15
```

matt

November 8, 2013 at 9:58 pm / Reply

function showName () {
setTimeout(function(){console.log (“First Name: ” + name)},0);
var name = “Ford”;
console.log (“Last Name: ” + name);
}

Johnzilla

November 22, 2013 at 4:49 am / Reply

Hi. Great article. I found it very helpful for the most part but I do have a two lingering questions.

1) In your first variable hoisting example, you say in a comment that “The reason undefined prints first is because the local variable name was hoisted to the top of the function.”
I understand why this is clearly indicative of the variable assignment *not* being hoisted, but not how this is at all indicative that the declaration *is* being hoisted.

2) With regard to functions, In one place you say that the format “var myFunction = function () {}” is called function assignment, but a bit further

you say that this is called function expression.
I see that both of these are something different from function declaration, which is written as “function myFunction () {}” but not whether there is a difference between function assignment and function expression.

Thank you and I look forward to reading more of your articles.

Richard Of Stanley (Author)

November 22, 2013 at 3:59 pm / Reply

1. All variable and function declarations are hoisted: this is simply a fundamental concept in JavaScript.

2. Ah ha. That is a good catch.
This, “var myFunction = function () {}”, is indeed both a function expression and a function assignment. Technically, it is called a **function expression. So keep that in mind always, because you will encounter and use that kind of expression a lot.**

Now, whenever you assign a value to a variable, the execution is called a variable assignment. For example, var city = “New York”, is a variable assignment. Similarly the expression, var cityFunc = function () {}, is also a variable assignment. But because it is an assignment of a function to the variable, we call it a function assignment.

Johnzilla

November 24, 2013 at 10:28 pm / Reply

Cool. Thank you! And thanks in general for all that you’ve done with this blog here. 🙂

Sivaram

November 28, 2013 at 7:22 am / Reply

Hi Richard,

I am starting to read your tutorial..its very very nice..
its amazing when u explain everything…

Shaikhul

November 29, 2013 at 7:28 am / Reply

Function Declaration Overrides Variable Declaration When Hoisted – OK

But variable assignment overrides the function declaration – not OK.

ex.
var hoistMe = ‘hoistme';
function hoistMe () {};
typeof hoistMe; // “function”

Shaikhul

November 29, 2013 at 7:34 am / Reply

ah sorry ‘But variable assignment overrides the function declaration’ is ok while running all the statement together. but if i run each statement in chrome console one by one it doesn’t override. Any explanation for this ?

haind

December 13, 2013 at 3:38 pm / Reply

Hi Richard. I’m just read your article. What I’m confused is I test some code with html file and command line of Firebug:
console.log(typeof myName);
var myName = “Richard”;

I show me 2 different results. ‘undefined’ for html file and ‘string’ for command line. I think it ‘s kind of weird. Maybe you can explain for me?

Dydrax

December 23, 2013 at 9:29 am / Reply

what about get a value from local variable and stored in global variable

in example i use a js library canvas like easeljs
there is a lot of function there

```
function main(){
var text=["A","B","C"];
var object;i=0;
object.on("click",function(A){i++});//assume when click the i is increase
console.log(text[i]);//still A why ?
}
```

Richard Of Stanley (Author)

January 7, 2014 at 12:46 am / Reply

Hi Dydrax, I don't understand your question. Could you ask the question again with a bit more detail?

saima

February 11, 2014 at 3:44 pm / Reply

One of the best articles on the subject.. Many thanks

sagar

March 23, 2014 at 3:00 am / Reply

I am a server side programmer and ignored JavaScript for last 13-14 years. With frameworks like AngularJS and dojo , JS has got lot of recognition.
I tried my hands on these frameworks recently and was damn scared with JS syntax.

Thanks a lot bro, you saved me from turning my back forever to JS.
You deserve a big HUG.

Vinayak

April 20, 2014 at 2:34 pm / Reply

Hi Richard,
Thank you very much for the post.. you cleared many things for me with very good examples and neat explanations.

Random Dyood

June 4, 2014 at 1:54 pm / Reply

Interesting thing about for-loop iterating variable being global.

Having said that, how does one implement a for-loop without polluting one's global scope with a multitude of distinct or, a lot worse, shared/overlapping iterator variables. Is there a way to implement these loop constructs with a local-scoped variable?

Thanks in advance. Btw, great blog and JS learning material.

Enjoy!

Steve

August 7, 2014 at 6:08 am / Reply

Hi Richard,

Love the tutorial / lesson style and workable examples btw!
I just have one question wrt to " example where you were showing how variable declaration outside functions have global scope and are accessible from the window object, ..however I don't understand the " tag declaration / assignment?

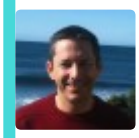
…
// or
var name; //
name;

Which you then say is accessible like so:

…
// or
console.log(“myName” in window); // true
console.log(“firstName” in window); // true

..i.e. what’s the significance of the trailing ‘pre’ tag in both declaration and accessibility examples above?

Thanks in advance,
Steve.



Steve

August 8, 2014 at 12:17 am / Reply

oops, I just noticed that all the > and < tags have been parsed in some way, ..the instances of two single quotes above ” in my post should read ‘<pre>’
(not sure if that will have worked either..)

Bipul

October 10, 2014 at 11:43 pm / Reply

Hi Richard,

Can you please the scope of “myName” in below 2 code snippets. Confused 8-(
1.
var myName=”Ram”;
var myName=function abc()
{

console.log(“Bipul”);
}

console.log(myName());

2.

var myName=”Ram”;
var myName=function abc()
{
console.log(myName);
}

myName(); // function

Igor Fedorov

November 7, 2014 at 4:39 pm / Reply

Hi,
I read about hoisting but could somebody explain why

alert(“Hello”);
var alert = 2;
alert(“World”);
does not show anything but

alert(“Hello”);
alert = 2;
alert(“World”);
shows “Hello”, ” World”

window.alert(“Hello”);
alert = 2;
window.alert(“World”);
shows “Hello”, ” World”

What is a difference between “var alert = 2;” and “alert = 2;”

Javier

December 8, 2014 at 11:21 pm / Reply

hi, i've followed your articles and they're awesome, but I'm having problems related to the last example of hoisting variables:

var myName = "Javier";

function myName() {console.log("My name is Javier");}

console.log(typeof myName); // this prints "function" i was hoping this prints "string". I've tried in Chrome's and Mozilla's webbrowser.and still getting the same result.

Siddhartha

April 14, 2015 at 1:41 am / Reply

Hey,

Nice article.

I'm not sure I understand why the 'this' in 'myObj.calculateIt()' wouldn't point to myObj and belong to the global scope?

Regards,
Siddhartha

kanhaiya

July 14, 2015 at 3:50 pm / Reply

Hi Richard,

In C or OOps –
function declaration – showName() // only call the method
function definition – function showName () { } // only define function

But in Javascript:-
function declaration :- function showName () { }

Question is why function declaration is different as per c language or oops?

Thanks,
Kanhaiya

Ravindra

July 23, 2015 at 11:31 am / Reply

I am unable to execute this code by simply copy & paste. Due to some encoding issue, I am getting below error

Uncaught SyntaxError: Unexpected token ILLEGAL

One invisible special characters has been added at the first character for every line. Even dos2unix is unable to fix the issue. Please let me know how to fix this issue

Naveen

August 18, 2015 at 2:36 am / Reply

Hi Richard,

/ The use of the “this” object inside the setTimeout function refers to the Window object, not to myObj

```
var highValue = 200;
var constantVal = 2;
var myObj = {
highValue: 20,
constantVal: 5,
calculateIt: function () {
setTimeout (function () {
console.log(this.constantVal * this.highValue);
```

```
}, 2000);
}
}

// The "this" object in the setTimeout function used the global highValue and constantVal variables, because the reference to "this" in the setTimeout function refers to the global window object, not to the myObj object as we might expect.

myObj.calculateIt(); // 400
// This is an important point to remember.
```

WHY "this" IN THE SETTIMEOUT FUNCTION REFERS TO THE GLOBAL WINDOW OBJECT.
As in your expalanation about "this" , you mentioned the calling object properties will be manipulated.
How is it different here ?????????

Deepal

October 5, 2015 at 3:14 am / Reply

I have seen many tutorials website, but the clarity with which you explain stuff is awesome. Keep up the good work.

Himanshu

December 16, 2015 at 7:00 am / Reply

Hello Richard,

Thanks for the nice & informative article(s)… 🙂

I ran this in FireFox/Chrome's console:

```
function showName () {
console.log ("First Name: " + name);
console.log ("Last Name: " + name);
}
```

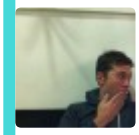
showName ();
shouldn't I have got a "Reference Error" for variable name here as I did not have a var name statement?
I saw this instead

First Name:
Last Name:
undefined

When I changed variable name to "Tname" then it showed a Reference Error but why not in the case "name"?

Please clarify.

Regards
Himanshu



Artas

January 23, 2016 at 2:37 pm / Reply

Thanks for such an informative article. I feel like I understand the variable scope much better now that I read your blog post. Looking forward to reading more of this stuff.

sasmita

January 25, 2016 at 12:16 am / Reply

Hi, i want to know .suppose within the function we don't use "var" keyword for creating a variable.It will be automaically global variable.what will happen if a local variable become a global. This question was asked me in interview. plz let clear abt this question.

Balasubramanian

February 3, 2016 at 12:16 am / Reply

```
var highValue = 200;
var constantVal = 2;
var myObj = {
highValue: 20,
constantVal: 5,
calculateIt: function () {
setTimeout (function () {
console.log(this.constantVal * this.highValue);
}, 2000);
}
}

// The “this” object in the setTimeout function used the global highValue and constantVal variables, because the reference to “this” in the setTimeout function refers to the global window object, not to the myObj object as we might expect.

myObj.calculateIt(); // 400
// This is an important point to remember
```

this is wrong the this.constantVal and this.highValue is not accesible directly inside setTimeout

Balasubramanian

February 3, 2016 at 1:06 am / Reply

Hello Richard,

Thanks for the nice article(s)… 🙂

I ran this in FireFox/Chrome’s console:

```
var highValue = 200;
var constantVal = 2;
var myObj = {
highValue: 20,
constantVal: 5,
```

```
calculateIt: function () {
setTimeout (function () {
console.log(this.constantVal * this.highValue);
}, 2000);
}
}
myObj.calculateIt();
```

I have got a “NaN” as a result.

When I stored this.constantVal and this.highValue inside calculateIt function and using those variables inside the setTimeOut function producing the results.

Please clarify.

Regards
Balasubramanian. K

krishna

February 10, 2016 at 5:23 am / Reply

for the concept of scope of variables follow the link:
http://java.meritcampus.com/core-java-topics/scope-of-variables-in-same-block

Thabo

February 23, 2016 at 4:14 am / Reply

Hi author,thank you, but i don’t understand why the variables declared in the for loop are accessible globally, i understood that JavaScript does not have the “Block scope” even if it has the braces(curly brackets), but a function also has braces but it has a block scope(you cannot access its variables outside that scope), how is a function different to the for loop? in respect to the block-scope part.

Prasanna

February 23, 2016 at 6:29 am / Reply

Hello Himanshu,
By now you might have figured out by name is not throwing exception.If not here is why,

browsers have global variable name ,which indicates the name of the current browser window.So it does not show throw error!!!.

Prasanna

February 23, 2016 at 6:30 am / Reply

Hello Himanshu,
By now you might have figured out why “name” is not throwing exception.If not here is why,

browsers have global variable name ,which indicates the name of the current browser window.So it did not throw error!!!.

Simran

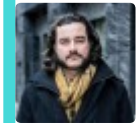
March 5, 2016 at 6:02 am / Reply

Nice article. Here’s a little more to add to javascript hoisting :).
http://www.yourtechchick.com/javascript/what-is-hoisting-in-javascript-javascript-hoisting-explained/

Himanshu

March 26, 2016 at 1:51 am / Reply

Nicely written.. 🙂



Caio Mathielo

March 21, 2016 at 12:46 pm / Reply

Hello Richard! Great article, extremely detailed and very simple to read/understand. Unfortunately some information is no longer true – i.e. “Unlike most programming languages, JavaScript does not have block-level scope (variables scoped to surrounding curly brackets);”.

The “let” statement (https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Statements/let) was implemented in JavaScript 1.7 and it specifically allows the declaration of block-level variables. Would be really nice to see the article updated as it looks like many people end up here when googling for JS scope!

Cheers!

Himanshu

March 26, 2016 at 1:52 am / Reply

A good point out..thanks.. 🙂

Eazie

April 28, 2016 at 1:16 am / Reply

Love your article, simple but detailed, thumbs up.

Devdutta Natu

October 12, 2016 at 3:20 am / Reply

Hi Richard,

I tried to hoist the very last thing where is mentioned that :-

It is important to note that function expressions, such as the example below, are not hoisted.

```
var myName;
var myName = function () {
console.log ("Rich");
}
```

but still by using the typeof operator i'm getting the same result as function itself.

Ganesan Natarajan

October 14, 2016 at 4:50 pm / Reply

sorry .. wrong comment on the post. i meant to post it here : http://javascriptissexy.com/understand-javascript-closures-with-ease/#comment-308368

Trackbacks for this post

1. 16 JavaScript Concepts You Must Know Well | JavaScript is Sexy
2. JavaScript Closures in Lovely Detail | JavaScript is Sexy
3. Learn Node.js Completely and with Confidence | JavaScript is Sexy
4. Learn Backbone.js Completely | JavaScript is Sexy
5. 12 Simple (Yet Powerful) JavaScript Tips | JavaScript is Sexy
6. Simple Yet Powerful JavaScript Tips | Hackya.com
7. Frontend Interview Practice | Hi, I'm Lillian.
8. sexy.com | JavaScript is Sexy
9. Best Practices for JavaScript Development (Parte I) - MediaNet Software
10. JavaScript Closures | Pradeep Reddy Kamatham
11. JavaScript Variable Scope and Hoisting Explaine...
12. Week 4: Oracle of Bacon and other stuff | jimmylocoding
13. The Odin Project Progress Map | Tilly Codes
14. JavaScript variable scope, variable hoisting | J.-H. Kwon
15. Encapsulation in Javascript - Aneesh Dogra's Blog
16. Game Development, JavaScript, and Persistence | Glimmerville
17. Javascript – variable scope | Majesty
18. Day 13: Meetup.com Sydney JavaScript Study Group – REJECTED | gourmetjavascript

19. How Not To Boil The Ocean-HTML/CSS and Javascript | Coding Over 40
20. Variables Hoisting in Javascript – Jagdeep Singh Bisht
21. The Little JavaScript Closures-IT大道
22. Day 17-28 Jan 6-7 | Mandy's road to code
23. html - JavaScript scope conflicts - CSS PHP
24. JavaScript Scope and Scope Chaining – Life as a Programmer

Leave a Reply

Name | Email (hidden) | Website

Comment:

Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

 JavaScript is Sexy About Contact Archive