**We can't change history, but we can change the future.
Be nice to each other. [@robertnyman](#)**

Explaining JavaScript scope and closures

Published on Thursday, October 9, 2008
This article is also available in Turkish, Russian and Traditional Chinese.

I thought I'd try to explain something which many people struggle with: JavaScript scope and closures.

Background

There are a number of articles and blog posts out there trying to explain scope and closures, but overall I'd say that a majority of them aren't crystal-clear. Besides, a number of them take for granted that everyone has developed in about 15 other languages before, while my experience is that a lot of people writing JavaScript come from a HTML and CSS background, instead of C and Java.

Therefore, my humble goal with this article is for everyone to finally grasp what scope and closures are, how they works, and especially how you can benefit from them. You do need to understand the basic concepts of variables and functions before reading this.

Scope

Scope refers to where variables and functions are accessible, and in what context it is being executed. Basically, a variable or function can be defined in a global or local scope. Variables have so-called function scope, and functions have the same scope as variables.

Global scope

When something is global means that it is accessible from anywhere in your code. Take this for example:

view sourceprint?
```
var monkey = "Gorilla";

function greetVisitor () {
        return alert("Hello dear blog reader!");
}
```

If that code was being run in a web browser, the function scope would be window, thus making it available to everything running in that web browser window.

Local scope

As opposed to the global scope, the local scope is when something is just defined and accessible in a certain part of the code, like a function. For instance;

view sourceprint?
```
function talkDirty () {
        var saying = "Oh, you little VB lover, you";
        return alert(saying);
}
alert(saying); // Throws an error
```

If you take a look at the code above, the variable saying is only available within the talkDirty function. Outside of it it isn't defined at all. Note of caution: if you were to declare saying without the var keyword preceding it, it would automatically become a global variable.

What this also means is that if you have nested functions, the inner function will have access to the containing functions variables and functions:

view sourceprint?
```
function saveName (firstName) {
        function capitalizeName () {
                return firstName.toUpperCase();
        }
        var capitalized = capitalizeName();
        return capitalized;
}
alert(saveName("Robert")); // Returns "ROBERT"
```

As you just saw, the inner function capitalizeName didn't need any parameter sent in, but had complete access to the parameter firstName in the outer saveName function. For clarity, let's take another example:

view sourceprint?
```
function siblings () {
        var siblings = ["John", "Liza", "Peter"];
        function siblingCount () {
                var siblingsLength = siblings.length;
                return siblingsLength;
        }
        function joinSiblingNames () {
                return "I have " + siblingCount() + " siblings:\n\n" + siblings.join("\n");
        }
        return joinSiblingNames();
}
alert(siblings()); // Outputs "I have 3 siblings: John Liza Peter"
```

As you just saw, both inner functions have access to the siblings array in the containing function, and each inner function have access to the other inner functions on the same level (in this case, joinSiblingNames can access siblingCount). However, the variable siblingsLength in the siblingCount is only available within that function, i.e. that scope.

Closures

Now when you hopefully have gotten a better grasp of what scope is, let's add closures to the mix. Closures are expressions, usually functions, which can work with variables set within a certain context. Or, to try and make it easier, inner functions referring to local variables of its outer function create closures. For instance:

view sourceprint?
```
function add (x) {
        return function (y) {
                return x + y;
        };
}
var add5 = add(5);
var no8 = add5(3);
alert(no8); // Returns 8
```

Whoa, *whoa*! What just happened? Let's break it down:

1. When the add function is called, it returns a function.
2. That function closes the context and remembers what the parameter x was at exactly that time (i.e. 5 in the code above)
3. When the result of calling the add function is assigned to the variable add5, it will always know what x was when it was initially created.
4. The add5 variable above refers to a function which will *always* add the value 5 to what is being sent in.
5. That means when add5 is called with a value of 3, it will add 5 together with 3, and return 8.

So, in the world of JavaScript, the add5 function actually looks like this in reality:

view sourceprint?

```
function add5 (y) {
        return 5 + y;
}
```

The infamous loop problem

How many times have you created some sort of loop where you wanted to assign the value of i in some way, e.g. to an element, and found out it just returned the last value i had?

Incorrect reference

Let's take a look at this faulty code, which creates 5 a elements, adds the value of i as a text to each element and an onclick which is expected to alert the value of i for that link, i.e. the same value as in the a element's text. It then appends them to your document body:

view sourceprint?

```
function addLinks () {
        for (var i=0, link; i<5; i++) {
                link = document.createElement("a");
                link.innerHTML = "Link " + i;
                link.onclick = function () {
                        alert(i);
                };
                document.body.appendChild(link);
        }
}
window.onload = addLinks;
```

Each a element gets the correct text, i.e. "Link 0", "Link 1" and so on. But whichever link you click, it alerts the number "5". Oh my God, *why*? The reason for this is that the variable i get its value increased with 1 for each iteration of the loop, and since the onclick event isn't being executed, just applied to the a element, it adds up.

Therefore, the loop continues until i is 5, which is the last value of i before the function addLinks exits. Then, whenever the onclick event is actually being triggered, it takes the last value of i.

Working reference

What you want to do instead is create a closure, so that when you apply the value of i to the onclick event of the a element, it gets the exact value of i at just that moment in time. Like this:

view sourceprint?

```
function addLinks () {
        for (var i=0, link; i<5; i++) {
                link = document.createElement("a");
                link.innerHTML = "Link " + i;
                link.onclick = function (num) {
                        return function () {
                                alert(num);
                        };
                }(i);
                document.body.appendChild(link);
        }
}
window.onload = addLinks;
```

With this code, if you click the first a element it will alert “0”, the second “1” etc; just what you probably expected with the first code I showed you to do. The solution here is that the inner function of what’s applied to the onclick event create a closure where it references the parameter num, i.e. what the i variable is at just that time.

That function then closes with that value safely tucked away, and can then return its corresponding number when the onclick event is being called.

Self-invoking functions

Self-invoking functions are functions who execute immediately, and create their own closure. Take a look at this:

view sourceprint?

```
(function () {
        var dog = "German Shepherd";
        alert(dog);
})();
alert(dog); // Returns undefined
```

Ok, so the dog variable was only available within that context. Big deal, man, hidden dogs… But, my friends, this is where it becomes really interesting! It solved our problem with the loop above, and it is also the base for the Yahoo JavaScript Module Pattern.

Yahoo JavaScript Module Pattern

The gist of the pattern is that it uses a self-invoking function to create a closure, hence making it possible to have private and public properties and methods. A simple example:

view sourceprint?

```
var person = function () {
        // Private
        var name = "Robert";
        return {
                getName : function () {
                        return name;
                },
                setName : function (newName) {
                        name = newName;
                }
        };
}();
alert(person.name); // Undefined
alert(person.getName()); // "Robert"
person.setName("Robert Nyman");
alert(person.getName()); // "Robert Nyman"
```

The beauty of this is that you now can decide on your own what will be publicly visible for your object (and can be overwritten), and what is private and no one can access nor alter. The variable name above is hidden outside the context of the function, but accessible from the returned getName respectively setName functions, since they create closures where they have a reference to the name variable.

Conclusion

My sincere hope is that after reading this, novice or experienced programmer, you have gotten a clear view of how scope and closures actually work in JavaScript. Questions and feedback are very welcome, and if any input is deemed important enough, I will update this article with it.

Happy coding!

Posted in Developing, JavaScript, Technology

226 Comments

- RSS feed for comments on this post

- RSS feed for comments on all posts

- *Norbert* says:
  October 9, 2008 at 22:40

  Thanks for explaining JS closures! It's very helpful.

  I also decided to give another try for the LISP book I was reading.

  Reply
- *Steven Clark* says:
  October 10, 2008 at 3:42

  Mmm I'm going to have to bookmark this fundamental JS stuff – you're training me by proxy I'm sure.

  I've definately got a far better handle on closures now. Thanks. ☺

  Reply
- *Harmen Janssen* says:
  October 10, 2008 at 11:46

  Nice read, Robert ☺

  I must say I already was familiar with most concepts you describe, but it's good to have them explained again all in one place for quick reference. Especially the infamous loop problem sometimes still bites me in the ass 😀

  Reply
- *HÃ¥kan* says:
  October 10, 2008 at 11:48

  Great post and good summary of important concepts!

  Thanks!

  Reply
- *Andrea Giammarchi* says:
  October 10, 2008 at 13:22

  Hi Robert,

  I would suggest a better addLinks example, since there is no reason to create two functions, one for the closure, and another for the current link onclick event.

  ```
  function addLinks () {

  for (var onclick = function(num){return function(){alert(num)}}, i=0, link; i<5; i++) {

  link = document.createElement("a");

  link.innerHTML = "Link " + i;

  link.onclick = onclick(i);

  document.body.appendChild(link);

  }

  };

  onload = addLinks;
  ```

  This is more about performances (maniacs) but it is good to understand closure portability, isn't it? ☺

  Reply
- *Robert Nyman* says:

October 10, 2008 at 13:47

Thanks guys, I'm glad that you liked it!

Andrea,

Really nice! ☺

I choose my approach partly for easier readability and understanding for everyone, but also honestly because I didn't think as far as you did.

I think it's a perfect example of making closures portable! Thanks!

Reply

- *Andreas Rydberg* says:
October 10, 2008 at 15:48

Thanks Robert for a great article! I really need to read up on closures.

Reply

- *Chris* says:
October 10, 2008 at 17:25

Thank you for that article. Made some things clear. But I have one question:

In your addLinks-example, why does link.onclick know that num gets the

value of i and not something else?

It's not the first time I see this and don't understand ?-)

Reply

- *Robert Nyman* says:
October 10, 2008 at 17:58

Andreas,

Thanks!

Chris,

Thank you!

The key is that the function is self-invoked, i.e. gets executed right away, and sends in the value <code>i</code> to be the value of <code>num</code>.

If you examine that part of the code more closely:

<code>link.onclick = function (num) {

return function () {

alert(num);

};

}(i);</code>

you can see that it is not just assigned, but actually gets called immediately by having two parentheses at the end, with <code>i</code> in them.

And since inner functions will always have access to their outer/containing functions' variables (i.e. a closure is created), it will always know that <code>num</code> is the value of <code>i</code> when it was initially called.

I hope it's clearer now! ☺

Reply

- *Chris* says:

October 10, 2008 at 18:11

It does clarify indeed! What I oversaw was just the parantheses in the end:

<code>}**(i)**;</code>

Thanks for explaining and for writing so much good stuff!

Reply

- *Robert Nyman* says:
  October 10, 2008 at 19:30

  Chris,

  Thanks! 🙂

  I'm glad that you understood it!

  Reply
- *Jeff L* says:
  October 10, 2008 at 21:43

  Thanks for a great article Robert. I also missed the (i) at the end of the onclick function and was wondering how num = i — makes sense now that you pointed it out, though.

  I also appreciate you using an example that most of us have problem run into (where i was not what we expected, but instead was the last known value). I think that makes a big difference in helping folks to understand.

  Reply
- *Robert Nyman* says:
  October 10, 2008 at 21:59

  Jeff,

  Thank you!

  Glad that it helped you as well!

  Reply
- *Weekly Links #22 | GrantPalin.com* says:
  October 12, 2008 at 21:38

  […] Explaining JavaScript scope and closures JavaScript handles scope somewhat differently from other common programming and scripting languages, and it also has an interesting capability for creating and using closures. Robert Nyman explains the concepts clearly. […]

  Reply
- *Bleyder* says:
  October 13, 2008 at 15:16

  Thanks for this article.

  I'm begining to undertand a little more how works JavaScript.

  Reply
- *Andrew Noyes* says:
  October 13, 2008 at 17:17

  Great article! This would have saved my ass on a project just a couple of months ago. The closure bit has caused numerous problems for me and most of the solutions I found on the internet were either half-assed or explained nothing, which left me with a viable solution but defenseless for the next time I ran into that problem. Thanks!

  Ironically, there are many C or Java programmers that are well educated but use poor form. My C++ professor in college taught us all kinds of bad practices, like using global variables and <code>system("PAUSE");</code> instead of <code>cin.ignore();</code> or some other less intrusive object. JavaScript is so much easier to work with if you coming from a strong C++ background where you are mindful of C++'s built-in precautions against scope problems that are essential to OOP, because JavaScript lacks any such thing, being a loosely typed language. It's easy to fall into sinkholes when beginning with JavaScript.

Reply

- *Robert Nyman* says:
  October 13, 2008 at 22:39

  Bleyder,

  Great!

  Andrew,

  Thank you! And absolutely, I can relate how developers coming from C++ or similar can have a hard time with some concepts, so I hope this is helpful for those as well!

  Reply
- *JavaScript: how to get private, privileged, public and static members (properties and methods) - Robert's talk - Web development and Internet trends* says:
  October 14, 2008 at 13:29

  […] reading JavaScript inheritance – how and why and Explaining JavaScript scope and closures, I thought we'd combine the knowledge gained to talk about private, privileged, public and […]

  Reply
- *JavaScript inheritance - experimenting with syntax alternatives and private variables - Robert's talk - Web development and Internet trends* says:
  October 22, 2008 at 16:17

  […] reading this article, I strongly recommend reading JavaScript inheritance – how and why and Explaining JavaScript scope and closures first, since many phenomenon below will have their explanation […]

  Reply
- *Internet Marketing* says:
  December 4, 2008 at 19:09

  Cool. I am working on a projec requiring clear mind of how JS module works. It helps a lot.

  Thanks

  Reply
- *JavaScript timers - using functions and scope - Robert's talk - Web development and Internet trends* says:
  December 16, 2008 at 13:07

  […] Explaining JavaScript scope and closures Posted in Developing, JavaScript, Technology | Share your thoughts […]

  Reply
- *Jeff* says:
  December 25, 2008 at 7:59

  Thanks for this article, i'm working on a javascript project full of scope issues and closures, very usefull page to keep mind clear 😉

  Reply
- *Robert Nyman* says:
  December 26, 2008 at 1:33

  Internet Marketing, Jeff,

  Glad that it helped!

  Reply
- *JavaScript timers - using functions and scope | How2Pc* says:
  December 27, 2008 at 21:44

  […] Explaining JavaScript scope and closures […]

Reply

- *Ryan Morr - Javascript, CSS, and Web Apps » Scope/Context in Javascript* says:
  February 16, 2009 at 4:11

  […] http://www.robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ […]

  Reply
- *Kevin Law* says:
  February 20, 2009 at 17:33

  Thank you so much!

  I don't understand the concept of closure in Javascript until I read your article.

  Very useful! ☺

  Reply
- *Robert Nyman* says:
  February 20, 2009 at 20:05

  Kevin,

  Great to hear that!

  Reply
- *Psychic Origami » Blog Archive » A Huffduffer Widget* says:
  March 22, 2009 at 18:37

  […] Then we use that id as a name for the function and create a callback that embeds the id of the div the widget will appear in via a closure: […]

  Reply
- *John DeHope* says:
  May 8, 2009 at 16:59

  Thanks for a great resource. Your explanation of what is going on in *the infamous loop problem* was exactly what I needed.

  Reply
- *Robert Nyman* says:
  May 8, 2009 at 18:02

  John,

  You're welcome, I'm glad it was helpful!

  Reply
- *Chanon's Blog » Blog Archive » Libraries and frameworks for my new Web 2.0 project: jQuery* says:
  May 9, 2009 at 7:41

  […] Object-Oriented Programming Part 1 and Part 2 (a little dated, but explains the basics nicely) Explaining JavaScript scope and closures jQuery for JavaScript programmers Improve your jQuery – 25 excellent tips 5 Tips for Better jQuery […]

  Reply
- *slier* says:
  May 23, 2009 at 7:32

  Sigh finally i grasp the concept behind the closure …

  Big thanks…

  Reply
- *Robert Nyman* says:
  May 23, 2009 at 23:56

  slier,

I'm very glad to hear that. 🙂

Reply

- *Skateside* says:
  June 19, 2009 at 1:36

  I was wondering why my module wasn't returning the array — thank you for showing me what i was doing wrong 🙂

  Reply
- *Robert Nyman* says:
  June 19, 2009 at 1:40

  Skateside,

  Glad it was of help!

  Reply
- *Steve tran* says:
  July 13, 2009 at 6:14

  the best explanation of closures around. I've been looking at many articles and this one is just perfect for me. Thanks alot Robert for writing it up

  Reply
- *Anirudh Vyas* says:
  August 14, 2009 at 19:11

  My Only complaint is : Why don't you have a printer-friendly page of this thing! 🙂

  Good stuff!

  Regards

  Vyas,Anirudh

  Reply
- *Robert Nyman* says:
  August 17, 2009 at 2:14

  Steve tran,

  Thanks, I'm glad you liked it!

  Anirudh Vyas,

  Well, it could be printed, but I agree – the print-out could definitely look better!

  Reply
- *Shawn Yuan* says:
  August 18, 2009 at 17:08

  Nice article. This solves my problem 😀

  Thanks, Robert.

  Reply
- *Robert Nyman* says:
  August 18, 2009 at 22:23

  Shawn,

  Glad it was of use to you! 🙂

  Reply

- *Vsync* says:
  August 24, 2009 at 21:56

  Just Awesome!

  I love you man.

  Reading your posts for a long time. excellent stuff!

  Reply
- *Robert Nyman* says:
  August 24, 2009 at 22:42

  Vsync,

  Glad you like it!

  Reply
- *Leo* says:
  August 25, 2009 at 21:16

  Thank you very much for the post, Robert!

  As you can see, it's still a very valuable read even after some months. And being such as it is, I believe it will stay for a long time, for people looking into Javascript a bit deeper than just the newbie stuff…

  Your explanation of closures finally cleared the concept for me – I read a couple of them before, but still wasn't entirely clear about what's going on – now I can use them wisely ☺

  Thanks again and keep up the great blog!

  Leo

  Reply
- *Robert Nyman* says:
  August 25, 2009 at 22:19

  Leo,

  Thank you very much; it sincerely makes me happy to hear it!.

  Glad that I made you understand! ☺

  Reply
- *Alan* says:
  August 30, 2009 at 7:01

  Hi Robert

  What a great article!!

  I'm a newbie coming from a HTML/CCS background and this was the best and clearest explanation of these topics that I've found on the web

  In the "Infamous Loop" example I have a question:

  'the Incorrect reference states "the onclick event is not being executed", which I understand, whilst in the Working reference the onclick event appears to be at least evaluated, because the "value is tucked away" '

  I would appreciate it if you would point out what am I missing

  Regards

  AlanB

  Reply

- *Robert Nyman* says:
  August 30, 2009 at 22:11

  Alan,

  Thank you very much!

  In regards to the loop, it's hard to catch the first time, but in the working example the <code>onclick</code> is assigned a function that is run *immediately*. If you check, you can see the two parentheses after the function that sends in the value of i, *just* as it is during that iteration in the loop.

  That means that the current value of is, indeed tucked away, and later referenced when the element is being clicked.

  Hope that made it a little clearer!

  Reply
- *Alan* says:
  August 30, 2009 at 23:59

  Robert

  Thank you for your quick response, I now get it!!

  I could see where the (i) variable came from but had overlooked the fact that this was a self-invoking function

  Thanks again

  Alan

  Reply
- *Robert Nyman* says:
  August 31, 2009 at 0:02

  Alan,

  No problem, glad you understood what I meant!

  Reply
- *6 Advanced JavaScript Techniques You Should Know* says:
  October 8, 2009 at 17:59

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *6 Advanced JavaScript Techniques You Should Know | Programming Blog* says:
  October 8, 2009 at 23:31

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *Julie* says:
  October 15, 2009 at 17:42

  Hi Robert,

  Many thanks for such a succinct article! I'm rather new to all this and it took a bit of time to get my head around self-invoking functions.

  If anyone is still struggling, correct me if I'm wrong but for your 'infamous loop' problem, if for the sake of argument you call the addLinks function declaration (all the loop and stuff) 'splurge', then am I right in thinking that the function call is done immediately by the declaration 'splurge'(i); ?

  I'm curious about several other points as well:

  Can you have a self-invoking function without a closure?

  If so, would there be any benefit to doing this?

If you use closures, what about memory leeks?

Reply

- *Robert Nyman* says:
  October 16, 2009 at 12:26

  Julie,

  Glad you liked it!

  About <code>addLinks</code>: it is rather that within each iteration of the loop, an inner function is called immediately to get a reference to the value of <code>i</code> at that specific time, and then assign that value to the <code>onclick</code> event of the link.

  > Can you have a self-invoking function without a closure?

  > If so, would there be any benefit to doing this?

  Well, a self-invoking rather creates a closure immediately. Then it is up to you if you want to take advantage of what's offered with that or not.

  > If you use closures, what about memory leeks?

  I believe older versions of Internet Explorer had some memory issues with this, but that they should be sorted out by long. Or at least so I hope! 🙂

  Reply

- *Julie* says:
  October 16, 2009 at 17:42

  Hi Robert,

  Many thanks for the quick reply!

  I've been playing around with self-invoking functions since your article, and came up with a simple way to "encapsulate" onload, thought you might like to see it 🙂

  (function()

  {

  window.onload=init;

  })();

  This is just like your "dog" function above. But would it technically be a closure though?

  Looks like closures give memory leaks to IE before version 8 (good ref is http://msdn.microsoft.com/en-us/library/dd361842(…), so they would still effect a substantial number of users right now. But are all closures bad??

  Reply

- *Robert Nyman* says:
  October 16, 2009 at 20:59

  Julie,

  Nice! However, you are overwriting something on a global object, so it will not allow you to have multiple <code>window.onload</code> – just thought I'd mention that. 🙂

  And sure, it's a closure. Whatever you put in there, i.e. that doesn't explicitly overwrite something that exists outside of it, is only valid and reachable from within that code. If you want to return something to the outside, though, you can use a <code>return</code> statement in that anonymous function, which will be returned globally.

  Also: no closures are bad; IE is terrible. 🙂

  But, to my knowledge, Microsoft released some separate script fixes as well. More info can be found in IE Memory Leaks Be-gone and IE's Memory Leak Fix Greatly Exaggerated.

However, really, don't worry about this. All major JavaScript libraries, such as jQuery and others, completely rely on closures as well. So, closures is a de-facto approach to efficient JavaScript programming.

Reply

- *Julie* says:
  October 16, 2009 at 23:35

  Hey Robert,

  Thanks for all the tips and and your full explanation about closures. I really get it now! I'll definitely check out the script fixes. Closures certaiinly look the way to go, especially as you explained above they're very handy for OOP and encapsulation.

  On an aside about onload, good job you pointed that out. I must admit I tend just to pile everything into init and have one onload, not the best way if you have a team working on a project I suppose, but never have any problem with it. I suppose one day I'll have to catch up with the modern approach… 😉

  Reply
- *6 Advanced JavaScript Techniques You Should Know | My Blog* says:
  October 17, 2009 at 8:36

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *Robert Nyman* says:
  October 18, 2009 at 14:16

  Julie,

  Great! And don't worry about <code>onload</code>: one smal step at a time. 🙂

  Reply
- *karalamalar » Blog Ar?ivi » JavaScript kapsam ve kaplamlar? anlamak* says:
  November 19, 2009 at 14:49

  […] kapsam ve kaplamlar? anlamak Bu makalenin orijinali (Explaining JavaScript scope and closures) Robert Nyman taraf?ndan kendi blogunda yaz?lm??t?r. Türkçe çevirisi için kendisinden izin […]

  Reply
- *Mikay* says:
  November 30, 2009 at 20:16

  Thanks,Robert.Great article.You help me completely understand the Closure,and this is the first article I read about Self-invoking functions,

  though I have learned it from the source code of Prototype.

  Reply
- *Robert Nyman* says:
  November 30, 2009 at 21:21

  Mikay,

  Glad that you liked it! 🙂

  Reply
- *6 excellent Advanced JavaScript Techniques « EWS* says:
  January 19, 2010 at 2:05

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *John* says:
  January 23, 2010 at 0:27

  I know this post is a year old, but the section on the loop problem really helped me today. Thanks!

Reply

- *Robert Nyman* says:
  January 25, 2010 at 11:23

  John,

  Great to hear!

  Reply
- *JavaScript: Tools, Coding Standard and Guidelines » select ** says:
  February 10, 2010 at 7:18

  […] Closures : Working with closures and Explaining JavaScript scope and closures. […]

  Reply
- *Jonathan Nieto* says:
  March 17, 2010 at 3:42

  Hey! this post was really helpful!!!

  Thx 😁

  Reply
- *Robert Nyman* says:
  March 17, 2010 at 12:54

  Jonathan,

  Glad you liked it!

  Reply
- *6 Advanced JavaScript Techniques You Should Know « HUE Designer* says:
  March 17, 2010 at 13:50

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *Robert Nyman* says:
  March 22, 2010 at 20:11

  Maria,

  Thanks!

  The reason that happens is because on of the trickier aspects of JavaScript. It is best described by Nicholas Zakas, and well worth a read.

  Reply
- *Maria* says:
  March 22, 2010 at 21:39

  Thank you , this is a clear and concise tutorial.

  I tested and modified the code that you are using in your examples.

  Here:

  function talkDirty () {

  saying = "Oh, you little VB lover, you";

  return alert(saying);

  }

  alert(saying);

as you can see i removed var from the "saying" variable declaration. Now saying becomes global variable. However, when i run the code i keep getting error: that saying is not defined.

I don't get it . It is a global variable now….

Reply

- *David* says:
  April 25, 2010 at 20:47

  This is one of the most helpful articles I have read yet on closures. Thanks!

  Reply

- *Robert Nyman* says:
  April 26, 2010 at 11:53

  David,

  Thank you!

  Reply

- *jokka's me2DAY* says:
  May 2, 2010 at 21:07

  ????? ??…

  Javascript ???…

  Reply

- *How to create an iPad menu with animation « Appcelerator Developer Center* says:
  June 12, 2010 at 20:15

  […] the loop, we need to add a callback that will do the menu selection. We wrap the handler logic in a Javascript closure so we can access the loop variables […]

  Reply

- *Alex C* says:
  July 13, 2010 at 5:16

  Thanks. Great tutorial. I had some code recently that had the closure issue and didn't want to work; didn't understand why until now. Thanks.

  Reply

- *Indie* says:
  August 7, 2010 at 17:33

  Great explanation, and lucid examples. Thank you!

  Reply

- *Jack* says:
  August 11, 2010 at 17:13

  Robert,

  Thanks for writing this up – two years later its still helping people (like me!)

  – Jack

  Reply

- *Tuan Le* says:
  August 13, 2010 at 18:03

  Greate article. It helps me very much to open my knowledge about javascript programming.

  Thanks.

  Reply

- *Robert Nyman* says:
  August 16, 2010 at 0:36

  Alex C, Indie, Jack, Tuan Le,

  Thank you all, glad it was of use to you!

  Reply
- *jaj* says:
  August 22, 2010 at 19:51

  Nice, thanks

  Reply
- *Matthew02* says:
  September 26, 2010 at 10:25

  Nice to see that you are still responding after 2 years. I just wanted to say thanks. I have been looking through the voodoo that is jQuery and this article really helped me understand scope and closures. Maybe you can help with this also…

  <code>

  ;(function(scope){

  var Benchmark = scope.Benchmark = function(tests){

  // store the argument { name: test }

  this.tests = tests;

  </code>

  The leading ; is evading my understanding. Also, I do not really understand the second line at all.

  Thanks for the great article.

  Reply
- *Robert Nyman* says:
  September 26, 2010 at 16:36

  jaj,

  You're welcome!

  Matthew02,

  Not sure about the first semi-colon: it can be about context, and that it's there depending on what's before, or for JavaScript magnification or similar.

  The second line is just about assigning several variables/values at the same time.

  Reply
- *Re : Registering events within an object - Jquery Home* says:
  October 4, 2010 at 6:48

  […] Then after that you should probably do some reading about scope and closures: http://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ Basically you are losing the "scope" of "this" inside your […]

  Reply
- *javascript: closure and other stuffs… « MushfiqRazib's Weblog* says:
  December 27, 2010 at 13:32

  […] from: robertnyman,Douglas […]

  Reply
- *JavaScript Closures « CHEN YU* says:
  January 12, 2011 at 18:15

[…] http://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ […]

Reply

- *Por qué los programadores Java odian Javascript | EtnasSoft* says:
  January 23, 2011 at 14:51

  […] Las variables son globales dentro de su contexto (scope). […]

  Reply

- *Headcircus* says:
  January 24, 2011 at 21:15

  Alternate way to escape the loop iterator problem with retrieve/store using mooTools, or with POJ below:

```
function addLinks () { //moo
for (var i=0, link; i<5; i++) {
link = document.createElement("a");
link.store("number",i);
link.innerHTML = "Link " + i;
link.onclick = function () {
alert(this.retrieve("number");
};
document.body.appendChild(link);
}
}
```

  ———————-

```
function addLinks () { // POJ as assoc array
for (var i=0, link; i<5; i++) {
link = document.createElement("a");
link["number"] = i;
link.innerHTML = "Link " + i;
link.onclick = function () {
alert(this.number); // or this["number"]
};
document.body.appendChild(link);
}
}

window.onload = addLinks;
```

  Reply

- *Headcircus* says:
  January 24, 2011 at 21:18

  ^^ Sorry, I know the article is about closures, but if I know anything about beginning JS developers, if they see your solution to the problem, they will always use that solution to the problem – even if the article is context of one issue: closures & scope. Its certainly what i did many, many years ago, but no longer do. Meaning, it may not be the best solution to the problem – but only 1 in context of the article.

  Reply

- *Headcircus* says:
  January 24, 2011 at 21:27

  Also, be careful if you reference a DOM element in the closure and don't nullify and then delete the reference. Because of IE garbage collection, you can create a circular reference and pull your hair out trying to find out why there is a memory leak. In an iteration, if you have a dom element that you want to reference later in the extended scope of a closure, be sure test for the object, then at the end of the iteration, nullify and then delete the local reference.

  Better yet, instead of adding an anonymous function, add a real function as an event instead, and you won't have to worry about circular references being created as you add events to DOM elements or even better: add a closure to protect against a circular reference, inside the closure… i know its odd but thats IE.

  This may be familiar
  https://developer.mozilla.org/en/a_re-introduction_to_javascript

  Reply

- *Robert Nyman* says:
  January 25, 2011 at 9:59

  Headcircus,

  Thanks for your comments.

  First of all, I think it's vital for developers to understand core JavaScript and not just depend on JavaScript libraries. When they understand the workings of JavaScript, absolutely, but some basic understanding is really desired. Therefore, it is anomy about core JavaScript here.

  And sure, there coupled be examples where you call an another, function-external function as well or declare the function in the loop initiation.

  Regarding circular references, if I'm not mistaken, is that not something that only applies to IE6 and has been fixed since?

  Reply
- *Balázs* says:
  February 11, 2011 at 22:24

  Great article, pal. Solved my problem immidiately.

  Reply
- *Emil* says:
  February 21, 2011 at 9:57

  Thanks for the lesson Robert, you've presented and explained this really well.

  The Yahoo js pattern is excellent and I'll definitely be using that a lot!

  Cheers!

  Reply
- *Robert Nyman* says:
  February 21, 2011 at 10:16

  Balázs,

  Good to hear!

  Emil,

  Thanks, glad you like it!

  Reply
- *Tom* says:
  February 26, 2011 at 21:43

  Closures are fine, I learned them from Danny Goodman's books, but they're difficult like certain Mathematics concepts. For instance, learning about logarithms, or sorting algorithms is easy, but you can find yourself in an unusual situation where the algorithms you need to use are different from what you may have practiced.

  Do you know of any resources more comprehensively studying the concept of closures in JavaScript?

  Nice article, btw. ☺

  Reply
- *Robert Nyman* says:
  February 27, 2011 at 20:41

  Tom,

  Thanks!
  Right now I can't think of anyone that delves much deeper into it – perhaps JavaScript – The Good Parts by Douglas Crockford?

Reply

- *Top 6 JavaScript which every one should know | Free download softwares, music, wordpress template, joomla template, Jquery tools and tips, Beauty tips, Free online video..* says:
  April 12, 2011 at 15:30

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *coder* says:
  April 28, 2011 at 16:23

  Hi! Thanks for the great explanation about javascript scope and closures! I also translated it into russian, which is available here: coders.ask-ru.net/question.aspx?qid2=292

  Reply
- *Robert Nyman* says:
  April 28, 2011 at 16:38

  coder,

  Nice, thanks!

  Reply
- *Daniel S.* says:
  May 29, 2011 at 19:38

  What a wonderful read, thanks for writing it!

  Reply
- *Robert Nyman* says:
  May 30, 2011 at 9:12

  Daniel,

  Glad you liked it!

  Reply
- *sridharjay* says:
  June 20, 2011 at 9:27

  A very nice post for folks like me who know javascript to some extent, and willing to explore more. I'm very sure tht lots of ppl like me wud have gained a world of knowledge on closures.

  Reply
- *Robert Nyman* says:
  June 25, 2011 at 23:46

  sridharjay,

  Thanks, glad you liked it!

  Reply
- *theartfuldodger* says:
  June 29, 2011 at 4:43

  you bloody clever bastard you! nice tut. verrrrry nice.

  Reply
- *Robert Nyman* says:
  June 29, 2011 at 15:18

  theartfuldodger,

Thanks! ☺

Reply

- *Advanced JavaScript Techniques You Should Know - Kappa Techie's Tips and Tricks* says:
  July 26, 2011 at 9:14

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *Nenad* says:
  August 17, 2011 at 4:09

  Is yahoo js code the same as:

```
var ClassPerson = function () {
// Private
var name = "Robert";
this.getName = function () {return name; },
this.setName = function (newName) {name = newName;}
}
}
var person = new ClassPerson();
alert(person.name); // Undefined
alert(person.getName()); // "Robert"
person.setName("Robert Nyman");
alert(person.getName()); // "Robert Nyman"
```

  Reply
- *Robert Nyman* says:
  August 17, 2011 at 11:40

  Nenad,

  You will get the same output, yes. The difference is that you need to instantiate and can't use the power of prototypes, whereas the Yahoo Module Pattern could be completely anonymous and needs no instantiation (but rather just execution by the parentheses at the end).

  Reply
- *Buzz Lightyear is here » Blog Archive » links for 2011-09-10* says:
  September 11, 2011 at 1:03

  […] Explaining JavaScript scope and closures – Robert's talk (tags: programming javascript web development tutorial) […]

  Reply
- *Closures: Links, News and Resources (1) « Angel "Java" Lopez on Blog* says:
  September 21, 2011 at 12:36

  […] Explaining JavaScript scope and closures – Robert's talk http://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ […]

  Reply
- *Por qué los programadores Java odian Javascript | ciberdix* says:
  October 5, 2011 at 16:44

  […] Las variables son globales dentro de su contexto (scope). […]

  Reply
- *[js] read The Infamous Loop Problem to know more about always get the last element when event trigger | code@butterflybone* says:
  October 21, 2011 at 6:06

  […] http://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ This entry was posted on Friday, October 21st, 2011 at 12:07 pmand is filed under web. You can follow any responses to this entry through the RSS 2.0 feed. You can leave a response, or trackback from your own site. […]

  Reply
- *Pawankumar Jajara* says:
  November 2, 2011 at 22:47

The best article on JavaScript Closures, I ever found on Web. I would say its very very simple and clear.

Keep it up !

Reply

- *Pawankumar Jajara* says:
  November 2, 2011 at 22:48

  The best article on JavaScript Closures, I ever found on Web. I would say its very very simple and clear. Looking forward for more tutorials on JavaScript 🙂

  Keep it up !

  Reply

- *Robert Nyman* says:
  November 3, 2011 at 3:36

  Pawankumar Jajara,

  Thank you, glad you liked it!

  Reply

- *dublintech* says:
  November 18, 2011 at 10:32

  Goo stuff, I like that.

  Reply

- *Fernando Sávio* says:
  November 28, 2011 at 18:50

  Congratulations, your explanation saved my life (ok, just my work)..
  Thank you so much!

  Reply

- *Robert Nyman* says:
  November 28, 2011 at 20:53

  Thanks, glad you liked it!

  Reply

- *A Most Useful Tutorial on Javascript Closures « The Trickle-Down* says:
  December 23, 2011 at 10:20

  […] explanatory code is from the blog post Explaining JavaScript scope and closures LD_AddCustomAttr("AdOpt", "1"); LD_AddCustomAttr("Origin", "other"); […]

  Reply

- *Tom* says:
  January 1, 2012 at 16:38

  Thank you very much for this. 🙂

  Reply

- *Robert Nyman* says:
  January 2, 2012 at 12:37

  Tom,

  Cool, glad you liked it!

  Reply

- *Krupar* says:

January 6, 2012 at 12:35

Thanks for this article. It’s a nice explanation.

regards
krupar

Reply

- *Patrik Krupar* says:
  January 6, 2012 at 13:56

  I think, there ist a mistake. Your Example “Self-invoking functions” returns “German Shepherd” not undefined. (FF5, Chrome 16)

  Reply
- *Robert Nyman* says:
  March 2, 2012 at 12:03

  Patrik,

  If you check the console in your web browser, you will see that you first get tjhe alert within the function and then an error for the undefined part (i.e., not an alert with the text “undefined”).

  Reply
- *Sougata* says:
  April 7, 2012 at 15:45

  Most lucid explanation of closures I’ve found anywhere, and that includes Crockford’s book, “Javascript: The Good Parts”.

  Kudos.

  Reply
- *Robert Nyman* says:
  April 13, 2012 at 12:31

  Sougata,

  Thanks, makes me very happy to hear that!

  Reply
- *Yawo* says:
  April 28, 2012 at 14:44

  Hooraah ! never understand the jquery (function()…
  Now I can confidently dive into.

  And above all nice comments, YOU ARE A TEACHER ! Just in the way you take time to explain and understand even seeming-stupid questions

  Reply
- *Robert Nyman* says:
  May 8, 2012 at 19:19

  Yawo,

  Thank you, glad you found it useful!

  Reply
- *Ragini* says:
  May 9, 2012 at 16:12

  Nice Article … very well explained.

  Reply

- *NP* says:
  [May 16, 2012 at 2:42]()

  This is very helpful Robert. Thanks a lot for detailed explanation

  [Reply]()
- *[Robert Nyman]()* says:
  [May 16, 2012 at 14:46]()

  Ragini, NP,

  Thanks, glad you liked it!

  [Reply]()
- *[More JavaScript Stuff « Input Coffee]()* says:
  [May 22, 2012 at 3:00]()

  […] Taken from: http://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ This is a great resource and I should refer back to it […]

  [Reply]()
- *[JavaScript gotcha: scoping and loops | Summa Blog]()* says:
  [May 22, 2012 at 20:23]()

  […] Explaining JavaScript Scope And Closures […]

  [Reply]()
- *Suresh Pydi* says:
  [June 4, 2012 at 12:32]()

  Great Article…….Thank you…Thank you very much for explaing about this………..

  [Reply]()
- *[Robert Nyman]()* says:
  [June 4, 2012 at 14:10]()

  Suresh,

  You're welcome!

  [Reply]()
- *webui* says:
  [June 6, 2012 at 10:56]()

  Thank u very much for the post.Really best Practical example I ever came to understand Closures. Keep Posting.

  [Reply]()
- *[Robert Nyman]()* says:
  [June 7, 2012 at 8:13]()

  webui,

  Glad you liked it!

  [Reply]()
- *Anderson Fortaleza* says:
  [June 10, 2012 at 19:25]()

  Excellent explanation, it was very useful to me, specially the loop problem, something overlooked on most closures articles.

  thank you

  [Reply]()
- *[Robert Nyman]()* says:

June 12, 2012 at 22:02

Anderson,

Thank you, glad it was useful!

Reply

- *Writing JavaScript | Jim White's Homepage* says:
  June 21, 2012 at 21:30

  […] from it with only one parameter and the other 2 set to specific values you can do this with closures. There are probably other tricks you can do, but I used this one to help troubleshoot one of my […]

  Reply
- *msn* says:
  July 20, 2012 at 17:02

  thanks man , you created a great article
  it's very useful because of simpleness

  Reply
- *Marek* says:
  August 12, 2012 at 16:19

  There is few more JavaScript confusing bits. 🙂 Thanks for the post though.

  The only thing I don't understand here is why you name Modular Pattern as Yahoo JavaScript Module Pattern?? It is not like they created something original right? It is used by many developers and hasn't been invented by Yahoo.

  Reply
- *Robert Nyman* says:
  August 13, 2012 at 14:34

  Marek,

  Thanks, glad you liked it!

  The pattern – or rather packaging and fame of it – came from Douglas Crockford when he was at Yahoo and got its name from there.

  Reply
- *Pete* says:
  September 21, 2012 at 14:52

  Thanks for this post on closures, now I only have another 97 key JavaScript concepts to go!

  Reply
- *Robert Nyman* says:
  September 21, 2012 at 14:53

  Ha ha! Good luck!

  Reply
- *appendChild in for loop only adds 1 child* says:
  October 4, 2012 at 17:54

  […] already looked at this page and this page, and they sound promising but I just can't figure out how to make this […]

  Reply
- *taiansu* says:
  October 16, 2012 at 19:16

  Just want to thanks for your excellent explain. Really helps me a lot. I've translate your work to Traditional Chinese(which use in Taiwan and Hong Kong), mark with your name and have a link to this post. Many tanks again!

Reply

- *Robert Nyman* says:
  October 17, 2012 at 16:02

  taiansu,

  Thank you, that's great!
  I've added a link to your translation at the top of the article.

  Reply
- *Stuart Dobson* says:
  October 19, 2012 at 1:56

  "Whoa, whoa! What just happened?" – I laughed out loud when I read that because it was exactly what I was thinking!

  Great job explaining it in the end – public and private variables! Is there nothing Javascript can't do?

  Reply
- *Robert Nyman* says:
  October 19, 2012 at 9:50

  Stuart,

  ☺ Glad you liked it!

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript | MyOfflineTheme.com Skyrocket Your Offline Business Just Now* says:
  November 5, 2012 at 16:41

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript - Steve deGuzman* says:
  November 6, 2012 at 5:31

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript « Books* says:
  November 6, 2012 at 8:09

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript | Diancin Designs* says:
  November 6, 2012 at 8:57

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript | wpdevlabs.com* says:
  November 7, 2012 at 13:47

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript | |* says:
  November 8, 2012 at 23:22

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

Reply

- *Performance: Writing Fast, Memory-Efficient JavaScript | Bitport* says:
  November 13, 2012 at 10:44

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *??? ????? ???? ???? ?????? ???? | NCODI's Blog* says:
  November 19, 2012 at 2:57

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript | PJExploration* says:
  November 27, 2012 at 17:08

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For example: […]

  Reply
- *Trenton D. Adams* says:
  December 4, 2012 at 0:27

  So, the explanation for "The infamous loop problem" seemed to be missing the common programming language I would expect. For a javascript function, the parameters are passed to the function by value, not be reference. In other words, the function gets a "copy" not a "pointer" to the original data.

  So, if you don't do the wrapper function, you get a reference to the exact variable defined in the loop, hence it's the last value it was assigned.

  Might be useful to update the wording, so it's not quite so confusing as to "why" javascript works that way.

  Reply
- *Robert Nyman* says:
  December 4, 2012 at 16:44

  Trenton,

  Thanks for the suggestion!
  The interesting part there is that, for me, who doesn't have too much experience with other programming languages, the copy and pointer wording isn't totally clear.

  I'll think about it. Thanks!

  Reply
- *Himanshu Makwana* says:
  December 6, 2012 at 11:31

  Great job simple and through.

  Reply
- *Vikas* says:
  January 4, 2013 at 11:31

  Very good and to the point article. Many thanks!!!

  Reply
- *Performance: Writing Fast, Memory-Efficient JavaScript | ShadesColour & Associates* says:
  January 9, 2013 at 11:47

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply

- *Nick* says:
  January 11, 2013 at 10:03

  thanks… nice explanation…

  Reply
- *Writing Fast, Memory-Efficient JavaScript » Xeno Joshua | Xeno Joshua* says:
  January 11, 2013 at 10:21

  […] ?? – ????????????????????????????????? […]

  Reply
- *Poornima* says:
  January 15, 2013 at 23:29

  great article!

  Reply
- *Understanding JavaScript Closures and Scope | TechSlides* says:
  January 21, 2013 at 20:53

  […] definition on WikiPedia JavaScript Execution Context and Closure Grokking V8 Closures javaScript Scope and Closures Closures and the Module Pattern Closures – Mozilla JS Closures Introduction to Closures […]

  Reply
- *piris* says:
  January 27, 2013 at 17:36

  great closure example!

  Reply
- *Michael Alexander* says:
  January 31, 2013 at 22:56

  Dear Robert,

  What is your definition of context as used in your article and how is that different from scope?

  -M

  Reply
- *Michael Alexander* says:
  February 2, 2013 at 1:02

  If you want to understand closures, walk or run to the following web page:

  http://www.javascriptkit.com/javatutors/closures.shtml

  It will save you lots of time.

  ;-))

  Reply
- *Alex* says:
  February 4, 2013 at 7:27

  Just want to say thanks, Robert. I went through quite many tutorials about closures but yours is the easiest to understand.

  Reply
- *Robert Nyman* says:
  February 4, 2013 at 9:42

  Thanks everyone who liked the article!

Michael,

In this article, context and scope would refer to the same thing. Context is just more ordinary speak, I'd say. ☺
Also, thanks for the link!

Reply

- *joshua* says:
  February 25, 2013 at 17:07

  If you say:

  1) When the add function is called, it returns a function.
  2) That function closes the context and remembers what the parameter x was at exactly that time (i.e. 5 in the code above)

  it seems as the closure is created only if we call the function but the closure is created when you define the code:

  ```
  function add (x) {
  .return function (y) {
  return x + y;
  };
  }
  ```

  so the inner function save in its scope the outer scope of its parent (that's the lexical scoping rules).

  Reply
- *Robert Nyman* says:
  March 1, 2013 at 16:20

  joshua,

  Yes, I believe you could say that.

  Reply
- *Writing Fast, Memory-Efficient JavaScript - Goodfav Howto* says:
  March 4, 2013 at 7:23

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *DD* says:
  March 9, 2013 at 8:59

  Hi Robert,
  Really a very good article.
  I read same examples before in other articles but with your article i understood them.
  It helped me a lot to understand closures.
  You explained it in simple words.
  Thanx a lot…:)

  Reply
- *How I achieved my MCSD: Windows Store App using HTML5 Certification; Study guide - Start* says:
  March 14, 2013 at 13:16

  […] Blog […]

  Reply
- *JavaScript: Tools, Coding Standard and Guidelines ← Select ** says:
  March 15, 2013 at 10:51

  […] Closures : Working with closures and Explaining JavaScript scope and closures. […]

  Reply
- *Dav* says:
  April 20, 2013 at 17:50

  Great article, closures became a clearer concept for me thanks to you.

Just a question: as the following code does the same job that your addLinks closure example, could you say which method is the best and why ?

```
function addLinks () {
for (var i=0, link; i<5; i++) {
link = document.createElement("a");
link.innerHTML = "Link " + i;
link.id = i;
link.onclick = function() { alert(this.id); }
document.body.appendChild(link);
}
}
window.onload = addLinks;
```

Reply

- *Robert Nyman* says:
  April 22, 2013 at 10:36

  Dav,

  With one, you need to apply a property to the element and reference it, whereas in the closure example, you keep it in memory.

  I'd prefer not to clutter the element with something like that, but it's nothing wrong with it and it works.

  Reply
- *Writing Fast, Memory-Efficient JavaScript | Smashing Coding* says:
  May 3, 2013 at 19:02

  […] will have access to the outer scope even after the outer function is executed. This is basically a closure — an expression which can work with variables set within a specific context. For […]

  Reply
- *marg* says:
  June 3, 2013 at 10:45

  Nice demonstration!
  About scope, another trick for me…

  ```
  var a = 4;
  function doIt(){
  alert(a);
  var a = 5;
  }
  doIt();
  ```

  Would you explain me why "doIt()" gives an "undefined" a and not the "global " value of 4?
  Many thanks again!

  Reply
- *Robert Nyman* says:
  June 4, 2013 at 12:34

  marg,

  It's due to something called variable hoisting, i.e. a gets sent to the top with no defined value. Please read more in JavaScript Scoping and Hoisting.

  Reply
- *Javascript infamous Loop problem? - Tech Forum Network* says:
  June 7, 2013 at 17:11

  […] above 2 snippets are quoted from here. As the author's explanation, seems the closure makes the […]

  Reply
- *Gary Johnson* says:
  July 7, 2013 at 0:16

This is very clear and concise. I have a much better hold on Scope and Closure.

Thanks Robert!

Reply

- *sample test1 | jeeva's blog* says:
  July 8, 2013 at 11:21

  […] Explaining JavaScript scope and closures (Robert Nyman) […]

  Reply
- *Javascript infamous Loop problem? | Think Aspx* says:
  August 9, 2013 at 1:05

  […] above 2 snippets are quoted from here. As the author's explanation, seems the closure makes the […]

  Reply
- *Esto no es lo que parece, el "this" en JavaScript | Mouseless Me* says:
  August 12, 2013 at 8:58

  […] Podemos tener efectos secundarios que no voy a explicar aquí, pero seguro que os parece interesante el artículo de Robert Nyman. […]

  Reply
- *Javascript infamous Loop problem? - oneuptime | oneuptime* says:
  August 20, 2013 at 21:27

  […] above 2 snippets are quoted from here. As the author's explanation, seems the closure makes the […]

  Reply
- *Javascript infamous Loop problem? | Ask Programming & Technology* says:
  November 1, 2013 at 9:44

  […] above 2 snippets are quoted from here. As the author's explanation, seems the closure makes the […]

  Reply
- *Robert Nyman: ?? JavaScript ? Scope ? Closures | ????????????……?* says:
  November 28, 2013 at 6:24

  […] Nyman ? 2008 ??????? Original Post […]

  Reply
- *????????JavaScript | ????* says:
  December 14, 2013 at 12:03

  […] ??? —— ?????????????????????? […]

  Reply
- *hestroy* says:
  January 15, 2014 at 14:01

  Self-invoking functions are in reality IIFE (Immediately-invoked function expression).

  There is a significant difference between FD (Function declaration) and FE.

  Reply
- *Guia para preparar el examen 70-480 HTML5 y JavaScript | joseadrien* says:
  February 6, 2014 at 14:19

  […] http://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ […]

  Reply
- *Exam 70-480 (programming in html5 with javascript and CSS3) Study Material | R. Suharta's Blog* says:
  February 23, 2014 at 2:27

  […] http://robertnyman.com/2008/10/09/explaining-javascript-scope-and-closures/ […]

  Reply
- *????????JavaScript | ??Web?? AlloyTeam ?? Blog | ??: ???????Web???* says:
  February 26, 2014 at 6:46

[…] ?? —— ?????????????????????? […]

Reply

- *Writing Fast, Memory-Efficient JavaScript - Internet Business* says:
  March 2, 2014 at 11:15

  […] will have access to the outer scope even after the outer function is executed. This is basically aclosure — an expression which can work with variables set within a specific context. For […]

  Reply
- *madhu* says:
  May 29, 2014 at 8:20

  very nice .I will suggest this post to my friends .

  Reply
- *Spiro* says:
  June 25, 2014 at 9:13

  The only thing I can't understand .. (and its probably obvious is.) is "y" argument to "function"
  How does it get it value ?

  ```
  var add5 = add(5);
  var no8 = add5(3);
  ```

  I understand that when add is called with 5 , X gets its value.
  when add5 is called with 3 , how does y get it value.

  ```
  function add (x) {
  return function (y) {
  return x + y;
  };
  }
  var add5 = add(5);
  var no8 = add5(3);
  alert(no8); // Returns 8
  ```

  Reply
  - *Robert Nyman* says:
    July 8, 2014 at 15:21

    The result of calling the add function with a value is then saved as a new function that you can call. That means that any value you send in to add5 (e.g. 3) that will be the y value in the function. It's basically like the x value is hard-coded from before, and the sent in value is the dynamic one, the y one.

    Reply
- *Alexa* says:
  October 2, 2014 at 17:55

  Your explanations to closures was perfect and zoned in on my grey area.

  thank you!!

  Reply
- *Frank Kim* says:
  October 25, 2014 at 0:42

  Best explanation I've read yet about closures. Thanks!

  Reply
- *Ajit* says:
  October 25, 2014 at 20:42

  Very nice explanation I have ever read !!! Thanks for posting such a nice article on closuers.

Reply

- *Elise* says:
  November 19, 2014 at 15:54

  I totally agree with the previously-posted comment, “Most lucid explanation of closures I’ve found anywhere, and that includes Crockford’s book, Javascript: The Good Parts.”

  You are an absolute angel!!

  Reply
- *Robert Nyman* says:
  November 20, 2014 at 6:24

  Thanks everyone for the kind words!

  Reply
- *varsha* says:
  December 21, 2014 at 11:50

  ```
  var abc = “hello”
  function () {
  var self = this;
  console.log (this.abc);
  console.log (self.abc);
  function () {
  console.log(this.abc);
  console.log(self.abc);
  }()
  }
  ```

  what will be o/p n why explain me plz

  }

  Reply
    - *Robert Nyman* says:
      December 21, 2014 at 20:12

      In short, the abc variable is global and available everywhere (with or without the this keyword). The self.abc property is only available within that function.

      Reply
- *Gary Hsu* says:
  January 6, 2015 at 19:52

  Just for your info, the Traditional Chinese translation link has been updated:

  http://blog.taian.su/2012-10-17-explaining-javascript-scope-and-closures-by-robert-nyman/

  Thank you for good work!

  Reply
    - *Robert Nyman* says:
      January 7, 2015 at 15:22

      Nice, thank you!

      Reply
- *abby* says:
  February 6, 2015 at 14:47

  You’re pretty smart in explaining hard things !! Thank you so much!!

  Reply

- *Robert Nyman* says:
  February 18, 2015 at 20:21

  Thanks, glad it was of use!

  Reply

- *Rahul mendonca* says:
February 14, 2015 at 11:56

Close to 7 years since this article but this one gem of an article.Im still in the initial phases as far as concerned but I look forward to posts like these.

Cheers from India

Reply

- *Robert Nyman* says:
  February 18, 2015 at 20:21

  Great to hear, thank you!

  Reply

- *ST* says:
March 16, 2015 at 19:07

Are these two code snippets equivalent (because of hoisting)?

1.)
```
(function(){
function foo(){
alert(a);
}
var a;
foo();
})();
```

2.)

```
(function(){
var a;
function foo(){
alert(a);
}
foo();
})();
```

Reply

- *Robert Nyman* says:
  April 30, 2015 at 12:35

  Yep.

  Reply

- *Bart McLeod* says:
April 2, 2015 at 22:24

Hi,

Just wanted to say thank you! You really saved my day. I was looking at the opposite problem today, having created a number of closures that keep the context of the creation time, while I needed them all to use the same reference to the latest state instead. I suspected it had something to do with closure scope and context and your post was the first hit on Google when I started researching.

Thanks!

Reply

- *Robert Nyman* says:
  April 30, 2015 at 12:35

  Thanks, good to hear it was useful!

  Reply

- *Bruno* says:
  October 13, 2015 at 4:52

  Wow, awesome post is awesome, even years after it's first appearance.

  This explanation laid ground for a talk I'm going to give for Nodeschool on closures.

  Thank you very much!

  Reply

- *Rauni* says:
  November 13, 2015 at 11:46

  Thanks! The part about "infamous loop problems" saved the day for me ☺

  Reply

- *Sal* says:
  November 21, 2015 at 15:22

  Scope – can global variable scope be extended between renders. Example:

  on initial page render global var x =1

  user performs some functions and then clicks on icon.

  the icon on click event is x=x + 1. x should be 2. It is 1. Do I have a bug or this not possible?

  Reply

- *Manish Jain* says:
  December 2, 2015 at 1:01

  Best closure article on the web. thank you!!

  Reply

- *Lin* says:
  January 3, 2016 at 3:29

  Thanks, it seems that I have understood it. And I'm a Chinese, my English is not good, but your writing is so concise. My expression may be not accurate, but you know what I want to say is 'Thanks'.

  Reply

- *Oliver* says:
  January 22, 2016 at 22:05

  Still reading in 2016. And still helpful. Thank you man for great explanation. Give this man a cookie…

  Reply

- *Vic* says:
  February 7, 2016 at 5:29

  I have a PHP back-end programming background and I've used JavaScript for minor things on the front-end, but now I'm learning how to program in it. I'm reading eloquent javascript and couldn't grasp closures, so in doing further research before continuing with my reading I came across your page. Your explanation is excellent. Thanks Robert!

  Reply

- *Robert Nyman* says:

February 8, 2016 at 10:00

Thanks all, glad this is still useful to all of you!

Reply

- *dogan* says:
  March 2, 2016 at 10:17

  Thanks for useful information. Keep continue to share more!

  Reply

- *Rodrigo* says:
  March 4, 2016 at 22:37

  Hey Robert,

  Thanks for the article!!

  I do have a question though. Turns out that I've been looking at closures lately because someone asked me about broken code that doesn't uses closures (besides the infamous loop example, which can also be considered using a list), and I'm having a lot of troubles coming up with something. I've been looking around for bad closures in JS but with no luck. Seems that everyone kinds of forget about the problem once it gets solved, I know I have 😛

  Do you know any other snippets that shows issues because closures are not created?

  Thanks,
  Rodrigo.

  Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

- @robertnyman
- Upcoming speaking
- Photos on Flickr