Understand JavaScript Closures With Ease

FEB. 2 2013 238

Closures allow JavaScript programmers to write better code. Creative, expressive, and concise. We frequently use closures in JavaScript, and, no matter your JavaScript experience, you will undoubtedly encounter them time and again. Sure, closures might appear complex and beyor your scope, but after you read this article, closures will be much more easily understood and thus more appealing for your everyday JavaScript programming tasks.

menu

This is a relatively short (and sweet) post on the details of closures in JavaScript. You should be familiar with JavaScript variable scope before you read further, because to understand closures you must understand JavaScript's variable scope.

Receive Updates

Your Email: Go

What is a closure?

A closure is an inner function that has access to the outer (enclosing) function's variables—scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function's variables, and it has access to the global variables.

The inner function has access not only to the outer function's variables, but also to the outer function's parameters. Note that the inner function cannot call the outer function's *arguments* object, however, even though it can call the outer function's parameters directly.

You create a closure by adding a function inside another function.

A Basic Example of Closures in JavaScript:

```
function showName (firstName, lastName) {
var nameIntro = "Your name is ";
    // this inner function has access to the outer function's variables, including the parameter
function makeFullName () {
return nameIntro + firstName + " " + lastName;
}

return makeFullName ();
}

showName ("Michael", "Jackson"); // Your name is Michael Jackson
```

Closures are used extensively in Node.js; they are workhorses in Node.js' asynchronous, non-blocking architecture. Closures are also frequently used in jQuery and just about every piece of JavaScript code you read.

A Classic jQuery Example of Closures:

```
$(function() {

var selections = [];
$(".niners").click(function() { // this closure has access to the selections variable
selections.push (this.prop("name")); // update the selections variable in the outer function's scope
});

});
```

Closures' Rules and Side Effects

1. **Closures have access to the outer function's variable even after the outer function returns:**
   One of the most important and ticklish features with closures is that the inner function still has access to the outer function's variables even after the outer function has returned. Yep, you read that correctly. When functions in JavaScript execute, they use the same scope chain that was in effect when they were created. This means that even after the outer function has returned, the inner function still has access to the outer function's variables. Therefore, you can call the inner function later in your program. This example demonstrates:

```
function celebrityName (firstName) {
    var nameIntro = "This celebrity is ";
    // this inner function has access to the outer function's variables, including the parameter
   function lastName (theLastName) {
        return nameIntro + firstName + " " + theLastName;
    }
    return lastName;
}

var mjName = celebrityName ("Michael"); // At this juncture, the celebrityName outer function has

// The closure (lastName) is called here after the outer function has returned above
// Yet, the closure still has access to the outer function's variables and parameter
mjName ("Jackson"); // This celebrity is Michael Jackson
```

2. **Closures store references to the outer function's variables**; they do not store the actual value. Closures get more interesting when the value of the outer function's variable changes before the closure is called. And this powerful feature can be harnessed in creative ways, such as this private variables example first demonstrated by Douglas Crockford:

```
function celebrityID () {
    var celebrityID = 999;
    // We are returning an object with some inner functions
    // All the inner functions have access to the outer function's variables
    return {
        getID: function ()  {
```

```
            // This inner function will return the UPDATED celebrityID variable
            // It will return the current value of celebrityID, even after the changeTheID function change
          return celebrityID;
        },
        setID: function (theNewID)  {
            // This inner function will change the outer function's variable anytime
            celebrityID = theNewID;
        }
    }

}

var mjID = celebrityID (); // At this juncture, the celebrityID outer function has returned.
mjID.getID(); // 999
mjID.setID(567); // Changes the outer function's variable
mjID.getID(); // 567: It returns the updated celebrityId variable
```

3. **Closures Gone Awry**

   Because closures have access to the updated values of the outer function's variables, they can also lead to bugs when the outer function's variable changes with a for loop. Thus:

```
// This example is explained in detail below (just after this code box).
function celebrityIDCreator (theCelebrities) {
    var i;
    var uniqueID = 100;
    for (i = 0; i < theCelebrities.length; i++) {
      theCelebrities[i]["id"] = function ()  {
        return uniqueID + i;
      }
    }

    return theCelebrities;
}

var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];

var createIdForActionCelebs = celebrityIDCreator (actionCelebs);

```

```
18 var stalloneID = createIdForActionCelebs [0];console.log(stalloneID.id()); // 103
```

In the preceding example, by the time the anonymous functions are called, the value of i is 3 (the length of the array and then it increments). The number 3 was added to the uniqueID to create 103 for ALL the celebritiesID. So every position in the returned array get id = 103, instead of the intended 100, 101, 102.

The reason this happened was because, as we have discussed in the previous example, the closure (the anonymous function in this example) has access to the outer function's variables by reference, not by value. So just as the previous example showed that we can access the updated variable with the closure, this example similarly accessed the i variable when it was changed, since the outer function runs the entire for loop and returns the last value of i, which is 103.

To fix this side effect (bug) in closures, you can use an **Immediately Invoked Function Expression** (IIFE), such as the following:

```
1  function celebrityIDCreator (theCelebrities) {
2      var i;
3      var uniqueID = 100;
4      for (i = 0; i < theCelebrities.length; i++) {
5          theCelebrities[i]["id"] = function (j)  { // the j parametric variable is the i passed in on invocati
6              return function () {
7                  return uniqueID + j; // each iteration of the for loop passes the current value of i into thi
8              } () // BY adding () at the end of this function, we are executing it immediately and returnin
9          } (i); // immediately invoke the function passing the i variable as a parameter
10     }
11
12     return theCelebrities;
13 }
14
15 var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];
16
17 var createIdForActionCelebs = celebrityIDCreator (actionCelebs);
18
19 var stalloneID = createIdForActionCelebs [0];
20 console.log(stalloneID.id); // 100
21
```

```
22  var cruiseID = createIdForActionCelebs [1];console.log(cruiseID.id); // 101
```


Posted in: 16 Important JavaScript Concepts, JavaScript / Tagged: Closures, JavaScript, Learn JavaScript

Richard

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

238 Comments

Nimesh

February 27, 2013 at 4:12 am / Reply

Thanks for the great post. I am bookmarking your website. A sexy collection of javascript tutorials.

Richard Bovell (Author)

February 27, 2013 at 11:11 am / Reply

Thanks much, Nimesh. I am happy to hear the website is helpful and… sexy, too. 🙂

vedran

August 12, 2013 at 10:33 am / Reply

Very good site !!!!

DeVontae Moore

October 17, 2014 at 8:41 pm / Reply

theCelebrities[i]["id"] = function (j) {

Im not familiar with the above syntax. Could you please explain?

MS

November 7, 2014 at 8:17 pm / Reply

This statement assigns a new value to a variable inside an array of objects.

Recall that theCelebrities now has the value of [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}]
The first set of square brackets, theCelebrities[i], calls an object by its array position. If i = 0 then theCelebreties[i] is equal to the first object in the array, in this case, {name:"Stallone", id:0}.
The second set of brackets, ["id"], call that object's property by its name "id" .
The rest of the statement "= function(j){….}(i)" simply reassigns the value of the property "id" from "0" to the return of the function.

On the macro level {name:"Stallone", id:0} becomes {name:"Stallone", id:100}.

This type of index referencing works with any type of nested array:
if a = [4, [apple, 7, triangle, [88, 69], 6, 6], tomorrow] then:
a[1][3][0] is equal to 88; and
a[2] is equal to tomorrow;
Hope that helps 😉

_redrobot

February 13, 2015 at 6:13 pm /

That was an amazing explanation. Thanks for taking the time to write that up!

pradeep

March 25, 2013 at 3:01 pm / Reply

very helpful

Lixiang

March 28, 2013 at 11:07 pm / Reply

Good post, It makes me feel Javascript is so sexy and interesting.

Richard Bovell (Author)

April 1, 2013 at 8:46 pm / Reply

Haha, you made my day. Great Comment, Lixiang.

Roberto

April 4, 2013 at 4:39 pm / Reply

Sorry man, I really don't get the example on the first rule,
when you say:

mjName ("Jackson"); // This celebrity is Michael Jackson

how are you calling the inner function there?

Richard Bovell (Author)

April 4, 2013 at 9:00 pm / Reply

No need to be sorry, Roberto, I understand how this part is confusing.

Here is the explanation.
The *celebrityName* function returns a **function**; the function it returns is the *lastName* inner function.
So when we do:
var mjName = celebrityName ("Michael");
The *mjName* variable now contains the *lastName* inner function that was returned when we called celebrityName (). And the inner function has all the variables, including the name "Michael" that was passed into the celebrityName function.

To prove that the mjName variable contains the lasName function, we can do this:

```
1  console.log (mjName);
2
3  // The output is this:
4  function lastName(theLastName) {
5        return nameIntro + firstName + " " + theLastName;
6    }
7
```

So *mjName* is a function expression: it is a variable with a value of a function.
Now, all we have to do to call the function stored in mjName is add the parentheses () to execute the function, like this:
mjName (); //When it executes, the following line from the lastName inner function runs;
return nameIntro + firstName + " " + theLastName;
And this is why the output is: "This celebrity is Michael Jackson"
When we do mjName ("Jackson");

Roberto

April 4, 2013 at 9:40 pm / Reply

Getting there!! but WOW! thanks for taking the time to write such a long and clear answer! I'm wrapping my head around the last one now.

It's weird though being a JavaScript Blog you dont have client side validation on the comments form. 😉

Richard Bovell (Author)

April 4, 2013 at 10:02 pm / Reply

You are welcome.

I haven't thought about the client-side form validation until you mentioned it, so I just added it. Thanks for the tip. We helped each other out 🙂

Vineet Sajwan

August 21, 2014 at 2:35 pm / Reply

Hey Roberto,
Its quite amazing that I have same two doubts, one is mjName thing and other absence of javascript form validation on a such a SEXY javascript tutorial site.

Thanks Richard Bovell for sharing your valuable knowledge.

Christopher

June 11, 2013 at 3:46 am / Reply

This was *profoundly* helpful to me. Thank you so much for taking the time to reply so thoroughly.

Brett Warner

August 10, 2013 at 3:14 pm / Reply

Thank you so much for this comment. I've been trying to understand clojures for a few days now and the fact that it was returning the function was the missing piece for me, I feel like all the code I've pored over just suddenly started to make sense.

Richard Bovell (Author)

August 15, 2013 at 2:08 pm / Reply

Great to hear that you have the moment of absolute clarity with JavaScript, Brett. All of us experience the same thing from time to time.

Pravin Waychal

January 24, 2014 at 8:03 am / Reply

Very Helpful and clear explanation. Thanks so much!

andy

April 14, 2014 at 2:07 am / Reply

everyone trying to understand closures should read this question and answer.

Bryson

November 21, 2014 at 8:13 am / Reply

Yep, agreed. I've been trying to understand closures on and off for two years... This might have just finally solved it for me!

jackson

May 9, 2014 at 7:16 pm / Reply

Thanks for this really helpful reply. Pointing out that the container function returns the inner function was key for me.

Although after typing out the example, I noticed that printing out mjName doesn't actually show the function. console.log(mjName); //prints [Function: lastname]

To show the actual function I had to use the prototype method of the function object:

console.log(mjName.toString()); //prints function source

Thbo

February 24, 2016 at 3:40 am / Reply

But how does the lastName callback function receive the ‘thelastName'(Jackson) parameter when calling the mjName variable?

Juan

April 9, 2013 at 11:31 am / Reply

This site rocks man!! Thanks for sharing your knowledge in an easy way. Saludos desde Argentina

ali

April 18, 2013 at 5:28 pm / Reply

i have one question about the following.
theCelebrities[i][“id”] = function (j) { // the j parametric variable is the i passed in on invocation of this IIFE

how does it here know the j should take the value of the current i ,maybe i am missing something obvious in here:)

many thanks

ali

April 18, 2013 at 5:31 pm / Reply

oops i missed this little devil in here
(i); // immediately invoke the function passing the i variable as a parameter

sorry

Richard Bovell (Author)

April 25, 2013 at 8:15 pm / Reply

Ali,
Are you good now or do you still have a question?

ali

April 26, 2013 at 5:13 am / Reply

Yeah its cool now ,thanks alot.

Jason Kotenko

May 26, 2013 at 7:37 pm / Reply

For the CelebrityIdCreator example, could you also use:

theCelebrities[i]["id"] = i;

So that the value is set right then, rather than have "id" be a function?

Richard Bovell (Author)

May 27, 2013 at 10:34 am / Reply

Very good question, Jason.

To do that, we can make one change:
Instead of returning a function, we can go ahead and call the function (execute it immediately) and return the uniqueID + J number.

I just updated the code in the article to store numbers in the returned array, instead of functions.

Piyush

June 23, 2013 at 4:15 am / Reply

Hi Richard,
Awesome blog....seriously ...going through your blog has made some of the

difficult concepts so easy. Thanks a lot.

Is it possible to refactor the code for removing the bug in Closure with the below way :

```
theCelebrities[i]["id"] = (function () {
return uniqueID + i;
})();
```

The one mentioned above somewht confused me at the start , so i guess this would too solve the purpose. In effect, both the code are doing the same things 🙂 🙂

Richard Bovell (Author)

June 25, 2013 at 1:11 am / Reply

HI Piyush,

I am glad you were able to refactor the code to a style that works better for you. That is an important aspect of programming that is underrated. In my opinion, you should feel very comfortable with the way your code looks and reads back in your head.

When I have some time, I will look at the code and see if it is worth refactoring.

Thanks.

soncu

June 26, 2013 at 7:33 am / Reply

Very good article. I loved your explanation really but when i saw:

```
theCelebrities[i]["id"] = function (j) {
return function () {
return uniqueID + j;
```

```
} ()
} (i);
```

i got uncomfortable. Because you introduce two IIFE here. And one can offer in a more complicated way as:

```
theCelebrities[i]["id"] = function (j) {
return function () {
return function() {
return function() {
return function() {
return uniqueID + j;
}()
}()
}()
} ()
} (i);
```

but these all make things more harder. I suggest to use just IIFE in a simple way as:

```
theCelebrities[i]["id"] = function (j) {
return uniqueID + j;
} (i);
```

and all these versions just use:

```
theCelebrities[i]["id"] = uniqueID + j;
```

Besides I want to add one more thing: your first example under "Closures Gone Awry" section all ids assigned to a function but the other example(bug fixing as your phrase) just assigns numeric variable using IIFE.I think it is impossible to solve this problem with assigning function to ids.

Hadar

March 11, 2014 at 3:51 pm / Reply

Exactly what I thought!
Too bad you didn't get an answer for that until today.. this could be *very* helpful.

Nirmal

July 2, 2013 at 7:21 am / Reply

Richard, this is the coolest blog on JavaScript that I ever saw! It helped me in understanding the fundamentals thoroughly. 🙂 Thanks!

Richard Bovell (Author)

July 5, 2013 at 6:35 pm / Reply

Thank you very much, Nirmal. I am glad the blog has been helpful.

Evox

July 12, 2013 at 12:59 am / Reply

– In example 2, why we have to add return before get and set function.

– why it have to include return before function and value, can I write it with out return

– Is the inner function excute and return separately from outer function?

Thanks you

Richard Bovell (Author)

July 17, 2013 at 3:01 pm / Reply

You can reformat the code to not return an object with the methods, but then you will have to change the celebrityID function and make it an object. And is you do, there will be no reason to assign it to

another variable to use the closure with references to the outer function's variables.

So the reason we used "return" to return an object in the celebrityID function is precisely because of the closure and accessing the outer function's variables later.

WebGyver

July 19, 2013 at 5:48 pm / Reply

First of all, thanks for this article. I greatly appreciate the time you took to work on the article itself but also on answers to questions.

I have a problem with the celebrityName example that returns a function. No matter how hard I try, I can only get the first name back. Instead of a lastName, I only get a 0.

I have set this up at http://jsfiddle.net/MmDvz/ if anybody else wants to help me see the errors of my ways.

Thank you for any help anyone could offer.

Ecovox

July 20, 2013 at 1:17 am / Reply

I think the mjName("Miller"); was return and destroyed before you call it. try this: $('#name').html(mjName("Miller"));.

I'm not sure why it be like this, maybe a concat method from primitive wrapper was call to produce it, after it is call, it will destroy.

sorry for my English.

Richard Bovell (Author)

July 31, 2013 at 1:04 am / Reply

You have everything correct except you need to call mjName as a function, as **Ecovox** correctly pointed out below.

```
1
2  /* The mjName variable is being assigned a function, which is the functio
3  */
4  var mjName = celebrityName("Mike");
5  // So you must call mjName like this:
6  $('#name').html(mjName("Miller"));
7
8  // Instead of like this
9  $('#name').html(mjName);
10
```

This line below by itself (which you have in your code) is not displaying anything to the screen or on the page because you are not logging (console.log) it to the console or displaying the results to the page:

```
1
2  mjName("Miller");
3
```

So the bottom line is that you can do either this:

```
1
2  console.log(mjName("Miller));
3
```

Or this:

```
1
2  $('#name').html(mjName("Miller"));
3
```

Ecovox, thanks for helping out. You answer is correct.

Noah Jerreel Guillen

August 1, 2013 at 8:47 pm / Reply

Your blog is really amazing. Keep it up.

Richard Bovell (Author)

August 15, 2013 at 2:06 pm / Reply

Thank you much, Noah.

And thanks, Dineshswamy—this is the coolest name, btw 🙂

dineshswamy

August 9, 2013 at 4:19 am / Reply

I like most of your articles !! Your articles stands because of one reason .. you dont give space for ads !! Nice.

tanyaka

August 12, 2013 at 3:46 pm / Reply

Dear Richard, your blog is awsome and I do enjoy reading it, thank very much! Still, I have the same questinon as soncu on June 26 (may by I missed something essential). Why not to use:

```
theCelebrities[i]["id"] = function (j) {
return uniqueID + j;
} (i);
```

instead of

```
theCelebrities[i]["id"] = function (j) {
return function () {
return uniqueID + j;
```

```
} ()
} (i);
```

?

Thank you so much for your time!

Richard Bovell (Author)

August 15, 2013 at 2:21 pm / Reply

The reason the code returns a function is because the point of closures is that the inner function has access to the outer function's variables even after the outer function has returned, so you can continue to execute code like this: stalloneID.id(). The demonstration was about closures, which are inner functions, hence the example.

Indeed, you can choose to not return a function and simply return the variable, but in some scenarios, you do need to return a function that you want to call later. The code shows the example of returning a function and calling that function at a later time.

tanyaka

August 15, 2013 at 5:58 pm / Reply

i see. thank you very much!

Pearce

September 28, 2013 at 2:12 pm / Reply

Hey Richard,
I believe you are missing soncu and tanyaka's critique. The second block of code assigns numbers to the id property of elements in the "theCelebrities" array, not functions. This trivializes the problem and makes the use of IIFE's

unnecessary. This is confusing your readers. I think the most instructive solution would be this:

```
function celebrityIDCreator (theCelebrities) {
var i;
var uniqueID = 100;
for (i = 0; i < theCelebrities.length; i++) {
theCelebrities[i]["id"] = function (j) {
return function() { // Return a function, not a number
return uniqueID + j;
}
}(i)
}

return theCelebrities;
}

var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];

var createIdForActionCelebs = celebrityIDCreator (actionCelebs);

var stalloneID = createIdForActionCelebs [0];
console.log(stalloneID.id()); // 100
```

Richard Of Stanley (Author)

October 1, 2013 at 4:06 pm / Reply

I must be missing something because the same code you are suggesting is what I provided in the article above: the first part of section "Closures Gone Awry."

Pearce

October 1, 2013 at 10:43 pm /

The code I'm suggesting has 1 IIFE as opposed to 2 (or 0). Sorry for the lack of formatting, I couldn't figure out how other commenters did the formatted code blocks.

San

May 7, 2014 at 7:01 am / Reply

Hello,
I have also the same confusion.
In the code if I give as follows the result as follows.
console.log(cruiseID.id); // 101
console.log(cruiseID.id()); // TypeError: stalloneID.id is not a function

why it shows this error?
//San

DeVontae Moore

December 11, 2014 at 1:13 pm / Reply

It could you point out your fact about the code returning a function and calling that function later.
I saw it clearly in your IIFE tutorial:
showName = function (name) {console.log(name || "No Name")}) (); // No Name

showName ("Rich"); // Rich
showName (); // No Name

but dont see those same principles here.Thank you for your time

sameer

August 20, 2013 at 3:01 pm / Reply

Thanks for the sexy website

Richard Bovell (Author)

August 21, 2013 at 1:09 pm / Reply

Thanks, Sameer.

I am hopeful the site is sexy and informative at the same time. The saying in English is “brain and brawn” :), which means sexy (or strong for male) and smart at the same time.

I added the explanation for “brain and brawn” in case English is not your first language.

Cosmo Scrivanich

August 21, 2013 at 10:07 am / Reply

For the last solution won’t just be alot easier to use bind?

```
function celebrityIDCreator (theCelebrities) {
var i;
var uniqueID = 100;

for (i = 0; i < theCelebrities.length; i++) {
theCelebrities[i]["id"] = function (j) { // the j parametric variable is the i passed in on invocation of this IIFE
return uniqueID + j; // each iteration of the for loop passes the current value of i into this IIFE and it saves the correct value to the array
}.bind(this, i) // BY adding () at the end of this function, we are executing it immediately and returning just the value of uniqueID + j, instead of returning a function.
}
```

```
return theCelebrities;
}

var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];

var createIdForActionCelebs = celebrityIDCreator (actionCelebs);

var stalloneID = createIdForActionCelebs [0];
console.log(stalloneID.id()); // 100

var cruiseID = createIdForActionCelebs [1];console.log(cruiseID.id()); // 101
```

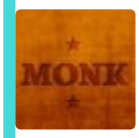
Richard Bovell (Author)

August 21, 2013 at 1:29 pm / Reply

Thanks for the bind () alternative, Cosmo.

This is why JavaScript is cool: you can always find an expressive way to execute your code, and the bind example you illustrated is perfectly fine, if you are interested in just returning the value and not a function.

Note that in some scenarios, as I have noted above in an earlier comment, you may want to return a function, which is more versatile than returning just the value. But your solution is a wonderful addition nonetheless.



Pierre

August 23, 2013 at 3:32 pm / Reply

Hi,

Thanks for this great page. I've been pouring over it and absorbing every line. I think I have it all except the following:

for (i = 0; i < theCelebrities.length; i++) {
theCelebrities[i]["id"] = function (j)

How does the value of i get passed into j if it's never explicitly told they are the same value?

Thanks!
Pierre

Pierre

August 23, 2013 at 3:53 pm / Reply

NM just had a facepalm moment. I was so fixated on that part of the code, I neglected to notice it was being passed in below!

Richard Bovell (Author)

August 26, 2013 at 5:35 pm / Reply

I have never heard the term “facepalm” before, but I take it you have figured it out. Glad to hear 🙂

hiren

September 3, 2013 at 11:55 pm / Reply

Hi Richard,

Great site. Found really helpful.
Thank you so much.

Hiren

Richard Bovell (Author)

September 4, 2013 at 9:37 pm / Reply

You are welcome, Hiren. And thank you.

olivier nguyen

September 10, 2013 at 6:29 pm / Reply

Hi there, this is a great website. The suggestion I would make is that the code examples are hard to read because of the narrow width of the main column. And the indentation doesn't help either.

Looking forward to more posts!

Richard Bovell (Author)

September 11, 2013 at 8:15 pm / Reply

Thanks, Olivier. I will definitely redo the layout within a few weeks.

Ahmed

September 22, 2013 at 1:07 am / Reply

Thanks for the post , its the best I read on Closures

Richard Of Stanley (Author)

September 23, 2013 at 1:02 pm / Reply

Thank you very much for the comment, Ahmed. I am happy to hear that the article has been helpful to you.

Roshan

September 27, 2013 at 11:55 am / Reply

Hello Richard,

This is one of the best explanations that I have read. It seems like this explanation will be understood even by a 6 year old. Remembering the famous Einstein quote.

I have subscribed and will be waiting to see more from this site.

One quick question
console.log(cruiseID.id);
what is this cruiseID?

Richard Of Stanley (Author)

October 1, 2013 at 3:55 pm / Reply

Thanks very much for wonderful compliment, Roshan.

The cruiseID is the variable that will hold the value of Tom Cruise's data. First, here is the object with all the celebrities:

```
1
2  var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0
3
```

Then we set up the array that will hold all the celebrities; we do this by executing the celebrityIDCreator function with the object we created in the line above:

```
1
2  var createIdForActionCelebs = celebrityIDCreator (actionCeleb
3
```

And with those, we then created variables for each celebrity, so the Tom Cruise variable is this one:

```
1
2  var cruiseID = createIdForActionCelebs [1];console.log(cruiseID
3
```

Sneha

September 27, 2013 at 2:35 pm / Reply

Hey Richard,
I am learning Javascript and I really appreciate the efforts u took to explain one of the readers “Roberto” because that really helped me understand clearly.
Thanks.
Sneha.

Danijel

September 29, 2013 at 4:50 am / Reply

Your blog helps me a lot, and reduce time needed to read all other blogs. I find the information on your blog and the way you are presenting the most informative. Just wanted to say thank you!!!

Foysal Mamun

September 30, 2013 at 9:41 am / Reply

really sweet post. Thanks.

Richard Of Stanley (Author)

October 1, 2013 at 4:15 pm / Reply

Thanks to Sneha, Danijel (nice name, btw), and Foysal Mamun for the wonderful compliments. Your comments are very encouraging to me and I really appreciate all of your feedback.

Vish

October 1, 2013 at 6:38 pm / Reply

Hi Richard,
Sorry for the long post. I wanted to explain my scenario as detailed as possible. I was experimenting with your example to see how it would work if I tweak things a bit. I changed it as follows

```
function celebrityIndicator(theCelebrities) {
var i;
var uniqueId = 100;
for (i = 0;i < theCelebrities.length; i++) {
theCelebrities[i]["id"] = function(j) {
return function() {
return uniqueId + j;
}();
}
} // did not pass i as a parameter here

return theCelebrities;
}

var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];

var createIdForActionCelebs = celebrityIndicator(actionCelebs);
var stallone = createIdForActionCelebs[0];
console.log(stallone.id());
```

Instead of passing i as a parameter, I assigned the variable theCelebrities[i]["id"] to the function reference and tried printing the id.

I got NaN as the output. Why would this happen ?

Sriram

October 10, 2013 at 2:27 pm / Reply

Hi Richard

This site is very flexible to learn javascript

good going.

Ankit Agarwal

October 10, 2013 at 6:43 pm / Reply

This is really an awesome site to learn javascript.
I have traversed many other sites for js but i haven't found such a simple and detailed description.

Bookmarked this site !!
Great work.

Richard Of Stanley (Author)

October 11, 2013 at 1:58 pm / Reply

Thank you very much, Ankit.

Ankit Agarwal

October 11, 2013 at 2:08 pm / Reply

Hi Richard,

If you have any js based project then please share i with me, i want to contribute so that i can enhance my skills and can learn.

Thanks
Ankit

Richard Of Stanley (Author)

October 15, 2013 at 8:29 pm / Reply

Ankit, you can check on Github. There are many small, medium, and large projects you can

contribute to.

Ankit Agarwal

October 11, 2013 at 2:09 pm / Reply

Hi Richard,

If you have any javascript based project then please share i with me, i want to contribute so that i can enhance my skills and can learn.

Thanks
Ankit

Alisa

October 11, 2013 at 12:46 am / Reply

Hello,
I'm still struggling with closures but getting um, closer. Where can I find the quiz you mentioned for Feb 7? Muchas gracias, Alisa

Richard Of Stanley (Author)

October 11, 2013 at 1:57 pm / Reply

Alisa, you caught me: I was supposed to post that quiz a while back and I completely forgot. I will think about doing it sometime soon; just for you 🙂

Which part of closures are you struggling with? Post your questions here and I will answer them.

Tyler

November 1, 2013 at 9:51 pm / Reply

Immediately invoking function expressions felt like a greased pig until I read your reply about testing a returned inner function. Once assigned to a variable, logging that variable without parenthesis dumps the function declaration. Logging the variable followed by () invokes the function. And so, assigning a function to a variable and following it with () assigns the function to the variable and runs it.

Ok, I just re-read what I wrote and I'm confused again… Maybe.

Curious how a passed parameter doesn't persist. Perhaps the assignment of the variable using 'var' gives the anonymous function a clean slate.

jeff

November 19, 2013 at 1:48 pm / Reply

Enjoying your site…

I was playing with a couple of the examples. Now I'm confused about the syntax difference of two of the examples. The 1st example (function showName) returns the closure function as follows:

return makeFullName (); // the parens are included

The 3rd example (function celebrityName) returns its closure function WITHOUT the trailing parens as follows:

return lastName; // no parens

The 3rd example's closure function has parameters, the 1st example's does not, otherwise things seem equal. Dropping parens from the 1st or adding them to the 3rd breaks things.

What's going on?

Thanks,
Jeff

Richard Of Stanley (Author)

November 22, 2013 at 3:30 pm / Reply

I see that you are paying attention to each line carefully there, Jeff. 🙂

In the first example, we are returning the result of executing the makeFullName function. We actually didn't need the makeFullName function at all; we could have just add the code right inside the showName function (not inside the makeFullName closure). This was example was just for demonstration of what a closure is hence the reason we added the makeFullName inner function.

But in the second example, we are returning the lastName function BEFORE it is executed, so that we can execute it later. If we return it like this, *return lastName()*, then it would have executed at that point in the code (where it is defined) and it would not have had access to the outer function's variables and parameters later because we couldn't call it later.

So the reason we return only lastName (without the params) is that we want to execute the function later and so that it can access the outer functions's variables.

Bart Kleijngeld

November 28, 2014 at 10:02 am / Reply

I had the exact same question as Jeff, and would like to make sure I understand your explanation. Can I summarize it as follows?

In the first example a closure is defined in the most basic way, the purpose to show what a closure is basically, _but_ it is a useless example and in practice one will always encounter the returning of closures for them to be useful?

Thanks for the great article!

Rakesh Sivan

December 3, 2013 at 12:11 am / Reply

I was running between several websites to get a clear picture about closures and also about the java script's most powerful features in general.Your website has the right stuff with the needed clarity. Thanks 🙂

vimal kumar

December 10, 2013 at 9:39 am / Reply

nice explanation….I liked very much:)

Petr

December 12, 2013 at 7:07 am / Reply

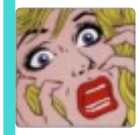
Hi,

firstly thanks for the articles, they are absolutely amazing. I'm reading through all of them and I will recommend them to other people every chance I get!

My question – in the last code block example – why does the inner function return another functions? Isn't that redundant to create a function whose only aim is to return another function? Or am I missing something?

P.S.: It seems that your comment section has the ability to highlight code, but users don't use it and I can't seem to find how to do it either. It'd make the comments section a lot more readable (sexy) though 🙂

Once again, thank you for the great articles!



hausFrauFromHades

December 17, 2013 at 10:24 am / Reply

I am really enjoying your articles, particularly this one and the one on callbacks. Thanks for your work.

JavaScript IS sexy– who knew?

Richard Of Stanley (Author)

January 7, 2014 at 1:01 am / Reply

It sounds like you have experienced JavaScript enlightenment, hausFrauFromHades. That's cool.

hausFrauFromHades is an interesting name, BTW.

Sarvesh Kesharwani

December 17, 2013 at 1:35 pm / Reply

You are just awesome.

Richard Of Stanley (Author)

January 7, 2014 at 12:58 am / Reply

🙂 I wish. But thanks, Sarvesh.

Shadab

January 3, 2014 at 5:36 am / Reply

function lastName (theLastName) {
return nameIntro + firstName + " " + theLastName;
}
return lastName;

in the case above you have written lastname without (),how does js treat this.Is it actually a function call?

Richard Of Stanley (Author)

January 7, 2014 at 12:13 am / Reply

In JavaScript, functions are object, so you can treat them like an object or a variable. When you return a function without the (), you are returning the function definition. So that you can call the function later. For example:

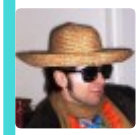
function showMe (){
console.log (“This is Rich”);
}

function showRich () {
return showMe;
}

If we run this line:
showRich ();

We get this result:
function showMe() {
console.log (“This is Rich”);
}

Yep, the actual definition of the showMe function.
But of course it is not very useful just return the showMe function, which is not executing or returning anything itself. The showmen function should execute some action or return something.



Axel

January 4, 2014 at 2:24 am / Reply

Hello Richard,

I think the last example with the nested IIFE´s does not match your intentions (as I suppose them :-).
The problem is, that all IIFE’s are executed immediately, so that ‘theCelebrities[i] [“id”]’ is a value and not a function anymore. The IIFE’s are equal to assigning the value directly: theCelebrities[i][“id”] = uniqueID + j; (as joncu correctly state

What I suppose you wanted is the following:

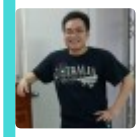
```
...
for (i = 0; i < theCelebrities.length; i++) {
theCelebrities[i]["id"] = function (j) {
return function () {
return uniqueID + j;
}
} (i);
}
...
```

This will create a new scope for the inner IIFE at "construction time" containing the current value of i.

Of course the id then must be called (or better – can be calleg) with parnthesises:

```
...
console.log(stalloneID.id()); // 100
...
console.log(cruiseID.id()); // 101
...
```

By the way, this is the best site I saw up to now explaining the (many) JS-obstacles in an easy and understandable way. Great work!



Binh Thanh Nguyen

January 16, 2014 at 11:03 pm / Reply

Thanks, nice post

Jeremy

January 16, 2014 at 11:48 pm / Reply

Richard,

This article is very good up until the end. Your example of IIFE is pretty bad, since the result, as others have pointed out, is that every celebrity is assigned an ID when celebrityIDCreator is called and you might have just as well written:

```
theCelebrities[i]["id"] = uniqueID + i;
```

Here is a version that uses and IIFE so that theCelebrities.id is still a function but it is not assigned until the id() function is first called. (Then the function is replaced with a new function that always returns the same number.)

```
function celebrityIDCreator (theCelebrities) {
var i;
var uniqueID = 100;
for (i = 0; i < theCelebrities.length; i++) {
(function() {
var thisCelebrity = theCelebrities[i];
thisCelebrity["id"] = function () {
var thisID = uniqueID++;
thisCelebrity["id"] = function() {
return thisID;
}
return thisID;
}
})();
}
return theCelebrities;
}

var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];

var createIdForActionCelebs = celebrityIDCreator (actionCelebs);


 createIdForActionCelebs[2].id(); // 100
createIdForActionCelebs[0].id(); // 101
```

```
createIdForActionCelebs[2].id(); // 100
```

Jack

January 21, 2014 at 2:56 pm / Reply

Hey man, your site is awesome! I have a question though, for your last example about immediately invoked functions, how come you nested functions instead of a simpler:

```
function celebrityIDCreator (celebs) {
var i;
var uniqueID = 100;
for (i = 0; i < celebs.length; i++) {
celebs[i]["id"] = function() {
return uniqueID + i;
} ( ); //INVOKE HERE IMMEDIATE
}
return celebs;
}
```

I'm not just providing a simpler solution because that's not the point of this lesson. But the point was a bit lost on me when I had to deal with more confusing IIFEs.

Could you expound on the point you were making with the last example? Thank you.

shams

March 5, 2014 at 6:35 am / Reply

I don’t understand the need of inner function.

```
theCelebrities[i]["id"] = function (j) { // the j parametric variable is the i passed in on invocation of this IIFE
return function () {
return uniqueID + j; // each iteration of the for loop passes the current valu
```

into this IIFE and it saves the correct value to the array
} () // BY adding () at the end of this function, we are executing it immediately and returning just the value of uniqueID + j, instead of returning a function.
} (i); // immediately invoke the function passing the i variable as a parameter
}

Cant it be simply ?
theCelebrities[i]["id"] = function (j) { // the j parametric variable is the i passed in on invocation of this IIFE
return uniqueID + j;
} (i); // immediately invoke the function passing the i variable as a parameter
}

Yui

March 12, 2014 at 2:11 am / Reply

Greatest post!! I learn a lot!
I think this might be a little shorter and it still works:
http://jsfiddle.net/GGGfW/

Ibrahim Islam

March 17, 2014 at 5:31 am / Reply

So, in the celebrity id where the id is being assigned a function that returns modified id example when is the anonymous function is executed actually?

Ibrahim Islam

March 17, 2014 at 7:08 am / Reply

In the last example, why are we returning a function and then the modified Id again inside? Why can't we just return the modified id?

Ibrahim Islam

March 17, 2014 at 9:15 am / Reply

Regarding the last code, this is making more sense to me:
http://jsfiddle.net/L39jM/

ashish

April 4, 2014 at 6:45 am / Reply

You really made Javascript sexy n awsome. !!! Thanks

Kedarnath G

April 9, 2014 at 6:10 am / Reply

Awesome Post :

Excellent Post on Closure. I didn't think Java script is this much sexy before

Thanks
Kedarnath

islam al khaldi

April 19, 2014 at 5:11 pm / Reply

First of all, thank you very much for your this nice and constructive blog.
Refering to your last example in "Understand JavaScript Closures with Ease", i think the inner IIFE could be removed, because it works fine without it.
I tried the example in Firebug and it works and gives the same result.

```
function celebrityIDCreator (theCelebrities)
{
var i;
var uniqueID = 100;
for (i = 0; i < theCelebrities.length; i++)
{

theCelebrities[i]["id"] = function (j)
{
```

```
return uniqueID + j;
} (i);
}
return theCelebrities;
}
var actionCelebs = [{name:"Stallone", id:0}, {name:"Cruise", id:0}, {name:"Willis", id:0}];
var createIdForActionCelebs = celebrityIDCreator (actionCelebs);
var stalloneID = createIdForActionCelebs [0];
console.log(stalloneID.id); // 100
var cruiseID = createIdForActionCelebs [1];console.log(cruiseID.id); // 101
```

Thanks 🙂

bhanu

April 24, 2014 at 8:25 am / Reply

Thank you
i had got a clear picture on closures

wdc

June 15, 2014 at 11:38 am / Reply

Call me old school (go ahead, I don't mind, I've been programming since 1974 :), but I honestly don't see why closure is a good thing. IMO it only allows for obscure, unreliable code. Even in what the first couple of side effect examples, #1 and #2, shows us, you're telling me that (in #1) the value of 'Michael' hangs around in memory, from a previous call to the celebrityName function, so that the subsequent call to the lastName function via the mjName returned function utilizes 'Michael' at that time? Why is something like this a good thing? Doesn't example #2 show why this isn't a good thing? I would rather have/write software that doesn't allow stuff like this to happen. Are you telling me that there is a case in which closure HAS to be used, because there is no way to provide a certain functionality by writing functions (most likely by not nesting functions) that does NOT utilize closure in that kind of way? If it IS true that I can write code that

provides a certain functionality by not nesting functions and not utilizing closure, in every case where I might see closure being utilized, then I would contend that code that performs that functionality without closure will always be more understandable and reliable than code that utilizes closure, even if the code not using closure takes more lines of code and isn't regarded as being as "cool" or concise. I would gladly trade more lines of code for better readability/understandability and better reliability/maintainability. If you could provide an example of code that shows the utilization of closure in a way in which the same functionality of that code CANNOT be performed with code NOT utilizing closure, then maybe I would be more open to the technique. It seems to me that sometimes languages working in such a way as this is more because of accident than by original intentional design, and then people start using it just because they can write supposedly "elegant" code that no one else can figure out how it works (at least not without a lot of effort).

jwalker

July 25, 2014 at 3:14 pm / Reply

Awesome man! Exactly what I was thinking. Writing code like this is just confusing and totally unnecessary.

John

June 26, 2014 at 2:50 pm / Reply

Hi Richard,

Thanks for writing such a helpful JS blog! I'm having trouble generally understanding when closures should be used. Why would we use a closure to create IDs in the celebrity example? Couldn't we just do the following with the same effect?

```
...
function celebrityIDCreator (theCelebrities) {
var i;
var uniqueID = 100;
```

```
for (i = 0; i < theCelebrities.length; i++) {
theCelebrities[i]["id"] = uniqueID + i;
}
return theCelebrities;
}
```

Kaustubh

July 21, 2014 at 1:56 pm / Reply

Thanks Richard for this useful blog.
function closure is the true heart of javascript and you helped understanding it better

wdc

July 28, 2014 at 12:02 pm / Reply

I'm still trying to understand why closure is a "good thing".

Let me take another short stab at justifying it as a language feature and coding technique to use.

In one of your comment responses, you say:

'So the reason we return only lastName (without the params) is that we want to execute the function later and so that it can access the outer functions's variables.'

That statement (particularly the word "later") made me think of an aspect of closure that I can actually accept as being useful. Could one say that the real, underlying usefulness of javascript closure (at least one useful thing, anyway) is that it allows for a kind of data persistence without having to use an actual datastore? IOW, that it is a kind of "in-memory" datastore, with respect to the outer function's variables and parameter values?

Basically (stated another way), that closure allows "setting up" a function (the inner function) for execution sometime LATER in time, knowing that the vali

the outer function's variables and parameter values will still be there (have been "stored" in memory for use later)? WITHOUT having to store those values in a file on disk, or in an actual database, etc.

This is what I have found lacking in every discussion/explanation of javascript closure – underlying, conceptual REASONS why closure is valuable, not just how the code works in a mechanical sense (and with "mickey mouse" examples that don't even seem useful in a real-world sense on top of it, which compounds the problem of old-timers like me being able to accept it as a good thing (if you read my previous post)).

ASSUMING that what I am observing in this post is correct – that at least one of the main, beneficial, underlying reasons for closure is that it allows "setting up" a function for later use, and "remembering", or persisting some data values in memory for use at that later time, without having to use a formal datastore… Can you verify this particular take on it, or am I off-base about it?

If so, then I can grudgingly see how javascript closures might be useful for that particular scenario 🙂

BECAUSE (to expand on the conceptual reasoning) from a software architecture / OOP viewpoint, it allows a function (the inner function) to be polymorphic in the sense that it will have different behavior when it is invoked, based on the values that were passed into the outer function at some previous time.

THAT I can understand and see as being useful.

bayloox

September 19, 2014 at 10:16 pm / Reply

Thanks very much for the post.

The "sexy" for JS is catching on man. If it becomes global all the credit will be yours!

Kevin

September 22, 2014 at 8:22 am / Reply

Hey Richard,

First of all I would like to thank you for this website because it has really aroused my interest in JavaScript so much so that I really wanna dive more into it.

Now coming to the doubt that I had; in the “Closures Rules and Side Effects” section no. 2 you made a statement that said,

“Closures get more interesting when the value of the outer function’s variable changes before the closure is called.”

My only doubt in the above scenario is that the example is simple to understand but can you please elaborate more on this topic since the statement is not clear enough from the example that you have given.

iJocuri

October 7, 2014 at 4:15 am / Reply

Hey there, thank you once again, you have been a great resource. I have been finding some great resources here.
Well article.. i just love Javascript!

maaz

October 10, 2014 at 7:12 am / Reply

aslamualykum
i am proud of you
because you solved lot of my problems.
thank you so much sir .

DeVontae Moore

October 17, 2014 at 8:42 pm / Reply

Im not familiar with the following syntax and its purpose:
theCelebrities[i]["id"]

ARSHAD

November 27, 2014 at 1:54 am / Reply

WOW!!! SUCH A NICE WAY TO EXPLAIN THE CORE CONCEPTS.. JAVASCRIPT IS REALLY GETTING SEXIER FOR ME NOW… 🙂

charles ross

November 28, 2014 at 10:06 am / Reply

It's a great post. But there's not so much ease. The rule 1 example never could make it into my brain, so I refactored it to make as clear as possible, with descriptive variable names and output harness included. Then the d-oh lamp lit up.

```
function celebrityNameMunger(firstName) {
var nameIntro = "This celebrity is ";
// this inner function has access to the outer function's variables, including
// the parameter
function lastNameAppender(theLastName) {
return nameIntro + firstName + " " + theLastName;
}
return lastNameAppender;
}

console.log (" celebrityNameMunger is global: " + celebrityNameMunger );
console.log (" ...and now the closure lastNameAppender gets assigned to
mjNameBuilder..." );
var mjNameBuilder = celebrityNameMunger("Michael");

console.log (" >>>>>>>>>mjNameBuilder >>>>>>>: " + mjNameBuilder );
// At this juncture, the celebrityNameMunger outer function has returned.?
```

// The closure (lastNameAppender) is called here after the outer function has returned above
// Yet, the closure still has access to the outer function's variables and parameters

console.log ("mjNameBuilder('Jackson'): " + mjNameBuilder("Jackson")); // This celebrity is Michael Jackson

We loggin', bro

Sakthivel

December 11, 2014 at 2:55 pm / Reply

Hi,

Excellent Stuff!! Appreciate your work. Nice to learn.

Anh Tran

January 28, 2015 at 1:36 am / Reply

Closure is one of the most headache issue in Javascript and I usually find hard to explain that to my developers. This post is very helpful and easy to understand. It helps me a lot for training devs of my team. Thanks for sharing.

Tarak

January 28, 2015 at 9:21 am / Reply

Really sexy stuff i found……

Mark

February 24, 2015 at 9:09 pm / Reply

In your last example, doesn't

```
function celebrityIDCreator(theCelebrities) {
var i; var uniqueID=100;
for (i = 0; i <theCelebrities.length; i++){
theCelebrities[i]["id"] = function (){
return uniqueID + i; }();
}
return theCelebrities;
}
```

create the same effect with one less level of convolution? although maybe I'm just missing the point you were trying to make.

Samir

March 17, 2015 at 3:38 am / Reply

Dude you are awesome and your website is sexy. Keep doing sexy work 🙂

Anthony

May 9, 2015 at 11:30 pm / Reply

Hi Richard,

Thanks a lot for this post. It's great, I've learned a lot from your site. FYI, I'm new to Javascript.

But I was wondering if the following paragraph could be made a little clearer, (right after the "Closures gone Awry" section).

"In the preceding example, by the time the anonymous functions are called, the value of i is 3 (the length of the array and then it increments). The number 3 was added to the uniqueID to create 103 for ALL the celebritiesID. So every position in the returned array get id = 103, instead of the intended 100, 101, 102."

changed to...

"In the preceding example, AFTER THE LAST CALL OF THE ANONYMOUS FUNCTION (singular) IN THE FOR LOOP, the value of i is 3 (the length of the array and then it increments). THE UPDATED VALUE OF i=3 IS THEN ALSO APPLIED TO THE PREVIOUS CALLS OF THIS ANONYMOUS FUNCTION, CREATING 103 FOR ALL celebritiesID. So every position in the returned array get id = 103, instead of the intended 100, 101, 102."

I read this section like 4-5 times and was still confused, but I think I finally understand.

Thanks again! Cheers!

-Anthony

Murari

May 20, 2015 at 4:16 am / Reply

Hi,

```
for (i = 0; i < theCelebrities.length; i++) {
theCelebrities[i]["id"] = function () {
return uniqueID + i;
}
}
```

Why do you say that value of i is 3 by the time annonymous functions are called? Isn't it that anonymous functions are being called with every for iteration?

Kreg

July 22, 2015 at 10:30 am / Reply

I could be wrong (as I am a total noob at js), but I think it's because the
anonymous functions are being called _after_ the outer function has gone out of

scope, at which time i has reached the value 3 and terminated the for loop.
Now, because it was a reference to an anonymous function being stored in the
array, the function hasn't been called yet, but when it is, it is referencing
the value of i at the time of call, which will yeild 3, the value that terminated the loop; I hope this helps (and I certianly hope it's correct lest
I give bad information).

santhosh

June 5, 2015 at 5:10 am / Reply

It's one of the best explination abt closure thks

Nikhil Fadnis

June 6, 2015 at 2:37 pm / Reply

Dude you are awesome.

Chalam CH

June 7, 2015 at 1:34 pm / Reply

Hi Richard,
It is the best explanation on closures. The difficulty in understaning is due to our habitual thinking.
1. In Closures gone awry section, in each iteration of the loop, what is assigned is NOT a value BUT a function. The same (anonymous) function is assigned(saved) but NOT executed. When we called the saved function later, the function has latest values.
2. The debate on whether closures are good or bad is another story.
Thanks a lot for the post.

Ananth

June 24, 2015 at 7:24 am / Reply

Example for closure holding reference is not correct. Because Number is immutable data type. If you change the Number then its reference will also be changed.
Reason because it has 3 is, thats what the value of its outer function's variable when the closure is called.
Closure is simply a function defined inside an object anonymously. We all know that Function is an Object, closure will have access to the variables of the object(ie., function) it defined.

Jessie

July 5, 2015 at 2:35 pm / Reply

I agree. Tried the same thing without the second inner function and it worked fine.

Kartik

July 6, 2015 at 10:34 pm / Reply

Hi Richard,
In the IIFE i guess this would suffice

```
function myClass(myList) {
var id=100;
for(var i=0;i<3;i++){
myList[i]["id"] = function(){
return id+i;
}();
}
return myList
}
var myObj_1 = [{"name":"kartik", "id":"0"},{"name":"Varun", "id":"0"},{"name":"Sid", "id":"0"}];
```

```
var myObj_2 = new myClass(myObj_1);
console.log(myObj_2[2].id);
```

There is no need to write a return function which will return Uniqueid+j, Using just IIFE without paramters (value of i) can be done. Refer the code above.

Shaik Amesh

July 15, 2015 at 4:20 am / Reply

Awesome Tutorial dude

Shabnam

July 15, 2015 at 2:39 pm / Reply

Came across this wonderful site!

JavaScript is sexier now 😉

Jason

July 26, 2015 at 8:09 am / Reply

Thank you for the great article, Richard.

I'm not sure if you're aware, but Hack Reactor is referencing this article for potential students who are trying to learn JavaScript, so I figured I would make a few suggestions…

First, in your second point, both the name of your outer function AND the variable inside it is the same (i.e. CelebrityID). This is confusing, and if I'm not mistaken, it produces an error. Additionally, in that same section, you make reference to a "changeTheID function" (which doesn't exist). I believe you meant to reference the "setID" function.

And finally, regarding section three, I don't quite understand why an IIFE inside of an IIFE is being used here (along with another parameter). Wouldn't the

following code suffice?

theCelebrities[i]["id"] = function () { return uniqueID + i; } ();

Or maybe I'm missing something? Either way, thanks again!

Jared Hensley

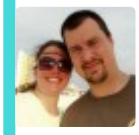
July 26, 2015 at 8:54 pm / Reply

I am a bit confused by this Richard. I know you are busy but if you have a moment, I'd appreciate some clarification. Thank you!

Jared Hensley

July 26, 2015 at 8:55 pm / Reply

I am a bit confused by this Richard. I know you are busy but if you have a moment, I'd appreciate some clarification. Thank you! whoops, double post but trying to get notifications 🙂 work around.



kmcguire

July 29, 2015 at 2:20 pm / Reply

You can also create a new variable instance and that will be referenced, not the original memory location.

```
for (var x = 0; x < 10; ++x) {
var newvariableinstance = x;
store(function () {
alert(newvariableinstance);
});
}
```

Yogesh

August 21, 2015 at 6:46 am / Reply

Am server side developer. can you please tell me what is the best way to debug J query and Java script.

1) console.log
2) Alert

please suggest.

Chella

August 25, 2015 at 6:46 am / Reply

OMG… Finally I understood the Closure… Thanks for make it simple…

Gonzalo

August 27, 2015 at 10:57 am / Reply

Thank you for such a great explanation. I had seen some articles explaining closures but I wasn't able to wrap my head around it until I read this comment. Perhaps you might consider adding it to the blogpost (maybe as a side note or something). It truly is a great explanation 🙂 Thanks again!

Joe Miller

September 4, 2015 at 10:25 pm / Reply

Thank you so much for this reply! I have been hung up on this exact example since I read it in eloquent JS and the explanation is SIMPLE, I just never got that the outer function returned the inner one. Thank you sexyJS.

Raghavendra Karteek

September 13, 2015 at 3:15 am / Reply

Please provide angularjs examples also

Sree

September 22, 2015 at 1:26 am / Reply

Really a very good website. Thanks a lot

Ozzy

September 25, 2015 at 2:42 pm / Reply

This is by far the best and simplest explanation of closures i have read. Simple and to the point. Awesome job.

Ashwani

September 27, 2015 at 12:41 am / Reply

This is incredibly helpful.. found about this website couple of days ago and couldn't get my eyes of it.. Thanks for the simplicity while explaining such complex concepts.

deejsis

September 27, 2015 at 11:33 am / Reply

But then you have changed the call itself.
Before: var stalloneID = createIdForActionCelebs [0];console.log(stalloneID.id()); // 103
After your workaround with the IFFE: var stalloneID = createIdForActionCelebs [0]; console.log(stalloneID.id); // 100
Notice you are not longer calling id() – you are simply getting the value back.

Is it possible to still keep the original call ie stalloneID.id() and still get the correct result (100) ?

Vinod

October 5, 2015 at 8:23 am / Reply

Great Post!
I have already used Immediately Invoked Function Expression (IIFE) but I don't know the actual use of IIFE.
Also I understood closures first time by the examples u explained.

Jatin Bansal

October 14, 2015 at 7:34 am / Reply

Some days before, I'd read about "Scope Chain" in detail. But now I was not able to find that page on this site. I urgently need to read that. Plz somebody can help me.

raj

October 19, 2015 at 3:19 am / Reply

First of all, thanks for this article. I greatly appreciate the time you took to work on the article.

gero

October 30, 2015 at 4:51 am / Reply

Hi! Many thanks for this!
gero (from Paris)

Johnny Chan

October 30, 2015 at 9:38 am / Reply

I was also wondering how that `x = function(j) {...} (i)` part work. This was a new type of syntax to me.

So I came up with two "equivalent" examples that have helped "convincing" me that this syntax works:

Example 1:

```
var i = 1;
console.log( function (j) {return j+100} (i) ) ;
// return 101
```

Example 2:

```
function testIt(j) {return j + 100};
var i = 1;
console.log( testIt(i) );
// returns 101 also
```

i.e. the syntax…

`testIt(i)`

is equivalent to…

`function (j) {return j+100} (i)`

I'm still a newbie and I hope this makes sense? 🙂

Johnny
@jAtlas7

gero

October 30, 2015 at 10:13 am / Reply

Hi again,
Is there any difference between the IIFE solution:
– theCelebrities[i]["id"] = function (j) { return function () {return uniqueID + j;} () } (i);

and a simpler Immediately Invoqued Function:
– theCelebrities[i]["id"] = return function () {return uniqueID + j;} () ;

or a basic:
– theCelebrities[i]["id"] = function (j) { return function () {return uniqueID + j;} () } (i);

Any comments?

Imbrod

November 6, 2015 at 3:40 pm / Reply

I agree. And probably DeathShadow (that I agree in most posts) from Digital Point Forums would too.

Showket

November 13, 2015 at 5:52 am / Reply

Really amazing stuff in a simple way. This is the contribution that makes javascript really Sexy. hats off to the stuff writer.

gel

November 13, 2015 at 11:28 am / Reply

My reuseable solution:

```
var starIDMaker = function(){
var uniqueID = 100;
return function(arr){
for(var i = 0; i < arr.length; i++){
arr[i]["id"] = function(id){
return ++uniqueID;
}(uniqueID)
}
```

```
}
}()
```

Rob

November 18, 2015 at 11:33 am / Reply

The fact that a contained function may access a variable in the closure, and that variable may change before the contained function executes is not a bug as you keep saying. It is intended functionality. You just have to know how closures work, which the rest of your article demonstrates very well.

treefish

November 19, 2015 at 11:49 pm / Reply

? Your first and last approaches are identical.

Rahul

November 22, 2015 at 4:17 am / Reply

Hello, I've just started learning Javascript. Your tutorial is so good that it clears my all doubts regarding closures. Thank you so much for this article.

Sravani

December 4, 2015 at 7:43 am / Reply

Hi,

Excellent Stuff.Very helpful in understanding .

I am pretty much confused with the "Closures gone awry" and IIFE .Would you please explain me further or let me know the reference books to understand clearly.

Thanks and Regards,
Sravani.

firos

December 14, 2015 at 2:52 am / Reply

Thanks very much. I am trying for a long time to understand closures concept but always end up with some confusion. Best example. Well explained.

Tay

December 21, 2015 at 12:05 pm / Reply

Thanks for that

Aurel

December 26, 2015 at 3:12 am / Reply

In the first points: there is a sintax error
mjName (“Jackson”);
Should be:
mjName.lastName (“Jackson”);

Is this correct?

Hari

January 4, 2016 at 8:59 pm / Reply

Very nice post, Thanks for posting. I bookmarked this link in my favourites.

Parish

January 13, 2016 at 1:55 am / Reply

In the last code example –

```
theCelebrities[i]["id"] = function (j) {
return function () {
return uniqueID + j;
}()
}(i);
```

why i need – return function() { return uniqueID + j; } when the code below is working fine.

```
theCelebrities[i]["id"] = function (j) {
return uniqueID + j;
}(i);
```

Çağlar ORHAN

January 16, 2016 at 3:49 pm / Reply

You should write a book about js, if you didn't write one yet. Clearly explained, keeped light to the dark corners which beginners couldn't see (also afraid of). You are awsome.
Thank you so much.

Marlo Dela Torre

January 27, 2016 at 7:53 am / Reply

such a great tutorial.. bookmarked your site and reading it when I have spare time.. thanks!

ugison

February 2, 2016 at 6:16 am / Reply

Wonderful solution, this IIFE! Love it..

mwas

February 6, 2016 at 1:50 am / Reply

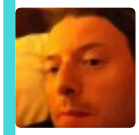
please help solve a tabular array to change the cars in sequence ona platform ‘demo’ when this displays make the image hover on the other cells.thanx var cars=[]; cars[0]= ida; cars up to INDEX[11] and their ids. when i use a closer it only shows the last image bt dont loop

Arifur Rahman

February 9, 2016 at 11:53 pm / Reply

To solve “3. Closures gone awry” we can just change line #6,7, &8. No need to write and return another anonymous function. instead of that we can just do this,

line:6. theCelebrities[i][“id”] = function (j) {
line:7. return uniqueID + j;
line:8. }(i)



Charles Robertson

February 10, 2016 at 6:33 pm / Reply

This is quite simply one of the finest JavaScript tutorials out there. In fact, I was asked about JavaScript closures in a phone interview today. I wish I had read this first. Doh! Can I ask what a parametric variable is?

= function (j) {

Ray

February 13, 2016 at 7:04 am / Reply

Thanks for a nicely laid out explanation. Years old and still helpful.
Try this:

function celebrityName (celebObj) {
var nameIntro = nameIntro?”The other celeb is “:”This celebrity is “;
var celebObj2 = {

```
name:celebObj.name
};

setTimeout((function lastName () {
console.log(nameIntro + this.name);
}).bind(celebObj),1000);

setTimeout((function lastName () {
console.log("Again, " + this.name);
}).bind(celebObj2),1100);

setTimeout((function lastName (celeb2) {
console.log("All 4 objects: " + this.name + " " + celeb2.name +
" " + celebObj.name + " " + celebObj2.name);
}).bind(celebObj,celebObj2),1200);
}
celebrityName ({name:"Michael"});
celebrityName ({name:"Peter"});
```

The result was as expected. The javascript engine could make copies of the functions and objects to maintain the scope. I'm wondering if persistent scope is implemented by referencing the objects or if copies are made containing only properties that have been used in declarations yet to be executed; and if it makes a difference to how callbacks are handled. It seems that it probably doesn't matter; the scope remains as though the outer function is still running. It starts to get confusing in things like angular.js

Bright Nkrumah

February 15, 2016 at 1:35 pm / Reply

Very nice tutorial

Anmol Dani

February 16, 2016 at 3:29 am / Reply

Your tutorials are indeed sexy! 😀

Jhony Grillet

February 26, 2016 at 7:33 pm / Reply

Definitively, i must write a (lot of) post(s) about this page… OMG, with this kind of posts, Javascript seems so SEXY!!!

Anusha

February 29, 2016 at 7:34 am / Reply

Thanks for the post. I am not clear with this line
“Note that the inner function cannot call the outer function’s arguments object, however, even though it can call the outer function’s parameters directly.”

Roshni

March 4, 2016 at 5:01 am / Reply

Great post!

sheik

March 10, 2016 at 4:35 am / Reply

Its very useful adnd good understandable explanation.

Bah Djibril

March 16, 2016 at 12:07 am / Reply

I know namy already replied but here is what I did. (mjName is still a function that has to be executed)

var fullName = mjName(“Miller”);

$(‘#name’).html(fullName);

Harish

March 16, 2016 at 6:41 am / Reply

when i cpoy and paste code after that i have always got illegal character error , why it is ??

Nagaraju

March 24, 2016 at 8:44 am / Reply

Very good example explained clearly .. Thanks for the article.

Nagaraju

March 24, 2016 at 8:45 am / Reply

Very good examples explained clearly .. Thanks for the article.

Nagaraju

March 24, 2016 at 9:14 am / Reply

I really did have confusions on javascript closure .. this is awesome and supreb .. very good and clear explanation ...I really fell in love with closure and javascript (sexy script).

But i tried some other approach that also works, please let me know is this correct and also rectify if i am wrong with the approach.

```
function celebrityIdCreator(celebList){
var uniqueID =100;
return function(){// returning inner function
for(var i=0;i<celebList.length;i++){
celebList[i]["id"] = uniqueID +i;

}
```

```
return celebList; // updated celebList of the inner function returned.
}

}

var actionCelebList = [{name:'Arnold',id:0},{name:'The Rock',id:0},
{name:'Stallon',id:0}];
var celebIdFn = celebrityIdCreator(actionCelebList); // celebIdFn has inner
function (closure)..
newActionCelebList= celebIdFn(); // By calling celebIdFn .. we get the updated
array ..
var theRock = newActionCelebList[1];
console.log(theRock.name+" :"+theRock.id);
```

Umesh

March 29, 2016 at 6:48 am / Reply

Nice explanation……really java script is sexy 🙂

Luke

April 5, 2016 at 9:45 pm / Reply

Awesome explanation Richard! When I initially tried to learn closures they seemed so intensely hard to learn. This makes it crystal clear. Thank you!

Awesome

April 7, 2016 at 4:24 am / Reply

Still awesome in 2016

Satish

April 7, 2016 at 6:36 am / Reply

I was always wondering what is this closure, prototype and brackets at the end of functions and they all are explained neatly and in a very simple way. Thanks a lot this great article.

thanhdong

April 9, 2016 at 4:32 pm / Reply

thank you very much. 😀

Sirisha

April 10, 2016 at 3:27 pm / Reply

Some real time JavaScript Closure example
****
Practical closures

Let's consider their practical implications. A closure lets you associate some data (the environment) with a function that operates on that data. This has obvious parallels to object oriented programming, where objects allow us to associate some data (the object's properties) with one or more methods.

Consequently, you can use a closure anywhere that you might normally use an object with only a single method.

Situations where you might want to do this are particularly common on the web. Much of the code we write in web JavaScript is event-based — we define some behavior, then attach it to an event that is triggered by the user (such as a click or a keypress). Our code is generally attached as a callback: a single function which is executed in response to the event.

Here's a practical example: suppose we wish to add some buttons to a page that adjust the text size. One way of doing this is to specify the font-size of the body element in pixels, then set the size of the other elements on the page (such as headers) using the relative em unit:
body {
font-family: Helvetica, Arial, sans-serif;

font-size: 12px;
}

h1 {
font-size: 1.5em;
}

h2 {
font-size: 1.2em;
}123456789101112

Our interactive text size buttons can change the font-size property of the body element, and the adjustments will be picked up by other elements on the page thanks to the relative units.

Here's the JavaScript:
function makeSizer(size) {
return function() {
document.body.style.fontSize = size + 'px';
};
}

var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);123456789

size12, size14, and size16 are now functions which will resize the body text to 12, 14, and 16 pixels, respectively. We can attach them to buttons (in this case links) as follows:
document.getElementById('size-12').onclick = size12;
document.getElementById('size-14').onclick = size14;
document.getElementById('size-16').onclick = size16;123
12
14
16123

sunil

April 21, 2016 at 3:12 pm / Reply

Great Post. I regularly visit your website to sharpen my skill in JS. Keep up the good work!

Vijay

April 23, 2016 at 12:43 am / Reply

You can also refer to this blog which explains the concepts of functions and closures http://www.applitter.com/introduction-to-javascript-functionsvariablesscopes-and-closures/

Luong

April 26, 2016 at 10:33 pm / Reply

This is very clear. You help me to dig deeply in JS. Thank you.

Bob Robertson

April 30, 2016 at 12:43 am / Reply

Great post, I learned a lot. Thanks.

AJ

May 2, 2016 at 9:17 pm / Reply

Your articles are so easy to understand and make learning advanced JS very very easy. You should start a JavaScript University. I have not seen such great articles anywhere else. Thank you very much.

Ganesan Natarajan

October 14, 2016 at 4:49 pm / Reply

celebrityName("Michael")("Jackson") // will return The celebrity is Michael Jackson. right?

pavithran

October 27, 2016 at 6:36 pm / Reply

The IIFE caught me by surprise. Do you have an article on this alone? I am trying to wrap my head around this concept.

Prabhat

November 13, 2016 at 10:32 pm / Reply

Closures have access to the outer function's variable even after the outer function returns….
I don't find this point correct. The lastname function is called before the the outer function returns.
I think the sequence is:
celebrityName function called
lastName function called
lastName function returns
celebrityName function returns.. I am applying the concepts of assembly language programming.

Cenob

June 1, 2014 at 6:12 am / Reply

The short answer:
remove the "()" that comes just before the comment starting in "// BY adding () at the end of this…"

By the way, the solution above can be deduced by reading the entirety of that comment, which is:
// BY adding () at the end of this function, we are executing it immediately and returning just the value of uniqueID + j, instead of returning a function.

Note that if you copy-pasted the IIFE code and you make the change noted above, you'll also need to change the console.log lines to reference the 'id' attribute as a function ".id()" instead of as a value ".id".

But wait, why was this question even asked (and the ones by soncu, tanyaka, and Pearce)?
Given that the article was written to explain closures, it doesn't make sense to return values instead of functions in the IIFE example code.

I'm just getting started on Javascript, so please check my facts here but I don't think the code using IIFEs is a closure since no references to celebrityIDCreator's local variables persist after the function returns, and as such there's no need for IIFEs.

I'm pretty sure the IIFE example was originally written to return functions, but that it was changed in conjunction with the note on May 27th, 2013 where Richard wrote "I just updated the code in the article to store numbers in the returned array, instead of functions."

Please change it back.

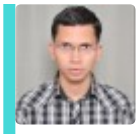
The change reduced the celebrityIDCreator function from a closure to a plain function, and as such removed the need for IIFEs, and also left the "closures gone awry" problem unsolved.

Thank you for writing this up Richard, it was very helpful and I'm sure it will continue to help people after me, which is why I bothered with this big post.

Shawn K.

July 19, 2014 at 2:01 am / Reply

i is passed in at the end of the function like this (i). Make sure you're reading the entire function. I also had a few minutes of not understanding how j got the value of i, but it does indeed get passed in.



Deepak

September 5, 2014 at 11:49 pm / Reply

Yes , I have also the same doubt. Why to write another IIFE inside an IIFE , as inside the first IIFE we can directly return the value.

Nick

September 23, 2014 at 11:18 pm / Reply

I've done the same thing.

Trackbacks for this post

1. 16 JavaScript Concepts You Must Know Well | JavaScript is Sexy
2. JavaScript Variable Scope and Hoisting Explained | 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. Learn Everything About JavaScript Callback Functions | JavaScript is Sexy
7. Closures. Definición y aspectos básicos. ← Javascript docs
8. JavaScript Closures | Geek Girl Arises
9. Simple Yet Powerful JavaScript Tips | Hackya.com
10. Apply, Call, and Bind Methods are Essential for JavaScript Professionals | JavaScript is Sexy
11. How to: How do JavaScript closures work? | SevenNet
12. How to: How do JavaScript closures work? | Technical information for you
13. JavaScript Closures | Pradeep Reddy Kamatham
14. 「译」不侧漏精通 Node.js – 千月殿 -
15. Solution: How do JavaScript closures work? #dev #it #computers | Good Answer
16. Як розмістити на сайті мапу від Google | Нотатки PHP розробника
17. PHP یا Node.js مساله این است ؟!
18. AngularJS Pattern: Handling Data From AJAX USING Javascript Objects: Part II | MO MOADELI
19. Week 1: What I Learned in JavaScript This Week | plane whimsy
20. ChadCollins.com › Understand JavaScript Closures With Ease
21. Learning Javascript I |

22. JavaScript Closure | J.-H. Kwon
23. WDI: The Final Weeks
24. JavaScript: escopo e contexto - Álvaro Guimarães
25. Javascript – Closure | Majesty
26. CallBack Function | Latest Programming
27. Preparing Before Hack Reactor Begins | bash $ cat bitchblog
28. 理解和使用 JavaScript 中的回调函数
29. Hack Reactor's Technical Interview | FunStackDeveloper.com
30. Hack Reactor Interview | Daniel Lin
31. Commonly Asked JavaScript Interview Questions and Answers | UI Tricks
32. Day 17-28 Jan 6-7 | Mandy's road to code
33. Day 21 -23 , (Jan 10 – 12) | Mandy's road to code
34. javascript - why do we need to pass a new variable in closures - javascript
35. Hack Reactor Remote-Getting Here | elizabeth sciortino
36. javascript closure | His story and technical story
37. JavaScript Closures | trulyoutrageous1984
38. The Ultimate JavaScript Closures Resource – Paolo Di Stefano

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