

Self-Executing Anonymous Functions or How to Write Clean Javascript

Feb 20th 2014 · [Permalink](#)

Let's talk about anonymous self-executing functions. Go ahead, say that out loud and I bet the kids next to you know that you're big time now. There is a wealth of information that we can learn about in regards to anonymous functions in Javascript. If you're a Javascript pro, you probably already know what these are. To be honest, I don't really care about most of those. I have been using anonymous functions primarily to make my Javascript more readable. Let's talk a bit about

ES BUENO

Noah Stokes

LINKS

[Follow me on Twitter](#)

[Subscribe to my Blog](#)

[Ask me a question](#)

SPONSORS

BLOGROLL

[Leslie Camacho](#)

[Jonathan Christopher](#)

[Phil Coffman](#)

[Harold Emsheimer](#)

[Jonnie Hallman](#)

[Naz Hamid](#)

[Rogie King](#)

[Dan Mall](#)

[Sean Sperte](#)

[Garrett St. John](#)

[Sean McCabe](#)

anonymous functions, making them self-executing and then walk through some examples. We'll keep it simple, you know, for the kids.

What is an anonymous function? Easy. It's a function without a name. NEXT!

This is an example of a function:

```
function helloWorld() {  
    alert('Hello World');  
}
```

This is an example of an anonymous function:

```
var helloWorld = function() {  
    alert('Hello World');  
}
```

Whoa, wait a minute, you assigned the function to a variable? Yes, yes I did. Don't worry about that now, the variable would just be a way of calling the function. What's important here is that the function in the second example has no name associated with it. That makes it an anonymous function. Cool, right? That's cool and all, but you said self-executing. Don't hold back on me. Right.

Let's look at another anonymous function:

```
(function() {  
    alert('Hello World');  
})
```

Looks like an anonymous function, right? It has no name, so yes, it's anonymous.

Now let's write it again adding three extra characters to the end:

```
(function() {  
    alert('Hello World');  
})();
```

You see those last two parentheses? Those are the self-executing part. They cause everything in the preceding parentheses to be executed immediately. Stop what you're doing and go ahead and raise your voice now. Your speech can go something like this. "Yada yada yada... self-executing anonymous functions... yada yada yada... show me the real thing."


No, shhhh. You're welcome.

Actually, this is really unimpressive. Let's take it a bit further. Let's talk about self-executing anonymous functions inside and outside of the browser. Go ahead, say it out loud. It has a nice ring to it, doesn't it. That's probably not even the correct technical term for it, but like I said, I don't re

care. Let's just dive right in and then ask forgiveness later.

Let's start with an example:

```
(function() {  
    var Person = {  
        sayHello: function() {  
            alert('Hello!');  
        }  
    }  
})();
```




Here we have a self-executing anonymous function with an object inside of it that contains its own anonymous function. I just blew your mind. I Let's look at it line by line. The first and last what we saw before, an anonymous self-executing function. Cool, looks familiar. Inside, we have a variable called Person that is equal to some curly brackets and another anonymous function with a name sayHello, but it's written in a way that we haven't seen before so it's causing my blood pressure to rise. Stay cool man. We got this. The Person variable is what's called an object literal in Javascript. We know this because it is equal to a couple of curly brackets. That's how some cool Javascript land decided objects (literal objects, know, let's not get into it right now)) should be written. The sayHello anonymous function is a method within the Person object. That's all a lot of fancy talk reserved for the dudes with tape on the

glasses. I come from the school of learn by ,
so let's just jump into some doing and I think
will see things start to come together.

Right now, if we ran the code above, nothing
happen. Yes, it is self-executing, but nothing
telling that 'Hello World' alert to run. It's there
waiting for us like a potential partner on a blind
date... waiting... nervously checking their phone
and dodging stares from other couples. Let's
our example and make it pop that 'Hello World'
alert right away.


```
(function() {  
    var Person = {  
        sayHello: function() {  
            alert('Hello World');  
        }  
    }  
  
    Person.sayHello();  
})();
```



Now, if we run this, wham-bam-thank-you-m
On line 9 we added a line of code that tells
Person object to run the sayHello method a
because this is inside the self-executing
anonymous function (ladies), we should see
'Hello World' alert pop up. Lick your thumb &
pinky and stroke back those eyebrows. Show
has arrived.

Ok, I feel like you're with me on this, so let's real. I'm gonna write a much larger example here. I want you to hang on and breathe thr Remember what we practiced. Two short br in, and exhale. Repeat.

```
(function() {  
  
    var Person = {  
  
        init: function()  
            this.for  
            this.bir  
  
        },  
  
        bindEvents: func  
            this.for  
  
        },  
  
        showName: functi  
            event.pr  
  
            alert(th  
  
        }  
    }  
  
    Person.init();  
  
})();
```



Take a look at this example [here](#) and then c back. Now look again, now back. How you c

That page is powered by the example code see above. Let's walk through each of the methods in the Person object and talk about what they're doing and how they're working together and how this very example of code could likely solve the world hunger problem. Bono, baby, call me.

Let's start with the init method as it is the first method called on our Person object when we call Person.init() at the bottom of our awesome self-executing anonymous function:

```
init: function() {  
    this.form = $('#form');  
    this.bindEvents();  
}
```

I should note that I'm calling this method init because it's short for initialize. That name here carries no special meaning. I could have called it tire or flamethrower or kitties. I'm just trying to be a little normal with my naming conventions. Good.

So, in the init method we are doing two things. We are setting a variable this.form which is the jQuery element that holds the element from our [demo page](#). This is our first glimpse at 'this'. Not to worry, but this 'this'. At this point one could go into a full lecture on 'this' and scope, but for now, I'll just say that 'this' is our object literal, Person. So we are assigning a variable to Person, a form element with the ID of 'form'. We do this by typing this.form

anywhere within the scope of Person, we can access the variable and have the form element. It's a bit of magic, but really it's just Javascript. The second thing the init method is doing is calling another method. So. Meta. We're calling the bindEvents method which is where I like to put any and all event listeners; things like on click, or on submit, or on fart, stuff like that.

So to summarize the init method we're setting a variable and then telling our script to listen for event handlers. But why didn't we just put our event handler into our init? Good question. 'A very good question and really the point of the entire article. This next sentence will be the purpose of your reading today—are you ready to write cleaner Javascript. Thank you thank you be here all week.


Actually, let's dive into that cleaner Javascript for a second. First, let's finish analyzing our script that we wanted to wet your appetite for the deconstruction that'll come later and take you to paradise with your sweet child o mine.

Let's take a look at the bindEvents method in

```
bindEvents: function() {  
    this.form.on('submit', t  
}
```


Remember this is the method I'm using to handle all my event listeners. In the case of our derivative is listening for someone to submit that form when it gets submitted it's going to call another method (meta, again) within our Person object called showName. Now, check it, we could have written this method like so:

```
bindEvents: function() {  
    $('#form').on('submit',  
}
```



We just switched our Person variable this for the value we had set it equal to, \$('#form'). This would have given us the same results. However, setting this.form in our Person object it affords a few luxuries. The first is that our script runs once and as soon as it grabs that element and stores it under this.form, it's done and stored and saved all warm and fuzzy. If we were to use it again further down in the showName method, we wouldn't need to go out and look up that element again. So we're saving some cycle or some amount of time in our script by setting that variable. Second, since we are using that variable multiple times in our script, if some intern popped in our code and was like, who gives their form the ID 'form', that's so lame and instead changed it to 'whatsapp', well then we'd have to go change every instance of our code with the new ID. If that ID was used like 16 billion times, well then there would be a lot of work, even for search/replace.

Finally, setting the variable `this.form` in one place and being able to use it yields cleaner Java:

Alright, now we're really moving. We're seeing how to set variables and call methods from other methods. We. Are. Big. Time.

Let's jump down to our final method, `showName`, that we're calling when the event handlers call `submit` in our `bindEvents` method. That was a lot of methods in one sentence. Remember when the only method you ever thought about was the `preventDefault` method and you were gonna finally pop in front of that girl you liked in junior high and she would finally see your potential and the video game instead of Tony Hawk's Pro Skater video game would be like Noah Stokes Pro Skater... any? I'm saying picture that for yourself, not for me, it never happened to me.

Here is our final method:

```
showName: function(event) {  
    event.preventDefault();  
  
    alert(this.form.find('input').val());  
}
```

This one is really straight forward. First we stop the form from submitting itself with the `event.preventDefault()`; Then we pop an alert

shows the value of the text field within our f
But wait, that doesn't work. If you ran the cc
is, you'd get an error telling you that it can't
value of the text input. Thanks alot Obama.
the heck is going on? This is an issue with s
Let's step back and check it out.

As we talked about earlier, I should be able
access this.form from anywhere within our s
and you're right you should. It's just that in t
instance of calling this.showName on submi
is no longer talking about our Person, it's tal
about our form element. Huh? Let's do an e:

```
$( 'form' ).on( 'submit', function(  
    alert( $(this).find( 'input'  
});
```

In the example above, you can see we are j
going to alert the value of the input. The 'thi
that example is talking about the form elem
because jQuery passes that in for us so we
some context. Very cool of John to do that,
But in our example, now we're futon'd beca
how are we supposed to get access to our
this.form variable?? Well, John thought of th
and there is a method in jQuery called \$.pro
With this method we are able to override th
of 'this' and keep it to what we originally wa
to be, Person. Here is a [link](#) to a quick 4 mir
video on \$.proxy. Take some time to watch i

a better understanding if you'd like. I would recommend it.

If it's still over your head, just take it on faith now as to what `$.proxy` is doing and let's move forward and jump back to our example. We'll use `$.proxy` and wrap our `this.showName` method in it to keep the context of 'this' where we want it.

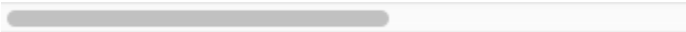
```
(function() {  
  
    var Person = {  
  
        init: function()  
            this.for  
            this.bir  
  
        },  
  
        bindEvents: func  
            this.for  
  
        },  
  
        showName: functi  
            event.pr  
  
            alert(th  
  
        }  
    }  
  
    Person.init();  
  
})();
```

See in line 12 how we wrapped this.`showName`
`$.proxy`? That is going to keep the scope of
where we want it and it's going to cause the
to function how we had originally planned. I
beautiful it brings a single solitary tear to my
Just one, one and no more. If I cried more for
would think that I had tickets to the new On
Direction tour which I don't or do I? Don't tell
kids. `#pForm`

Well that's it... and this entire time I'm sure you
were convinced that there was some wiccan
magic going on. Boy did I have you fooled. You
should take tricks like these to parties and call
them on white boards. All your friends and
neighbors will be `_so_` impressed.

Joking aside, you can see again in our example
how clean our Javascript is. That really is the
point of this article. I'm 100% confident that in writing
I used incorrect terminology and I probably
made statements that aren't entirely true—but please
don't let those things distract you. What you
see below is perfectly working code that is clean
organized, full of reusable variables all scoped to
our `Person` object and firing inside of a self-
executing anonymous function. That's pretty
fucking ass if you ask me. This article represents just
the tip of the iceberg when it comes to what you can
do with Javascript and with writing code this way.
I hope this sparked your interest, and I hope it did,
go off and learn more. Then write your own
teaching the rest of us what you learned.

```
(function() {  
  
    var Person = {  
  
        init: function()  
            this.for  
            this.bir  
  
        },  
  
        bindEvents: func  
            this.for  
  
        },  
  
        showName: functi  
            event.pr  
  
            alert(th  
  
        }  
    }  
  
    Person.init();  
  
})();
```



If you liked this post, and would like me to do more posts like this on a particular topic, please let me know. You can submit your topic [here](#) and I'll do my best to start creating more of these types of

[← Previous Post](#) [Next Post →](#)

