# 12 Simple (Yet Powerful) JavaScript Tips

feb. 20 2013 38

**(For Badass JavaScript Development)**

Table of Contents



**NOTICE:** I have written only 2 of the 12 tips so far, but I plan to post **all** 12 Powerful Tips eventually.

I provide you with 12 simple, yet powerful, JavaScript tips and detailed explanation of each. These are techniques that all JavaScript programmers can use now; you needn't be an advanced JavaScript developer to benefit from these tips. After you read all of the detailed explanations of how each technique works and when to use it, you will have become a more enlightened JavaScript developer, if you aren't already one.

Indeed, notable JavaScript masters and enlightened JavaScript developers have been using many of these techniques to write powerful, efficient JavaScript. And in a bit, you will, too.



## 1. Powerful JavaScript Idiomatic Expressions With && and ||

You see these idiomatic expressions in JavaScript frameworks and libraries. Let's start off with a couple of basic examples:



**Example 1: Basic "short circuting" with || (Logical OR)**
To set default values, instead of this:

```
function documentTitle(theTitle)
if (!theTitle) {
  theTitle  = "Untitled Document";
  }
}
```

Use this:

```
function documentTitle(theTitle)
  theTitle  = theTitle || "Untitled Document";
}
```

Explanation:
— First, read the "Important Note" box below for a refresher on JavaScript's Falsy and Truthy values
— The || operator first evaluates the expression on the left, if it is truthy, it returns that value. If it is falsy, it evaluates and returns the value of the right operand (the expression on the right).
— If *theTitle* variable is falsy, the expression on the right will be returned and assigned to the variable. On the other hand, if *theTitle* is truthy, its value will be returned and assigned to the variable.

> **An Important Note**
>
> **JavaScript Falsy Values:** null, false, 0 undefined, NaN, and "" (this last item is an empty string).
> — Note that *Infinity*, which is a special number like *NaN,* is truthy; it is not falsy, while NaN is falsy.
>
> **JavaScript Truthy Values:** Anything other than the falsy values.

**Example 2: Basic short circuting with && (Logical AND)**
Instead of this:

```
function isAdult(age) {
  if (age && age > 17) {
  return true;
}
else {
  return false;
  }
}
```

Use this:

```
function isAdult(age) {
   return age && age > 17 ;
}
```

Explanation:
— The && operator first evaluates the expression on the left. If it is falsy, false is returned; it does not bother to evaluate the right operand.
— If the the first expression is truthy, also evaluate the right operand (the expression on the right) and return the result.

## It Gets More Exciting!

**Example 3:**
Instead of this:

```
if (userName) {
  logIn (userName);
}
 else {
   signUp ();
}
```

Use this:

```
 userName && logIn (userName) || signUp ();
```

Explanation:
— If userName is truthy, then call the logIn function with userName as the parameter.
— If userName is falsy, call the signUp function

**Example 4:**
Instead of this:

```
var userID;
if (userName && userName.loggedIn) {
  userID = userName.id;
}
else {
  userID = null;
}
```

Use this:

```
var userID = userName && userName.loggedIn && userName.id
```

Explanation:
— If userName is truthy, call userName.loggedIn and check if it is truthy; if it is truthy, then get the id from userName.
— If userName is falsy, return null.

## 2. Powerful Uses of Immediately Invoked Function Expressions

**(How Immediately Invoked Function Expressions Work and When to Use Them)**
IIFE (pronounced "Iffy") is an abbreviation for Immediately Invoked Function Expression, and the syntax looks like this:

```
(function () {
 // Do fun stuff
 }
)()
```

It is an anonymous function expression that is immediately invoked, and it has some particularly important uses in JavaScript.

### How Immediately Invoked Function Expressions Work?

- The pair of parenthesis surrounding the anonymous function turns the anonymous function into a function expression or variable expression. So instead of a simple anonymous function in the global

scope, or wherever it was defined, we now have an unnamed function expression.
□

- It is as if we have this:

```
□// Shown without the parentheses here:
? = function () {};

// And with the parentheses here:
(? = function () {});
// An unknown variable assigned the value of a function, wrapped in a parentheses, which turns it int
o an unnamed function expression.
```

Similarly, we can even create a **named** , immediately invoked function expression:

```
(showName = function (name) {console.log(name || "No Name")}) (); // No Name

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

— Note that you cannot use the *var* keyword inside the opening pair of parentheses (you will get a syntax error), but it is not necessary in this context to use var since any variable declared without the var keyword will be a global variable anyway.
— We were able to call this named function expression both immediately and later because it has a name.
— But we can't call the anonymous function expression later, since there is no way to refer to it. This is the reason it is only useful when it is immediately invoked.

- By placing the anonymous function in parentheses (a group context), the entire group is evaluated and the value returned. The returned value is actually the entire anonymous function itself, so all we have to do is add two parentheses after it to invoke it.

- Therefore, the last two parentheses tell the JS compiler to invoke (call) this anonymous function immediately, hence the name "Immediately Invoked Function Expression."□

- Because JavaScript has function-level scope, all the variables declared in this anonymous function are local variables and therefore cannot be accessed outside the anonymous function.

- So we now have a powerful piece of anonymous code inside an unnamed function expression, and the code is meaningless unless we invoke the anonymous function, because nothing can access the code. It is the immediate invocation of the anonymous function that makes it powerful and useful.

- You can pass parameters to the anonymous function, just like you would any function, including variables. The anonymous function's scope extend into any outer function that contains it and to the global scope. Read my article, JavaScript Variable Scope and Hoisting Explained, for more.

## When To Use Immediately Invoked Function Expressions?

1. **To Avoid Polluting the Global Scope**
   □The most popular use of the IIFE is to avoid declaring variables in the global scope. Many JavaScript libraries use this technique, and of course many JS pros, too. It is especially popular amongst jQuery plugin developers. And you should use an IIFE in the top-level (main.js) of your applications.
   □
   In this first example, I am using it in the global scope to keep all my variables local to the anonymous function, and thus outside the global scope where variables can shadow (block) other already-defined variables with the same name (probably from an included library or framework). All of my code for the application will start in the IIFE:□□

```
// All the code is wrapped in the IIFE
(function () {
var firstName = "Richard";
  function init () {
    doStuff (firstName);
    // code to start the application
  }

function doStuff () {
  // Do stuff here
}

function doMoreStuff () {
  // Do more stuff here
}

// Start the application
init ();
}) ();
```

— Note that you can also pass jQuery or any other object or variable via the parameter (the last 2 parentheses).

2. **Use With the Conditional Operator**
   The use of the IIFE in this manner is not as well known, but it quite powerful since you can execute complex logic without having to setup and call a named function:

   — Note the two anonymous functions in the conditional statement
   — Why would you do this? Because it is powerful and badass.
   — I purposely added enough space between each section so the code can read better.

```
var unnamedDocs = [], namedDocs = ["a_bridge_runover", "great_dreamers"];

function createDoc(documentTitle) {
   var documentName = documentTitle

       ?

 (function (theName) {
       var newNamedDoc = theName.toLocaleLowerCase().replace(" ", "_");
       namedDocs.push(newNamedDoc);

       return newNamedDoc;
   })(documentTitle)



       :



       (function () {
          var newUnnamedDoc = "untitled_" + Number(namedDocs.length + 1);
          unnamedDocs.push(newUnnamedDoc);
          return newUnnamedDoc;
       })();


   return documentName;
}
createDoc("Over The Rainbow"); // over_the rainbow
createDoc(); // untitled_4
```

□□

3. **Use it in Closures to Prevent Fold Over**
   I discussed this particular use of the IIFE before in my post, Understand JavaScript Closures With Ease. So I will copy the section of that code and reuse it here.

   To prevent close over in for loops, we can use an Immediately Invoked Function Expression to prevent a common bug when closures are used with for loops:

   To fix side effects (bug) in closures, you can use an IIFE, such as if this example:

```
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 function () {
                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
            }
        } (i); // immediately invoke the function passing the i variable as a parameter
    }

    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
```

Be good. Sleep well. And enjoy coding.



Posted in: 12 Powerful JavaScript Tips, JavaScript / Tagged: 12 Powerful JavaScript Tips, Advanced JavaScript, Idiomatic Expressions, Intermediate Javascript, JavaScript Tips, Logical AND, Logical OR

### Richard

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