Javascript Enlightenment summary

Official website

Introduction

> In JavaScript, objects are king: Almost everything is an object or acts like an object. – p.15

Notation

Constructor

Literal

Function

> A function is a container of code statements that can be invoked using the parentheses () operator. Parameters can be passed inside of the parentheses during invocation so that the statements in the function can access certain values when the function is invoked.

Object

Variable

Style guide – Coding rules

Best practices

Use === Instead of ==

JavaScript utilizes two different kinds of equality operators: === | !==

and == | != It is considered best practice to always use the former set when comparing.

> If two operands are of the same type and value, then === produces true and !== produces false. – JavaScript: The Good Parts

However, when working with == and !=, you'll run into issues when working with different types. In these cases, they'll try to coerce the values, unsuccessfully.

Eval = Bad

For those unfamiliar, the "eval" function gives us access to JavaScript's compiler. Essentially, we can execute a string's result by passing it as a parameter of "eval". Not only will this decrease your script's performance substantially, but it also poses a huge security risk because it grants far too much power to the passed in text. **Avoid it!**

Don't Use Short-Hand

Technically, you can get away with omitting most curly braces and semi-colons. Most browsers will correctly interpret the following:

```
if(someVariableExists)
x = false
```

However, consider this:

```
if(someVariableExists)
x = false
anotherFunctionCall();
```

One might think that the code above would be equivalent to:

```
if(someVariableExists) {
   x = false;
   anotherFunctionCall();
}
```

Unfortunately, he'd be wrong. In reality, it means:

```
if(someVariableExists) {
   x = false;
}
anotherFunctionCall();
```

As you'll notice, the indentation mimics the functionality of the curly brace. Needless to say, this is a terrible practice that should be avoided at all costs. The only time that curly braces should be omitted is with one-liners, and even this is a highly debated topic.

```
if(2 + 2 === 4) return 'nicely done';
```

Always Consider the Future. What if, at a later date, you need to add more commands to this if statement. In order to do so, you would need to rewrite this block of code. Bottom line – tread with caution when omitting.

Place Scripts at the Bottom of Your Page

This tip has already been recommended in the previous article in this series. As it's highly appropriate though, I'll paste in the information.

Place JS at bottom Remember -- the primary goal is to make the page load as quickly as possible for the user. When loading a script, the browser can't continue on until the entire file has been loaded. Thus, the user will have to wait longer before noticing any progress.

If you have JS files whose only purpose is to add functionality -- for example, after a button is clicked -- go ahead and place those files at the bottom, just before the closing body tag. This is absolutely a best practice.

```
Better

<p>And now you know my favorite kinds of corn. </p>
<script type="text/javascript" src="path/to/file.js">
</script>
<script type="text/javascript"
src="path/to/anotherFile.js"></script>
</body>
</html>
```

Declare Variables Outside of the For Statement

When executing lengthy "for" statements, don't make the engine work any harder than it must. For example:

```
Bad

for(var i = 0; i < someArray.length; i++) {
   var container =
document.getElementById('container');
   container.innerHtml += 'my number: ' + i;
   console.log(i);
}
```

Notice how we must determine the length of the array for each iteration, and how we traverse the dom to find the "container" element each time – – highly inefficient!

```
Better

var container = document.getElementById('container');
for(var i = 0, len = someArray.length; i < len;  i++)
{
   container.innerHtml += 'my number: ' + i;
   console.log(i);
}
```

The Fastest Way to Build a String

Don't always reach for your handy-dandy "for" statement when you need to loop through an array or object. Be creative and find the quickest solution for the job at hand.

```
var arr = ['item 1', 'item 2', 'item 3', ...];
var list = '<ul><li>' + arr.join('</li><li>') +
'</li></ul>';
```

I won't bore you with benchmarks; you'll just have to believe me (or test for yourself) – this is by far the fastest method!

> Using native methods (like join()), regardless of what's going on behind the abstraction layer, is usually much faster than any non-native alternative. – James Padolsey, james.padolsey.com

Reduce Globals

> "By reducing your global footprint to a single name, you

> significantly reduce the chance of bad interactions with other applications, widgets, or libraries." – Douglas Crockford

```
var name = 'Jeffrey';
var lastName = 'Way';

function doSomething() {...}

console.log(name); // Jeffrey -- or window.name

Better

var DudeNameSpace = {
name : 'Jeffrey',
lastName : 'Way',
doSomething : function() {...}
}
console.log(DudeNameSpace.name); // Jeffrey
```

Notice how we've "reduced our footprint" to just the ridiculously named "DudeNameSpace" object.

Comment Your Code

It might seem unnecessary at first, but trust me, you WANT to comment your code as best as possible. What happens when you return to the project months later, only to find that you can't easily remember what your line of thinking was. Or, what if one of your colleagues needs to revise your code? Always, always comment important sections of your code.

```
// Cycle through array and echo out each name.
for(var i = 0, len = array.length; i < len; i++) {
console.log(array[i]);
```

```
  }
```

Embrace Progressive Enhancement

Always compensate for when JavaScript is disabled. It might be tempting to think, "The majority of my viewers have JavaScript enabled, so I won't worry about it." However, this would be a huge mistake.

Have you taken a moment to view your beautiful slider with JavaScript turned off? (Download the Web Developer Toolbar for an easy way to do so.) It might break your site completely. As a rule of thumb, design your site assuming that JavaScript will be disabled. Then, once you've done so, begin to progressively enhance your layout!

Don't Pass a String to "SetInterval" or "SetTimeOut"

Consider the following code:

```
setInterval(
"document.getElementById('container').innerHTML +=
'My new number: ' + i", 3000
);
```

ot only is this code inefficient, but it also functions in the same way as the "eval" function would. Never pass a string to SetInterval and SetTimeOut. Instead, pass a function name.

```
setInterval(someFunction, 3000);
```

Don't Use the "With" Statement

At first glance, "With" statements seem like a smart idea. The basic concept is that they can be used to provide a shorthand for accessing deeply nested objects. For example...

```
with (being.person.man.bodyparts) {
arms = true;
legs = true;
}
```

-- instead of --

```
being.person.man.bodyparts.arms = true;
being.person.man.bodyparts.legs= true;
```

Unfortunately, after some testing, it was found that they "behave very badly when setting new members." Instead, you should use var.

```
var o = being.person.man.bodyparts;
o.arms = true;
o.legs = true;
```

Use {} Instead of New Object()

There are multiple ways to create objects in JavaScript. Perhaps the more traditional method is to use the "new" constructor, like so:

```
var o = new Object();
o.name = 'Jeffrey';
o.lastName = 'Way';
o.someFunction = function() {
```

```
console.log(this.name);
}
```

However, this method receives the "bad practice" stamp without actually being so. Instead, I recommend that you use the much more robust object literal method.

Better

```
var o = {
    name: 'Jeffrey',
    lastName = 'Way',
    someFunction : function() {
    console.log(this.name);
    }
};
```

Note that if you simply want to create an empty object, {} will do the trick.

```
var o = {};
```

> "Objects literals enable us to write code that supports lots of features yet still make it a relatively straightforward for the implementers of our code. No need to invoke constructors directly or maintain the correct order of arguments passed to functions, etc." – dyn-web.com

Use [] Instead of New Array()

The same applies for creating a new array.

Okay

```
var a = new Array();
a[0] = "Joe";
a[1] = 'Plumber';
```

Better

```
var a = ['Joe','Plumber'];
```

> "A common error in JavaScript programs is to use an object when an array is required or an array when an object is required. The rule is simple: when the property names are small sequential integers, you should use an array. Otherwise, use an object." – Douglas Crockford

Long List of Variables? Omit the "Var" Keyword and Use Commas Instead

```
var someItem = 'some string';
var anotherItem = 'another string';
var oneMoreItem = 'one more string';
```

Better

```
var someItem = 'some string',
anotherItem = 'another string',
oneMoreItem = 'one more string';
```

...Should be rather self-explanatory. I doubt there's any real speed improvements here, but it cleans up your code a bit.

Always, Always Use Semicolons

Technically, most browsers will allow you to get away with omitting semi-colons.

```
var someItem = 'some string'
function doSomething() {
    return 'something'
}
```

Having said that, this is a very bad practice that can potentially lead to much bigger, and harder to find, issues.

Better

```
var someItem = 'some string';
function doSomething() {
    return 'something';
}
```

"For in" Statements

When looping through items in an object, you might find that you'll also retrieve method functions as well. In order to work around this, always wrap your code in an if statement which filters the information

```
for(key in object) {
    if(object.hasOwnProperty(key) {
        ...then do something...
    }
}
```

As referenced from JavaScript: The Good Parts, by Douglas Crockford.

Use Firebug's "Timer" Feature to Optimize Your Code

Need a quick and easy way to determine how long an operation takes? Use Firebug's "timer" feature to log the results.

```
function TimeTracker(){
    console.time("MyTimer");
    for(x=5000; x > 0; x--){}
    console.timeEnd("MyTimer");
}
```

Self-Executing Functions

Rather than calling a function, it's quite simple to make a function run automatically when a page loads, or a parent function is called. Simply wrap your function in parenthesis, and then append an additional set, which essentially calls the function.

```
(function doSomething() {
    return {
        name: 'jeff',
        lastName: 'way'
    };
})();
```

Raw JavaScript Can Always Be Quicker Than Using a Library

JavaScript libraries, such as jQuery and Mootools, can save you an enormous amount of time when coding -- especially with AJAX operations. Having said that, always keep in mind that a library can never be as fast as raw JavaScript (assuming you code correctly).

jQuery's "each" method is great for looping, but using a native "for" statement will always be an ounce quicker.

Crockford's JSON.Parse

Although JavaScript 2 should have a built-in JSON parser, as of this writing, we still need to implement our own. Douglas Crockford, the creator of JSON, has already created a parser that you can use. It can be downloaded HERE.

Simply by importing the script, you'll gain access to a new JSON global object, which can then be used to parse your .json file.

```
var response = JSON.parse(xhr.responseText);

var container = document.getElementById('container');
for(var i = 0, len = response.length; i < len; i++) {
    container.innerHTML += '<li>' + response[i].name
+ ' : ' + response[i].email + '</li>';
}
```

Remove "Language"

Years ago, it wasn't uncommon to find the "language" attribute within script tags.

```
<script type="text/javascript" language="javascript">
...
</script>
```

However, this attribute has long since been deprecated; so leave it out.