# **Buenas prácticas para programar con JavaScript**

Les comparto un **resumen básico de buenas prácticas** que debemos tener en cuenta a la hora de programar con **JavaScript**.

También, tenemos una publicación actualizada sobre las [nuevas caracteristicas JavaScript anunciadas en el último Google I/O 2019](http://www.solocodigoweb.com/blog/2019/06/25/nuevas-funciones-y-especificaciones-javascript-anunciadas-en-el-google-i-o-2019/).

**1. No usar taquigrafía**

Si bien puede resultar muy útil y rápido usar las distintas técnicas abreviadas para programar, debe tener mucho cuidado con algunas, por ejemplo, puedes tener algo similar a:

|  |  |
| --- | --- |
| 1  2 | if (variableExiste)      x = false; |

Sin embargo considera esto:

|  |  |
| --- | --- |
| 1  2  3 | if (variableExiste)      x = false;      llamarOtraFuncion(); |

Uno podría pensar que el código es equivalente a:

|  |  |
| --- | --- |
| 1  2  3  4 | if (variableExiste) {      x = false;      llamarOtraFuncion();  } |

Lo cual estaría mal. En realidad sería de la siguiente forma:

|  |  |
| --- | --- |
| 1  2  3  4 | if (variableExiste) {      x = false;  }  llamarOtraFuncion(); |

Si deseas omitir las llaves ({}) debería de ser cuando se trate de una sentencia de una sola línea, por ejemplo:

|  |  |
| --- | --- |
| 1 | if (edad > 18) return "Es mayor de edad"; |

**2. Utilice === en vez de ==**

**JavaScript**utiliza dos tipos diferentes de operadores de igualdad: === | !== y == | !=. Se considera una buena practica utilizar la primera forma.

**Javascript Good Patterns** dice lo siguiente: «*If two operands are of the same type and value, then === produces true and !== produces false.*«

Eso quiere decir que si dos operandos son del **mismo tipo** y **valor**, entonces **===** produce verdadero y **!==** produce falso.

**3. Usa JSLint**

[**JSLint**](http://www.jslint.com/) es un depurador escrito por [**Douglas Crockford**](https://twitter.com/CrockfordFacts). Simplemente pega tu script, y la herramienta va a escanear rápidamente cualquier problema y errores que tuviera su código.

«JSLint takes a JavaScript source and scans it. If it finds a problem, it returns a message describing the problem and an approximate location within the source. The problem is not necessarily a syntax error, although it often is. JSLint looks at some style conventions as well as structural problems. It does not prove that your program is correct. It just provides another set of eyes to help spot problems.»

*JSLint Documentation*

**4. Coloque su script al final de la página**

Es una buena práctica que el **código Javascript cargue después de haber cargado los elementos HTML** de su página web.

Esto **optimizará la velocidad** en que se muestra su página web para el usuario. Cuando se carga un script, el navegador no puede continuar hasta que todo el archivo se haya cargado. De este modo, el usuario tendrá que esperar más tiempo antes de notar algún progreso.

Si usted tiene archivos JavaScript cuyo único propósito es añadir funcionalidad – por ejemplo, después de hacer clic en un botón – debe poner los archivos en la parte inferior, justo antes de la etiqueta del cuerpo de cierre. Esto es absolutamente una buena práctica.

|  |  |
| --- | --- |
| 1 | <!-- fin contenedor --><script type="text/javascript" src="archivoJavascript.js"></script><script type="text/javascript" src="funcionalidad.js"></script> |

**5. Declara las variables fuera de la sentencia FOR**

Cuando ejecute sentencias **FOR** muy extensos, no haga que el motor trabaje mucho, por ejemplo:

|  |  |
| --- | --- |
| 1  2  3  4  5 | for(var i = 0; i < algunArray.length; i++) {     var contenedor = document.getElementById('contenedor');     contenedor.innerHtml += 'mi numero: ' + i;     console.log(i);  } |

Observe cómo debemos determinar la longitud de la matriz para cada iteración, y cómo se recorre el **DOM**para encontrar el elemento «contenedor», esto es muy ineficiente!

Quedaría mejor de esta forma:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var contenedor = document.getElementById('contenedor');  var longitud = algunArray.length;  var contenido = '';  for(var i = 0; i < longitud;  i++) {     contenido += 'mi numero: ' + i;     console.log(i);  }  contenedor.innerHtml = contenido; |

**6. La manera mas rápida de construir una cadena**

Si tiene un array de cadenas y necesita armar una cadena puede optar por utilizar el método nativo join que le permite unir elementos de un array dentro de una cadena.

|  |  |
| --- | --- |
| 1  2  3 | var ciudades = ["Lima", "Ica", "Arequipa", "Chiclayo"];  var lista = ciudades.join(",");  document.write(lista); |

**7. Reducir variables globales**

Al reducir sus variables globales a una sola variable (a un solo nombre), reduce significativamente la posibilidad de malas interaciones con otras aplicaciones, widgets o librerias JS.

Por ejemplo, puedes tener el siguiente código.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var nombres = 'Gonzalo';  var apellidos = 'Chacaltana';    function hacerAlgo() {...}    console.log(nombres); // Gonzalo |

Mejor puedes programar así:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var Persona = {     nombres : 'Gonzalo',     apellidos : 'Chacaltana',     hacerAlgo : function() {...}  }  console.log(Persona.nombres); // Gonzalo |

**8. Comenta tu código**

Puede parecer innecesario al principio, pero la verdad que **es muy importante comentar nuestro código** de la mejor manera posible. ¿Qué sucede cuando después de meses vuelves a revisar tu código, te darás cuenta que no puedes recordar fácilmente como era la línea de pensamiento o proceso que implementaste. La siguiente imagen resume muy bien cuándo no documentamos nuestro código.

O bien, ¿qué pasa si uno de tus compañeros tiene que revisar tu código? Lo mas probable que tarde un poco en entenderlo o quizás ni lo entienda. Por ello se recomienda que siempre, siempre comenten las secciones importantes de su código.

**9. Usar {} en vez de New Object()**

Existen muchas formas de crear objetos en Javascript, y quizás el método mas tradicional sea el siguiente:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var o = new Object();  o.nombres = 'Gonzalo';  o.apellidos = 'Chacaltana';  o.algunaFuncion = function() {     console.log(this.nombres);  } |

Sin embargo, este método se le conocer como una «mala practica» sin realmente serlo. En su lugar se recomienda utilizar el método literal que es mucho mas robusto que instanciar un Object().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | var o = {     nombres: 'Gonzalo',     apellidos: 'Chacaltana',     algunaFuncion : function() {        console.log(this.nombres);     }  }; |

Si solo necesita crear un objeto vacio, solo utilice las llaves {}

|  |  |
| --- | --- |
| 1 | var o = {} |

**10. Usar [] en vez de New Array()**

Lo mismo se aplica para crear un array.

|  |  |
| --- | --- |
| 1  2  3 | var cursos = new Array();  a[0] = "Lenguaje de Programación";  a[1] = 'Administración de Proyectos'; |

Mejor, podríamos hacerlo de la siguiente forma:

|  |  |
| --- | --- |
| 1 | var cursos = ['Lenguaje de Programación','Administración de Proyectos']; |

**11. Omitir var keyword para declarar muchas variables**

|  |  |
| --- | --- |
| 1  2  3 | var edad = 14;  var peso = 56;  var talla = 166; |
| 1  2  3 | var edad = 14,      peso = 56,      talla = 166; |

Y por último y no menos importante, darse un tiempo para leer el libro **[Javascript Cookbook](http://www.javascriptcookbook.com/" \t "_blank)** de **Shelley Powers**, un libro muy completo para profundizar en este grandioso lenguaje de programación.

Enlace de Referencia: <http://www.solocodigoweb.com/blog/2014/04/30/buenas-practicas-javascript/>