APRENDERAPROGRAMAR.COM

JAVASCRIPT. REGLAS DE ESTILO (MANUALES O CONVENCIONES). CÓMO CREAR OBJETOS Y ARRAYS. EJEMPLOS Y EJERCICIO (CU01191E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº91 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

REGLAS DE ESTILO JAVASCRIPT

Los programadores JavaScript suelen atenerse a unas reglas de estilo a la hora de escribir código. Estas reglas son definidas por programadores expertos (gurús) o por empresas como Google ó Microsoft y no son de obligado cumplimiento. Tan sólo son recomendaciones que buscan que el código sea más fácil de leer y entender y siga unos estándares.

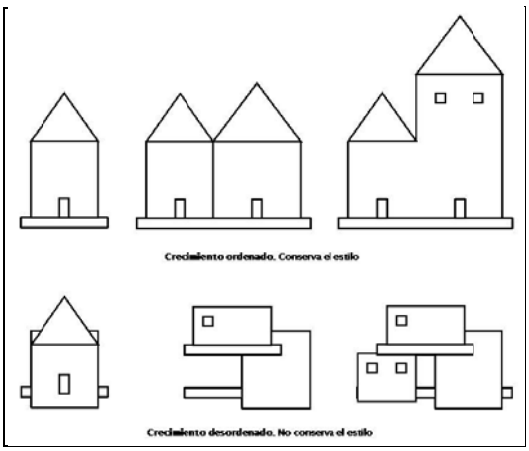
MANUALES DE ESTILO

En una empresa en la que trabajan 100 ó 1000 programadores cada programador no puede "hacer las cosas a su manera". En estas empresas es frecuente que cada programador desarrolle un fragmento de una aplicación y luego esos fragmentos hay que unirlos. Para que esa unión sea sólida y no haya problemas de integración y mantenimiento posterior, todos los programadores deben atenerse a unas mismas normas.

Lo ideal sería que en una aplicación donde han trabajado 20 programadores, un supervisor no fuera capaz de detectar qué parte del código ha hecho cada programador. Esto es difícil, porque cada programador deja su impronta, pero siguiendo unas reglas predefinidas se puede conseguir como resultado un código bien estructurado y bastante homogéneo.

Si una aplicación se ha creado siguiendo unas reglas, cuando se hagan ampliaciones o correcciones del código, se deberían seguir usando esas reglas para mantener el código homogéneo y hacerlo mantenible. Si no se mantiene un orden y un criterio de estilo, las aplicaciones terminan por hacerse demasiado complicadas y difíciles de entender.

La idea puede ser similar a la construcción de un edificio: en todas las plantas deben respetarse unas normas (altura de las puertas, pintura aplicada, dimensiones de los pasillos, etc.). Además cuando el edificio se amplíe deben seguir respetándose esas normas. Si no se hace así, el aspecto será el de "una chapuza" o una unión de cosas en lugar de un edificio.



REGLAS DE ESTILO

A continuación presentamos una lista de reglas de estilo que suelen recomendarse para la programación JavaScript. Como hemos indicado, son opcionales, pero un programador que no sigue unas reglas de estilo no suele considerarse un buen programador. Más que seguir estrictamente estas reglas que presentamos a continuación, lo importante es que cuando crees código sigas unas reglas de estilo (que pueden ser estas o similares a estas) y que no las modifiques. De esa manera crearás código homogéneo.

También ten en cuenta que las reglas aquí presentadas son una forma resumida de reglas respecto a lo que sería un manual de estilo propio de una empresa, que sería mucho más extenso. El objetivo de presentar aquí estas reglas es didáctico y para familiarizarnos con ellas, no consideres que son obligatorias ni que aquí tienes todas las reglas necesarias.

Reglas de estilo recomendadas a la hora del uso del lenguaje JavaScript:

REGLA DE ESTILO	EJEMPLO
Declarar siempre las variables usando var. No hacerlo dificulta interpretar en qué ámbito se encuentra la variable y puede ocasionar que una variable esté en ámbito global indebidamente.	var nodosDiv;

Usa siempre puntos y coma para delimitar el final de una instrucción o línea. Cuando una función funciona como una expresión, debe delimitarse con punto y coma final. En cambio, en las declaraciones de funciones no se usa punto y coma final (ver ejemplo)	var mivar1 = function() {
	  return true;
	}; // Aquí incluir punto y coma final
	
	function miFuncion1() {
	  return true;
	} // Aquí no incluimos punto y coma final

No declarar funciones dentro de bloques (condicionales, bucles…). Si se requiere crear funciones dentro de bloques, usarlas como expresiones que se asignan a una variable.	if (x) {
	  function miFuncion1() {…} // ¡incorrecto!
	}
	if (x) {
	  var mivar1 = function() {…}; // ¡correcto!
	}

Hay que tener cuidado a la hora de crear closures porque pueden generar referencias circulares que impliquen desbordamiento de memoria.	No utilizar:
	function miFuncion1(element, a, b) {
	  element.onclick = function() { /* código que usa a y b, crea el closure que mantiene una referencia a element a pesar de no usarlo, mientras que element mantiene una referencia al closure, circularidad */ };
	  }
	Para resolver esta situación se usa:
	function miFuncion1(element, a, b) {
	  element.onclick = miFun2(a, b);
	}
	function miFun2(a, b) {
	  return function() { /* código que usa a y b */ };
	  }

REGLA DE ESTILO	EJEMPLO
No usar with	Evitar el uso de esta sentencia
Limitar el uso de this	this puede tener distintos significados según el contexto. Usarlo sólo cuando sea realmente necesario.
No usar for-in para recorrer arrays.	Para recorrer arrays debe usarse un bucle for tradicional.
Crear arrays usando sintaxis de literal en lugar de new Array	Recomendado: var a = [x1, x2, x3]; No usar: var a = new Array(x1, x2, x3);
Crear objetos usando sintaxis de literal en lugar de new Object	Recomendado: var a = { }; No usar: var a = new Object ();
Crea el javascript en archivos js independientes en lugar de embeberlo en ficheros html	Esto hará reusable el código y hará más pesada la carga de los archivos html.
Usa === y !== en condicionales para realizar comparaciones preferentemente sobre == y !=	if (a === b) { ... }
Evita el uso de eval	Busca alternativas y evita usar la función eval

Incluye código previendo una llamada a una función en la que falten parámetros	function myFunction(x, y) {
	if (y === undefined) { y = 0; }
	}
	Esto se escribe más fácil y cómodo así:
	function myFunction(x, y) {
	y = y || 0;
	}

Reduce el acceso al DOM.	Si tienes que acceder varias veces a un elemento del DOM, no uses el acceso tipo getElementById repetidas veces. Usalo una sola vez y almacena el resultado en una variable. De esta manera mejoras el rendimiento.

Reglas de estilo recomendadas para la presentación de código JavaScript:

REGLA DE ESTILO	EJEMPLO
Usar la sintaxis camelCase para declarar nombres de variables y funciones (primera letra minúscula e intercalar mayúsculas).	nombreDeFuncionAsi, nombreDeVariableAsi, nombreDeMetodoAsi,
Crear los nombres de funciones y variables usando sólo 26 letras (a hasta z, sin la ñ), diez números (0 al 9) y el guión bajo _. Evitar uso de la ñ o signos como $, %, &, etc. Tampoco uses el guión medio.	var anyo = 2089;

REGLA DE ESTILO	EJEMPLO
Si una variable no va a cambiar de valor nunca se declara con todas sus letras en mayúsculas.	var NUMERODEAVOGADRO;
Los nombres de ficheros se escriben completamente en minúsculas. En el caso de ficheros html, se usa la extensión html en lugar de htm.	nombredeficherotodominusculas.js otroNombre.html

Incluye las llaves en la misma línea de aquello que abren (no pongas las llaves por ejemplo debajo de una condición de un if, sino al lado)	if (condicion) {
	// ...
	} else {
	// ...
	}

Usa siempre indentado para aquello que está dentro de un bloque. Por ejemplo, el código dentro de un bucle for estará desplazado hacia la derecha. Usa siempre el mismo indentado.	for (var i=0; i<10; i++) {
	//aquí codigo
	}

Las líneas no se deben alargar tanto como para no ser visibles. Se usa la referencia de 80 caracteres. La línea no debe tener más de 80 caracteres. Si se excede esta longitud, la línea deberá dividirse.	esto.aquello.alli.hacer = function(
	unArgumentoParaLaFuncionDeNombre1,
	unArgumentoParaLaFuncionDeNombre2,
	unArgumentoParaLaFuncionDeNombre3,
	unArgumentoParaLaFuncionDeNombre4) {
	// código
	};

Usa comillas simples en lugar de comillas dobles	var nombre = 'carlos';

Separa las sentencias en distintas líneas. No escribas una detrás de otra separadas por ;	var x=1; var y=2; // No
	var x=1;
	var y=2;

Usa espacios separadores: esto hace más fácil de leer el código	while(x<3&&y<5&&z==10){llamarA();} //No
	while (x < 3 && y < 5 && z == 10) {
	llamarA();
	} //Sí

El código debe estar correctamente comentado. Los bloques /* ... */ se usarán para documentación formal (explicación de funciones, parámetros que intervienen, etc. y los comentarios en línea // para aclaraciones puntuales.	Para documentar proyectos extensos se recomienda seguir unas normas de comentarios y usar herramientas específicas que crean la documentación de forma automática, como JSDoc.
Respetar el estilo del código en edición	Si nos encargan realizar unos cambios en un código que tiene un estilo, respetar el estilo de dicho código aunque no coincida con el que nosotros usemos habitualmente. Es importante que dentro de un código se mantenga siempre el mismo criterio.

EJERCICIO

Un programador ha creado este código y nos han pedido que lo mejoremos. Revísalo y responde a las siguientes cuestiones:

```
<HTML><HEAD><TITLE>JavaScript Index</TITLE><script Language="JavaScript">
function goback(){alert("Good Bye!");history.go(-1);}
function getthedate() {Todays = new Date();
TheDate = "" + (Todays.getMonth()+ 1) +" / "+ Todays.getDate() + " / " +
Todays.getYear()
document.clock.thedate.value = TheDate;
}

var timerID = null;
var timerRunning = false;
function stopclock (){
        if(timerRunning)
                clearTimeout(timerID);
        timerRunning = false;
}

function startclock () { stopclock(); getthedate()
        showtime();
}

function showtime () {
        var now = new Date(); var hours = now.getHours(); var minutes = now.getMinutes();
        var seconds = now.getSeconds()
        var timeValue = "" + ((hours >12) ? hours -12 :hours)
        timeValue += ((minutes < 10) ? ":0" : ":") + minutes
        timeValue += ((seconds < 10) ? ":0" : ":") + seconds
        timeValue += (hours >= 12) ? " P.M." : " A.M."
        document.clock.face.value = timeValue;
        timerID = setTimeout("showtime()",1000); timerRunning = true;
}
</script>
</HEAD>
<BODY bgcolor="#00FFFF" onLoad="startclock()">
<CENTER><h2>Esto es un reloj hecho con JavaScript</h2>
<table border><tr>
  <td><form name="clock" onSubmit="0"></td>
</tr>
<tr>
  <td colspan=2>Hoy es: <input type="text" name="thedate" size=12  value=""></td>
  <td colspan=2>La hora es: <input type="text" name="face" size=12
      value=""></td></form>
</tr>
</table>
</CENTER>
<hr>
<center>
<h3>
[<a href="http://aprenderaprogramar.com">Volver</a>]
</h3></center></BODY></HTML>
```

a) Reescribe el código HTML que presenta distintas deficiencias y no se ajusta a las normas de estilo habituales.

b) Reescribe el código JavaScript para que cumpla con las normas de estilo que hemos estudiado.

c) Corrige el código del reloj para que se vea una mejor presentación y funcione correctamente.

d) Incluye comentarios en el código indicando qué es lo que hacen las diferentes partes del código JavaScript.

e) Como resultado de este ejercicio debes presentar el código con todos los cambios antes mencionados.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01192E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206