Tema 2
Estructura del lenguaje javascript

ÍNDICE

Estructura del lenguaje javascript

1. Fundamentos de javascript.

El lenguaje que vas a estudiar ahora se llama JavaScript, pero quizás habrás oído otros nombres que te resulten similares como JScript (que es el nombre que le dio Microsoft a este lenguaje). Microsoft le dio el nombre de JScript para evitar problemas relacionados con la marca, pero no pudo evitar otros problemas surgidos por las incompatibilidades que su versión de JavaScript tenía con múltiples navegadores. Para evitar esas incompatibilidades, el W3C, diseñó el DOM (Modelo de Objetos del Documento), que incorporaron las versiones de Internet Explorer 6, Netscape Navigator, Opera 7 y Mozilla Firefox desde su primera versión.

A partir de 1997 este lenguaje se rige por un estándar denominado ECMA, que se encarga de gestionar las especificaciones de este lenguaje de script (da igual el nombre que reciba). En el documentoECMA-262 es dónde se detallan dichas especificaciones. Tanto JavaScript como JScript son compatibles con el estándar ECMA-262.

JavaScript se diseñó con una sintaxis similar al lenguaje C y aunque adopta nombres y convenciones del lenguaje Java, éste último no tiene relación con JavaScript ya que tienen semánticas y propósitos diferentes.

JavaScript fue desarrollado originariamente por Brendan Eich, de Netscape, con el nombre de Mocha, el cual se renombró posteriormente a LiveScript y quedó finalmente como JavaScript.

Hoy en día JavaScript es una marca registrada de Oracle Corporation, y es usado con licencia por los productos creados por Netscape Communications y entidades actuales, como la fundaciónMozilla.

Más información sobre Brendan Eich: http://es.wikipedia.org/wiki/Brendan_Eich

1.1. Comentarios en el código.

A la hora de programar en cualquier lenguaje de programación, es muy importante que comentes tu código.

Los comentarios son sentencias que el intérprete de JavaScript ignora. Sin embargo estas sentencias permiten a los desarrolladores dejar notas sobre cómo funcionan las cosas en sus scripts.

Los comentarios ocupan espacio dentro de tu código de JavaScript, por lo que cuando alguien se descargue vuestro código necesitará más o menos tiempo, dependiendo del tamaño de vuestro fichero. Aunque ésto pueda ser un problema, es muy recomendable el que documentes tu código lo máximo posible, ya que esto te proporcionará muchas más ventajas que inconvenientes.

JavaScript permite dos estilos de comentarios. Un estilo consiste en dos barras inclinadas hacia la derecha (sin espacios entre ellas), y es muy útil para comentar una línea sencilla. JavaScript ignorará cualquier carácter a la derecha de esas barras inclinadas en la misma línea, incluso si aparecen en el medio de una línea.

Ejemplos de comentarios de una única línea:

```
// Este es un comentario de una línea
var nombre="Marta" // Otro comentario sobre esta línea
// Podemos dejar por ejemplo
//
// una línea en medio en blanco
```

Para comentarios más largos, por ejemplo de una sección del documento, podemos emplear en lugar de las dos barras inclinadas el `/*` para comenzar la sección de comentarios, y `*/` para cerrar la sección de comentarios.

Por ejemplo:

```
/* Ésta es una sección
de comentarios
en el código de JavaScript */
```

O también:

```
/* ----------------------------------
función imprimir()
Imprime el listado de alumnos en orden alfabético
----------------------------------*/
function imprimir()
{
// Líneas de código JavaScript
}
```

¿Sabías que usando comentarios podemos desactivar un bloque de código para que JavaScript deje de ejecutarlo sin tener que borrar nada en el código?

¿Sabías además, que los comentarios en JavaScript son muy útiles para dejar por ejemplo tu información de contacto para que otros programadores contacten contigo?

Más información y ejemplos sobre comentarios en JavaScript.

http://www.htmlpoint.com/javascript/corso/js_07.htm

1.2. Variables.

La forma más conveniente de trabajar con datos en un script, es asignando primeramente los datos a una variable. Es incluso más fácil pensar que una variable es un cajón que almacena información. El tiempo que dicha información permanecerá almacenada, dependerá de muchos factores. En el momento que el navegador limpia la ventana o marco, cualquier variable conocida será eliminada.

Dispones de dos maneras de crear variables en JavaScript: una forma es usar la palabra reservada **var** seguida del nombre de la variable. Por ejemplo, para declarar una variable edad, el código de JavaScript será:

```
var edad;
```

Otra forma consiste en crear la variable, y asignarle un valor directamente durante la creación:

```
var edad = 38;
```

o bien, podríamos hacer:

```
var edad;
edad = 38; // Ya que no estamos obligados a declarar la variable pero es
//una buena práctica el hacerlo.
var altura, peso, edad: // Para declarar más de una variable en la misma
//línea.
```

La palabra reservada **var** se usa para la declaración o inicialización de la variable en el documento. Una variable de JavaScript podrá almacenar diferentes tipos de valores, y una de las ventajas que tenemos con JavaScript es que no tendremos que decirle de qué tipo es una variable u otra.

A la hora de dar nombres a las variables, tendremos que poner nombres que realmente describan el contenido de la variable. No podremos usar palabras reservadas, ni símbolos de puntuación en el medio

de la variable, ni la variable podrá contener espacios en blanco. Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). No podremos utilizar caracteres raros como el signo `+`, `un espacio`, `%` , `$`, etc. en los nombres de variables, y estos nombres no podrán comenzar con un número.

Si queremos nombrar variables con dos palabras, tendremos que separarlas con el símbolo "_" o bien diferenciando las palabras con una mayúscula, por ejemplo:

```
var mi_peso;
var miPeso; // Esta opción es más recomendable, ya que es más cómoda de
//escribir.
```

Deberemos tener cuidado también en no utilizar nombres reservados como variables. Por ejemplo, no podremos llamar a nuestra variable con el nombre de **return** o **for**.

Más información y ejemplos de Variables. http://www.w3schools.com/js/js_variables.asp

1.3. Tipos de datos.

Las variables en JavaScript podrán contener cualquier tipo de dato. A continuación, se muestran los tipos de datos soportados en JavaScript:

Tipos de datos soportados por JavaScript		
Tipo	**Ejemplo**	**Descripción**
Cadena.	"Hola mundo"	Una serie de caracteres dentro de comillas dobles.
Número.	9.45	Un número sin comillas dobles.
Boolean.	true.	Un valor verdadero o falso.
Null.	null.	Desprovisto de contenido, simplemente es un valor null.
Object.		Es un objeto software que se define por sus propiedades y métodos (los arrays también son objetos).
Function.		La definición de una función.

El contener solamente este tipo de datos, simplifica mucho las tareas de programación, especialmente aquellas que abarcan tipos incompatibles entre números.

A continuación te presentamos un vídeo sobre tipos de datos en JavaScript.

http://www.youtube.com/watch?feature=player_embedded&v=cw5sFqlDBtE#!

Resumen: Vamos a ver dos artículos uno de ellos más genérico y otro más específico sobre los tipos de datos. Abrimos el editor web Komodo y creamos un directorio, en el cuál crearemos un archivo `ejemplos-tipos-datos.html`. Abrimos el código de JavaScript con `<script lenguaje="javascript">` y `</script>` . Primero hablaremos de los tipos de datos numéricos, los cuales no se diferencian en JavaScript. Es indiferente que se usen números reales o números enteros. Para JavaScript siempre serán tipos de datos numéricos. Creamos 4 variables que almacenan diferentes ejemplos de números. Para crear variables numéricas también se puede utilizar notación científica. Para ello usamos un exponente (que se define con una letra "e"), seguido a continuación del exponente al cual está elevado. También podremos crear números en bases diferentes (base 8, base 16). Para crear un número en base 8 comenzamos con un 0 seguido de números del 0 al 7. Para los números hexadecimal se pone un 0 seguido de una "x" y cualquier carácter hexadecimal (del 0 al 9 y de la A a la F).

Para crear variables de tipo cadena, simplemente tenemos que poner el texto entre comillas. Si tenemos una cadena que sólo contiene caracteres numéricos, para JavaScript eso será una cadena, independientemente del contenido que tenga. Dentro de las cadenas podemos usar caracteres de escape (como saltos de línea con `\n`, comillas dobles `\"`, tabuladores `\t`, etc.). Cuando sumamos cadenas en JavaScript lo que hacemos es concatenar esas cadenas (poner una a continuación de la otra).

Los tipos de datos booleano, son un `true` o un `false`. Y se usan mucho para tomar decisiones. Los únicos valores que podemos poner en las variables booleanas son `true` y `false`. Se muestra un ejemplo de uso de una variable booleana dentro de una sentencia `if`, modificando la comparación usando `==` y `===`.

Existen otros tipos de datos especiales que son de tipo Objeto, por ejemplo los **arrays**. Un array es una variable con un montón de casillas a las que se accede a través de un índice.

El programa final quedará así:

```
<script lenguaje=”javascript”>
// tipo de datos numéricos
// pueden ser enteros o números con decimales
var miEntero = 33;
var miDecimales = 2.5;
var comaFlotante = 2344.983338;
var numeral = 0.573;
// pueden tener notación científica
var numCientifico = 2.9e3;
var otroNumCientifico = 2e-3;
//alert(otroNumCientifico);
// podemos escribir números en otras bases
var numBase10 = 2200;
var numBase8 = 0234;
var numBase16 = 0x2A9F;
//alert(numBase16);
// tipo de datos cadena de caracteres
var miCadena = “Hola! esto es una cadena!”;
var otraCadena = “2323232323”; // parece un numérico pero es una cadena
// caracteres de escape en cadenas
var cadenaConSaltoDeLinea = “linea1\nLínea2\nLínea3”;
var cadenaConComillas = “cadena „con comillas“ \”comillas dobles\””;
var cadenaNum = “11”;
var sumaCadenaConcatenacion = otraCadena + cadenaNum;
//alert(sumaCadenaConcatenacion);
// tipos de datos booleano
var miBooleano = true;
var falso = false;
if (niBooleano){
//alert (“era true”);
}else{
//alert(“era false”);
}
var booleano = (23===”23”);
//alerts(booleano)
//esos eran los tipos de datos principales
```

```
//pero existen otros tipos especiales que veremos más adelante
//arrays
var miArray = (2,5,7);
//objetos
var miObjeto = {
propiedad: 23,
otracosa: “hola”
}
//operador typeof para conocer un tipo
alert(“El tipo de miEntero es: “ + typeof(miEntero));
alert(“El tipo de miCadena es: “ + typeof(miCadena));
alert(“El tipo de miEntero es: “ + typeof(miBooleano));
alert(“El tipo de miEntero es: “ + typeof(miArray));
alert(“El tipo de miEntero es: “ + typeof(miObjeto));
</script>
```

1.3.1. Conversiones de tipos de datos.

Aunque los tipos de datos en JavaScript son muy sencillos, a veces te podrás encontrar con casos en los que las operaciones no se realizan correctamente, y eso es debido a la conversión de tipos de datos. JavaScript intenta realizar la mejor conversión cuando realiza esas operaciones, pero a veces no es el tipo de conversión que a ti te interesaría.

Por ejemplo cuando intentamos sumar dos números:

```
4 + 5 // resultado = 9
```

Si uno de esos números está en formato de cadena de texto, JavaScript lo que hará es intentar convertir el otro número a una cadena y los concatenará, por ejemplo:

```
4 + "5" // resultado = "45"
```

Otro ejemplo podría ser:

```
4 + 5 + "6" // resultado = "96"
```

Esto puede resultar ilógico pero sí que tiene su lógica. La expresión se evalúa de izquierda a derecha. La primera operación funciona correctamente devolviendo el valor de 9 pero al intentar sumarle una cadena de texto "6" JavaScript lo que hace es convertir ese número a una cadena de texto y se lo concatenará al comienzo del "6".

Para convertir cadenas a números dispones de las funciones: **parseInt()** y **parseFloat()**.

Por ejemplo:

```
parseInt("34") // resultado = 34
parseInt("89.76") // resultado = 89
```

parseFloat devolverá un entero o un número real según el caso:

```
parseFloat("34") // resultado = 34
parseFloat("89.76") // resultado = 89.76
4 + 5 + parseInt("6") // resultado = 15
```

Si lo que deseas es realizar la conversión de números a cadenas, es mucho más sencillo, ya que simplemente tendrás que concatenar una cadena vacía al principio, y de esta forma el número será convertido a su cadena equivalente:

```
("" + 3400) // resultado = "3400"
("" + 3400).length // resultado = 4
```

En el segundo ejemplo podemos ver la gran potencia de la evaluación de expresiones. Los paréntesis fuerzan la conversión del número a una cadena. Una cadena de texto en JavaScript tiene una propiedad asociada con ella que es la longitud (length), la cuál te devolverá en este caso el número 4, indicando que hay 4 caracteres en esa cadena "3400". La longitud de una cadena es un número, no una cadena.

1.4. Operadores.

JavaScript es un lenguaje rico en operadores: símbolos y palabras que realizan operaciones sobre uno o varios valores, para obtener un nuevo valor.

Cualquier valor sobre el cual se realiza una acción (indicada por el operador), se denomina un operando. Una **expresión** puede contener un operando y un operador (denominado operador unario), como por ejemplo en **b++**, o bien dos operandos, separados por un operador (denominado operador binario), como por ejemplo en **a + b**.

Categorías de operadores en JavaScript	
Tipo	**Qué realizan**
Comparación.	Comparan los valores de 2 operandos, devolviendo un resultado de true o false (se usan extensivamente en sentencias condicionales como `if...` `else` y en instrucciones `loop`). `==` `!=` `===` `!==` `>` `>=` `<` `<=`
Aritméticos.	Unen dos operandos para producir un único valor que es el resultado de una operación aritmética u otra operación sobre ambos operandos. `+` `-` `*` `/` `%` `++` `- -` `+valor` `-valor`
Asignación.	Asigna el valor a la derecha de la expresión a la variable que está a la izquierda. `=` `+=` `-=` `*=` `/=` `%=` `<<=` `>=` `>>=` `>>>=` `&=` `|=` `^=` `[]`
Boolean.	Realizan operaciones booleanas aritméticas sobre uno o dos operandos booleanos. `&&` `||` `!`
Bit a Bit.	Realizan operaciones aritméticas o de desplazamiento de columna en las representaciones binarias de dos operandos. `&` `|` `^` `~` `<<` `>>` `>>>`
Objeto.	Ayudan a los scripts a evaluar la herencia y capacidades de un objeto particular antes de que tengamos que invocar al objeto y sus propiedades o métodos. `.` `[]` `()` `delete` `in` `instanceOf` `new` `this`
Misceláneos.	Operadores que tienen un comportamiento especial. `,` `?:` `typeof` `void`

Operadores. **http://www.htmlpoint.com/javascript/corso/js_30.htm**

1.4.1. Operadores de comparación.

Operadores de comparación en JavaScript			
Sintaxis	Nombre	Tipos de operandos	Resultados
==	Igualdad.	Todos.	Boolean.
!=	Distinto.	Todos.	Boolean.
===	Igualdad estricta.	Todos.	Boolean.
!==	Desigualdad estricta.	Todos.	Boolean.
>	Mayor que .	Todos.	Boolean.
>=	Mayor o igual que.	Todos.	Boolean.
<	Menor que.	Todos.	Boolean.
<=	Menor o igual que.	Todos.	Boolean.

En valores numéricos, los resultados serían los mismos que obtendríamos con cálculos algebraicos.

Por ejemplo:

```
30 == 30 // true
30 == 30.0 // true
5 != 8 // true
9 > 13 // false
7.29 <= 7.28 // false
```

También podríamos comparar cadenas a este nivel:

```
"Marta" == "Marta" // true
"Marta" == "marta" // false
"Marta" > "marta" // false
"Mark" < "Marta" // true
```

Para poder comparar cadenas, JavaScript convierte cada carácter de la cadena de un string, en su correspondiente valor ASCII. Cada letra, comenzando con la primera del operando de la izquierda, se compara con su correspondiente letra en el operando de la derecha. Los valores ASCII de las letras mayúsculas, son más pequeños que sus correspondientes en minúscula, por esa razón "Marta" no es mayor que "marta". En JavaScript hay que tener muy en cuenta la sensibilidad a mayúsculas y minúsculas.

Si por ejemplo comparamos un número con su cadena correspondiente:

```
"123" == 123 // true
```

JavaScript cuando realiza esta comparación, convierte la cadena en su número correspondiente y luego realiza la comparación. También dispones de otra opción, que consiste en convertir mediante las funciones **parseInt()** o **parseFloat()** el operando correspondiente:

```
parseInt("123") == 123 // true
```

Los operadores `===` y `!==` comparan tanto el dato como el tipo de dato. El operador `===` sólo devolverá `true`, cuando los dos operandos son del mismo tipo de datos (por ejemplo ambos son números) y tienen el mismo valor.

Operadores de Comparación. http://www.webestilo.com/javascript/js06.phtml

1.4.2. Operadores aritméticos.

Operadores aritméticos en JavaScript			
Sintaxis	**Nombre**	**Tipos de Operando**	**Resultados**
+	Más.	integer, float, string.	integer, float, string.
-	Menos.	integer, float.	integer, float.
*	Multiplicación.	integer, float.	integer, float.
/	División.	integer, float.	integer, float.
%	Módulo.	integer, float.	integer, float.
++	Incremento.	integer, float.	integer, float.
--	Decremento.	integer, float.	integer, float.
+valor	Positivo.	integer, float, string.	integer, float.
-valor	Negativo.	integer, float, string.	integer, float.

Veamos algunos ejemplos:

```
var a = 10; // Inicializamos a al valor 10

var z = 0; // Inicializamos z al valor 0

z = a; // a es igual a 10, por lo tanto z es igual a 10.

z = ++a; // el valor de a se incrementa justo antes de ser asignado a z,
//por lo que a es 11 y z valdrá 11.

z = a++; // se asigna el valor de a (11) a z y luego se incrementa el
//valor de a (pasa a ser 12).

z = a++; // a vale 12 antes de la asignación, por lo que z es igual a 12;
//una vez hecha la asignación a valdrá 13.
```

Otros ejemplos:

```
var x = 2;

var y = 8;

var z = -x; // z es igual a -2, pero x sigue siendo igual a 2.

z = - (x + y); // z es igual a -10, x es igual a 2 e y es igual a 8.

z = -x + y; // z es igual a 6, pero x sigue siendo igual a 2 e y igual a 8
```

1.4.3. Operadores de asignación.

Operadores de asignación en JavaScript			
Sintaxis	**Nombre**	**Ejemplo**	**Significado**
=	Asignación.	x = y	x = y
+=	Sumar un valor.	x += y	x = x + y
-=	Substraer un valor.	x -= y	x = x - y
*=	Multiplicar un valor.	x *= y	x = x * y
/=	Dividir un valor.	x /= y	x = x / y
%=	Módulo de un valor.	x %= y	x = x % y
<<=	Desplazar bits a la izquierda.	x <<= y	x = x << y
>=	Desplazar bits a la derecha.	x >= y	x = x > y
>>=	Desplazar bits a la derecha rellenando con 0.	x >>= y	x = x >> y
>>>=	Desplazar bits a la derecha.	x >>>= y	x = x >>> y
&=	Operación AND bit a bit.	x &= y	x = x & y
|=	Operación OR bit a bit.	x |= y	x = x | y
^=	Operación XOR bit a bit.	x ^ = y	x = x ^ y
[]=	Desestructurando asignaciones.	[a,b]=[c,d]	a=c, b=d

Operadores. http://www.w3schools.com/js/js_operators.asp

1.4.4. Operadores booleanos.

Debido a que parte de la programación tiene un gran componente de lógica, es por ello, que los operadores booleanos juegan un gran papel.

Los operadores booleanos te van a permitir evaluar expresiones, devolviendo como resultado `trae` (verdadero) o `false` (falso).

Operadores de boolean en JavaScript			
Sintaxis	Nombre	Operandos	Resultados
&&	AND.	Boolean.	Boolean.
||	OR.	Boolean.	Boolean.
!	Not.	Boolean.	Boolean.

Ejemplos:

```
!true // resultado = false
!(10 > 5) // resultado = false
!(10 < 5) // resultado = true
!("gato" == "pato") // resultado = true
5 > 1 && 50 > 10 // resultado = true
5 > 1 && 50 < 10 // resultado = false
5 < 1 && 50 > 10 // resultado = false
5 < 1 && 50 < 10 // resultado = false
```

T

Tabla de valores de verdad del operador AND			
Operando Izquierdo	Operador AND	Operando Derecho	Resultado
True	&&	True	True
True	&&	False	False
False	&&	True	False
False	&&	False	False

Tabla de valores de verdad del operador OR			
Operando Izquierdo	Operador OR	Operando Derecho	Resultado
True	||	True	True
True	||	False	True
False	||	True	True
False	||	False	False

Ejemplos:

```
5 > 1 || 50 > 10 // resultado = true
5 > 1 || 50 < 10 // resultado = true
5 < 1 || 50 > 10 // resultado = true
5 < 1 || 50 < 10 // resultado = false
```

1.4.5. Operadores bit a bit.

Para los programadores de scripts, las operaciones bit a bit suelen ser un tema avanzado. A menos que tú tengas que gestionar procesos externos en aplicaciones del lado del servidor, o la conexión con applets de Java, es raro que tengas que usar este tipo de operadores.

Los operandos numéricos, pueden aparecer en JavaScript en cualquiera de los tres formatos posibles (decimal, octal o hexadecimal). Tan pronto como el operador tenga un operando, su valor se convertirá a representación binaria (32 bits de longitud). Las tres primeras operaciones binarias bit a bit que podemos realizar son **AND**, **OR** y **XOR** y los resultados de comparar bit a bit serán:

Bit a bit AND: 1 si ambos dígitos son 1.

Bit a bit OR: 1 si cualquiera de los dos dígitos es 1.

Bit a bit XOR: 1 si sólo un dígito es 1.

Tabla de operador Bit a Bit en JavaScript			
Operador	Nombre	Operando izquierdo	Operando derecho
&	Desplazamiento AND.	Valor integer.	Valor integer.
|	Desplazamiento OR.	Valor integer.	Valor integer.
^	Desplazamiento XOR.	Valor integer.	Valor integer.
~	Desplazamiento NOT.	(Ninguno).	Valor integer.
<<	Desplazamiento a la izquierda.	Valor integer.	Cantidad a desplazar.
>>	Desplazamiento a la derecha.	Valor integer.	Cantidad a desplazar.
>>>	Desplazamiento derecha rellenando con 0.	Valor integer.	Cantidad a desplazar.

Por ejemplo:

```
4 << 2 // resultado = 16
```

La razón de este resultado es que el número decimal **4** en binario es `00000100`. El operador `<<` indica a JavaScript que desplace todos los dígitos dos lugares hacia la izquierda, dando como resultado en binario `00010000`, que convertido a decimal te dará el valor **16**.

En el siguiente enlace podrás ver ejemplos de operaciones a nivel de bits:

http://www.mundoprogramacion.com/net/dotnet/operar_con_bits.aspx

1.4.6. Operadores de objeto.

El siguiente grupo de operadores se relaciona directamente con objetos y tipos de datos. La mayor parte de ellos fueron implementados a partir de las primeras versiones de JavaScript, por lo que puede haber algún tipo de incompatibilidad con navegadores antiguos.

1.4.6.1. . (punto)

El operador punto, indica que el objeto a su izquierda tiene o contiene el recurso a su derecha, como por ejemplo: **objeto.propiedad** y **objeto.método()**.

Ejemplo con un objeto nativo de JavaScript:

```
var s = new String('rafa');
var longitud = s.length;
var pos = s.indexOf("fa"); // resultado: pos = 2
```

1.4.6.2. [] (corchetes para enumerar miembros de un objeto).

Por ejemplo cuando creamos un array: `var a =["Santiago","Coruña", "Lugo"];`

Enumerar un elemento de un array: `a[1] = "Coruña";`

Enumerar una propiedad de un objeto: `a["color"]= "azul";`

1.4.6.3. Delete (para eliminar un elemento de una colección).

Por ejemplo si consideramos:

```
var oceanos = new Array("Atlantico", "Pacifico", "Indico","Artico");
```

Podríamos hacer:

```
delete oceanos[2];
```

Esto eliminaría el tercer elemento del array ("Indico"), pero la longitud del array no cambiaría. Si intentamos referenciar esa posición oceanos[2] obtendríamos undefined.

1.4.6.4. In (para inspeccionar métodos o propiedades de un objeto).

El operando a la izquierda del operador, es una cadena referente a la propiedad o método (simplemente el nombre del método sin paréntesis); el operando a la derecha del operador, es el objeto que estamos inspeccionando. Si el objeto conoce la propiedad o método, la expresión devolverá `true`.

Ejemplo: `"write" in document` o también `"defaultView" in document`

1.4.6.5. instanceof (para comprobar si un objeto es una instancia de un objeto nativo de JavaScript).

Ejemplo:

```
a = new Array(1,2,3);

a instanceof Array; // devolverá true.

//new (para acceder a los constructores de objetos incorporados en el
//núcleo de JavaScript).
```

Ejemplo:

```
var hoy = new Date();

// creará el objeto hoy de tipo Date() empleando el constructor por
//defecto de dicho objeto. this (para hacer referencia al propio objeto en
//el que estamos localizados).
```

Ejemplo:

```
nombre.onchange = validateInput;

function validateInput(evt)

{

var valorDeInput = this.value;

// Este this hace referencia al objeto nombre que estamos validando.

}
```

1.4.7. Operadores misceláneos.

1.4.7.1. El operador coma ,

Este operador, indica una serie de expresiones que van a ser evaluadas en secuencia, de izquierda a derecha. La mayor parte de las veces, este operador se usa para combinar múltiples declaraciones e inicializaciones de variables en una única línea.

Ejemplo:

```
var nombre, direccion, apellidos, edad;
```

Otra situación en la que podemos usar este operador coma, es dentro de la expresión **loop**. En el siguiente ejemplo inicializamos dos variables de tipo contador, y las incrementamos en diferentes porcentajes. Cuando comienza el bucle, ambas variables se inicializan a 0 y a cada paso del bucle una de ellas se incrementa en 1, mientras que la otra se incrementa en 10.

```
for (var i=0, j=0 ; i < 125; i++, j+10)
{
// más instrucciones aquí dentro
}
```

Nota: no confundir la coma, con el delimitador de parámetros ";" en la instrucción **for**.

1.4.7.2. ? : (operador condicional)

Este operador condicional es la forma reducida de la expresión **if .... else**. La sintaxis formal para este operador condicional es: **condicion ? expresión si se cumple la condición: expresión si no se cumple;**

Si usamos esta expresión con un operador de asignación:

```
var = condicion ? expresión si se cumple la condición: expresión si no se cumple;
```

Ejemplo:

```
var a,b;
a = 3; b = 5;
var h = a > b ? a : b; // a h se le asignará el valor 5;
```

1.4.7.3. typeof (devuelve el tipo de valor de una variable o expresión).

Este operador unario se usa para identificar cuando una variable o expresión es de alguno de los siguientes tipos: **number, string, boolean, object, function** o **undefined.**

Ejemplo:

```
if (typeof miVariable == "number")
{
miVariable = parseInt(miVariable);
}
```

1.5. Condiciones y bucles.

En esta sección te mostraremos cómo los programas pueden tomar decisiones, y cómo puedes lograr que un script repita un bloque de instrucciones las veces que quieras.

Cuando te levantas cada día tomas decisiones de alguna clase; muchas veces ni te das cuenta de ello, pero lo estás haciendo. Por ejemplo, imagínate que vas a hacer la compra a un supermercado; desde el momento que entras en el supermercado ya estás tomando decisiones: ¿compro primero la leche o

compro la verdura?, ¿ese precio es barato o es caro?, ¿el color de ese tinte es azul claro u oscuro?, ¿tengo suficiente dinero para pagar o no?, ¿me llegan estos kilogramos de patatas o no?, ¿pago en efectivo o tarjeta?, etc.

Es decir, tomamos innumerables decisiones a lo largo del día y la mayor parte de las veces no nos damos ni cuenta de ello.

En las siguientes secciones, verás cómo puedes ejecutar unas u otras instrucciones, dependiendo de ciertas condiciones, y cómo puedes repetir una o varias instrucciones, las veces que te hagan falta.

1.5.1. Estructuras de control.

En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bucles de ejecución, se denominan "Estructuras de Control". Una estructura de control, dirige el flujo de ejecución a través de una secuencia de instrucciones, basadas en decisiones simples y en otros factores.

Una parte muy importante de una estructura de control es la "condición". Cada condición es una expresión que se evalúa a `true` o `false`.

En JavaScript tenemos varias estructuras de control, para las diferentes situaciones que te puedas encontrar durante la programación. Tres de las estructuras de control más comunes son: construcciones **if**, construcciones **if...else** y los **bucles**.

1.5.1.1. Construcción if

La decisión más simple que podemos tomar en un programa, es la de seguir una rama determinada si una determinada condición es `true` .

Sintaxis:

```
if (condición) // entre paréntesis irá la condición que se evaluará a true
//o false.
{
// instrucciones a ejecutar si se cumple la condición
}
```

Ejemplo:

```
if (miEdad >30)
{
alert("Ya eres una persona adulta");
}
```

1.5.1.2. Construcción if … else

En este tipo de construcción, podemos gestionar que haremos cuando se cumpla y cuando no se cumpla una determinada condición.

Sintaxis:

```
if (condición) // entre paréntesis irá la condición que se evaluará a true
//o false.
{
// instrucciones a ejecutar si se cumple la condición
}
else
{
// instrucciones a ejecutar si no se cumple la condición
}
```

Ejemplo:

```
if (miEdad >30)
{
alert("Ya eres una persona adulta.");
}
else
{
alert("Eres una persona joven.");
}
```

Ejemplos de if..else. http://www.gratismil.com/apuntes/javascript/21-if-else-en-javascript.htm

1.5.2. Bucles.

Los bucles son estructuras repetitivas, que se ejecutarán un número de veces fijado expresamente, o que dependerá de si se cumple una determinada condición.

1.5.2.1. Bucle for.

Este tipo de bucle te deja repetir un bloque de instrucciones un número limitado de veces.

Sintaxis:

```
for (expresión inicial; condición; incremento)
{
// Instrucciones a ejecutar dentro del bucle.
}
```

Ejemplo:

```
for (var i=1; i<=20; i++)
{
// instrucciones que se ejecutarán 20 veces.
}
```

1.5.2.2. Bucle while().

Este tipo de bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo de comprender que el bucle **FOR**, ya que no incorpora en la misma línea la inicialización de las variables, su condición para seguir ejecutándose y su actualización. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración o repetición.

Sintaxis:

```
while (condición)
{
// Instrucciones a ejecutar dentro del bucle.
}
```

Ejemplo:

```
var i=0;
while (i <=10)
{
// Instrucciones a ejecutar dentro del bucle hasta que i sea mayor que 10
//y no se cumpla
la condición.
i++;
}
```

1.5.2.3. Bucle do … while().

El tipo de bucle **do…while** es la última de las estructuras para implementar repeticiones de las que dispone JavaScript, y es una variación del bucle **while()** visto anteriormente. Se utiliza generalmente, cuando no sabemos el número de veces que se habrá de ejecutar el bucle. Es prácticamente igual que el bucle **while()**, con la diferencia, de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

Sintaxis:

```
do {
// Instrucciones a ejecutar dentro del bucle.
}while (condición);
```

Ejemplo:

```
var a = 1;
do{
alert("El valor de a es: "+a); // Mostrará esta alerta 2 veces.
a++;
}while (a<3);
```

1.5.3. Ejemplo sencillo con JavaScript.

Aquí se muestra el código fuente, de un pequeño ejemplo de una aplicación en JavaScript. Simplemente copia el código, pégalo en tu editor favorito y guarda el archivo con la extensión .html

Para probar la aplicación haz doble click en el fichero .html y visualízalo con tu navegador.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo Básico de JavaScript</title>
</head>
<body>
<h2>Código fuente de un ejemplo básico con JavaScript
</h2>
<script type="text/javascript">
// Definimos variables.
var nombre, apellidos, edad, hermanos;
nombre="Rafa";
apellidos="Martinez Díaz";
```

```
edad=38;
hermanos=3;
// Imprimo el nombre y los apellidos.
document.write("Hola " + nombre + " " + apellidos);
// Imprimo un salto de línea.
document.write("<br/>");
// Imprime la edad y el número de hermanos.
document.write(nombre + " tienes " + edad + " años y además tienes " +
hermanos + "
hermanos.<br/>");
// Fíjate en la diferencia entre las dos siguientes lineas.
document.write("Dentro de 15 años tu edad será " + edad + 15 + ".<br/>");
document.write("Dentro de 15 años tu edad será " + (edad+15) + ".<br/>");
// Tu nombre escrito 50 veces.
for (i=1; i<=50; i++)
{
document.write(nombre + ",");
}
</script>
</body>
</html>
```

Información y ejemplos sobre JavaScript. **http://www.w3schools.com/js/default.asp**