JAVASCRIPT

Donde incluirlo

Antes siquiera de que conozcamos la sintaxis o una primera orden de JavaScript, debemos saber primero cómo se incluye un script dentro de un documento HTML.
El código JavaScript se inserta directamente en nuestra página HTML. Hay cuatro (4) maneras de hacerlo:

1. En el cuerpo del documento

Es decir entre los comandos > usando el comando **<script>**

```
<html>
<head><title>Titulo</title></head>
<body>
<script Language="JavaScript">
   <!-- escondemos el codigo>
  document.write("Hola que tal");
  // fin de esconder -->
</script>
</body>
</html>
```

Este código se ejecuta inmediatamente al cargar la pagina y lo que produce es un texto sobre la pagina, para ello use el metodo **write** del objeto **document**, que representa al documento actual.
Nota que use un comentario <!-- --> para ocultar el código a los navegadores que no soportan JavaScript.

2. En archivo aparte

En este caso todo el código del script está situado en otro archivo y se hace una llamada.

```
<html>
<head><title>Titulo</title></head>
<body>
<script src="codigo.js">
</script>
</body>
</html>
```

Nota que aquí no fue necesario esconder ningún código y que los navegadores que no soporte el comando **script** simplemente lo ignoraran.

3. Usando manejadores de evento

Los comandos de JavaScript son evaluados inmediatamente al cargarse la página. Las funciones son almacenadas, pero no ejecutadas, hasta cierto evento.

```
<html>
<head><title>Titulo</title></head>
<body>
<a  href="" evento=metodo o llamada a
funcion interna>algo </a>
</body>
</html>
```

Nota que aquí es un evento el que dispara.

4. Haciendo una llamada a función

Dentro de la cabecera, después del título. Es decir, entre los comandos </title> y </head> y luego la llamada a la función en el cuerpo.

```
<html>
<head>
<title>Titulo</title>
<script Language="JavaScript">
   <!-- escondemos el codigo>
   function Ver() {
                alert("Le dije que NO !");
                    }
    // fin de esconder -->
</script>
</head>
<body>
No haga Clic <a  href="Ver()">AQUÍ </a>
</body>
</html>
```

Observa que aquí se definió la función en la cabecera, pero recién se ejecuta al hacer clic en el enlace, que es el evento que llama a la función a la cual se le para un parámetro

Tipos de datos

JavaScript acepta diferentes tipos de datos:

Tipo	Descripción	Ejemplo
Números		
Enteros Decimales (base 10)	Es una secuencia de dígitos (0-9) que no comiencen con 0	1999
Enteros Hexadecimales (base 16)	Una secuencia de dígitos (0-9) y letras (A-F) que comienza con 0x	0xE477
Enteros Octales (base 8)	Secuencia de dígitos (0-7) que comiencen con 0	0777
Punto flotante	Puede tener un entero decimal, un punto, una fracción (otro numero decimal), un exponente que consiste en la letra **e** seguida de un entero, el cual puede llevar un signo (+ o -).	3.14159, -2e4, 5e-12
Cadenas		
Cadenas de caracteres	Consta de uno o mas caracteres encerrados entre comillas simples o dobles	"Hola", '1999'
También pueden usar los siguientes caracteres	\f indica un avance de pagina (Form feed) \n Indica nueva línea (New Line) \r Indica un retorno de carro (Carruage return) \t Indica un tabulador (Tab) \" se puede incluir comillas. Ej: "José \"Chemo\" del Solar"	
Lógicas		
Lógicas	Verdadero o falso	true o false
Nulas		
Nulas	Es cuando la variable no toma ningún valor	null

Palabras Reservadas

JavaScript posee un número de “palabras reservadas”, o palabras que son especiales dentro del mismo lenguaje. Debe utilizar estas palabras cuando las necesite para su uso específico.

abstract	**default**	**for**	**native**	**synchronized**	**with**
boolean	**delete**	**function**	**new**	**this**	
break	**do**	**goto**	**package**	**throw**	
byte	**double**	**if**	**private**	**throws**	
case	**else**	**implements**	**protected**	**transient**	
catch	**enum**	**import**	**public**	**try**	
char	**export**	**in**	**return**	**typeof**	
class	**extends**	**instanceof**	**short**	**var**	
const	**final**	**int**	**static**	**void**	
continue	**finally**	**interface**	**super**	**volatile**	
debugger	**float**	**long**	**switch**	**while**	

Variables

Las variables son usadas para almacenar valores a ser evaluados. En JavaScript no es necesario declarar las variables ya que automáticamente se convierten al tipo necesario mientras se ejecutan los comandos. Por ejemplo podemos definir **indica=true** y luego asignarle un valor de otro tipo **indica="prendido"**
El nombre de una variable debe empezar por una letra o por el símbolo de subrayado (_). Lo que siga a esto es indiferente. Pero diferencia mayúsculas de minúsculas.

Por ejemplo podríamos definir como variables:

```
Nombre
_Opción15
mes3
```

Estarían mal definidas las siguientes variables:

```
7opcion
&inicio
¿nombre
```

Además también estaría mal si usáramos como variable:

```
goto
new
null
```

Debido a que son palabras reservadas del lenguaje. Una variable puede tener alcance **local** o **global**. Cuando es global se puede emplear en cualquier parte del programa. Las locales solo se pueden usar en la función donde fueron definidas. Para crear una variable local le antepondremos la palabra **var**.

Variables tipo matriz

```
<script language=“JavaScript”>
function HacerMatriz(n) {
    this.length=0;
     for (var i = 1; i < n ; i++) {
        this[i]=0;
        return this;
     }}
</script>
```

La sintaxis para crear la matriz es la siguiente: *nombre=new Array(elementos)*, posteriormente para usarla se usa *nombre[# de elemento]*. Una forma de crear las matrices de forma automática es con el siguiente código:

Operadores

Aritméticos	
+	Adición
-	Sustracción
*****	Multiplicación
/	División
%	Modulo
++	Incremento
--	Decremento
-	Negación
Cadena	
+	Concatenación.
Nota: Cuando se opera un valor de cadena con un numérico, el resultado es una cadena.	
Sobre BITs	
AND o &	Devuelve un 1 en cada bit para el cual ambos operandos sean 1 y 0 en el resto.
OR o |	Devuelve un 0 en cada bit para el cual ambos operandos sean 0 y 1 en el resto.
O-EX o ^	Devuelve un 1 en cada bit para el cual uno de los operandos sea 1 y el otro 0 y 0 en el resto.
NOT o ~	Devuelve un 1 en cada bit para el cual el operandos sean 0 y viceversa.
Lógicos	
&&	Devuelve verdadero si ambos operandos son verdaderos y falsos en el resto.
||	Devuelve falso si ambos operandos son falsos y verdaderos en el resto.
!	Devuelve falso si operando es verdadero y viceversa.
Comparación	
==	Devuelve verdadero si ambos operador son iguales.
!=	Devuelve verdadero si ambos operador son difrenetes.
>	Devuelve verdadero si operador izquierdo es mayor al derecho.
>=	Devuelve verdadero si operador izquierdo es mayor o igual al derecho.
<:	Devuelve verdadero si operador izquierdo es menor al derecho.
<=	Devuelve verdadero si operador izquierdo es menor o igual al derecho.
Asignación:	
x=y	Asigna a x el valor de y
x+=y	Asigna a x el valor de x+y
x-=y	Asigna a x el valor de x-y
x*=y	Asigna a x el valor de x*y
x/=y	Asigna a x el valor de x/y
x%=y	Asigna a x el valor de x%y
x<<=y	Asigna a x el valor de x=x<x<y
x>>y	Asigna a x el valor de x=x>x>y
x>>>=y	Asigna a x el valor de x=x>x>x>y
x&=y	Asigna a x el valor de x=x&y
x^=y	Asigna a x el valor de x=x^y
x|=y	Asigna a x el valor de x=x|y

Prioridades de operación

1	[] O ()	Agrupar
2	!, ~, ++, --	Negación, incremento y decremento.
3	*, / y %	Multiplicación, división y modulo.
4	+ y -	Adición y sustracción.
5	>> y <<	Desplazamiento
6	>, >=, <, <=	Relaciones
7	= y !=	Igualdad y desigualdad
8	&	Y sobre bits
9	^	O-EX sobre bits
10	|	O sobre bits
12	&&	AND
13	||	OR
14	?:	Condicional
15	=, +=, -=	Asignación

Código Condicional

A veces se desea ejecutar un bloque de código bajo ciertas condiciones. Las estructuras de control de flujo a través de la utilización de las declaraciones if y else permiten hacerlo.

```
var foo = true;
var bar = false;
if (bar) {
        // este código nunca se ejecutará
    console.log('hello!');
}
if (bar) {
        // este código no se ejecutará
    }
 else {
    if (foo) {
            // este código se ejecutará
        }
     else {
            // este código se ejecutará si foo y bar son falsos (false)
        }
     }
```

Control del flujo

Nota
En una línea singular, cuando se escribe una declaración if, las l laves no son estrictamente necesarias; sin embargo es recomendable su utilización, ya que hace que el código sea mucho más legible. Debe tener en cuenta de no definir funciones con el mismo nombre múltiples veces dentro de declaraciones if/else, ya que puede obtener resultados no esperados

Bucles

Un bucle es un conjunto de comandos que se ejecutan repetitivamente un cierto número de veces.

for

Permite un bucle repetitivo sabiendo de antemano el número de ejecuciones que sera necesario.
Sintaxis:

for ([inicial;][final;][incremento]) {instrucciones }

Ejemplo: for (i=0; i<4; i++){alert("Ahora van "+i)}

while

Permite un bucle repetitivo cuyo número de repeticiones dependera de una condicion. Aqui normalmente no sabemos de antemano el número de repeticiones.
Sintaxis:

while (condicion) { instrucciones }

Control de bucle

Tenemos dos comandos para el control de bucles: *break* que termina el bucle y transfiere el control del programa a la siguiente instruccion a continuacion del bucle. *continue* interrumpe la ejecucion de comandos y regresa el control al inicio del bucle.

Estructura Switch:

```
switch (foo){
   case 'bar':
     alert('el valor es bar');
   break;
   case 'baz':
     alert('el valor es baz');
   break;
   default:
    alert('de forma predeterminada se ejecutará este código');
   break;
           }
```

También existen ocasiones o programas donde se exige evaluar muchas condiciones a la vez en estos casos o se usan una condición compuesta muy grande o se debe intentar convertir el problema a uno que se pueda resolver usando la instrucción SWITCH. Esta instrucción es una instrucción de decisión múltiple donde el compilador prueba o busca el valor contenido en una variable contra lista de constantes, cuando el computador encuentra el valor de igualdad entre variale y constante entonces ejecuta el grupo de instrucciones asociados a dicha constante si no encuentra el valor de igualdad entre variable y constante, entonces ejecuta un grupo de instrucciones asociados a un default, aunque este último es opcional.
Sintaxis:

Objetos

Los objetos son elementos que pueden contener cero o más conjuntos de pares de nombres claves y valores asociados a dicho objeto. Los nombres claves pueden ser cualquier palabra o número válido. El valor puede ser cualquier tipo de valor: un número, una cadena, un arreglo, una función, incluso otro objeto.

[Definición: Cuando uno de los valores de un objeto es una función, ésta es nombrada como un método del objeto.] De lo contrario, se los llama propiedades. Curiosamente, en JavaScript, casi todo es un objeto arreglos, funciones, números, incluso cadenas y todos poseen propiedades y métodos.

Creación de un “objeto literal”

```
var myObject = {
sayHello : function() {
console.log('hello');
},
myName : 'Rebecca'
};
myObject.sayHello(); // se llama al método sayHello,
// el cual muestra en la consola 'hello'
console.log(myObject.myName); // se llama a la propiedad
//myName, la cual muestra en la consola 'Rebecca'
```

Nota: Notar que cuando se crean objetos literales, el nombre de la propiedad puede ser cualquier identificador JavaScript, una cadena de caracteres (encerrada entre comillas) o un número:

```
var myObject = {
    validIdentifier: 123,
    'some string': 456,
    99999: 789
};
```

Los objetos literales pueden ser muy útiles para la organización del código. Existen los objetos en javascript y existe la herencia donde objetos heredan de otros objetos (lo veremos más abajo), existe el foreach y existen los arrays estos últimos con estructura limitada. Hay 3 formas de crear un objeto en javascript

1-Mediante literales, declaración directa

Cuando el objeto se crea usando literales, posee un link al prototipo Object, de esta manera si extendemos a Object extendemos también a los objetos. Esto es: agrego un nuevo método a Object y todos los objetos pasan a tener ese método sin importar en qué lugar del código fueron creados, con esto digo , creo un objeto 1 luego extiendo a Objecto con el método 2 luego el objeto 1 por más que fue creado con anterioridad pasa a contener el método 2.

```
var objeto = new Objet();

//ó también

var objeto ={} // objeto vacio
var objeto ={name:'lucas',lastname:'forchino'}; //objeto con atributos

//objeto con métodos
var objeto ={
        name:'lucas',
       lastname:'forchino',
       getName:function(){return this.name}
}
```

```
var objeto={name:'lucas',lastname:'forchino'};//objeto con atributos a clonar

Creamos la funcion vacia

var F=function(){} ;

Copiamos el objeto dentro del prototipo de la función
 F.prototype = objeto;

Obtenemos el objeto nuevo
objeto2 = new F();
```

2-Crear un objeto clonando usando un truco muy simple

Para clonar un objeto de otro objeto se usa truco es muy simple, se crea una función vacía, a esta se le extiende el prototipo asignándole el objeto a clonar, de esta manera obtenemos una función constructora con la estructura del objeto, posteriormente se crea el nuevo objeto usando new. Una vez aprendida esta técnica se puede meter dentro de una función para hacerlo más rápido y despreocuparse de su implementación. En si lo que se hace es usar el método de creación 3 (a continuación) adaptado a objetos que no tienen constructor.

3- Si contiene constructor, invocarlo mediante el operador new

Una variable que contiene una función es utilizada como constructora de un nuevo objeto. Estas variables por convención deben empezar con mayúscula y serian de la forma:

```
var Objeto = function(){
this.name='lucas';
this.lastname='forchino';
}
```

```
var objectFactory=function(){
// variable solo accesible por la función siguiente debido a
//closure
var variable_privada='31';

// función u objeto que se retorna, que tiene acceso a la
//variable anterior
     return function(){
          this.name ="lucas";
          this.apellido="forchino";
          this.getEdad=function(){
                    return variable_privada;
               }
          }
}
```

Utilizando la función constructora genero un objeto nuevo.

var objeto2 = new Objeto;

Otra forma de crear un objeto de deriva de las formas anteriores es retornando el objeto por medio de una función, en si la creación se produce dentro de la función y se asigna su resultado a una variable que contendrá el objeto. Este método es muy interesante ya que, debido al closure, permite hacer cosas como mantener atributos privados dentro de un objeto. Luego para obtener un objeto nuevo hacemos

var objeto2= objectFactory();

Herencia en javascript

Hay varias formas de crear herencia en javascript, solo voy mostrar las dos más simples y comunes.

```
var Objeto = function(){
    this.name='lucas';
   this.lastname='forchino';
}

var objeto1= new Objeto;
var objeto2= new Objeto;

//extiendo el objeto 2 añadiendo un atributo y un método
objeto2.middleName='francisco';
objeto2.getMiddleName= funcion(){
        return this.middleName;
}
```

1-Sobrecargando directamente un objeto
Algo muy simple de pensar si tengo dos objetos iguales creados de la misma forma, incorporando métodos a uno estoy extendiendo el objeto original. Tenemos

De esta forma el objeto1 se mantiene su estructura original mientras que el objeto2 fue extendido de Objeto agregando un atributo y un método.

2-Añadir el objeto padre en el prototipo del objeto hijo

```
// creo el constructor del objeto padre
var Padre=function(){  this.name='jose';  this.apellido='forchino';}
// constructor del objeto hijo
var Hijo=function(){ this.name='lucas';}

// heredo del padre añadiendo el objeto padre al prototipo del hijo usando new ! muy importante de otra
//manera asignariamos la función al prototipo y no el objeto
Hijo.prototype=new Padre();
var obj= new Hijo; // creo un objeto hijo

//el objeto hijo contiene la estructura del padre y del hijo, los atributos o métodos que prevalecen son los del
// hijo por sobrecargar al padre
document.writeln(obj.name); // lucas
document.writeln(obj.apellido); //forchino
Padre.prototype.edad=56; // si cambio el prototipo de la función padre
// todos lo hijos cambian su prototypo , sin importar si ya fueron creados, todos pasan a leer el atributo
//añadido.
document.writeln(obj.edad); // 56
```

Scope de variables en javascript.

Este es uno de los temas más simples y más complejos a la vez un error podría hacer que nuestro programa tenga comportamientos extraños. Hay 4 tipos de scope que puede obtener el metodo this en javascript

1- Dentro de la función u objeto donde se hace referencia a la misma función u objeto.
Todos los métodos de un objeto cuando invocan "this" hacen referencia a su mismo objeto, esto es

```
Objeto={
     name:'lucas'.
   getName:function(name){
           return this.name;
              }
}
```

El return devolverá el valor guardado en name del objeto ya que "this" se refiere al objeto mismo, para complicar mas las cosas pase por parametro un valor name, este valor name es una variable local que no influye en el valor devuelto y no tiene nada que ver con el valor name del objeto, de esta manera podemos trabajar con ambas variables name sin problemas haciendo referencia a una u otra con o sin "this".

2- Fuera del objeto donde se hace referencia al entorno global.

Cualquier función que no está dentro de un objeto , es decir no es un método de algún objeto , “this” hará referencia a window, esto si uno se lo imagina como funciones generales podría llegar a pensar que no hay problemas ya que si tenemos una función suelta por ahí podría ser válido que “this” llame a lo que tiene inmediatamente arriba que es el objeto window del cual depende.
El problema surge cuando tenemos una función dentro de otra función y es ahi donde uno espera que las funciones internas hagan referencia con “this” a las externas cuando en realidad por un error del lenguage todas las funciones apuntan con this al ámbito global. Es por esto que si tenemos

```
var a= function(){
  this.contenido=”a”;
 this.b = function(){
        this.contenido=”b”;
     }
}
var contenido=”;
a();
// contenido vale “a”

b(); // puedo llamar a b mágicamente
// contenido vale “b”
```

Lo que sucede es que a modifica la variable global porque “this” hace referencia a ese ámbito, luego crea un función que se llama b sobre el entorno global porque la misma situación, si ejecutamos b() también modificará la variable global porque al no ser un método de un objeto también modifica el ámbito global.

3- Dentro de un constructor donde se referencia a un nuevo objeto.

Dentro del constructor o función constructora, “this” hace referencia al objeto que se está creando. La creación de un objeto usando un constructor funciona de la siguiente manera hacemos new sobre una variable que contiene una función y esto internamente crea objeto vacio que se pasa como ámbito u objeto “this” a la función constructora, la función cuando trabaja sobre métodos u atributos usando “this” está trabajando sobre ese nuevo objeto creado, luego simplemente lo retorna por más que nuestra función no tenga un return.
Si nuestra función constructora tuviera un return, la función trabajaría sobre el objeto nuevo pero devolvería lo que retornamos con return. Hay que tener en cuenta que en estos casos solo se pueden retornar objetos, como una función es un objeto, también es válido retornar una función, si lo que se retorna no es un objeto la función retornara el objeto inicialmente creado dejando de lado el valor puesto en return. Esto es:

```
var Objeto = function(){
this.name=’lucas’;
}
var ob= new Objeto(); // ob = {name:’lucas’}
```

a- No tengo Return
Como resultado obtengo un nuevo objeto que fue previamente pasado a la función y al cual esta hizo referencia con “this”

```
var Objeto = function(){
this.name=’lucas’;
}
var ob= new Objeto(); // ob = {name:’lucas’}
```

b- Retorno un string
Obtengo el mismo resultado que el caso 1 , ya que el return no es un objeto

```
var Objeto = function(){
   this.name=’lucas’;
   return {};}
var ob= new Objeto();// ob ={}
```

c- Retorno un Objeto
Obtengo el objeto retornado, en este caso es un objeto vacio

```
var Objeto = function(){
   this.name=’lucas’;
   return function(){}; }
ob= new Objeto();// ob =function
```

d- Retorno una función
Obtengo una función puesto que como ya dijimos las funciones son objetos también.

4- Usando el método apply forzamos el scope a que tome un valor definido.

El método apply toma dos argumentos, el primero es el ámbito y el segundo son parámetros, de esta forma podemos ejecutar una función diciéndole cual es el ámbito que queremos que tome y cual son los parámetros que tomará.

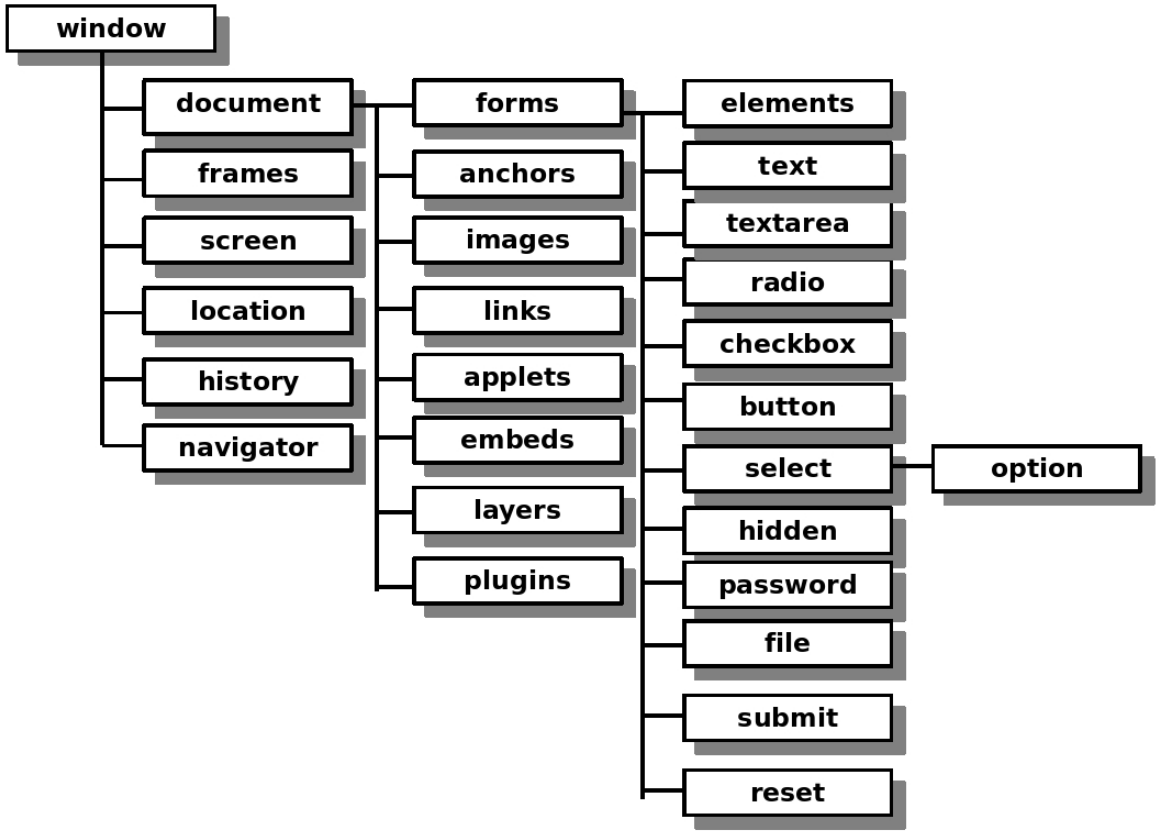
En este caso la función a correrá con el ámbito “this” que se paso por parámetro y hacen referencia al ámbito en el que se encuentra a, también podríamos reemplazar “this” con un objeto function o lo que sea que se quiera como ámbito, los parámetros en este caso representados por params deben ser un array con valores que serán los que recibirá la función “a”

```
var a = function (params){}

a.apply(this,params);
```

Jerarquía de Objetos

Cada vez que se carga una página en el navegador, el intérprete de JavaScript crea automáticamente una serie de objetos que pueden ser usados al programar en JavaScript. De entre los distintos objetos generados automáticamente destacan los siguientes:

Window: es el objeto principal del cual "cuelgan" los demás como si fueran propiedades suyas. Se crea un objeto de este tipo para cada ventana que pueda ser lanzada desde una página Web.
Navigator: contiene información acerca del navegador, tal como nombre, versión, tipos MIME soportados por el cliente y plugs-in instalados.
Location: con información acerca del URL que estemos visualizando.
History: historial en el que se guardan los URL ya visitados.
Document: es el contenedor de todos los objetos que se hayan insertado en la página web: enlaces, formularios, imágenes o marcos.
Frames: Matriz conteniendo los distintos objetos frame que puede contener una ventana. Cada frame es a su vez otra ventana.
El siguiente cuadro muestra la jerarquía de objetos creados por el navegador al cargar una página Web:



Objeto window:

Como se ha dicho, el objeto **window** es el más importante. A partir de él se pueden crear nuevos objetos **window** que serán nuevas ventanas ejecutando el navegador. Tales ventanas se pueden controlar desde la ventana padre. Además permite cerrar ventanas, actúa sobre los marcos, puede sacar mensajes (de error u otro tipo), confirmación y entrada de datos, y como se ha dicho, mantiene una matriz por cada tipo de elemento que puede haber en un documento, formularios, enlaces, imágenes y marcos.

Propiedades	**DefaultStatus**	Mensaje por omisión en la barra de estado de la ventana
	document	como document
	Frame	como Frame
	frames	Es un arreglo que contiene todos los recuadros de la ventana.
	length	Indica el numero de recuadros de la ventana madre
	location	Como location.
	name	refleja el argumento nombre con que se creó la ventana
	parent	Es un sinónimo del argumento nombre y se refiere a una ventana con recuadros.
	self	Es un sinónimo del argumento nombre y se refiere a la ventana en uso.
	status	Especifica un mensaje a presentar en la barra de estado de la ventana.
	top	Es un sinónimo del argumento nombre y se refiere a una ventana de nivel superior del navegador.
	window	Es un sinónimo del argumento nombre y se refiere a la ventana en uso
Métodos	**open**	Abre una nueva ventana en tu navegador, con una página en blanco o la URL que tú indiques.
	close	Cierra la ventana activa o indicada.
	alert	Muestra una caja de alerta con un mensaje y el botón de aceptar.
	confirm	Muestra una caja de información con un mensaje y los botones aceptar y cancelar.
	prompt	Muestra una caja de dialogo con un mensaje y un campo para entrada de datos y los botones aceptar y cancelar.
	setTimeout	Evalúa una expresión después de transcurrido un tiempo en milisegundos.
	clearTimeout	Elimina la ejecución de sentencias asociadas a un tiempo de espera indicadas con el método setTimeout().
Eventos	**onLoad**	Se ejecuta cuando el navegador termina de cargar una ventana.
	onUnload	Se ejecuta cuando el navegador descarga la página.

Objeto document:

Por cada página cargada en una ventana hay un objeto **document** contenedor de todos los elementos visibles de la página. Estos elementos se tratan con más detalle en los capítulos sucesivos. Aquí nos vamos a encargar de hablar de las propiedades y métodos específicos de este objeto.

Metodos	**write**	Escribe una o más expresiones de HTML en el documento contenido en la ventana indicada. document.write(exp1[,exp2]...[,expN])
	writeln	Escribe una o más expresiones de HTML en el documento contenido en la ventana indicada, seguidas de un caracter de nueva linea. document.writeln(exp1[,exp2]...[,expN])
Propiedades	**bgColor**	Corresponde al parámetro bgcolor del comando “body”.
	fgColor	Corresponde al parámetro text del comando “body”
	linkColor	Corresponde al parámetro link del comando “body”
	vlinkColor	Corresponde al parámetro vlink del comando “body”
	location	URL completo del documento

Objeto history

```
<html><head><title>Ejemplos en JavaScript</title>
<body>
<form>
  <input type="button" value="Atras" onClick="history.back();">
  <input type="button" value="2 Atras" onClick="history.go(-2);">
</form></body></html>
```

Es bien sabido que el visualizador guarda un historial de los sitios que se han visitado en cada sesión, de modo que podemos utilizar los botones de anterior y siguiente para movernos por las páginas. Todas las direcciones que se han visitado se guardan en esta matriz, de modo que podemos utilizarla para crear nuestros propios botones de retroceso y avance dentro de las páginas. Sin embargo, no podemos acceder a los nombres de las direcciones URL ya que son información privilegiada y solo las puede saber el usuario.

Métodos	**current**	Cadena que contiene la URL completa de la entrada actual en el historial.
	next	Cadena que contiene la URL completa de la siguiente entrada en el historial.
	length	Entero que contiene el número de entradas del historial (i.e., cuántas direcciones han sido visitadas).
	previous	Cadena que contiene la URL completa de la anterior entrada en el historial.
Propiedades	**back()**	Vuelve a cargar la URL del documento anterior dentro del historial.
	forward()	Vuelve a cargar la URL del documento siguiente dentro del historial.
	go(posicion)	Vuelve a cargar la URL del documento especificado por posición dentro del historial. Posición puede ser un entero, en cuyo caso indica la posición relativa del documento dentro del historial; o puede ser una cadena de caracteres, en cuyo caso representa toda o parte de una URL que esté en el historial.

Objeto location

Este objeto contiene la URL actual así como algunos datos de interés respecto a esta URL. Su finalidad principal es, por una parte, modificar el objeto location para cambiar a una nueva URL, y extraer los componentes de dicha URL de forma separada para poder trabajar con ellos de forma individual si es el caso. Recordemos que la sintaxis de una URL era: protocolo://maquina_host[:puerto]/camino_al_recurso

Métodos	**reload().**	Vuelve a cargar la URL especificada en la propiedad href del objeto location.
	replace(cadenaURL)	Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.
Propiedades	**hash**	Cadena que contiene el nombre del enlace, dentro de la URL.
	host	Cadena que contiene el nombre del servidor y el número del puerto, dentro de la URL.
	hostname	Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL.
	href	Cadena que contiene la URL completa.
	pathname	Cadena que contiene el camino al recurso, dentro de la URL.
	port	Cadena que contiene el número de puerto del servidor, dentro de la URL.
	protocol	Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
	search	Cadena que contiene la información pasada en una llamada a un script, dentro de la URL.

```
<script LANGUAGE="JavaScript">
   document.write("Location <b>href</b>: " + location.href + "<br>");
   document.write("Location <b>host</b>: " + location.host + "<br>");
   document.write("Location <b>hostname</b>: " + location.hostname + "<br>");
   document.write("Location <b>pathname</b>: " + location.pathname + "<br>");
   document.write("Location <b>port</b>: " + location.port + "<br>");
   document.write("Location <b>protocol</b>: " + location.protocol + "<br>");
</script>
```

Objeto navigator

```
<script LANGUAGE="JavaScript">
   document.write("appCodeName</b>:" + navigator.appCodeName +"<br>");
   document.write("appName</b>:" + navigator.appName + "<br>");
   document.write("appVersion</b>:" + navigator.appVersion + "<br>");
   document.write("language</b>:" + navigator.language + "<br>");
   document.write("platform</b>:" + navigator.platform + "<br>");
   document.write("userAgent</b>:" + navigator.userAgent + "<br>");
</script>
```

Este objeto simplemente nos da información relativa al navegador que esté utilizando el usuario.

Metodos	**javaEnabled().**	Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.
Propiedades	**appCodeName**	Cadena que contiene el nombre del código del cliente.
	appName	Cadena que contiene el nombre del cliente.
	appVersion	Cadena que contiene información sobre la versión del cliente.
	language	Cadena de dos caracteres que contiene información sobre el idioma de la versión del cliente.
	mimeTypes	Array que contiene todos los tipos MIME soportados por el cliente. A partir de NS 3.
	platform	Cadena con la plataforma sobre la que se está ejecutando el programa cliente.
	plugins	Array que contiene todos los plug-ins soportados por el cliente. A partir de NS 3.
	userAgent	Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades appCodeName y appVersion.

Objeto link

```
<HTML> <HEAD>  <title>Ejemplo de JavaScript</title> </HEAD>
<BODY>
<a href="http://www.yahoo.com" target="_blank">Yahoo!!</a><br><br>
<a href="http://www.google.com/search?q=crear+paginas+web">Google!</a><br><br>
<script LANGUAGE="JavaScript"> var i;
for (i=0;i<document.links.length;i++)
  {
  document.write("Target : " + document.links[i].target + "<br>");
  document.write("Host : " + document.links[i].host + "<br>");
  document.write("Href : " + document.links[i].href + "<br>");
  document.write("Search : " + document.links[i].search + "<br>");
  document.write("<br><br>");
   }
</script> </BODY> </HTML>
```

Este objeto engloba todas las propiedades que tienen los enlaces externos al documento actual.

Propiedades	**target**	Es una cadena que tiene el nombre de la ventana o del frame especificado en el parámetro TARGET
	hash	Es una cadena con el nombre del enlace, dentro de la URL
	host	Es una cadena con el nombre del servidor y número de puerto, dentro de la URL
	hostname	Es una cadena con el nombre de dominio del servidor (o la dirección IP) dentro de la URL
	href	Es una cadena con la URL completa
	pathname	Es una cadena con el camino al recurso, dentro de la URL
	port	Es una cadena con el número de puerto, dentro de la URL
	protocol	Es una cadena con el protocolo usado, incluyendo los : (los dos puntos), dentro de la URL
	search	Es una cadena que tiene la información pasada en una llamada a un script, dentro de la URL

Objeto image

Gracias a este objeto (disponible a partir de la versión 3 de Netscape, aunque Microsoft lo adoptó en la versión 4 de su navegador) vamos a poder manipular las imágenes del documento, pudiendo conseguir efectos como el conocido rollover (cambio de imágenes al pasar el ratón sobre la imagen).

Propiedades	**border**	Contiene el valor del parámetro 'border' de la imagen.
	complete	Es un valor booleano que nos dice si la imagen se ha descargado completamente o no.
	height	Contiene el valor del parámetro 'height' de la imagen.
	hspace	Definen el espacio horizontal o vertical de una imagen flotante respecto al texto que la rodea.
	lowsrc	Fija la ruta de una imagen accesoria, generalmente una versión de menos peso en Ks de la imagen principal, que se debe cargar en pantalla mientras se recibe ésta.
	name	Contiene el valor del parámetro 'name' de la imagen.
	src	Contiene el valor del parámetro 'src' de la imagen o sea la ruta.
	vspace	Definen el espacio horizontal o vertical de una imagen flotante respecto al texto que la rodea.
	width	Contiene el valor del parámetro 'width' de la imagen.

```
<html>
<head>
   <title>Ejemplo de JavaScript</title>
</head>
<script language="JavaScript">
   img1 = new Image();
   img1.src = "/graficos/nni1b.gif";
   img2 = new Image();
   img2.src = "/graficos/nni1a.gif";
   function cambia(nombre,imagen)
   {
      nombre.src = imagen.src
   }
   function dobleancho()
   {
      imagen1.width=imagen1.width*2;
   }
   function doblealto()
   {
      imagen1.height=imagen1.height*2;
   }
   function mitadancho()
   {
      imagen1.width=imagen1.width/2;
   }
   function mitadalto()
   {
      imagen1.height=imagen1.height/2;
   }
</script>
<body>
<a href="" onmouseover="cambia(imagen1,img1)" onmouseout="cambia(imagen1,img2)"><img
src="/graficos/nni1a.gif" width="68" height="68" border="0" name="imagen1"></a><br><br>
<a href="javascript:dobleancho()">Doble ancho</a><br>
<a href="javascript:doblealto()">Doble Alto</a><br>
<a href="javascript:mitadancho()">Mitad ancho</a><br>
<a href="javascript:mitadalto()">Mitad Alto</a><br>
</body>
</html>
```

Objeto String:

El objeto String se usa para manipular cadenas de caracteres. En JavaScript todo texto encerrado entre comillas, dobles o simples, se interpreta como una cadena, así '45' no es el número cuarenta y cinco sino la cadena formada por los caracteres 4 y 5. El objeto String permite realizar operaciones con cadenas como concatenar o dividir cadenas, buscar texto, extraer parte de un texto, etc... La operación de crear una variable de este tipo se lleva a cabo como es habitual con el operador new pudiéndole pasar un argumento para inicializar la variable. Al usar un método o referirnos a una propiedad podemos usar el nombre de la variable o una constante de cadena así el ejemplo

var mitexto = "Esta es una cadena";
var pos = mitexto.indexOf("una")

Puede también escribirse en la siguiente forma: var pos = "Esta es una cadena". indexOf("una");

Propiedades	**length:**	Devuelve la longitud de la cadena. Sintaxis: string.lenght(nombre)
	prototype:	Permite agregar métodos y propiedades al objeto
Métodos	**anchor**	Crea un marcador HTML // var refer = "Referencia num. 1"; var ancla = refer.anchor("Refer1"); ancla→Referencia num. 1
	big	Hace que una cadena se muestre con tipos grandes, como si estuviera entre comandos <big> //var mitexto = "Este es el texto"; mitexto = mitexto.big(); Mitexto→<big>Este es el texto</big>
	blink	Hace que una cadena parpadee, como si estuviese entre comandos <blink> //var mitexto = " intermitente";mitexto = mitexto.blink();mitexto→<blink> intermitente</blink>
	bold	Hace que una cadena este en negritas, como si estuviese entre comandos <b>
	charAt	Devuelve el carácter en la posición indicada (0 a length-1) de la cadena //var nombre = "abcdefghij"; var car3 = nombre.charAt(2); Devolverá "c"
	fixed	Hace que la cadena se muestre como si estuviese entre comandos <tt>
	fontcolor	Hace que la cadena se muestre en el color indicado, como si estuviese entre comandos <font color=>
	fontsize	Hace que la cadena se muestre del tamaño indicado, como si estuviese entre comandos <font size=>
	indexOf	Devuelve la posición dentro de la cadena donde se encuentra el texto de búsqueda. De izquierda a derecha.
	italics	Hace que la cadena se muestre en cursiva, como si estuviese entre comandos <i>
	lastIndexOf	Devuelve la última posición dentro de la cadena en que se encuentra el texto de búsqueda. De derecha a izquierda.
	link	Crea un enlace de hipertexto que apunta a otro URL
	small	Hace que un cadena se muestre con tipos pequeños, como si estuviera entre etiquetas <small>
	strike	Hace que una cadena se muestre atravesada, como si estuviese entre etiquetas <strike>
	sub	Hace que una cadena se muestre en subíndice, como si estuviese entre etiquetas <sub>
	substring	Devuelve una subcadena de la cadena indicada, entre las posiciones indicadas.
	toLowerCase	Devuelve la cadena convertida en minúsculas
	toUpperCase	Devuelve la cadena convertida en mayúsculas

Objeto Math

Este objeto se utiliza para poder realizar cálculos en nuestros scripts. Tiene la peculiaridad de que sus propiedades no pueden modificarse, sólo consultarse. Estas propiedades son constantes matemáticas de uso frecuente en algunas tareas, por ello es lógico que sólo pueda consultarse su valor pero no modificarlo.

Propiedades	**E**	La constante de EULER. Aproximadamente 2.7818
	LN2	El logaritmo natural de 2. Aproximadamente 0.693
	LN10	El logaritmo natural de 10. Aproximadamente 2.302
	LOG2E	El logaritmo en base 2 de e. Aproximadamente 1.442
	LOG10E	El logaritmo vulgar de e. Aproximadamente 0.434
	PI	La constante PI. Aproximadamente 3.14159
	SQRT1_2	La raíz cuadrada de 1/2. Aproximadamente 0.707
	SQRT2	La raíz cuadrada de 2. Aproximadamente 1.414
Métodos	**abs(numero)**	Función valor absoluto. var numabs = Math.abs(- 45) la variable numabs contendrá el valor 45.
	acos(numero)	Función arcocoseno. Devuelve un valor cuyas unidades son radianes o NaN. 'numero' debe pertenecer al rango [-1,1], en otro caso devuelve NaN. var arco = Math. acos(1); la variable arco contendrá el valor 0.
	asin(numero)	Función arcoseno. Devuelve un valor cuyas unidades son radianes o NaN. 'numero' debe pertenecer al rango [-1,1], en otro caso devuelve NaN. var arco = Math.asin(1); la variable arco contendrá el arco cuyo seno es 1, o sea, 1.57 o lo que es lo mismo pi / 2 radianes.
	atan(numero)	Función arcotangente. Devuelve un valor cuyas unidades son radianes o NaN. var arco = Math.atan(1); La variable arco contendrá el arco cuya tangente es 1, o sea, 0.7853981633974483 o lo que es lo mismo pi / 4 radianes (45º).
	atan2(x,y)	Devuelve el ángulo formado por el vector de coordenadas (x,y) con respecto al eje OX. var argum= Math.atan2(10, 4); La variable argum contendrá el arco cuya tangente es 10/4.
	ceil(numero)	Devuelve el entero obtenido de redondear 'numero' "por arriba". var redexceso = Math.ceil(4.25); la variable redexceso contendrá el valor 5.
	cos(numero)	Devuelve el coseno de 'numero' (que debe estar en radianes) o NaN. var coseno = Math.cos(Math.PI/2); la variable coseno contendrá el valor 0, que es el coseno de pi/2 radianes (90º).
	exp(numero)	Devuelve el valor enumero. var e4 = Math.exp(4); la variable e4 contendrá el valor e^4 . El número e es la base de los logaritmos neperianos por lo que esta función sirve para calcular antilogaritmos.
	floor(numero)	Devuelve el entero obtenido de redondear 'numero' "por abajo". var redexceso = Math.floor(4.75); la variable redexceso contendrá el valor 4.
	log(numero)	Devuelve el logaritmo neperiano de 'numero'. var logaritmo = Math.log(1000); la variable logaritmo contendrá el valor 6.907755278982137 .
	max(x,y)	Devuelve el máximo de 'x' e 'y'. var mayor = Math.wax(12, 5); la variable mayor contendrá el valor 12.
	min(x,y)	Devuelve el mínimo de 'x' e 'y'. var menor = Math.min(12, 5); la variable menor contendrá el valor 5.
	pow(base,exp)	Devuelve el valor baseexp. var potencia = Math.pow(2, 4); la variable potencia contendrá el valor 16.
	random()	Devuelve un número pseudoaleatorio entre 0 y 1. var azar = Math.random()*10; la variable azar contendrá un número al azar entre 0 y 10.
	round(numero)	Redondea 'numero' al entero más cercano.
	sin(numero)	Devuelve el seno de 'numero' (que debe estar en radianes) o NaN.
	sqrt(numero)	Devuelve la raíz cuadrada de número.
	tan(numero)	Devuelve la tangente de 'numero' (que debe estar en radianes) o NaN.

Objeto Date

El objeto Date contiene un valor que representa fecha y hora de un instante dado. Para crear una instancia de este objeto usamos alguna de las siguientes sintaxis:
var fecha= new Date();
var fecha= new date(número);
var fecha= new date(cadena) :
var fecha= new date(año, mes, día[, hora[, minutos[, seg[,ms]]]]);

Los argumentos encerrados entre corchetes son opcionales. En la primera forma la variable fecha contendrá la fecha del día actual. La segunda opción almacena en fecha la fecha dada por el argumento como el número de milisegundos transcurridos desde la media noche del 1 de Enero de 1970. El tercer tipo se usa cuando la fecha se pasa en forma de cadena. Por último la fecha puede crearse pasándole como argumento los números de año, mes, día, hora y opcionalmente, hora, minuto, segundo y milisegundo. Los años posteriores a 1970 puede escribirse con dos dígitos, pero es aconsejable usar siempre cuatro dígitos por aquello de los efectos 2000.
var hoy = new date() /*fecha del día en hoy */
var evento = new Date("November 10 1990");
var otro = new Date("10 Nov 1990");
var otro = new Date("10/02/2000"); //Oct, 2, 2000

Estas son tres posibles formas de declarar objetos de tipo fecha. Las dos últimas almacenan el mismo día, pero en la última además se guarda la hora. Donde se usen cadenas para indicar una fecha podemos añadir al final las siglas GMT (o UTC) para indicar que la hora se refiere a hora del meridiano Greenwich, si no se toma como hora local, o sea, según la zona horaria configurada en el ordenador donde se ejecute el script.

Métodos	**getDate()**	Obtiene el dia del mes (1 a 31) var fecha = new Date(); document.write("Hoy es día: "+fecha.getDate());
	getDay()	Obtiene el dia de la semana (0=dom, 1=lun...6=sabado)
	getMonth()	Obtiene el mes en numeros (0=ene, 1=feb...11=dic)
	getYear()	Obtiene el año con dos digitos
	getHours()	Obtiene la hora (0 a 24)
	getMinutes()	Obtiene los minutos (0 a 59)
	getSeconds()	Obtiene los segundos (o a 59)
	getTime()	Obtiene los milisegundos trascurridos desde el 01-01-1970
	getTimezoneoffset()	Obtiene la diferencia horaria con GMT en minutos. var ahora = new Date(); document.write(ahora.getTimezoneOffset());
	setDate(number)	Establece el dia del mes (1 a 31) var fecha = new Date("1 Sep 2000"); fecha.setDate(31); document.write("Hoy es día: "+fecha.toString()); Como verás si pruebas el ejemplo la fecha es corregida y pasa a 1 de Octubre.
	setMonth(number)	Establece el mes del año (1 a 11)
	setYear(number)	Establece el año a partir de 1900
	setHours(number)	Establece la hora del dia (0 a 23)
	setMinutes(number)	Establece los minutos (0 a 59)
	setSeconds(number)	Establece los segundos (0 a 59)
	setTime(number)	Establece el valor del objeto Date, en milisegundos a partir de las 0:00:00 del 1º de enero de 1970
	toGMTString()	Convierte un objeto Date a una cadena usando un formato GMT para Internet
	toLocaleString()	Convierte un objeto Date a una cadena usando el formato local var fecha = new Date("10 Apr 2000 02:00:00"); document.write(fecha.toUTCString()); Como ves existe una diferencia en la hora almacenada y la devuelta por la función, esto es debido a que la cadena devuelta es la hora correspondiente a Greenwich, no la local del ordenador.
	parse(string)	Convierte una cadena representando una fecha al tiempo en milisegundos a partir de 0:00:00 del 1º de enero de 1970
	UTC()	Convierte una fecha al tiempo en milisegundos a partir de las 0:00:00 del 1º de

		enero de 1970

Objeto Array

La creación de un array se lleva a cabo con la sentencia: mi_array = new Array();
Se pueden asignar a un array valores de cualquier tipo (incluyendo objetos).mi_array[0] = 1; mi_array[1] = “Cadena de texto”; mi_array[2] = Date();
Tanto el tamaño (longitud) del array como el tamaño de las celdas individuales se asigna de forma dinámica (automática) a medida que se asignan elementos.Hay que tener en cuenta que el tamaño de los arrays en JavaScript puede crecer pero no decrecer, es decir, para el siguiente trozo de código:var array_grande = Array(); array_grande[1000]=1; Se construye un array de 1001 posiciones (aunque realmente sólo se utiliza una de ellas).

Propiedades	**length**	Como su nombre indica esta propiedad nos devuelve la longitud del array, es decir, elnúmero de elementos que puede almacenar. Su uso es muy simple: var lista = new Array(50); tamagno = lista.length; /*tamagno almacenaría el valor 50 */
	prototype	Esta es una propiedad muy potente en el sentido que nos permite agregar al objeto Array las propiedades y métodos que queramos. Array.protoype.descriptor = null; dias = new Array ('lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes'); dias.descriptor = "Dias laborables de la semana"; En este ejemplo hemos creado una nueva propiedad para el objeto array, la propiedad descriptor que podría utilizarse para darle un título a la matriz.
Metodos	**concat(objArray)**	Une dos arrays y devuelve como resultado un array con la unión.
	join()	Devuelve una cadena de texto que contiene la unión de los elementos del array.
	pop()	Borra y devuelve el último elemento de un array.
	push()	Añade un elemento a un array y devuelve ese elemento.
	reverse()	Refleja el contenido de un array (el primer elemento pasa a ser el último).
	shift()	Borra y devuelve el primer elmento de un array.
	slice()	Extrae una sección de un array y la devuelve como un nuevo array: datos = new Array () datos[1]=”uno”; datos[2]=”dos”; datos[3]=”tres”; dosprimeros = datos.slice(1,2); // dosprimeros[0]=”uno” // dosprimeros[1]=”dos”
	splice()	Añade y/o elimina elementos de un array.
	sort()	Ordena los elementos de un array. Por defecto utiliza ordenación lexicográfica (alfabética). Se puede definir una función externa que implemente el criterio de ordenación: function compareNumbers(a, b) { return a – b } numberArray.sort(compareNumbers)
	toString ()	Devuelve una cadena que representa al array.
	unshift ()	Añade uno o más elementos al comienzo de un array y devuelve el número actualizado de elementos.

Objeto Boolean

Las variables booleanas o lógicas son las que sólo pueden tomar dos valores: true, verdadero, y false, falso. Este tipo de variables está implementado en JavaScript como un objeto**.**

Metodos	toString ()	Si el valor es false devuelve la cadena "false" y si es true devuelve la cadena "true"
	valueOf()	Devuelve el valor booleano (true o false)
Propiedades	**constructor**	heredada del objeto genérico Object, devuelve la referencia al constructor: function Boolean() { [native code] }
	prototype	Es una propiedad utilizada para asignar nuevos métodos o propiedades, heredado del objeto genérico Object. Por ejemplo podemos traducir los valores true o false. function valor () { return this.valueOf()?'cierto':'falso' } Boolean.prototype.valor = valor; var item = new Boolean(false);

		document.write(item.valor()); Con este ejemplo logramos que true se devuelva como la cadena "cierto" y false como la cadena "falso".

FORMULARIOS

Objeto form

Los formularios siempre han sido la mejor manera de facilitar la comunicación entre los usuarios y los creadores de una web. Sin embargo, la implementación de los mismos en el lenguaje HTML ha provocado ciertos problemas debido a sus carencias. Estos problemas han intentado solventarse con scripts, situados tanto en el servidor como en el navegador.

Métodos	**submit()**	Para hacer que el formulario se submita, aunque no se haya pulsado el botón de submit.
	reset()	Para reinicializar todos los campos del formulario, como si se hubiese pulsado el botón de reset.
Propiedades	**action**	Es la acción que queremos realizar cuando se submite un formulario. Se coloca generalmente una dirección de correo o la URL a la que le mandaremos los datos. Corresponde con el atributo ACTION del formulario. Por ejemplo podríamos cambiar la URL que recibiría la información del formulario con la instrucción. document.miFormulario.action = "miPágina.asp"
	elements array	La matriz de elementos contiene cada uno de los campos del formulario.
	encoding	El tipo de codificación del formulario
	length	El número de campos del formulario.
	method	El método por el que mandamos la información. Corresponde con el atributo METHOD del formulario.
	name	El nombre del formulario, que corresponde con el atributo NAME del formulario
	target	La ventana o frame en la que está dirigido el formulario. Cuando se submita se actualizará la ventana o frame indicado. Corresponde con el atributo target del formulario. O cambiar el target para submitir un formulario en una posible ventana secundaria llamada mi_ventana. document.miFormulario.target = "mi_ventana"

```
<html>
<head>
  <title>Ejemplo de formularios</title>
  <script language="JavaScript">
    function validar(direccion) {
       if (direccion.indexOf("@") != -1)
          return true;
       else {
          alert('Debe escribir una dirección
válida');
          return false;
       }
    }
  </script>
</head>
<body>
<form name="miFormulario"
     method="POST"
     action="mailto:yo@miproveedor.mipais"
     enctype="text/plain"
     onSubmit="return validar(this.email.value)">
Mandame tu e-mail:
<input name="email" type="text"><br>
<input type="submit" value="Mandame tu e-mail">
</form>
</body>
</html>
```

El código es sencillo: el código llamado por el controlador de evento onSubmit debe devolver false si deseamos que el formulario no sea enviado. Así pues, llamamos a la función que comprueba si hay alguna arroba en el campo email del formulario.

La manera de llamar a esta función es quizás lo más complicado. La función validar recibe una cadena de caracteres, devolviendo verdadero o falso dependiendo de que haya o no una arroba dentro de ella. El controlador utiliza para llamar a esta función lo siguiente:

this.email.value

this es una referencia estándar. Cuando veamos this en algún código en realidad deberemos sustituirlo mentalmente por el nombre del objeto en el que está el código. En este caso, como estamos dentro de miFormulario, this será equivalente a document.miFormulario. email es el nombre del campo que queremos investigar y value el

el atributo que contiene lo que haya tecleado el usuario.

Los objetos text, textarea y password

```
<html>
<head>
   <title> Ejemplo de JavaScript </title>
</head>
<script language="JavaScript">
   function Mostrar()
   {
      alert('Su nombre: ' + formulario.nombre.value);
      alert('El password: ' + formulario.pass.value);
   }
</script>
<body>
<form action="procesa.phtml" name="formulario" id="formulario"
   method="GET">
   Nombre: <input type="text" name="nombre" value="Tu nombre"
   maxlength="15"><br>
   Password: <input type="password" name="pass"
    maxlength="10"><br>
</form>
<a href="javascript:Mostrar();">Mostrar datos</a><br>
```

Estos objetos representan los campos de texto dentro de un formulario. Además, el objeto password es exactamente igual que el text salvo en que no muestra los caracteres introducidos por el usuario, poniendo asteriscos (*) en su lugar.

Métodos	**blur()**	Pierde el foco del ratón sobre el objeto especificado.
	focus()	Obtiene el foco del ratón sobre el objeto especificado.
	select()	Selecciona el texto dentro del objeto dado.
Propiedades	**dafaultValue**	Es una cadena que contiene el valor por defecto que se le ha dado a uno de estos objetos por defecto.
	name	Es una cadena que contiene el valor del parámetro NAME.
	value	Es una cadena que contiene el valor del parámetro VALUE.
	maxlength	Número máximo de caracteres que puede contener el campo de texto.

El objeto button

Tenemos tres tipos de botones: un botón genérico, 'button', que no tiene acción asignada, y dos botones específicos, 'submit' y 'reset'. Estos dos últimos sí que tienen una acción asignada al ser pulsados: el primero envía el formulario y el segundo limpia los valores del formulario.

```
<html> <head> <title>Ejemplo de JavaScript</title> </head>
<script language="JavaScript">
   function Mostrar(boton)
   { alert('Ha hecho click sobre el boton: ' + boton.name+', de
      valor:'+boton.value); return true; }
</script>
<body>
<form action="procesa.phtml" name="formulario" id="formulario"
method="GET">
Un boton: <input type="button" name="Boton1" value="El boton 1"
OnClick="Mostrar(this);"><br><br>
Un boton: <input type="button" name="Boton2" value="El boton 2"
OnClick="Mostrar(this);"><br><br>
Un boton: <input type="button" name="Boton3" value="El boton 3"
OnClick="Mostrar(this);"><br>
</form> </body> </html>
```

Metodos	**click()**	Realiza la acción de pulsado del botón
Propiedades	**name**	Es una cadena que contiene el valor del parámetro NAME.
	value	Es una cadena que contiene el valor del parámetro VALUE.

El objeto checkbox

```
<HTML> <HEAD> <title>Ejemplo de JavaScript</title> </HEAD>
<script LANGUAGE="JavaScript">
   function Mostrar(boton)
   {
      msg="Opcion 1:"+formulario.check1.checked+"\n"
      msg+="Opcion 2:"+formulario.check2.checked+"\n"
      msg+="Opcion 3:"+formulario.check3.checked+"\n"
      alert(msg);
   }
</script>
<body>
<form action="procesa.phtml" name="formulario" id="formulario"
method="GET">
<input type="checkbox" name="check1" checked> Opcion 1<br>
<input type="checkbox" name="check2"> Opcion 2<br>
<input type="checkbox" name="check3" checked> Opcion 3<br>
</form>
<a href="javascript:Mostrar()">Ver valores</a>
</body> </html>
```

Las "checkboxes" nos permiten seleccionar varias opciones marcando el cuadrito que aparece a su izquierda. El cuadrito pulsado equivale a un "sí" y sin pulsar a un "no" o, lo que es lo mismo, a "true" o "false".

Métodos	click()	Realiza la acción de pulsado del botón
Propiedades	checked.	Valor booleano que nos dice si el checkbox está pulsado o no
	defaultChecked	Valor booleano que nos dice si el checkbox debe estar seleccionado por defecto o no
	name	Es una cadena que contiene el valor del parámetro NAME.
	value	Es una cadena que contiene el valor del parámetro VALUE.

El objeto radio

```
<html> <head> <title>Ejemplo de JavaScript</title> </head>
<script language="JavaScript">
   function Mostrar(boton) {
      msg="Elementos:"+formulario.edad.length+"\n";
      msg+="Menor de 18 años:"+formulario.edad[0].checked+"\n";
      msg+="Entre 18 y 60
años:"+formulario.edad[1].checked+"\n";
      msg+="Mayor de 60 años:"+formulario.edad[2].checked+"\n";
      alert(msg); } </script>
<body>
<form action="procesa.phtml" name="formulario"
   id="formulario" method="GET">
Edad:<br> <input type="radio" name="edad" value="<18">
Menor de 18 años.<br> <input type="radio" name="edad" value=">18
y <60" checked> Entre 18 y 60 años.<br>
<input type="radio" name="edad" value=">60"> Mayor de 60
años.<br>
</form>
<a href="javascript:Mostrar()">Ver valores</A>
</body> </html>
```

Al contrario que con los checkbox, que nos permiten elegir varias posibilidades entre las dadas, los objetos radio sólo nos permiten elegir una de entre todas las que hay. Están pensados para posibilidades mútuamente excluyentes (no se puede ser a la vez mayor de 18 años y menor de 18 años, no se puede estar a la vez soltero y casado, etc.).

Métodos	click()	Realiza la acción de pulsado del botón
Propiedades	checked	Valor booleano que nos dice si el radio está seleccionado o no.
	defaultChecked	Valor booleano que nos dice si el radio debe estar seleccionado por defecto o no.
	length	Valor numérico que nos dice el número de opciones dentro de un grupo de elementos radio.
	name	Es una cadena que contiene el valor del parámetro NAME.
	value	Es una cadena que contiene el valor del parámetro VALUE.

El objeto select

Este objeto representa una lista de opciones dentro de un formulario. Puede tratarse de una lista desplegable de la que podremos escoger alguna (o algunas) de sus opciones.

Propiedades	length	Valor numérico que nos indica cuántas opciones tiene la lista
	name	Es una cadena que contiene el valor del parámetro NAME
	options	Se trata de un array que contiene cada una de las opciones de la lista. Este array tiene, a su vez, las siguientes propiedades: - **defaultSelected**. Valor booleano que nos indica si la opción está seleccionada por defecto. - **index**. Valor numérico que nos da la posición de la opción dentro de la lista. - **length**. Valor numérico que nos dice cuántas opciones tiene la lista. - **options.** Cadena con todo el código HTML de la lista. - **selected**. Valor booleano que nos dice si la opción está actualmente seleccionada o no. - **text**. Cadena con el texto mostrado en la lista de una opción concreta. - **value**. Es una cadena que contiene el valor del parámetro VALUE de la opción concreta de la lista.
	selectedIndex	Valor numérico que nos dice cuál de todas las opciones disponibles está actualmente seleccionada.

```
<html>
<head>
   <title>Ejemplo de JavaScript</title>
</head>
<script LANGUAGE="JavaScript">
   function Mostrar(boton)
   {
      msg="Elementos:"+formulario.edad.length+"\n";
      msg+="Edad:
"+formulario.edad.options[formulario.edad.selectedIndex].value+"\n";

      alert(msg);
   }
</script>
<body>
<form action="procesa.phtml" name="formulario" id="formulario" method="GET">

Edad:<br>
<select name="edad">
   <option value="<18" SELECTED>Menor de 18 años</option>
   <option value=">18 y <60">Entre 18 y 60 años</option>
   <option value=">60">Mayor de 60 años</option>
</select>

</form>
<a href="javascript:Mostrar()">Ver valores</A>
</body>
</html>
```

EVENTOS EN JAVASCRIPT

Un evento es un suceso que ocurre cuando el usuario realiza alguna acción. Por ejemplo cuando el usuario pasa el ratón encima de un objeto de la página, cuando pulsa una tecla,... Incluso algunos eventos no los produce el usuario, sino el navegador, como por ejemplo la carga de la página. De tal forma que cuando se produce el evento en cuestión, se ejecuta el código en JavaScript que está entre comillas.

```
<a href="cities.htm"onMouseClick="alert('Pulsaste!');">
//Cuando el usuario pulsa en el mensaje, aparece un cuadro de alerta que pone "Pulsaste!"
```

EVENTOS JAVASCRIPT		
onAbort	Se produce si el usuario pulsa el botón Detener mientras se estaba cargando una imagen. La etiqueta IMG es la que maneja este evento.	Image
onBlur	Se produce cuando un objeto pierde el foco (deja de ser el objeto activo).	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window
onChange	Se produce cuando el usuario cambia el contenido de un cuadro de texto de un formulario.	FileUpload, Select, Text, Textarea
onClick	Se produce cuando el usuario hace clic en el objeto o formulario.	Button, document, Checkbox, Link, Radio, Reset, Submit
onDblClick	Se genera cuando el usuario hace doble clic con el ratón o formulario.	document, Link
onDragDrop	El usuario arrastra y suelta un objeto en la ventana	window
onError	La carga de un documento o imagen produce un error	Image, window
onFocus	Sucede cuando un objeto gana el foco (pasa a ser el objeto activo).	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window
onKeyDown	El usuario pulsa una tecla	document, Image, Link, Textarea
onKeyPress	El usuario mantiene pulsada una tecla	document, Image, Link, Textarea
onKeyUp	El usuario libera una tecla	document, Image, Link, Textarea
onLoad	El navegador termina la carga de una ventana	Image, Layer, window
onMouseDown	El usuario pulsa un botón del ratón	Button, document, Link
onMouseMove	El usuario mueve el puntero	Ninguno (debe asociarse a uno)
onMouseOut	El puntero abando una área o enlace	Layer, Link
onMouseOver	El puntero entra en una área o imagen	Layer, Link
onMouseUp	El usuario libera un botón del ratón	Button, document, Link
onMove	Se mueve una ventana o un marco	window
onReset	El usuario limpia un formulario	Form
onResize	Se cambia el tamaño de una ventana o marco	window
onSelect	Se selecciona el texto del campo texto o área de texto de un formulario	Text, Textarea
onSubmit	El usuario envía un formulario	Form
onUnload	El usuario abandona una página	window

Métodos Globales.

JavaScript incluye las siguientes funciones, que no son métodos de ningún objeto sino propias del lenguaje:

eval(cadena)	Trata de evaluar una cadena y devolver un valor numérico, si el argumento es una expresión, la expresión se evalúa, si el argumento consiste en uno o más comandos, se ejecutan.
parseFloat(cadena)	Convierte una cadena a un número en punto flotante. Si se encuentra un carácter que no es numero, signo (+ o -), punto decimal o exponente, la funcion ignora la cadena a partir de esa posición y la evalúa hasta el carácter anterior. Si el primer carácter no se puede convertir, la función devuelve uno de estos valores: o en las plataformas Windows y "NaN" (Not a Number) para otras plataformas.
parseInt(cadena [,base])	Convierte una cadena a un entero en la base especificada. Si no se especifica la base o se especifica como 0, se opta por lo siguiente: Si la cadena comienza con "0x", la base es 16 (hexadecimal), si la cadena empieza con 0, la base es 8 (octal), si la cadena comienza con otro valor, la base es 10 (decimal). Si se encuentra un carácter que no es numérico, la función ignora la cadena a partir de esa posición y la evalúa hasta la anterior. Si el primer carácter no se puede convertir, la función devuelve uno de estos valores: 0 para plataformas Windows y "NaN" (Not a Number) para otras plataformas.
isNaN(valor prueba).	Evalúa un argumento para determinar si es "NaN", en plataformas UNIX, devolviendo un valor Booleano true o false.
GetElementById()	Permite referirnos a los elemento por su identificador ID y nodificarlos. document.getElementById(id).style.propiedad. Ejemplo de Sintaxis: document.getElementById('imagen').style.backgroundColor
setTimeOut()	Evalúa una expresión después de transcurrido un tiempo en milisegundos. Este es un método del Objeto window

Captura de excepciones con try y catch

Las sentencias try y catch, que se relacionan con el objeto Exception, se generado cuando se produce algún error durante la ejecución del script.Aunque puede parecer de poca utilidad, hay que observar que determinadas condiciones de error pueden no implicar la terminación abrupta de la ejecución del programa. Utilizando la captura de excepciones (catch en inglés significa precisamente esto), podemos incorporar en el script una forma elegante de eludir el error.

En lenguajes de programación como Java, un uso típico de la secuencia try ... catch es el acceso a ficheros o recursos que no se sabe si estarán disponibles. Si el recurso no está disponible se producirá una excepción, la cual puede ser capturada para evitar la terminación abrupta del programa, y mostrar en su lugar un mensaje de error o de revisión de los datos introducidos por el usuario (por poner un ejemplo). La captura de errores se realiza incluyendo el código que puede potencialmente contener un error (o provocar una excepción) entre las llaves de la sentencia try, como en el siguiente ejemplo:

```
try {
  //ejemplo de línea con error
  writte("Esta linea probocará una excepción.");
}
catch(exception) {
  if (exception.description == null) {
    alert("Excepción: " + exception.message);
  } else {
    alert("Excepción: " + exception.description);
  }
}
```

Como se observa en el ejemplo, la sentencia writte provocará una excepción, al estar escrita de forma incorrecta (la forma correcta es write). Cuando se ejecuta la pieza de código anterior, se crea un objeto Exception. Este objeto puede ser accedido mediante la sentencia catch.

MODELO DE OBJETOS DE DOCUMENTO (DOM)

La creación del Document Object Model o DOM es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.
DOM permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML. A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

Árbol de nodos

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la manipulación de las páginas web. De esta forma, es habitual obtener el valor almacenado por algunos elementos (por ejemplo los elementos de un formulario), crear un elemento (párrafos, , etc.) de forma dinámica y añadirlo a la página, aplicar una animación a un elemento (que aparezca/desaparezca, que se desplace, etc.).

Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, para poder utilizar las utilidades de DOM, es necesario "transformar" la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular. Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

DOM transforma todos los documentos XHTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama "árbol de nodos".

La siguiente página XHTML sencilla:

```
<!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=iso-
8859-1" />
<title>Página sencilla</title>
</head>
<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se transforma en el siguiente árbol de nodos:

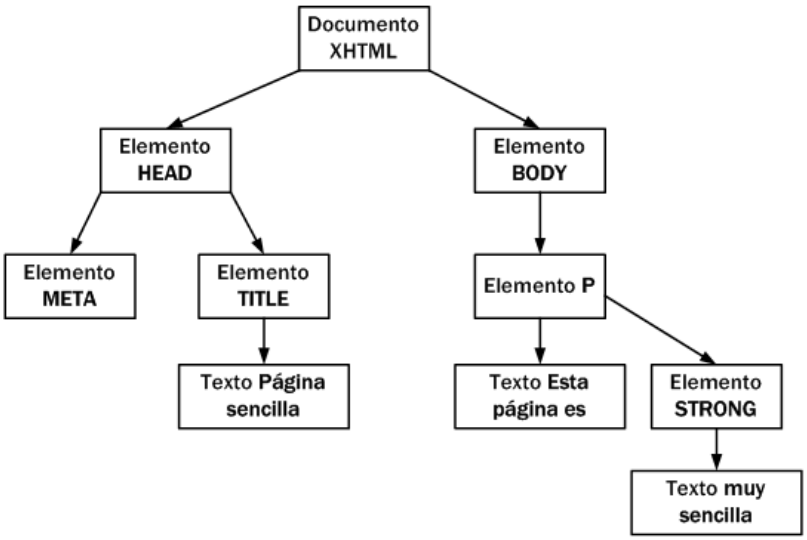


Figura 5.1. Árbol de nodos generado automáticamente por DOM a partir del código XHTML de la página

En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.La raíz del árbol de nodos de cualquier página XHTML siempre es la misma: un nodo de tipo especial denominado "Documento".

A partir de ese nodo raíz, cada etiqueta XHTML se transforma en un nodo de tipo "Elemento". La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada etiqueta XHTML se transforma en un nodo que deriva del nodo correspondiente a su "etiqueta padre". La transformación de las etiquetas XHTML habituales genera dos nodos: el primero es el nodo de tipo "Elemento" (correspondiente a la propia etiqueta XHTML) y el segundo es un nodo de tipo "Texto" que contiene el texto encerrado por esa etiqueta XHTML.

Así, la siguiente etiqueta XHTML:

```
<title>Página sencilla</title>
```

Genera los siguientes dos nodos:

Figura 5.2. Nodos generados automáticamente por DOM para una etiqueta XHTML sencilla

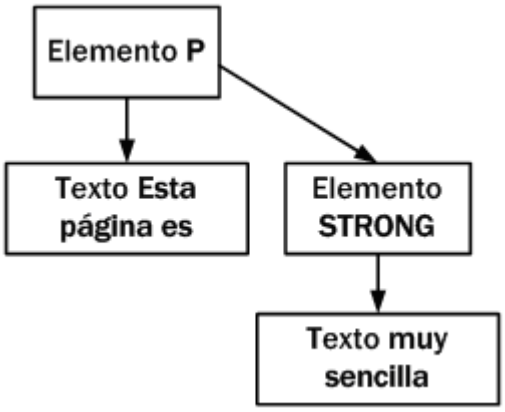
De la misma forma, la siguiente etiqueta XHTML:

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

Genera los siguientes nodos:

- Nodo de tipo "Elemento" correspondiente a la etiqueta <p>.
- Nodo de tipo "Texto" con el contenido textual de la etiqueta <p>.
- Como el contenido de <p> incluye en su interior otra etiqueta XHTML, la etiqueta interior se transforma en un nodo de tipo "Elemento" que representa la etiqueta y que deriva del nodo anterior.

- El contenido de la etiqueta genera a su vez otro nodo de tipo "Texto" que deriva del nodo generado por .



La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:

Las etiquetas XHTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.
Si una etiqueta XHTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.
Como se puede suponer, las páginas XHTML habituales producen árboles con miles de nodos. Aun así, el proceso de transformación es rápido y automático, siendo las funciones proporcionadas por DOM (que se verán más adelante) las únicas que permiten acceder a cualquier nodo de la página de forma sencilla e inmediata.

Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- Document, nodo raíz del que derivan todos los demás nodos del árbol.
- Element, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- Attr, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.
- Text, nodo que contiene el texto encerrado por una etiqueta XHTML.
- Comment, representa los comentarios incluidos en la página XHTML.

Los otros tipos de nodos existentes que no se van a considerar son DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.

Acceso directo a los nodos

Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Como acceder a un nodo del árbol es equivalente a acceder a "un trozo" de la página, una vez construido el árbol, ya es posible manipular de forma sencilla la página: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc. DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.
Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por ese motivo, no se van a presentar las funciones necesarias para el acceso jerárquico de nodos y se muestran solamente las que permiten acceder de forma directa a los nodos. Por último, es importante recordar que el acceso a los nodos, su modificación y su eliminación solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página XHTML se cargue por completo. Más adelante se verá cómo asegurar que un código JavaScript solamente se ejecute cuando el navegador ha cargado entera la página XHTML.

getElementsByTagName()

Como sucede con todas las funciones que proporciona DOM, la función getElementsByTagName() tiene un nombre muy largo, pero que lo hace autoexplicativo. La función getElementsByTagName(nombreEtiqueta) obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que se indica delante del nombre de la función (en este caso, document) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor document como punto de partida de la búsqueda. El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales. Por lo tanto, se debe procesar cada valor del array de la forma que se muestra en las siguientes secciones.

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++) {
  var parrafo = parrafos[i];
}
```

La función getElementsByTagName() se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");
```

getElementsByName()

La función getElementsByName() es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo name sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementsByName("especial");

<p name="prueba">...</p>
<p name="especial">...</p>
<p>...</p>
```

Normalmente el atributo name es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado. En el caso de los elementos HTML radiobutton, el atributo name es común a todos los radiobutton que están relacionados, por lo que la función devuelve una colección de elementos.
Internet Explorer 6.0 no implementa de forma correcta esta función, ya que sólo la tiene en cuenta para los elementos de tipo <input> y <img>. Además, también tiene en consideración los elementos cuyo atributo id sea igual al parámetro de la función.

getElementById()

La función getElementById() es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.La función getElementById() devuelve el elemento XHTML cuyo atributo id coincide con el parámetro indicado en la función. Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

```
var cabecera = document.getElementById("cabecera");

<div id="cabecera">
  <a href="/" id="logo">...</a>
</div>
```

La función getElementById() es tan importante y tan utilizada en todas las aplicaciones web, que casi todos los ejemplos y ejercicios que siguen la utilizan constantemente. Internet Explorer 6.0 también interpreta incorrectamente esta función, ya que devuelve también aquellos elementos cuyo atributo name coincida con el parámetro proporcionado a la función.

Creación y eliminación de nodos
Acceder a los nodos y a sus propiedades (que se verá más adelante) es sólo una parte de las manipulaciones habituales en las páginas. Las otras operaciones habituales son las de crear y eliminar nodos del árbol DOM, es decir, crear y eliminar "trozos" de la página web.

Creación de elementos XHTML simples

Como se ha visto, un elemento XHTML sencillo, como por ejemplo un párrafo, genera dos nodos: el primer nodo es de tipo Element y representa la etiqueta <p> y el segundo nodo es de tipo Text y representa el contenido textual de la etiqueta <p>. Por este motivo, crear y añadir a la página un nuevo elemento XHTML sencillo consta de cuatro pasos diferentes:

- Creación de un nodo de tipo Element que represente al elemento.
- Creación de un nodo de tipo Text que represente el contenido del elemento.
- Añadir el nodo Text como nodo hijo del nodo Element.
- Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

De este modo, si se quiere añadir un párrafo simple al final de una página XHTML, es necesario incluir el siguiente código JavaScript:

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");

// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola
Mundo!");

// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);

// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

El proceso de creación de nuevos nodos puede llegar a ser tedioso, ya que implica la utilización de tres funciones DOM:

- createElement(etiqueta): crea un nodo de tipo Element que representa al elemento XHTML cuya etiqueta se pasa como parámetro.
- createTextNode(contenido): crea un nodo de tipo Text que almacena el contenido textual de los elementos XHTML.
- nodoPadre.appendChild(nodoHijo): añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo Text como hijo del nodo Element y a continuación se añade el nodo Element como hijo de algún nodo de la página.

Eliminación de nodos

Afortunadamente, eliminar un nodo del árbol DOM de la página es mucho más sencillo que añadirlo. En este caso, solamente es necesario utilizar la función removeChild():

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);

<p id="provisional">...</p>
```

La función removeChild() requiere como parámetro el nodo que se va a eliminar. Además, esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad nodoHijo.parentNode. Así, para eliminar un nodo de una página XHTML se invoca a la función removeChild() desde el valor parentNode del nodo que se quiere eliminar. Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.

Acceso directo a los atributos XHTML

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página.

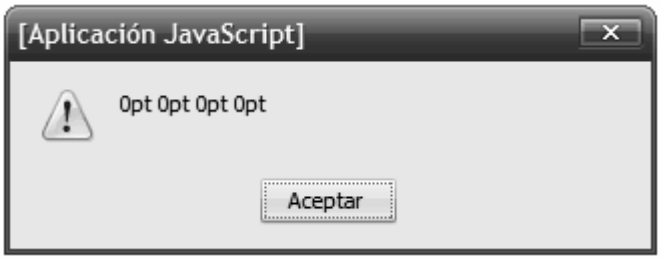
Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.
El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
var enlace = document.getElementById("enlace");
alert(enlace.href); // muestra http://www...com

<a id="enlace" href="http://www...com">Enlace</a>
```

En el ejemplo anterior, se obtiene el nodo DOM que representa el enlace mediante la función document.getElementById(). A continuación, se obtiene el atributo href del enlace mediante enlace.href. Para obtener por ejemplo el atributo id, se utilizaría enlace.id. Las propiedades CSS no son tan fáciles de obtener como los atributos XHTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style. El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin);

<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Aunque el funcionamiento es homogéneo entre distintos navegadores, los resultados no son exactamente iguales, como muestran las siguientes imágenes que son el resultado de ejecutar el código anterior en distintos navegadores:



Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.style.fontWeight);  // muestra "bold"

<p id="parrafo" style="font-weight: bold;">...</p>
```

La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio. A continuación se muestran algunos ejemplos:

- font-weight se transforma en fontWeight
- line-height se transforma en lineHeight
- border-top-style se transforma en borderTopStyle
- list-style-image se transforma en listStyleImage

El único atributo XHTML que no tiene el mismo nombre en XHTML y en las propiedades DOM es el atributo class. Como la palabra class está reservada por JavaScript, no es posible utilizarla para acceder al atributo class del elemento XHTML. En su lugar, DOM utiliza el nombre className para acceder al atributo class de XHTML:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.class); // muestra "undefined"
alert(parrafo.className); // muestra "normal"

<p id="parrafo" class="normal">...</p>
```