

# Fundamentos de la Interacción Persona-Ordenador



## 7. JavaScript

Luis Rodríguez Baena (luis.rodriguez@upsam.es)

Universidad Pontificia de Salamanca  
Escuela Superior de Ingeniería y Arquitectura

# Introducción

- ❑ Javascript es un lenguaje de script multiplataforma y orientado a objetos.
- ❑ Es un lenguaje interpretado, pequeño y ligero.
  - No es demasiado útil como lenguaje independiente.
  - Está diseñado para ser incrustado en otros productos y aplicaciones.
    - ✓ Adobe Acrobat, Flash y Flex (poseen un lenguaje ActionScript que es un dialecto de JavaScript), Photoshop, navegadores web, etc.
- ❑ Como lenguaje para la web se puede utilizar tanto del lado del cliente como del servidor.
  - En el lado del cliente proporciona objetos para el control del navegador y el Modelo de Objeto del Document (DOM).
    - ✓ Permite manejar los elementos de un formulario para la verificación, responder a eventos del usuario para realizar acciones con los elementos xhtml, elementos del formulario, comportamiento del navegador, etc.
  - En el lado del servidor proporciona objetos interesantes para el manejo de un servidor web.
    - ✓ Conexión con bases de datos, manipulación de archivos en el servidor, etc.

# Introducción (II)

- ❑ Posee una sintaxis similar a otros lenguajes como C, C++ o Java.
  - Algunas diferencias básicas:
    - ✓ Es un lenguaje poco tipeado.
      - No es necesaria la declaración de variables ni asignarlas un tipo de forma explícita.
        - Al declarar una variable no hay que indicar su tipo, por lo que una variable puede almacenar distintos tipos de datos durante la ejecución del script.
    - ✓ No es necesario terminar cada sentencia con un punto y coma.
      - Es conveniente hacerlo.
- ❑ Limitaciones.
  - Con ellas sólo se pueden ejecutar scripts en un entorno limitado para permitir a los usuarios confiar en su ejecución.
    - ✓ Los scripts no pueden comunicarse con otros recursos que pertenezcan a otro dominio desde dónde se descargó.
    - ✓ No pueden cerrar ventanas que ellos no hayan abierto.
    - ✓ No pueden acceder al sistema de archivos local.
    - ✓ No pueden leer o modificar las preferencias del navegador.

# Introducción

## Historia

- ❑ Se crea para evitar tener que conectarse con un servidor para que las páginas web tengan que hacer algunas tareas.
  - Permite delegar la ejecución de programas al lado del cliente.
    - ✓ Por ejemplo, la verificación de datos de un formulario.
- ❑ Se crea en 1995, al salir el navegador Netscape 2.
  - Inicialmente, Netscape lo llamó LiveScript.
  - La alianza con Sun Microsystems rebautizó el lenguaje para llamarlo JavaScript.
    - ✓ No tiene que ver con Java: es una cuestión de marketing.
- ❑ Poco después Microsoft saca su versión JScript.
  - Para evitar incompatibilidades, Netscape decide llevar el lenguaje a la ECMA (*European Computer Manufacturers Association*) para su estandarización.
    - ✓ Nace el lenguaje ECMAScript.
      - JavaScript es la implementación de Netscape para ECMAScript.
- La versión soportada actualmente por los navegadores el Javascript 1.8, compatible con ECMAScript 5ª edición).
  - La última edición de ECMAScript es la 5.1 de junio de 2011 ([www.ecma-international.org/publications/files/ECMA-ST/ECma-262.pdf](http://www.ecma-international.org/publications/files/ECMA-ST/ECma-262.pdf))

# Introducción

## Introducir JavaScript en xhtml

### □ JavaScript dentro de un documento.

- Las etiquetas `<script></script>` deben encerrar el código JavaScript.
- Se puede incluir en cualquier parte del documento.
  - ✓ Lo normal es agrupar todas las funciones dentro del elemento `<head>`.
- La etiqueta debe incluir el atributo `type="text/javascript"`.
- Puede que sea necesario incluir el código dentro de una sección CDATA.
  - ✓ En caso contrario el símbolo `<` se interpretará como el comienzo de una etiqueta.
  - ✓ Como CDATA no forma parte de JavaScript, se debe incluir como comentario (`//`).

```
<script type="text/javascript">
  //
    /* Código JavaScript
       ...
    */
  //]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="3 947 467 991" data-label="Page-Footer"><p>Universidad Pontificia de Salamanca. Escuela Superior de Ingeniería y Arquitectura<br/>(CC) Luis Rodríguez Baena, 2013</p></div><div data-bbox="979 959 994 981" data-label="Page-Footer"><p>5</p></div>
```

# Introducción

## Introducir JavaScript en xhtml (II)

### ❑ JavaScript en un documento externo.

- Incluir JavaScript en el documento...
  - ✓ Retarda la carga del documento si existe mucho código.
  - ✓ Si el script cambia, obliga a modificar todas las páginas que lo utilizan.
- El atributo `src` de la etiqueta `<script>` permite hacer referencia a un archivo de texto con extensión `.js` que contiene el código.
- La etiqueta `<script>` requiere siempre una etiqueta de cierre `</script>`.

```
<script type="text/javascript" src="PrimerJavaScript.js">
</script>
```

- Es recomendable en estos casos colocar la llamada justo antes de `</body>`. De esta forma primero se procesa la página y, una vez procesada se procede a la carga del script.

### ❑ JavaScript en elementos xhtml.

- Mediante los atributos de evento es posible incluir código JavaScript en elementos xhtml.
- Este método no es recomendable, ya que ensucia el código xhtml.

```
<elemento atributoDeEvento="código javascript">
```

# Introducción

## Introducir JavaScript en xhtml (III)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <title>PrimerJavaScript</title>
  <!-- JavaScript en la sección head del documento -->
  <script type="text/javascript">
    //<![CDATA[
      alert("Hola, mundo! (de la sección head)");
    //]]> </script>
  <!-- JavaScript en un archivo externo -->
  <script type="text/javascript" src="PrimerJavaScript.js"></script>
</head>
<body>
  <!-- JavaScript en el cuerpo del documento -->
  <script type="text/javascript">
    //<![CDATA[
      alert("Hola, mundo! (de la sección body)");
    //]]>
  </script>
  <!-- JavaScript en un elemento xhtml -->
  <p onclick="alert('Hola, mundo! (al hacer clic)');">Si está activado
    JavaScript, al hacer clic aquí aparecería un mensaje.</p>
  <noscript>
    <p>Si no está activado aparecería esta línea.</p>
  </noscript>
</body>
</html>
```

# Introducción

## Introducir JavaScript en xhtml (IV)

- ❑ Hay que tener en cuenta que no todos los agentes de usuario soportan JavaScript y el usuario puede que lo tenga desactivado.
  - ✓ El script no podrá ejecutarse, quitando funcionalidad a la página.
- ❑ En estos casos es posible insertar código xhtml que se visualizará sólo cuando JavaScript no se pueda ejecutar.
  - El código xhtml se encerrará dentro de la etiqueta `<noscript></noscript>`.
- ❑ En el peor de sus usos se mostrará un mensaje que indique la necesidad de que el usuario active los scripts.
  - Si queremos que la página sea usable y accesible, la funcionalidad básica debería mantenerse sin necesidad de JavaScript mediante métodos alternativos, como por ejemplo:
    - ✓ Si no se puede hacer la validación de datos en el cliente, se deberá habilitar un sistema alternativo de validación en el servidor.
    - ✓ Si se utilizan menús en cascada basados en JavaScript u otros elementos como calendarios, hay que aportar algún sistema de navegación o de selección alternativo entro de la etiqueta `<noscript>`.



# Elementos del lenguaje

## Tipos de datos

- ❑ Numéricos.
  - No hace distinción entre datos reales o enteros.
- ❑ Lógicos.
  - Puede tomar los valores true o false.
- ❑ `undefined`.
  - Representa el contenido de una variable a la que no se le ha asignado un valor.
- ❑ `null`.
  - Representa el contenido de un objeto no instanciado.
- ❑ Cadenas.
  - Como separador puede utilizar tanto las comillas simples como las dobles.
  - Puede incluir las secuencias de escape...

Secuencia	Significado	Secuencia	Significado
<code>\n</code>	Nueva línea	<code>\r</code>	Retorno de carro
<code>\t</code>	Tabulador	<code>\b</code>	Retroceso (backspace)
<code>\'</code>	Comilla simple	<code>\nnn</code>	Carácter Unicode nnn en octal
<code>\"</code>	Comilla doble	<code>\xnn</code>	Carácter Unicode nn hexadecimal
<code>\\</code>	Barra inclinada	<code>\xnxxx</code>	Carácter Unicode nxxx hexadecimal

# Elementos del lenguaje

## VARIABLES Y CONSTANTES

- ❑ Se nombran mediante un identificador.
  - Un identificador JavaScript puede incluir caracteres Unicode alfabéticos, dígitos, el guión bajo, el signo de dólar o una secuencia de escape Unicode de un carácter alfabético (`\uxxxx`).
  - Debe comenzar con un carácter alfabético, guión bajo o dólar.
  - El lenguaje es sensible a mayúsculas.

- ❑ Declaración.

- Declaración explícita.

```
var identificador [= expresión de inicialización]  
var precio = 0;  
var ciudad = "Madrid";
```

- Declaración implícita.

- ✓ Asignando directamente un valor a un identificador.

```
identificador = expresión de inicialización  
precio = 0;  
ciudad = "Madrid";
```

- ✓ La declaración implícita declara siempre variables globales.
- ✓ El interprete JavaScript genera una advertencia.
- ✓ No es recomendable.
- La declaración `use strict` de la versión 1.8.5 obliga a hacer una declaración explícita.

# Elementos del lenguaje

## Variables y constantes(II)

---

### ❑ Evaluación de variables.

- Una variable declarada mediante `var` sin valor inicial tiene el valor `undefined`.
- Acceder a una variable no declarada lanza la excepción `ReferenceError`.

### ❑ Ámbito de variables.

- Cualquier valor declarado de forma explícita dentro de una función tiene ámbito local.
- Cualquier valor declarado fuera de una función tiene ámbito global.
  - ✓ Las variables declaradas de forma implícita son siempre globales.
  - ✓ Una variable global es una propiedad del *objeto global*.
  - ✓ En un navegador el objeto global es la ventana, por lo que se puede hacer referencia a ella como `window.variable`.

# Elementos del lenguaje

## Variables y constantes(III)

### ❑ Declaración de constantes.

- Las constantes simbólicas se declaran mediante la palabra reservada `const`.

```
const identificador = expresión
```

- ✓ Los identificadores y las reglas de ámbito son iguales que en las variables.

### ❑ Conversión de tipos de datos.

- JavaScript es un lenguaje dinámico.
  - ✓ No es necesario especificar el tipo de una variable.
  - ✓ El tipo del contenido de una variable puede cambiar a lo largo de la ejecución del script.
  - ✓ El tipo de las variables se puede convertir cuando sea necesario.

```
var dato = 42 //Dato es numérico
dato = "Madrid" //Ahora dato es de cadena

//En una expresión con datos numéricos y de cadena,
//el operador + convierte los datos a cadena
dato = "Edad: " + 30 //Dato es "Edad: 30"

//Si la expresión incluye otros operadores convierte las cadenas a número
dato = "28" - 2 //Dato es 26
dato = "28" + 2 //Dato es 282

//Si no puede hacer la conversión a número devuelve NaN
dato = "Madrid" - 2 //Devuelve NaN (Not a Number)
```

# Elementos del lenguaje

## Operadores de asignación

Operador	Significado
$x = y$	
$x += y$	$x = x + y$ Puede actuar tanto como operador aritmético como de cadena
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% = y$	$x = x \% y$

# Elementos del lenguaje

## Operadores aritméticos

Operador	Descripción	Ejemplo
+	Suma aritmética	3+4 devuelve 7
-	Resta	5-2 devuelve 3
*	Multiplicación	4*2 devuelve 8
/	División real (no existe la división entera)	5/2 devuelve 2.5
% (Módulo)	Operador binario. Devuelve el resto de la división entera entre sus dos operandos.	12 % 5 devuelve 2.
++ (Incremento)	Operador unitario. Suma uno a su operando. Si se usa como prefijo (++x), devuelve el valor de su operando después de la suma; si se usa como sufijo (x++), devuelve el valor de su operando antes de sumarle uno.	Si x es 3, entonces ++x establece x a 4 y devuelve 4, mientras que x++ establece x a 4 y devuelve 3.
-- (Decremento)	Operador unitario. Resta uno a su operando. Su funcionamiento es análogo al del operador de incremento.	Si x es 3, entonces --x establece x a 2 y devuelve 2, mientras que x-- establece x a 2 y devuelve 3.
- (Cambio de signo)	Operador unitario. Devuelve su operando cambiado de signo.	Si x es 3, entonces -x devuelve -3.

# Elementos del lenguaje

## Operadores de relación

Operador	Descripcion	Ejemplos que devuelven verdadero <sup>1</sup>
Igual (==)	Devuelve true si los operandos son iguales.	3 == var1 "3" == var1 3 == '3'
Distinto (!=)	Devuelve true si los operandos no son iguales.	var1 != 4 var2 != "3"
Igual estricto (===)	Devuelve true si los operandos son iguales y del mismo tipo.	3 === var1
Distinto estricto (!==)	Devuelve true si los operandos no son iguales y/o no son del mismo tipo.	var1 !== "3" 3 !== '3'
Mayor que (>)	Devuelve true si el operando izquierdo es mayor que el derecho.	var2 > var1 "12" > 2
Mayor o igual que (>=)	Devuelve true si el operando izquierdo es mayor o igual que el derecho.	var2 >= var1 var1 >= 3
Menor que (<)	Devuelve true si el operando izquierdo es menor que el derecho.	var1 < var2 "12" < "2"
Menor o igual que (<=)	Devuelve true si el operando izquierdo es menor o igual que el derecho.	var1 <= var2 var2 <= 5

Se supone que se han hecho las declaraciones:

```
var var1=3;  
var var2=4;
```

# Elementos del lenguaje

## Operadores lógicos

Operador	Uso	Descripción
&&	expr1 && expr2	AND lógico
	expr1    expr2	OR lógico
!	!expr	NOT lógico

❑ En los operadores lógicos se produce cortocircuito.

- Con el operador &&, si el primer operando es falso, la expresión es falsa.
- Con el operador ||, si el primer operando es verdadero, la expresión es verdadera.



# Elementos del lenguaje

## Operadores especiales

### ❑ Operador condicional (?).

```
expresión_lógica ? expresión1 : expresión2
```

- Evalúa la expresión lógica, si es verdadera devuelve la expresión1, si no la expresión2.

### ❑ Operador coma (,).

- Evalúa las dos expresiones y devuelve la segunda.
- Se utiliza en bucles `for` para modificar el valor de varias variables en cada pasada del bucle.

```
for (var i=0, j=9; i <= 9; i++, j--)
```

✓ En cada pasada incrementa la `i` y decrementa la `j`.

### ❑ Operador `delete`.

- Borra un objeto, una propiedad de un objeto, o un elemento de un array a partir de su índice.

```
delete Nombreobjeto
```

```
delete Nombreobjeto.propiedad
```

```
delete NombreArray[índice]
```

# Elementos del lenguaje

## Operadores especiales (II)

### ❑ Operador `in`.

- Devuelve verdadero si una propiedad existe dentro de un objeto o si un elemento existe dentro de un array.

`Nombre_o_NúmeroPropiedad in objeto_o_array.`

- ✓ `Nombre_o_NúmeroPropiedad` es el nombre de una propiedad o el índice de un array.

### ❑ Operador `instanceof`.

- Devuelve verdadero si el objeto especificado pertenece a la clase indicada.

`nombreObjeto instanceof tipoObjeto`

### ❑ Operador `new`.

- Crea una nueva instancia de un objeto.

### ❑ Operador `this`.

- Hace referencia al objeto actual.

### ❑ Operador `typeof`.

- Devuelve una cadena con el nombre de la clase del operando.

`typeof operando`

# Elementos del lenguaje

## Prioridad de operadores

Tipo de operador	Operadores individuales
miembro	. []
llamada/crear instancia	() new
negación/incremento	! - + ++ -- typeof delete
multiplicación/división	* / %
suma/resta	+ -
relacionales	< <= > >= in instanceof
igualdad	== != === !==
and lógico	&&
or lógico	
condicional	?:
asignación	= += -= *= /= %=
coma	,

# Elementos del lenguaje

## Estructuras alternativas

### □ Sentencia `if..else`.

```
if(condición) {  
    bloque_sentencias  
}  
[else if(condición) {  
    bloque_sentencias  
}]...  
[else {  
    bloque_sentencias  
}]
```

- Cualquier valor que no sea `undefined`, `null`, `0`, `NaN` o cadena nula se evalúa como verdadero cuando se utiliza en la condición de un `if`.

```
var b = 234;  
if (b) // esta condición se evalúa como verdadera
```

# Elementos del lenguaje

## Estructuras alternativas (II)

### □ Sentencia `switch`.

```
switch (expresión) {  
    case valor1:  
        bloque_sentencias  
        [break;]  
    case valor2:  
        bloque_sentencias  
        [break;]  
    ...  
    [default:  
        bloque_sentencias]  
}
```

- La expresión puede ser de cualquier tipo.

# Elementos del lenguaje

## Estructuras alternativas (III)

```
function comprobarDatos(){
  if (document.form1.txtCodigoPostal.value.length == 5){
    return true; }
  else
  { alert("El código postal debe tener 5 caracteres. "
    + document.form1.txtCodigoPostal.value + " no es válido.");
    return false; }
}
```

```
switch (diaSemana){
  case "Lunes": numeroDiaSemana = 1;
    break;
  case "Martes": numeroDiaSemana = 2;
    break;
  case "Miércoles": numeroDiaSemana = 3;
    break;
  case "Jueves": numeroDiaSemana = 4;
    break;
  case "Viernes": numeroDiaSemana = 5;
    break;
  case "Sábado": numeroDiaSemana = 6;
    break;
  case "Domingo": numeroDiaSemana = 0;
    break;
  default: alert("Entrada de datos errónea");
}
```

# Elementos del lenguaje

## Estructuras repetitivas

□ Funcionan de forma similar a C o Java.

- Sentencia `for`.

```
for ([Expresión inicial]; [condición]; [incremento de la Expresión])  
    bloque_sentencias
```

- Sentencia `do..while`.

```
do  
    bloque_sentencias  
while (condición);
```

- Sentencia `while`.

```
while (condición)  
    bloque_sentencias
```

- Sentencia `break`.
- Sentencia `continue`.

# Elementos del lenguaje

## Estructuras repetitivas (II)

### □ Sentencia `for..in`.

```
for (variable in objeto) {  
    bloque_sentencias }
```

- La variable *variable* itera por todas las propiedades de *objeto*.
- Suponiendo que el objeto *alumno* tiene las propiedades *expediente* y *nombre*...

```
var alumno= {expediente:"091234",nombre:"Rodriguez Baena, Luis"};  
for (var dato in alumno){  
    alert(dato+" = " + alumno[dato]);  
}
```



# Elementos del lenguaje

## Funciones

### ❑ Definición de funciones.

```
function nombreFunción([argumentos]) {  
    cuerpo_función  
    [return expresión;]  
}
```

- *argumentos* es una lista de identificadores separados por comas.
- Los argumentos se pasan por valor.
  - ✓ En el caso de que se trate de objetos (los arrays también se consideran un objeto), las propiedades del mismo (o los elementos del array) pueden reflejar los cambios producidos dentro de la función.
- Una función se puede definir dentro de una expresión.
  - ✓ Se trata de funciones anónimas.

```
const cuadrado = function(número) {return número * número};  
document.write(cuadrado(2)); //Devuelve 4
```

- ✓ Pueden ser útiles si una función utiliza otra función como argumento.

```
function cuadrados(función, arrayValores) {  
    var resultado = new Array;  
    for(var i=0; i<arrayValores.length; i++){  
        resultado[i] = función(arrayValores[i]);  
    }  
    return resultado;  
}  
var a = cuadrados(function(x) {return x*x}, [0,1,3,5,7]);  
//a = [0,1,9,25,49]
```

# Elementos del lenguaje

## Funciones (II)

### □ Argumentos opcionales.

- Los argumentos de una función se almacenan en el objeto `arguments`.
  - ✓ `arguments` es un array que almacena cada uno de los parámetros actuales de la llamada.
  - ✓ Utilizando `arguments`, es posible definir una función con un número de argumentos variable.

```
function suma(números){
    var suma=0;
    for(i=0;i<arguments.length;i++){
        suma += arguments[i]; }
    return suma;
}

document.write(suma(3) + "<br>");           //Devuelve 3
document.write(suma(3,4,5) + "<br>");       //Devuelve 12
document.write(suma(3,10,20,30) + "<br>");  //Devuelve 63
```

# Elementos del lenguaje

## Funciones globales

- ❑ Función `isFinite(número)`.
  - Devuelve `false` si `número` es NaN, infinito positivo o infinito negativo.
- ❑ Función `isNaN(número)`.
  - Devuelve `true` si `número` no puede ser tratado como número.
- ❑ Función `parseFloat(cadena)`.
  - Intenta convertir `cadena` a un número real.
  - Si encuentra un carácter distinto de signo, un número, un punto decimal o un exponente devuelve todos los caracteres anteriores a ese.
  - Si no puede convertir ningún carácter devuelve NaN.

```
if isNaN(parseFloat(dato)) {  
    //Acciones si dato no es un número
```

- ❑ Función `parseInt(cadena [, base])`.
  - Intenta convertir `cadena` a un número entero.
  - `base` indica la base en la que está representado el número.

# Elementos del lenguaje

## Arrays

- ❑ No se trata de tipos de datos predefinidos, sino objetos de la clase `Array`.

- ❑ Creación de arrays (constructores).

```
nombreArray = new Array(elemento0, elemento1, ..., elementoN)
```

- ✓ Los elementos pueden ser de distinto tipo.

```
nombreArray = new Array(longitud)
```

- ✓ Todos los elementos serían `undefined`.

- ❑ Propiedad `length`.

- Devuelve el número de elementos del array.

- ❑ La referencia a los elementos del array y su proceso mediante bucles se hace como en cualquier otro lenguaje.

```
var diaSemana = new Array("lunes", "martes", ..., "sábado", "domingo")
for (var i=0; i<diaSemana.length; i++)
    document.write(diaSemana[i] + "<br>");
```

# Elementos del lenguaje

## Arrays (II)

### □ Literales de array.

- Se pueden crear literales de tipo array mediante una lista de varios elementos encerrados entre corchetes.

```
["Alava", "Albacete", "Almería", "Zaragoza"]
```

- ✓ Al definir el literal se pueden dejar comas intermedias para elementos vacíos.

```
["primero",,,, "último"]
```

- ✓ El propio literal define un array que se puede asignar a una variable para crear el array, utilizar como argumento a una función que precise de un array, iterar dentro de un `for each`, etc.

```
var provincias = ["Alava", "Albacete", "Almería", "Zaragoza"]

//Esto sólo funciona con Js 1.6 Funciona en Firefox 1.5 y posteriores
//(no funciona con las versiones navegadores de hace dos o tres años)
for each(var provincia in ["Alava", "Albacete", "Almería", "Zaragoza"])
    document.write(provincia + "<br>");

for(var i=0; i<provincias.length; i++)
    document.write(provincias[i] + "<br>");
```

# Elementos del lenguaje

## Arrays: métodos

### ❑ Método `concat`.

```
array1.concat(array2)
```

- Devuelve un array formado por los elementos de `array1` y `array2`.

### ❑ Método `join`.

```
array.join(delimitador)
```

- Devuelve una cadena formada por los elementos de `array` separados por `delimitador`.

### ❑ Método `pop`.

```
array.pop()
```

- Elimina el último elemento de `array` y devuelve ese último elemento.

### ❑ Método `push`.

```
array.push()
```

- Añade un elemento al final de `array` y devuelve el número de elementos del nuevo array.

### ❑ Método `reverse`.

```
array.reverse()
```

- Devuelve un array con los elementos de `array` en orden inverso.

# Elementos del lenguaje

## Arrays: métodos (II)

```
var provincias = ["Alava","Albacete","Almería","Zaragoza"]

var otrasProvincias = ["Madrid","Barcelona"];
document.write(provincias.concat(otrasProvincias)+"<br>");
//Escribe: Alava,Albacete,Almería,Zaragoza,Madrid,Barcelona

var cadenaProvincias=provincias.join(":");
document.write(cadenaProvincias+"<br>");
//Escribe: Alava:Albacete:Almería:Zaragoza

var ultimo=provincias.pop();
document.write(provincias+"<br>");
document.write("Elemento sacado:" + ultimo + "<br>");
//Escribe: Alava,Albacete,Almería
//Elemento sacado:Zaragoza

var nuevo=provincias.push("Fin");
document.write(provincias+"<br>");
document.write("Elemento nuevo:" + nuevo + "<br>");
//Escribe: Alava,Albacete,Almería,Fin
// Elemento nuevo:4

var nuevo=provincias.reverse();
document.write(provincias+"<br>");
//Escribe: Fin,Almería,Albacete,Alava
```

# Elementos del lenguaje

## Arrays: métodos (III)

### ❑ Método `shift`.

```
array.shift()
```

- Elimina el primer elemento de `array` y devuelve dicho elemento.

### ❑ Método `slice`.

```
array.slice(indice1, indice2)
```

- Devuelve una copia del array entre `indice1` e `indice2` (sin incluir éste).

### ❑ Método `splice`.

```
array.splice(indice1, numElementos, elemento1, elemento2, ...)
```

- Sustituye a partir del elemento `indice1`, tantos elementos como indique `numElementos`, por `elemento1`, `elemento2`, etc. Devuelve un array con los elementos cambiados.

### ❑ Método `unshift`.

```
array.unshift(elemento1, elemento2, ...)
```

- Añade los elementos indicados al comienzo del array y devuelve el número de elementos.



# Elementos del lenguaje

## Arrays: métodos (IV)

```
//provincias es Fin,Almería,Albacete,Alava

var primero=provincias.shift();
document.write(provincias+"<br>");
document.write("Primer elemento:" + primero + "<br>");
//Escribe: Almería,Albacete,Alava
//Primer elemento:Fin

document.write(provincias.slice(1,3)+"<br>");
//Escribe: Albacete,Alava (el array sigue siendo [Almeria, Albacete, Alava]

provincias.splice(1,1,"un elemento","otro elemento","un elemento más");
document.write(provincias + "<br>");
//Escribe: Almería,un elemento,otro elemento,un elemento más,Alava
provincias.splice(0,0,"primero");
document.write(provincias + "<br>");
//Escribe: primero,Almería,un elemento,otro elemento,un elemento más,Alava

document.write("Número de elementos:" + provincias.unshift("-1","-2")+ "<br>");
document.write(provincias + "<br>");
//Escribe: Número de elementos:8
//-1,-2,primero,Almería,un elemento,otro elemento,un elemento más,Alava
```

# Elementos del lenguaje

## Arrays: métodos (V)

### □ Método `sort`.

`array.sort()`

- Ordena `array` de forma ascendente y devuelve dicho array ordenado.

- ✓ La ordenación se realiza considerando los elementos como cadenas.

- `sort` puede usar como argumento una función para determinar cómo ordenar el array.

`array.sort(función)`

- ✓ La función recibe como argumentos dos valores `a` y `b` y devuelve 1, 0 o -1 según un elemento sea mayor, igual o menor que el otro.

- Si al ordenar `a` es menor que `b`, devuelve -1.
- Si al ordenar `a` es mayor que `b`, devuelve 1.
- Si al ordenar `a` es igual a `b`, devuelve 0.

# Elementos del lenguaje

## Arrays: métodos (VI)

```
var colores = ["rojo","verde","azul","amarillo"]
document.write(colores.sort() + "<br>");
//Escribe: amarillo,azul,rojo,verde

//Crea un array de alumnos. Cada alumno es un objeto con nombres y apellido
var alumnos = new Array(3);
alumnos[0] = {nombre:"Pedro",apellidos:"Pérez"};
alumnos[1] = {nombre:"Juan",apellidos:"Pérez"};
alumnos[2] = {nombre:"Luis",apellidos:"Martínez"};

//Función anónima para determinar el orden de los elementos
var ordenarPorApellido = function(a,b){
    if (a.apellidos < b.apellidos) return -1;
    if (a.apellidos > b.apellidos) return 1;
    if (a.apellidos = b.apellidos){
        if(a.nombre < b.nombre) return -1;
        if(a.nombre > b.nombre) return 1
        if(a.nombre = b.nombre) return 0;
    }
}

alumnos.sort(ordenarPorApellido);
for(var i=0;i<alumnos.length;i++)
    document.write(alumnos[i].nombre + " " + alumnos[i].apellidos + "<br>");
//Escribe: Luis Martínez
//Juan Pérez
//Pedro Pérez
```

# Elementos del lenguaje

## Objeto Date

- ❑ Representa una fecha como el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 0 horas (un día tiene 86.400.000 milisegundos).
  - Admite valores entre -100.000.000 y 100.000.000 de días a partir del 1 de enero de 1970.
- ❑ Constructores:
  - `new Date()`, representa la fecha y hora actual.
  - `new Date(milisegundos)`, crea una fecha a partir del número de milisegundos.
  - `new Date(cadena)`, crea una fecha a partir de una cadena.
    - ✓ La cadena podrá ser como como "Dec 25, 1995", "Mon, 25 Dec 1995 13:30:00 GMT" o similar.
  - `new Date(año, mes, día [, hora, minuto, segundo, milisegundo])`, crea una fecha a partir de los datos que se le pasan.
    - ✓ Los meses van entre 0 y 11.
- ❑ Métodos:
  - Método `parse`.
    - ✓ `Date.parse(cadena)` devuelve el número de milisegundos que representa `cadena`.
  - Métodos `get`: devuelven la información de una fecha.
    - ✓ `getDate()` (día del mes), `getDay()` (día de la semana), `getFullYear()` (año en cuatro cifras), `getHours()`, `getMilliseconds()`, `getMinutes()`, `getMonth()`, `getSeconds()`, `getTime()` (número de milisegundos).
  - Métodos `set`: establecen los valores de una fecha.
    - ✓ `setDate(díaMes)`, `setFullYear(año4cifras)`, `setHours(hora24horas)`, `setMilliseconds(milisegundos)`, `setMinutes(minutos)`, `setMonth(mes)`, `setSeconds(segundos)`, `setTime(milisegundosFecha)`.

# Elementos del lenguaje

## Objeto String: propiedades y métodos

### ❑ Propiedad `length`.

`cadena.length`

- Devuelve el número de caracteres de una cadena.

### ❑ Método `charAt`.

`cadena.charAt(índice)`

- Devuelve el carácter *índice* de la cadena.

### ❑ Método `charCodeAt`.

`cadena.charCodeAt(índice)`

- Devuelve el valor Unicode del carácter *índice* de la cadena.

### ❑ Método `fromCharCode`.

`String.fromCharCode(num1, num2, ...)`

- Crea una cadena a partir de los caracteres Unicode correspondientes a los números especificados.

### ❑ Métodos `indexOf` y `lastIndexOf`.

`cadena.indexOf(cadenaBuscada, [posición])`

`cadena.lastIndexOf(cadenaBuscada, [posición])`

- Devuelve la primera o última posición de *cadenaBuscada* a partir de *posición*.

# Elementos del lenguaje

## Objeto String: propiedades y métodos (II)

### ❑ Método `concat`.

`cadena.concat(cadena1, cadena2, ...)`

- Concatena a `cadena` las cadenas que se pasan como argumento.

### ❑ Método `substring`.

`cadena.substring(inicio, [fin])`

- Devuelve una subcadena de `cadena` a partir de la posición `inicio` hasta la posición `fin` (sin incluir ésta).

### ❑ Método `replace`.

`cadena.replace(cadena1, cadena2, [flag])`

- Sustituye las apariciones de `cadena1` por `cadena2`.
- Sustituye la primera aparición a no ser que se incluya el `flag` "g".

### ❑ Método `split`.

`cadena.split([delimitador] [, numDivisiones])`

- Separa la cadena por el `delimitador` y devuelve un array con cada uno de las partes. Si se omite el delimitador devuelve un elemento con la cadena completa. `numDivisiones` permite especificar el número de elementos a dividir.

### ❑ Métodos `toUpperCase` y `toLowerCase`.

# Elementos del lenguaje

## Objeto String: propiedades y métodos (III)

```
var cad = "cocodrilo";
document.write(cad.charAt(2)+"<br>"); //Devuelve c

document.write("cojín".charCodeAt(3)+"<br>"); //Devuelve 237

document.write(String.fromCharCode(65,66,67) +"<br>"); //Devuelve ABC

document.write(cad.indexOf("o")+"<br>"); //Devuelve 1
document.write(cad.indexOf("o",2)+"<br>"); //Devuelve 3
document.write(cad.lastIndexOf("o")+"<br>"); //Devuelve 8
document.write(cad.lastIndexOf("o",5)+"<br>"); //Devuelve 3

document.write(cad.concat(" 1"," 2 ","3")+"<br>"); //Devuelve 1 2 3

document.write(cad.substring(2)+"<br>"); //Devuelve codrilo
document.write(cad.substring(4,7)+"<br>"); //Devuelve dri

document.write(cad.replace("o","u","g")+"<br>"); //Devuelve cucudrilu

var datos="Juan:Pérez:23";
document.write(datos.split(":")+"<br>"); //Devuelve Juan,Pérez,23
document.write(datos.split(":",2)+"<br>"); //Devuelve Juan,Pérez

document.write(datos.toUpperCase()+"<br>"); //Devuelve JUAN:PÉREZ:23
document.write(datos.toLowerCase()+"<br>"); //Devuelve juan:pérez:23
```

# Document Object Model

- ❑ El Document Object Model (DOM) es una interfaz de programación para documentos HTML y XML.
  - Establece una representación estructurada del documento.
  - Proporciona interfaces para acceder y modificar la estructura, el contenido y la presentación del documento.
- ❑ Representa el documento como un conjunto de nodos estructurados con sus propiedades y métodos.
- ❑ Mediante las API de DOM, dicha representación puede modificarse desde cualquier lenguaje de programación.
  - Ofrece métodos para navegar entre los elementos del documento, acceder a elementos determinados, a su contenido o propiedades y modificarlos o crear nuevos elementos.
- ❑ Se trata de un estándar del W3C ([www.w3.org/DOM/](http://www.w3.org/DOM/)) no plenamente implementado por todos los navegadores.



# Document Object Model

## El árbol del documento

- ❑ DOM considera un documento xhtml (bien formado) como un árbol de nodos.
- ❑ Para el siguiente código html...

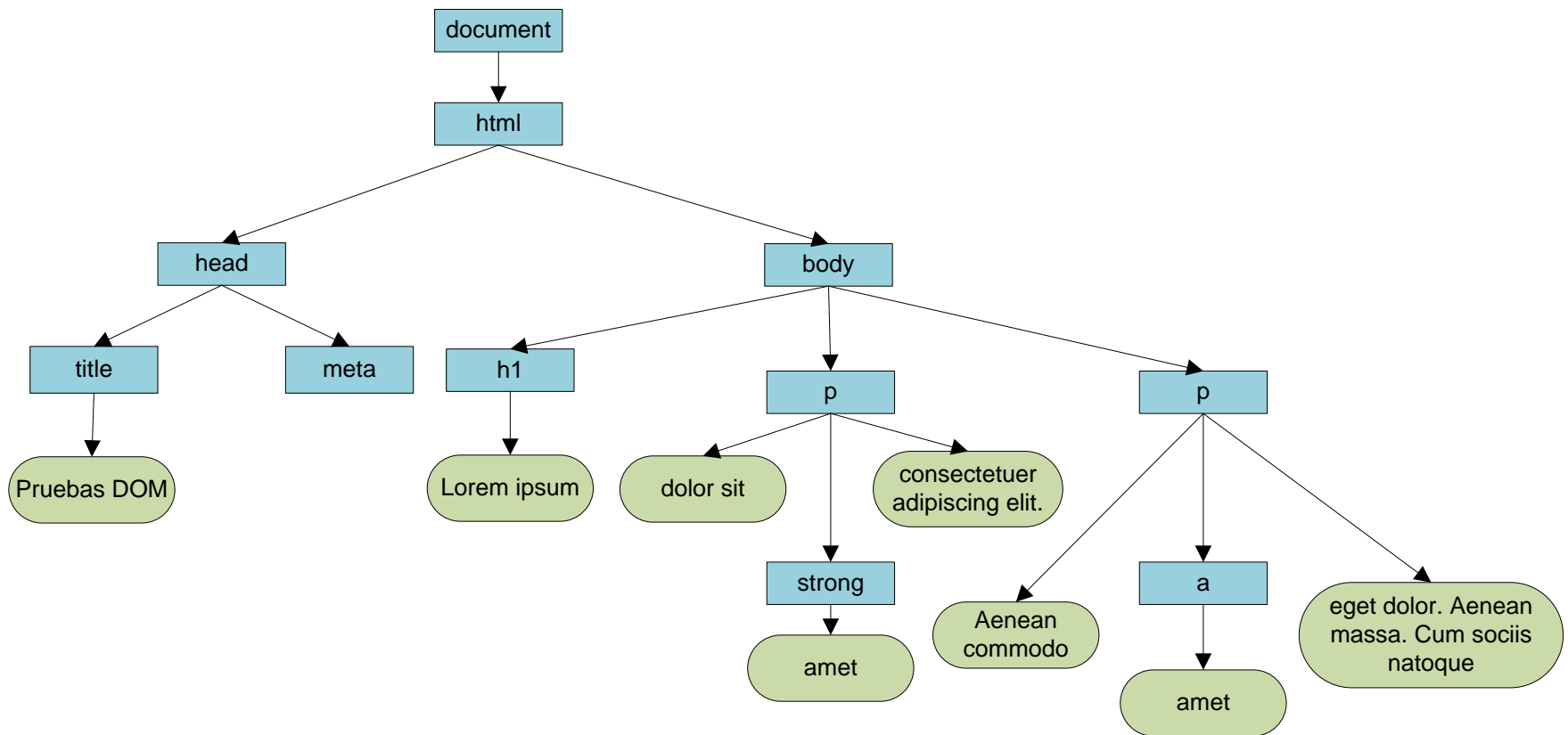
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
<head>
  <title>Pruebas DOM</title>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
<body><h1>Lorem ipsum</h1>
<p>dolor sit <strong>amet</strong>, consectetur adipiscing elit.</p>

<p>Aenean commodo <a href="http://www.upsam.com">ligula</a> eget dolor. Aenean massa.
  Cum sociis natoque
</p>
</body>
</html>
```

# Document Object Model

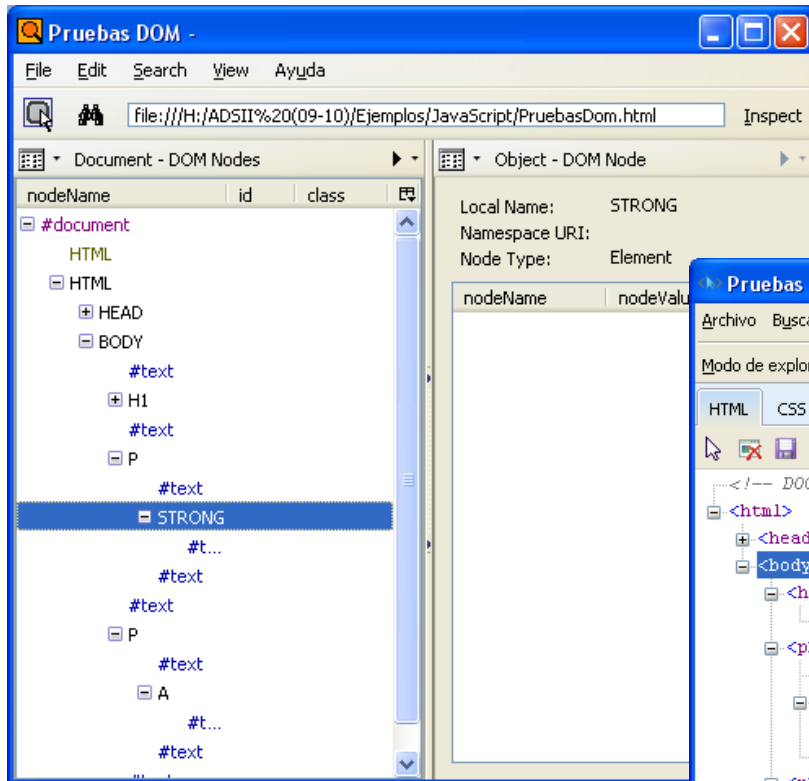
## El árbol del documento (II)

□ Este sería el árbol de nodos resultante...

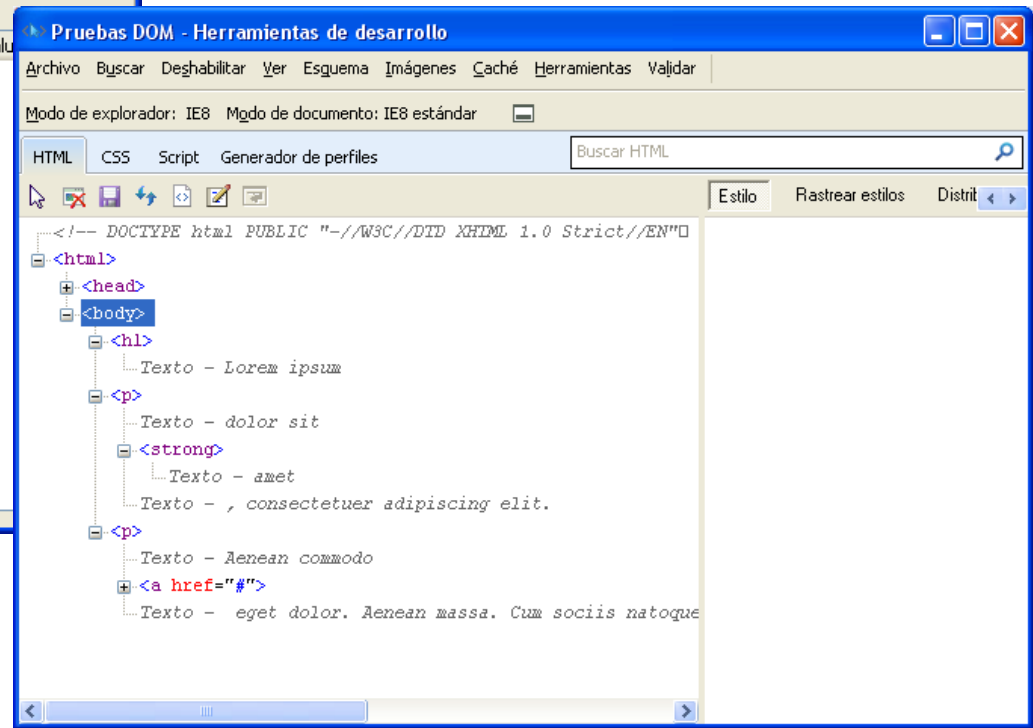


# Document Object Model

## El árbol del documento (III)



Árbol del documento con DOM Inspector para Firefox



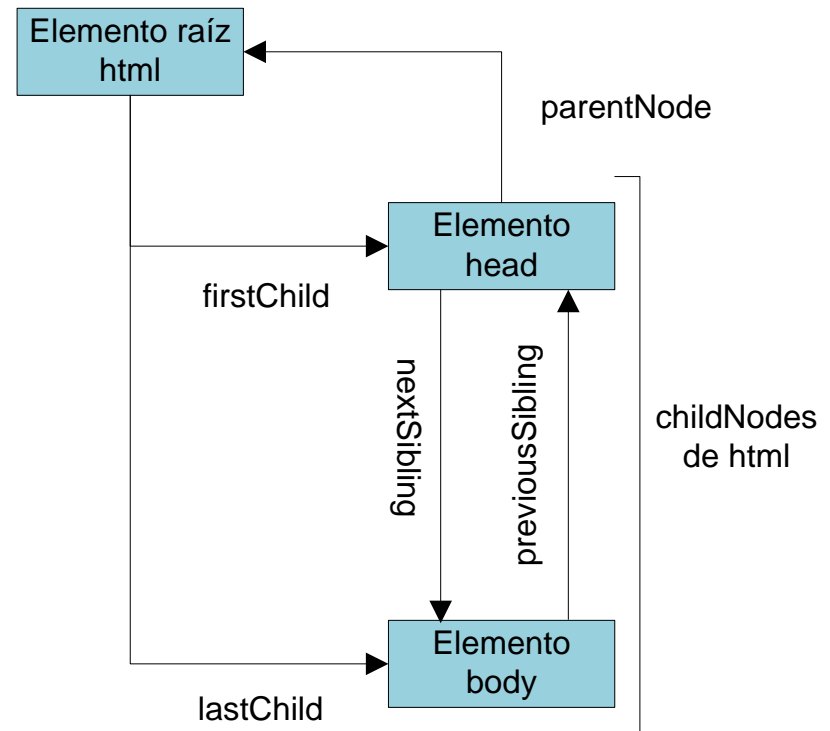
Árbol del documento con las herramientas de desarrollo de Internet Explorer ( F12)

# Document Object Model

## El árbol del documento (IV)

### □ Relaciones entre elementos.

- Los términos padre (parent), hijo (child) y hermanos (sibling) definen las relaciones entre nodos.
- El nodo de jerarquía superior sería el nodo raíz.
- Cada nodo, excepto el raíz tiene un nodo padre.
- Un nodo puede tener cualquier número de hijos.
- Una hoja es un nodo sin hijos.
- Los nodos hermanos, son nodos del mismo padre.



# Document object model

## Tipos de nodos

---

- ❑ DOM define 12 tipos de nodos. Los más importantes son:
  - `Document`. Hace referencia al nodo raíz del que derivan todos los demás nodos.
  - `Element`. Representa cada una de las etiquetas del documento.
    - ✓ Puede contener atributos y de él pueden derivar otros nodos.
  - `Attr`. Representa cada uno de los atributos de una etiqueta, representados por parejas *nombreAtributo = valor*.
  - `Text`. Representa el contenido de un elemento.
  - `Comment`. Representa un comentario.
- ❑ El resto de tipos de nodos son `DocumentType`, `CDataSection`, `DocumentFragment`, `Entity`, `EntityReference`, `ProcessingInstruction` y `Notation`.

# Document object model

## Tipos de datos

---

- DOM utiliza los siguientes tipos de datos (clases):
  - `document`. Hace referencia al elemento raíz. Se trataría de un nodo de tipo documento.
  - `element`. Hace referencia a un nodo de tipo elemento.
  - `nodeList`. Se trata de un conjunto de nodos.
    - ✓ Para hacer referencia a cada uno de ellos se utiliza la sintaxis de arrays.
      - Si `lista` es un `nodeList`, se accedería al primer elemento mediante `lista[i]`.
  - `attribute`. Se corresponde a un atributo de un elemento.
  - `namedNodeMap`. Una lista de nodos especial a la que se puede acceder tanto a partir del índice como del nombre del elemento.

# Document Object Model

## Acceso a los nodos

- ❑ DOM proporciona distintas interfaces para acceder a los nodos.
- ❑ Acceso a los elementos raíz del documento.
  - `document.body`, hace referencia al elemento `body` del documento.
  - `document.documentElement`, hace referencia al elemento `html`.
- ❑ Existen dos formas de acceder:
  - Acceso a partir de otros nodos.
    - ✓ Los elementos del árbol tienen las propiedades `parentNode`, `firstChild`, `lastChild`, `nextSibling` y `previousSibling` que devuelven nodos a partir de un nodo dado.
    - ✓ La propiedad `childNodes` devuelve un `nodeList` de los elementos hijos de un nodo dado.
  - Acceso directo a partir de las características de un nodo.
    - ✓ Tanto el objeto `document` como el objeto `element` tienen métodos para acceder a un nodo a partir de la etiqueta `html`, el valor de la propiedad `name` (obsoleto), mediante su clase o mediante el `id` de un elemento.
      - `getElementsByTagName`, `getElementsByName`, `getElementsByClassName` y `getElementById`.

# Document Object Model

## Acceso a los nodos (II)

### ❑ Método `getElementsByTagName`.

- Devuelve un `nodeList` con los nodos que correspondan a una etiqueta html.

`nodo.getElementsByClassName(etiquetaHTML)`

- ✓ Devuelve los nodos cuya etiqueta sea igual a `etiquetaHTML` que se encuentre dentro de `nodo`.

`var párrafos = document.getElementsByClassName("p")`

- párrafos se cargaría con todos los elementos `p` del documento.

```
<body>
<h1>Lorem ipsum</h1>
<p>dolor sit <strong>amet</strong>, consectetur adipiscing elit.</p>
<p>Aenean commodo <a href="http://www.upsam.com">ligula</a> eget dolor.
Aenean <a href="http://www.colimbo.net">massa</a>. Cum sociis natoque</p>
<script type="text/javascript">
//
    var parrafos = document.getElementsByClassName("p");
    document.write("Num. de párrafos del documento:" + parrafos.length + "&lt;br&gt;");
    var enlaces = parrafos[1].getElementsByTagName("a");
    for(var i=0; i &lt; enlaces.length;i++)
        document.write(enlaces[i].innerHTML + "&lt;br&gt;");
    // Escribe ligula y masa
//]]&gt;
&lt;/script&gt;
&lt;/body&gt;</pre></div><div data-bbox="3 947 467 991" data-label="Page-Footer"><p>Universidad Pontificia de Salamanca. Escuela Superior de Ingeniería y Arquitectura<br/>(CC) Luis Rodríguez Baena, 2013</p></div><div data-bbox="972 959 993 980" data-label="Page-Footer"><p>48</p></div>
```



# Document Object Model

## Acceso a los nodos (III)

### □ Método `getElementById`.

- Devuelve el nodo que tenga como valor del atributo `id` el dato que se pasa como argumento.

`nodo.getElementsById(valorID)`

- ✓ Devuelve el elemento html, descendiente de `nodo` cuyo identificador sea igual a `valorID`.

```
<body>
<h1>Lorem ipsum</h1>
<p>dolor sit <strong>amet</strong>, consectetur adipiscing elit.</p>
<p id="segundoparrafo">Aenean commodo <a href="http://www.upsam.com">ligula</a> eget dolor.
Aenean <a href="http://www.colimbo.net">massa</a>. Cum sociis natoque</p>
<script type="text/javascript">
//
    var nodo = document.getElementById("segundoparrafo");
    var enlaces = nodo.getElementsByTagName("a");
    for(var i=0; i &lt; enlaces.length;i++)
        document.write(enlaces[i].innerHTML + "&lt;br&gt;");
    // Escribe ligula y masa
//]]&gt;
&lt;/script&gt;
&lt;/body&gt;</pre></div><div data-bbox="3 947 467 991" data-label="Page-Footer"><p>Universidad Pontificia de Salamanca. Escuela Superior de Ingeniería y Arquitectura<br/>(CC) Luis Rodríguez Baena, 2013</p></div><div data-bbox="975 960 993 980" data-label="Page-Footer"><p>49</p></div>
```

# Document Object Model

## Algunas propiedades de los nodos

---

### ❑ `nodo.nodeName`.

- Devuelve una cadena con el nombre de `nodo`.
- Propiedad de sólo lectura.
- Según el tipo del nodo devolverá:
  - ✓ `Document`, "#document".
  - ✓ `Element`, en un documento html, el nombre de la etiqueta html.
  - ✓ `Attr`, el nombre del atributo.
  - ✓ `text`, "#text".
  - ✓ `Comment`, "#coment".

### ❑ `nodo.nodeValue`.

- Devuelve o establece el valor de `nodo`.
- El valor será para los distintos tipos de nodos...
  - ✓ `Document`, `null`.
  - ✓ `Element`, `null`.
  - ✓ `Attr`, valor del atributo.
  - ✓ `text`, contenido del texto.
  - ✓ `Comment`, contenido del comentario.

# Document Object Model

## Algunas propiedades de los nodos (II)

- ❑ `nodo.nodeType`.
  - Devuelve valor numérico con el tipo de `nodo`.
  - Propiedad de sólo lectura.
  - Según el tipo del nodo devolverá:
    - ✓ `Document`, 9.
    - ✓ `Element`, 1.
    - ✓ `Attr`, 2.
    - ✓ `text`, 3.
    - ✓ `Comment`, 8.
- ❑ `nodo.innerHTML`
  - Devuelve o establece el contenido HTML de `nodo`.
    - ✓ Aunque no forma parte del estándar del W3C, la gran mayoría de los navegadores la utiliza.
    - ✓ Se emplea comúnmente para modificar de forma dinámica el código html de un documento.
- ❑ `nodo.childNodes`
  - Devuelve un `nodeList` con los nodos hijos de `nodo`.
  - Funciona de forma distinta en Mozilla (Firefox y Chrome) y en versiones antiguas de IE. En Mozilla cuenta como nodo los espacios entre elementos, mientras que IE sólo cuenta como elementos los elementos html.
    - ✓ Para acceder a los nodos es mejor utilizar el método `getElementById`.

# Document Object Model

## Modificar la estructura DOM

---

- ❑ El modelo de objetos DOM proporciona los métodos necesarios para modificar la estructura de DOM.
  - Algunos métodos...
    - ✓ Los métodos `createElement` y `createTextElement` permiten crear nuevos nodos.
    - ✓ Los métodos `appendChild` e `insertBefore` permiten insertar nuevos nodos en la estructura DOM.
    - ✓ El método `removeChild` permite eliminar nodos.
    - ✓ El método `replaceChild` permite sustituir un nodo por otro.
    - ✓ El método `cloneNode` permite copiar un nodo.

# Document Object Model

## Crear nodos html

- ❑ En el árbol de nodos, los elementos html con contenido presentan, al menos, dos nodos:
  - Un nodo `Element` con la etiqueta.
  - Un nodo `Text` con el contenido de la etiqueta que será hijo del nodo `Element`.
- ❑ Para añadir un nuevo nodo html habrá que...
  - Crear un nuevo nodo de tipo `Element` que represente a la etiqueta mediante el método del objeto `Document createElement`.
  - Crear un nuevo nodo de tipo `Text` que con el contenido del elemento mediante el método del objeto `Document createTextNode`.
  - Añadir el nodo de tipo `Text` al elemento con el método `appendChild`.
  - Añadir el elemento en la página en el lugar correspondiente.
    - ✓ El método `appendChild` inserta el elemento como último nodo del padre.
    - ✓ El método `insertBefore` inserta el elemento dentro del nodo padre, antes de otro nodo hijo.

# Document Object Model

## Crear nodos html (II)

### ❑ Método `createElement`.

`document.createElement(etiquetaHTML)`

- `etiquetaHTML` es una cadena con la etiqueta.
- Devuelve un nodo de tipo `Element` con la etiqueta especificada.

### ❑ Método `createTextElement`.

`document.createTextElement(contenido)`

- Devuelve un nodo de tipo `Text` con el contenido especificado.

### ❑ Método `appendChild`.

`nodoPadre.appendChild(nodoHijo)`

- Hace que `nodoHijo` se coloque como último hijo de `nodoPadre`.
- Si `nodoHijo` ya existe, lo elimina de dónde esté y lo coloca en la nueva posición.

### ❑ Método `insertBefore`.

`nodoPadre.insertBefore(nodoAñadido, nodoSiguiente)`

- Inserta el `nodoAñadido` como hijo de `nodoPadre` antes del `nodoSiguiente` referenciado.
- Si `nodoSiguiente` no existe, lo inserta como último nodo de `nodoPadre`.

# Document Object Model

## Crear nodos html (III)

```
function insertarNodoAlFinal(){
    var elemento= document.createElement("p");
    var texto = document.createTextNode("Insertado al final...");
    elemento.appendChild(texto);
    document.body.appendChild(elemento);
}
funcion moverNodoAlFinal(nodo){
    document.body.appendChild(nodo);
}
function insertarNodoAntes(){
    var elemento= document.createElement("p");
    var texto = document.createTextNode("Insertado antes del último párrafo...");
    elemento.appendChild(texto);
    var nodo = document.getElementById("ultimoParrafo");
    document.body.insertBefore(elemento,nodo);
}
<body>
<p id="primerparrafo" onclick="insertarNodoAlFinal()">Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Aenean commodoligula eget dolor. Aenean massa. ...</p>

<p onclick="moverNodoAlFinal(this)">Donec pede justo, fringilla vel,... </p>

<p id="ultimoParrafo" onclick="insertarNodoAntes()">Etiam rhoncus. Maecenas
tempus, tellus eget ... </p>
</body>
```

# Document Object Model

## Eliminación de nodos

---

- ❑ El método `removeChild` permite eliminar un nodo hijo de un nodo.
  - `nodoPadre.removeChild(nodo)`
  - Elimina `nodo` de `nodoPadre`.
  - Devuelve el nodo eliminado.
    - ✓ Aunque `nodo` no esté dentro de DOM se mantiene en memoria, por lo que es posible reutilizarlo.
    - ✓ Si `nodo` no existe, se genera una excepción.
  - Para asegurarse de quién es `nodoPadre` se puede utilizar la propiedad `parentNode` de `nodo`.



# Document Object Model

## Eliminación de nodos (II)

```
function eliminarNodo(){
    nodoViejo = document.getElementById("enlace");
    //No se declara con var para que nodoViejo sea global
    nodoViejo.parentNode.removeChild(nodoViejo);
}
function recuperarEnlace(){
    document.getElementById("ultimoParrafo").appendChild(nodoViejo);
}
<p>Aliquam lorem ante <a href="#" id="enlace"
onclick="eliminarNodo()">,dapibus in</a>, viverraquis, feugiat a,
tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque
rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue.
Curabitur ullamcorper ultricies nisi. Nam eget <strong
id="ultimapalabra" onclick="recuperarEnlace()">dui.</strong></p>
<p id="ultimoParrafo" onclick="insertarNodoAntes()">Etiam rhoncus.
Maecenas tempus, tellus eget condimentum rhoncus, sem quam ... </p>
```

# Document Object Model

## Reemplazar y copiar nodos

---

□ El método `replaceChild` permite cambiar un nodo por otro.

*nodoPadre.replaceChild(nodoNuevo, nodoViejo)*

- Cambia *nodoViejo* por *nodoNuevo*.
- Devuelve *nodoViejo*.
- Si *nodoNuevo* ya existe, primero lo elimina.
- Si *nodoViejo* no existe, genera una excepción.

# Document Object Model

## Reemplazar y copiar nodos (II)

□ El método `cloneNode` devuelve una copia de un nodo.

`nodo.cloneNode(copiarHijos)`

- `cloneNode` no inserta nada, sólo devuelve una copia de `nodo` con todos sus atributos.
  - ✓ Si se desea incluir esa copia en el árbol de nodo habría que recurrir al método `appendChild`.
- `copiarHijos` es un valor lógico.
  - ✓ Si se pone a falso, no se clonan los nodos hijos, incluido el contenido del nodo.

```
function remplazarÚltimoPorPrimero() {
    var nodoViejo = document.getElementById("ultimoParrafo");
    var nodoNuevo =
        document.getElementById("primerparrafo").cloneNode(true);
    document.body.replaceChild(nodoNuevo, nodoViejo);
}
```

# Document Object Document

## Acceso a los atributos de un elemento

---

- ❑ La propiedad `attributes` permite acceder a los atributos de un elemento.

`elemento.attributes`

- Devuelve un dato de tipo `namedNodeMap`.
  - ✓ Un dato de tipo `namedNodeMap` es una colección a la que se puede acceder tanto por el índice de cada elemento, como por el nombre del atributo.
    - DOM no determina el orden de los elementos.
- Cada elemento de la colección es un nodo de tipo `Attr`.
  - ✓ La propiedad `name` del nodo devuelve el nombre del atributo.
  - ✓ La propiedad `value` del nodo devuelve el valor del atributo.

# Document Object Document

## Acceso a los atributos de un elemento (II)

```
<p id="primerparrafo" class="normal"
onclick="enmarcarPárrafo()">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. <a href="http://www.colimbo.net">Aenean commodo</a>
ligula eget dolor...</p>
```

```
var nodo = document.getElementById("primerparrafo");
//Devuelve 3
alert("Número de atributos del primer párrafo:" +
      nodo.attributes.length);
//Puede devolver id, class u onclick
alert("Nombre del segundo atributo:" + nodo.attributes[1].name);
//Devuelve primerparrafo
alert("Valor del atributo id: " + nodo.attributes["id"].value);
```

# Document Object Model

## Acceso a los atributos de un elemento (III)

---

- ❑ Los elementos html de DOM, tienen propiedades para cada uno de los atributos definidos en los elementos html.
  - Se accede mediante `elemento.nombreAtributo`.
    - ✓ Los nombre de los atributos son los mismos que en html, con la excepción del atributo `class` que es `className`.
  - Devuelve el valor del atributo `nombreAtributo` de `elemento`.
  - Se puede utilizar para establecer el valor de un atributo.
    - ✓ Para modificar el valor de un atributo, no se debe utilizar la colección `attributes`.

# Document Object Model

## Acceso a los atributos de un elemento (III)

```
<style type="text/css">
    .recuadro {padding: 0.5em;
                background-color:#EBEBEB;
                border: #d6d6d6 solid 1px;}
</style>

//Sobre el párrafo de la diapositiva anterior
//Cambia el enlace
document.body.getElementsByTagName("a")[0].href =
"http://www.upsam.com";

function enmarcarPárrafo() {
    var nodo = document.getElementById("primerparrafo");
    nodo.className="recuadro";
}
```

# Document Object Model

## Crear y eliminar atributos

- ❑ El método `setAttribute` permite crear un nuevo nodo de atributo en un elemento DOM.

`nodo.setAttribute(nombreAtributo, valorAtributo)`

- Crea o establece el valor de un atributo del elemento `nodo`.
  - ✓ Si `nombreAtributo` ya existe modifica su valor.
  - ✓ Si `nombreAtributo` no existe, crea un nuevo atributo.
- No devuelve nada.

- ❑ El método `removeAttribute` elimina un atributo de un elemento.

`nodo.removeAttribute(nombreAtributo)`

- Elimina `nombreAtributo` de `nodo`.
- Si el atributo no existe genera una excepción.
  - ✓ Se puede utilizar el método `nodo.hasAttribute(nombreAtributo)` para determinar si un elemento tiene un atributo.

- ❑ El método `getAttribute` permite obtener el valor de un atributo.

`nodo.getAttribute(nombreAtributo)`

- Devuelve una cadena con el valor del atributo.
- Si el atributo no existe, devuelve una cadena nula o el valor nulo.



# Document Object Model

## Acceder a los estilos

---

- La propiedad `style` de los elementos permite acceder a los distintos estilos **en línea** de un nodo.
  - Devuelve un objeto de tipo `style` que representa el conjunto de las propiedades de estilos en línea establecidas para el elemento mediante el atributo `html style`.
    - ✓ Como los estilos en línea tienen mayor prioridad que el resto dentro de CSS, permite cambiar el estilo de los elementos.

# Document Object Model

## Acceder a los estilos (II)

- ❑ El objeto `style` devuelto permite acceder a todos los atributos de estilo definidos en CSS.

*nodo.style.nombreAtributoCSS*

- El nombre de los atributos varía ligeramente respecto a CSS.
  - ✓ En los atributos CSS con guiones, los guiones desaparecen.
  - ✓ Cuando hay palabras compuestas, la primera va en minúsculas y las siguientes en mayúsculas.
    - Por ejemplo, la propiedad `color` de CSS llamaría igual que la propiedad `color` del objeto `style` de DOM, la propiedad `background-color` de CSS se llamaría `backgroundColor` en DOM o la propiedad `border-top-color` de CSS se llamaría `borderTopColor` en DOM.
  - ✓ La especificación de DOM del W3C da una correspondencia de todas las propiedades CSS con DOM en [www.w3.org/TR/DOM-Level-2-Style/css.html#CSS-CSS2Properties](http://www.w3.org/TR/DOM-Level-2-Style/css.html#CSS-CSS2Properties).

# Document Object Model

## Modificar los estilos

- ❑ El objeto `style` es de sólo lectura por lo que no se puede modificar directamente.
  - Para modificar un estilo en línea hay que utilizar las propiedades del objeto `style` (los nombre de los atributos del estilo) que son de lectura escritura.

```
var nodo=document.getElementById("ultimoParrafo");
nodo.style.color = "blue"
```

- ❑ Si se desea cambiar varios atributos de estilo...
  - Crear una regla para una clase con todos los atributos que se desea modificar.
  - Cambiar el atributo `class` del elemento mediante `nodo.className`.

```
<style type="text/css">
  .recuadro {padding: 0.5em; background-color:#EBEBEB; border: #d6d6d6
solid 1px;}
</style>
function enmarcarPárrafo(){
  var nodo = document.getElementById("primerparrafo");
  if(nodo.className=="normal"){
    nodo.className="recuadro"; }
  else{
    nodo.className="normal"; }
}
```

# Browser Object Model

## Introducción

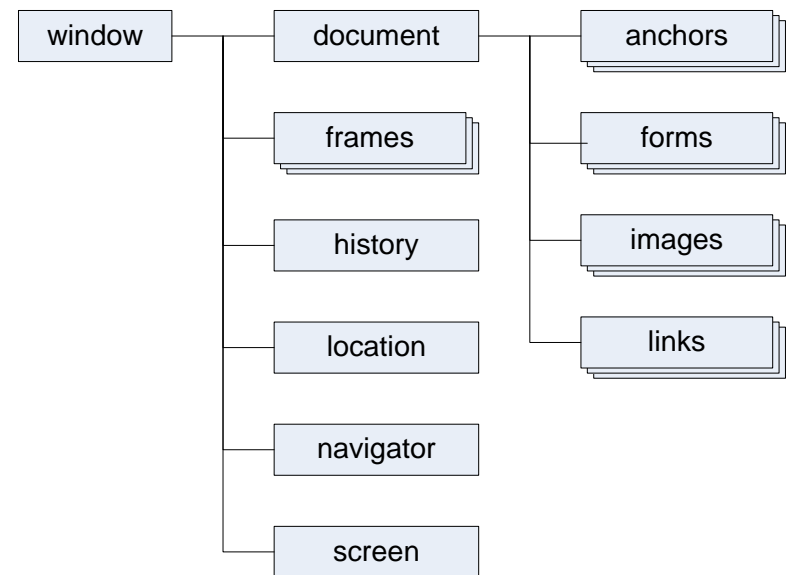
---

- ❑ El Browser Object Model (BOM) permite representar un documento html desde el punto de vista del navegador.
  - Se encuentra en proceso de estandarización por el W3C.
  - Hay que tener cuidado con la compatibilidad entre navegadores.
- ❑ Permite interactuar con las ventanas del navegador mediante JavaScript.
  - Crear, mover, redimensionar y cerrar ventanas de navegador.
  - Obtener información sobre el propio navegador.
  - Propiedades de la página actual y de la pantalla del usuario.
  - Obtener colecciones de elementos (formularios, imágenes, frames, etc.).
  - ...

# Browser Object Model

## Introducción (II)

- ❑ Aunque no está estandarizado, la mayoría de los navegadores admiten el objeto `window`, `history`, `location`, `navigator` y `screen`.
- ❑ El objeto `document` se correspondería al objeto `document` de DOM.
  - El objeto `document` contiene las colecciones de objetos `anchors`, `forms`, `images` y `links`.
- ❑ En [www.w3schools.com/jsref/obj\\_window.asp](http://www.w3schools.com/jsref/obj_window.asp) está disponible una lista de las propiedades y métodos de BOM y los navegadores que las admiten.



Jerarquía de objetos BOM

# Browser Object Model

## Objeto Window

---

- ❑ Representa la ventana donde está cargado el documento.
  - La propiedad `document` hace referencia al objeto `document` de DOM.
- ❑ Los métodos `alert` y `prompt` ya utilizados son métodos del objeto `window`.
- ❑ La colección `frames` hace referencia a los marcos que pudiera tener la página.
  - Devuelve una colección de objetos `frame`.
  - Se accede a cada uno de los elementos mediante el índice: `window.frames[índice]`.

# Browser Object Model

## Objeto Window (II)

### □ Temporizadores.

`setTimeout(código_JavaScript, milisegundos)`

- Retarda la ejecución del `código_JavaScript` el tiempo indicado por `milisegundos`.
- Devuelve un identificador que representa el retardo.
  - ✓ Se puede cancelar la ejecución mediante el método `clearTimeout(identificador)`.
    - Identificador es el valor devuelto por `setTimeout`.

`setInterval(código_JavaScript, milisegundos)`

- Ejecuta `código_JavaScript` cada vez `milisegundos` milisegundo.
- Al igual que `setTimeout`, devuelve un identificador que se puede utilizar para cancelar el retardo mediante `clearInterval(identificador)`.

# Browser Object Model

## Trabajar con ventanas

### □ Abrir ventanas.

- El método `window.open` abre una nueva ventana.

`window.open(URL, nombre [, características])`

- ✓ Devuelve una referencia a la ventana abierta.
- ✓ *URL* es una cadena con el url de la ventana.
- ✓ *nombre* es una cadena que especifica el nombre de la ventana. Puede tomar alguno de los siguientes valores:
  - `_blank`. Valor por omisión. El url se carga en una nueva ventana.
  - `_self`. El url reemplaza la página actual.
  - `_parent`, `_top` o *nombre* se utiliza para trabajar con marcos o con el atributo `target` de los enlaces.
- ✓ *características* es una cadena que indica algunas características especiales de la ventana (tamaño, si tiene barra de título, barra de estado o barra de desplazamiento, etc.).
  - En [developer.mozilla.org/en/DOM/window.open](http://developer.mozilla.org/en/DOM/window.open) se pueden encontrar algunos de las características que se pueden utilizar en distintos navegadores.

```
var upsam = window.open("http://www.upsam.com", "_blank");
```



# Browser Object Model

## Trabajar con ventanas (II)

### ❑ Cerrar ventanas.

- El método `close` permite cerrar la ventana actual o alguna de las ventanas abiertas por el método `open`.

```
window.close()
```

- ✓ Cierra la ventana actual.

```
ventana.close()
```

- ✓ Cierra la ventana referenciada.

- *ventana* debe ser una ventana abierta por el script.

```
window.setTimeout("upsam.close()",10000); //Funciona bien con Chrome
```

### ❑ Controlar el foco.

- El método `ventana.focus()` hace que *ventana* tome el foco actual.
- El método `ventana.blur()` hace que *ventana* pierda el foco.
- *ventana* puede ser el objeto `window` para hacer referencia a la propia ventana o una referencia a una ventana abierta con `open`.

# Browser Object Model

## Otros objetos

---

### ❑ Objeto `window.history`.

- Hace referencia al historial de navegación de la ventana.

- Métodos:

`forward()`

- ✓ Va a la página siguiente del historial de navegación.

`back()`

- ✓ Va a la página anterior del historial de navegación.

`go(índice)`

- ✓ Se mueve a la página indicada por *índice*.

- Si *índice* es positivo se mueve hacia delante y si es negativo hacia atrás.

# Browser Object Model

## Otros objetos (II)

### ❑ Objeto `window.location`.

- Obtiene información del url de la ventana actual.
- Propiedades:
  - ✓ `hash`. Devuelve lo que hay después del símbolo # incluido.
  - ✓ `host`. Devuelve el nombre del servidor entre corchetes y el puerto separado por dos puntos.
  - ✓ `hostname`. Devuelve el nombre del servidor.
  - ✓ `href`. Devuelve el url completo.
  - ✓ `pathname`. Devuelve directorio del recurso.
  - ✓ `port`. Devuelve el puerto.
  - ✓ `protocol`. Devuelve el protocolo.
  - ✓ `search`. Devuelve lo que hay después del símbolo ? Incluido.
- Métodos:
  - ✓ `assign(url)`. Carga el *url* indicado en la página actual.
  - ✓ `reload(forzar)`. Refresca la página actual. Si *forzar* es `true` recarga de nuevo el recurso, si no lo toma de la caché.
  - ✓ `replace(url)`. Carga el *url* indicado en la página actual, sin incluirlo en el historial de navegación.

# Browser Object Model

## Otros objetos (III)

- ❑ Los objetos `window.history` y `window.location` utilizan propiedades y métodos más o menos estándar.
- ❑ Las propiedades de los objetos `window.screen` y `window.navigator` están mucho menos estandarizadas.
- ❑ Con este script se pueden ver las propiedades de cada navegador (hay que ejecutarlo en el navegador a examinar).

```
<script type="text/javascript">
  //
  document.write("&lt;h1&gt;Propiedades del objeto screen&lt;/h1&gt;");
  for (propiedad in screen){
    document.write(propiedad + "=" + screen[propiedad]+"&lt;br&gt;");
  }

  document.write("&lt;h1&gt;Propiedades del objeto navigator&lt;/h1&gt;");
  for (propiedad in navigator){
    document.write(propiedad + "=" + navigator[propiedad]+"&lt;br&gt;");
  }
  //]]&gt;
&lt;/script&gt;</pre></div><div data-bbox="3 947 467 991" data-label="Page-Footer"><p>Universidad Pontificia de Salamanca. Escuela Superior de Ingeniería y Arquitectura<br/>(CC) Luis Rodríguez Baena, 2013</p></div><div data-bbox="972 959 993 980" data-label="Page-Footer"><p>76</p></div>
```

# Browser Object Model

## Otros objetos (IV)

---

### □ Objeto `screen`.

- Devuelve las características de la pantalla.
- `screen` no es objeto estandarizado.
  - ✓ Aquí se van a comentar algunas de las propiedades presentes en las últimas versiones de Firefox, Internet Explorer y Chrome.
- `screen.availHeight`. Devuelve la altura de la pantalla en pixels, descontando lo elementos ocupados por el sistema operativo, como la barra de tareas.
- `screen.availWidth`. Devuelve la anchura en pixels del navegador.
- `screen.colorDepth`. Devuelve la profundidad del color.
- `screen.height`. Devuelve la altura en pixels del navegador.
- `screen.width`. Devuelve la anchura en pixels del navegador.

# Browser Object Model

## Otros objetos (V)

---

### ❑ Objeto `navigator`.

- Devuelve las características del navegador.
- Algunas propiedades comunes a Firefox, Internet Explorer y Chrome.
  - ✓ `navigator.appCodeName`. Devuelve el nombre en código del navegador.
    - En los tres navegadores devuelve "Mozilla".
  - ✓ `navigator.appName`. Devuelve el nombre oficial del navegador.
    - En Internet Explorer devuelve "Microsoft Internet Explorer", en los otros dos "Netscape".
  - ✓ `navigator.appVersion`. Devuelve la versión del navegador y de algunos de los elementos que lo forman (por ejemplo el CLR en IE).
    - La cadena varía mucho de un navegador a otro.

# Browser Object Model

## Otros objetos (VI)

### □ Objeto `navigator` (*continuación*).

- ✓ `navigator.cookieEnabled`. Devuelve un valor lógico indicando si están admitidas las cookies.
- ✓ `navigator.language`. Devuelve una cadena con el idioma del navegador.
  - En Firefox devuelve una cadena formada por el idioma y el dialecto (por ejemplo "es-ES" o "es-LA"), en Chrome sólo el idioma ("es").
  - En Internet Explorer, la propiedad es `navigator.systemLanguage` y devuelve sólo el idioma.
- ✓ `navigator.mimeTypes`. Devuelve un array con la lista de tipos mime instalados en el navegador.
- ✓ `navigator.plugins`. Devuelve un array con la lista de plugins instalados en el navegador.
  - Aunque Internet Explorer la tiene, no devuelve nada.
- ✓ `navigator.userAgent`. Devuelve la cadena que el cliente envía al servidor.
  - Similar a `navigator.appVersion`, puede variar mucho de un navegador a otro.

# Browser Object Model

## Otros objetos (VII)

### ❑ Objeto `navigator` (*continuación*).

- Detectar el navegador.

- ✓ Mediante la propiedad `navigator.appName`, sería posible detectar si estamos con Internet Explorer...

```
if(navigator.appName=="Microsoft Internet Explorer"){
    document.write("Internet Explorer <br>");}
else if(navigator.appName=="Opera"){
    document.write("Opera <br>");}
else{
    document.write("Puede que se trate de Firefox,
    Chrome, Safari... <br>");
}
```



# Browser Object Model

## Otros objetos (VIII)

### ❑ Detectar el navegador.

- Es difícil saber con exactitud el navegador.
  - ✓ Es mejor utilizar los estándares del W3C.
  - ✓ En caso de que sea necesario utilizar alguna propiedad o método no estándar, se puede preguntar por ella.

```
//Esta propiedad sólo la tiene FireFox y devuelve el
//sistema operativo
if(navigator.oscpu) {
    document.write("Este navegador tiene la propiedad
navigator.oscpu <br>");
}
else {
    document.write("Este navegador no tiene la propiedad
navigator.oscpu <br>");
}
```

# Browser Object Model

## Otros objetos (VII)

- ❑ Ejemplo: función JavaScript que devuelve el idioma del navegador.

```
function idiomaUsuario(){
    //La propiedad userLanguage la tiene Internet Explorer y Opera
    var idioma;
    if(navigator.userLanguage){
        idioma = navigator.userLanguage;}
    else{
        idioma = navigator.language;
    }
    //En ambos casos puede devolver una cadena formada por lenguaje y país
    //o sólo el lenguaje.
    //Por eso es mejor coger sólo los dos primeros caracteres.
    return(idioma.substring(0,2));
}
```

# Gestión de eventos

## Asociar controladores de eventos

- ❑ La ejecución de scripts se asociará normalmente a la ejecución de un evento.
  - Se pueden asociar mediante los eventos intrínsecos de html o mediante técnicas de script.
- ❑ La mayoría de los elementos html tienen eventos intrínsecos definidos en las especificaciones:
  - La lista de eventos y elementos a los que afectan se pueden encontrar en [www.sidar.org/recur/desdi/traduc/es/html401-es/interact/scripts.html#h-18.2.3](http://www.sidar.org/recur/desdi/traduc/es/html401-es/interact/scripts.html#h-18.2.3).
- ❑ Se podrá asociar código JavaScript a un evento de un elemento determinado:

```
<elemento nombreEvento = "códigoJavaScript">
```

  - El `nombreEvento` aparece en el elemento en forma de atributo.
  - El `códigoJavaScript` será el manejador de eventos (event handler).
    - ✓ Puede estar formado por una o más instrucciones JavaScript.
    - ✓ Lo normal es que se trate de una llamada a una función JavaScript contenida en el elemento `head` de la página o en un archivo externo.

# Gestión de eventos

## Asociar controladores de eventos (II)

---

- ❑ Ejemplo: Al pasar el ratón sobre el párrafo, aparece con fondo negro sobre blanco. Cuando sale se vuelve al modo normal.
  - El uso de `this` en el código evita tener que buscar la referencia al párrafo en las instrucciones JavaScript.
    - ✓ `this` pasa la referencia al elemento actual, en este caso, el propio párrafo.
    - ✓ Si no se utilizara habría que haber puesto `document.getElementById("otroparrafo").style.color='white'...`

# Gestión de eventos

## Asociar controladores de eventos (III)

```
<p id="otroparrafo"
onmouseover="this.style.color='white';this.style.backgroundColor='black';"
onmouseout="this.style.color='black';this.style.backgroundColor='white';">
Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus.
Vivamus elementum semper nisi. </p>
```

```
function videoInverso(elemento) {
    elemento.style.color = "white";
    elemento.style.backgroundColor = "black";
}
function videoNormal(elemento) {
    elemento.style.color = "black";
    elemento.style.backgroundColor = "white";
}
```

```
<p onmouseover="videoInverso(this)" onmouseout="videoNormal(this)">Nullam
dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus.
Vivamus elementum semper nisi. </p>
```

# Gestión de eventos

## Asociar controladores de eventos (IV)

- Ejemplo: al hacer clic en el párrafo aparece el elemento "párrafo oculto".

```
<style type="text/css">
    .visible {border: black solid 1px;
              background-color: yellow;
              display:block;}
    .oculto  {display:none;}
</style>

function mostraryocultar(){
    var elemento=document.getElementById("parrafooculto");
    if(elemento.className=="visible"){
        elemento.className="oculto";    }
    else{
        elemento.className="visible";    }
}

<p onclick="mostraryocultar()">dolor sit <strong>amet</strong>, consecteturadipiscing
elit.</p>
<div id="parrafooculto" class="oculto">Donec pede justo, fringilla vel, ...</div>
```

# Gestión de eventos

## Asociar controladores de eventos (V)

- ❑ Asociar *event handlers* mediante script.
  - Los eventos se representan como propiedades de un elemento, por lo que podemos asignar su valor mediante instrucciones JavaScript.
    - ✓ `elemento.nombreEvento = referenciaAFuncion`
    - ✓ `referenciaAFuncion` sería el nombre de la función **sin los paréntesis**.
      - Si aparecen los paréntesis sería una llamada a función.
- ❑ Esta técnica permite “limpiar” el código html, separando la gestión de eventos de la estructura y contenido de la página.
- ❑ Para realizarlo es necesario...
  - Asignar un `id` a cada elemento al que se quiera asociar un evento.
    - ✓ Esto es así, porque hay que localizar el elemento.
  - Crear la función que controle el evento.
  - Asociar al atributo de evento del elemento la referencia a la función.
    - ✓ Eso se debe hacer después de haber cargado todos los elementos de la página, generalmente en el evento `window.onload`.

# Gestión de eventos

## Asociar controladores de eventos (VI)

- ❑ Ejemplo: resaltar un párrafo al pasar el ratón sobre él.
  - Las funciones reciben el argumento event, que puede guardar información sobre el evento como la tecla que se ha pulsado (keyCode) o la posición horizontal del cursor (clientX).

```
//Función anónima que se ejecuta al terminar la carga del documento.
window.onload = function(){
    document.getElementById("ejemplo").onmouseover = resaltar;
    document.getElementById("ejemplo").onmouseout = normal;
}

//this hace referencia al elemento que generó el evento
function resaltar(event){
    this.style.color = "white";
    this.style.backgroundColor = "black";
}

function normal(event){
    this.style.color = "black";
    this.style.backgroundColor = "grey";
}

<p id="ejemplo" style="background-color:grey">Aenean vulputate eleifend tellus. Aenean
leo ligula, porttitor eu, consequatvitae, eleifend ac, enim. </p>
```



# Gestión de eventos

## Asociar controladores de eventos (VII)

- ❑ Ejemplo: En un formulario, resaltar los campos cuando están activos.
  - Podemos asignar eventos a la vez a distintos elementos, recorriéndolos mediante bucles.

```
window.onload = function(){
    var inputs = document.getElementsByTagName("input");
    for(var i=0; i < inputs.length; i++){
        inputs[i].onfocus = resaltar;
        inputs[i].onblur = normal;    }
}
function resaltar(){
    this.style.backgroundColor = "yellow";
    this.style.border = "black solid 1px";}
function normal(){
    this.style.backgroundColor = "transparent";
    this.style.border = "black solid 1px";}

<form id="frmPrueba" method="get" action="http://miservidor.com/bin/aplicacion.pl">
<fieldset><legend>Datos personales</legend>
<label for="apellidosynombre">Apellidos y nombre:<br />
<input name="nombre" type="text" id="apellidosynombre" size="60" /><br />
</label>
Direcci&oacute;n:<br /><input name="direccion" type="text" size="60" /><br />
Poblaci&oacute;n: <br /><input name="poblacion" type="text" size="60" /><br />
C&oacute;digo postal:<br /><input name="cp" type="text" size="20" maxlength="5"/>
</fieldset>
</form>
```

# Gestión de eventos

## Asociar controladores de eventos (VII)

□ Ejemplo: hacer un rollover de una imagen.

```
//Función anónima que se ejecuta al terminar la carga del documento.
window.onload = function(){
    document.getElementById("botónImagen").onmouseover=cambiarImagen;
    document.getElementById("botónImagen").onmouseout=restaurarImagen;
}

function cambiarImagen(event){
    this.src ="rollover.jpg";
}

function restaurarImagen(event){
    this.src ="boton.jpg";
}


```

# Gestión de eventos

## Eventos del ratón

- ❑ Los eventos de ratón son estándar del W3C.
- ❑ `click` (se corresponde con la propiedad html `onclick`).
  - Cuando se pulsa sobre un elemento se producen los siguientes eventos en una determinada secuencia: (1) `mousedown`, (2) `mouseup`, (3) `click`.
    - ✓ `click` se detecta si se ha producido el evento `mousedown` y `mouseup` en el mismo elemento.
  - Información adicional proporcionada a través del objeto `event`:
    - ✓ `screenX`, `screenY`, `clientX`, `clientY`, `altKey`, `ctrlKey`, `shiftKey`, `button`.
- ❑ `mousedown` (se corresponde con la propiedad html `onmousedown`).
  - Información adicional proporcionada a través del objeto `event`:
    - ✓ `screenX`, `screenY`, `clientX`, `clientY`, `altKey`, `ctrlKey`, `shiftKey`, `button`.
- ❑ `mouseup` (se corresponde con la propiedad html `onmouseup`).
  - Información adicional proporcionada a través del objeto `event`:
    - ✓ `screenX`, `screenY`, `clientX`, `clientY`, `altKey`, `ctrlKey`, `shiftKey`, `button`.

# Gestión de eventos

## Eventos del ratón (II)

- ❑ `mouseover` (se corresponde con la propiedad `html.onmouseover`).
  - Información adicional proporcionada a través del objeto `event`:
    - ✓ `screenX`, `screenY`, `clientX`, `clientY`, `altKey`, `ctrlKey`, `shiftKey`, `button`, `relatedTarget`.
- ❑ `mousemove` (se corresponde con la propiedad `html.onmousemove`).
  - Información adicional proporcionada a través del objeto `event`:
    - ✓ `screenX`, `screenY`, `clientX`, `clientY`, `altKey`, `ctrlKey`, `shiftKey`, `button`.
- ❑ `mouseout` (se corresponde con la propiedad `html.onmouseout`).
  - Información adicional proporcionada a través del objeto `event`:
    - ✓ `screenX`, `screenY`, `clientX`, `clientY`, `altKey`, `ctrlKey`, `shiftKey`, `button`, `relatedTarget`.
      - `relatedTarget` hace referencia al control dónde sale el ratón en `mouseover` y a dónde va en `mouseout`.

# Gestión de eventos

## Eventos del ratón (III)

### ❑ Información adicional...

- Se accede a ella mediante `event.propiedad` (en FireFox) o `window.event.propiedad` en IE.

Propiedad del objeto Event	Descripción
<code>altKey</code> , <code>ctrlKey</code> , <code>shiftKey</code>	Valor lógico que indica si se pulsó la tecla alt, ctrl o shift
<code>button</code>	Indica el botón que se ha pulsado; 0 para el botón izquierdo, 1 para el derecho y 2 para el central. En el modelo de eventos de Internet Explorer los valores son 1, 2 y 4.
<code>clientX</code> , <code>clientY</code>	Coordenadas en pixel de dónde se ha producido el evento dentro del área cliente.
<code>relatedTarget</code>	En <code>mouseover</code> indica el nodo que ha dejado el ratón, en <code>mouseout</code> indica el nodo al que ha salido. En Internet Explorer la información se guarda en las propiedades <code>toElement</code> y <code>fromElement</code>
<code>screenX</code> , <code>screenY</code>	Coordenadas en pixel dónde se ha producido el evento dentro de la pantalla.

# Gestión de eventos

## Eventos del teclado

---

- ❑ La especificación DOM actual no recoge todavía los eventos de teclado.
  - Son bastante incompatibles entre navegadores.
  - Se producen en el orden `keydown`, `keypress` y `keyup` respectivamente.
    - ✓ `keydown` y `keyup` permiten detectar teclas especiales (`alt`, `ctrl`, retroceso, etc.).
- ❑ Información adicional sobre el evento.
  - `altKey`, `ctrlKey`, `shiftKey` devuelven un valor lógico que indica si se ha pulsado la tecla `alt`, `ctrl` o `shift`.

# Gestión de eventos

## Eventos del teclado (II)

### ❑ Detección de la tecla pulsada.

- En Firefox.
  - ✓ En el evento `keypress`, `charCode` y `which` devuelven el código de la tecla pulsada.
  - ✓ En los eventos `keyup` y `keydown`, `keyCode` y `which` devuelven el código de la **mayúscula** de la tecla pulsada (si se ha pulsado "a" devuelve 65).
- En Internet Explorer (hasta la versión 8).
  - ✓ En el evento `keypress`, `keyCode` devuelve el código de la tecla pulsada.
  - ✓ En los eventos `keyup` y `keydown`, `keyCode` devuelve el código de la mayúscula de la tecla pulsada.
- Distinguir entre IE y FireFox.
  - ✓ En Firefox, se manda el evento en forma de un objeto event a la función que lanza el evento.
    - Si el argumento es nulo, es que el script se está ejecutando en IE.

```
function eventoKeyPress(event) {
    //Si event no es nulo (estamos en Firefox), la condición del if es verdadera
    if(event)
        alert("keypress \ncharCode:" + event.charCode +
            "\nkeyCode: " + event.keyCode + "\nwhich: " + event.which);
    else
        //Si event es falso estamos en FireFox
        alert("keypress \nkeyCode: " + window.event.keyCode)
}
```

# Gestión de eventos

## Eventos html

---

- Los eventos html sí son estándar y están recogidos en la especificación del W3C.
- `load`.
  - Se produce cuando se ha acabado de cargar el documento.
- `unload`.
  - Se produce cuando se elimina el documento de una ventana.
  - Aplicable a los elementos `body` y `frameset`.
- `abort`.
  - Se produce cuando se cancela la carga de un documento.
- `error`.
  - Se produce cuando no se carga correctamente un documento, objeto o marco.
  - Aplicable a los elementos `body`, `object` y `frameset`.
- `select`.
  - Se produce cuando el usuario selecciona texto.
  - Aplicable a los elementos `input` y `textarea`.
- `change`.
  - Se produce cuando un elemento pierde el foco y se ha modificado su contenido.
  - Aplicable a los elementos `input`, `select` y `textarea`.



# Gestión de eventos

## Eventos html (II)

---

- ❑ `submit`.
  - Se produce cuando se envía un formulario.
  - Sólo se aplica el elemento `form`.
- ❑ `reset`.
  - Se produce cuando inicializa un formulario.
  - Sólo se aplica el elemento `form`.
- ❑ `focus`.
  - Se produce cuando un elemento entra en foco.
  - Aplicable a los elementos `label`, `input`, `select`, `textarea`, y `button`.
- ❑ `blur`.
  - Se produce cuando un elemento pierde el foco.
  - Aplicable a los elementos `label`, `input`, `select`, `textarea`, y `button`.
- ❑ `resize`.
  - Se produce cuando el documento se redimensiona.
- ❑ `scroll`.
  - Se produce cuando se hace un desplazamiento del documento.

# Formularios

## Acceso a los formularios

- ❑ El acceso a los formularios se puede realizar de la misma forma que en el resto de elementos:
  - `getElementById()` o `getElementsByTagName()`.
- ❑ También se puede realizar a través del array `forms` del nodo `document`.
  - `document.forms` devolvería un array con todos los formularios del documento.
  - Para acceder a cada formulario, se puede utilizar:
    - `document.forms[índice]`
    - `document.forms[nombre]`
      - Esta última forma es más recomendable.
      - `nombre` sería una cadena con el valor de la propiedad `id` o `name` del formulario.
  - En general, es más recomendable acceder al formulario a través del nombre (`getElementById`) en lugar de hacerlo a través del array `forms`.

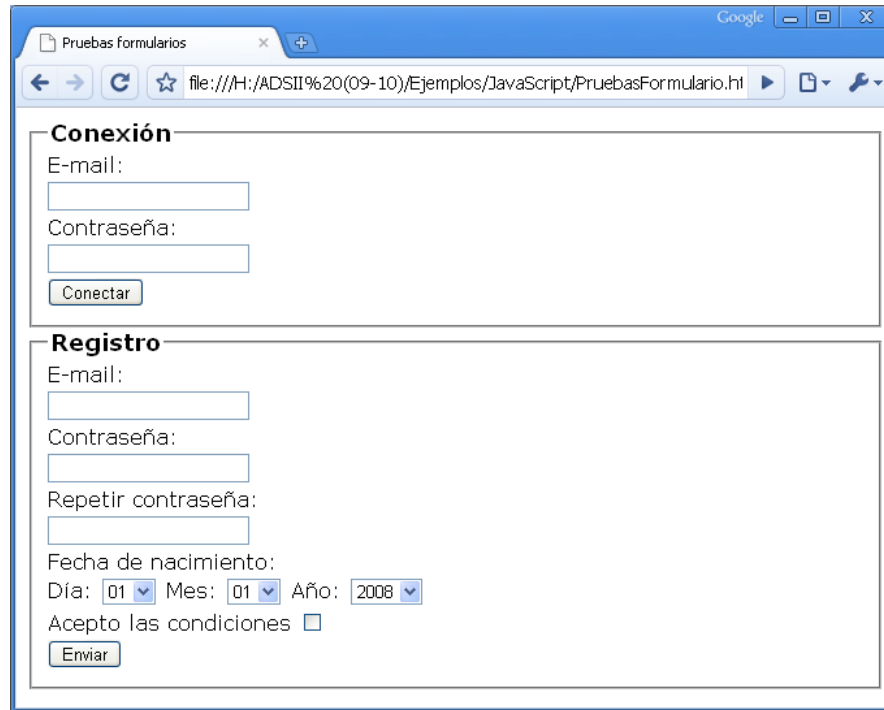
# Formularios

## Acceso a los formularios (II)

```
<form action="#" method="get" id="frmConectar"><fieldset><legend>Conexión</legend>
  <label for="email-registrado" class="descripcion">E-mail:</label>
  <input type="text" class="textbox" id="email-registrado" />
  <label for="password-registrada" class="descripcion">Contraseña:</label>
  <input type="password" class="textbox" id="password-registrada" />
  <input type="submit" class="boton" id="conectar" value="Conectar" />
</fieldset>
</form>
<form action="#" method="get" id="frmRegistro"><fieldset><legend>Registro</legend>
  <label for="email" class="descripcion">E-mail:</label>
  <input type="text" class="textbox" id="email" />
  <label for="password" class="descripcion">Contraseña:</label>
  <input type="password" class="textbox" id="password" />
  <label for="repetirpassword" class="descripcion">Repetir contraseña:</label>
  <input type="password" class="textbox" id="repetirpassword" />
  <label for="dia" class="descripcion">Fecha de nacimiento: </label>
  <label>Día:</label>
  <select id="dia">
    <option value="01">01</option>...
  </select>
  <label>Mes:</label>
  <select id="mes">
    <option value="01">01</option>...
  </select>
  <label>Año:</label>
  <select id="anno">
    <option value="1900">1900</option>...
  </select>
  <label for="acepto" class="descripcion">Acepto las condiciones
    <input type="checkbox" id="acepto" /></label>
  <input class="boton" type="submit" id="enviar" value="Enviar" />
</fieldset>
</form>
```

# Formularios

## Acceso a los formularios (III)



The screenshot shows a web browser window with two forms. The first form, titled 'Conexión', has fields for 'E-mail:' and 'Contraseña:', and a 'Conectar' button. The second form, titled 'Registro', has fields for 'E-mail:', 'Contraseña:', 'Repetir contraseña:', 'Fecha de nacimiento:' (with sub-fields for 'Día:', 'Mes:', and 'Año:'), and an 'Acepto las condiciones' checkbox, followed by an 'Enviar' button.

### □ Dado el formulario anterior...

`document.forms[0].id`

- Devolvería `frmConectar`

`document.forms['frmRegistro'].id`

- Devolvería `frmRegistro`

# Formularios

## Acceso a los elementos de un formulario

- ❑ Cada formulario contiene un array `elements` que referencia a cada uno de los elementos del formulario.
  - Se consideran elementos del formulario los elementos `html input`, `textarea`, `button` y `select`.
  - Al igual que al array `forms`, se puede acceder a los elementos por el índice o por el identificador del elemento.

```
document.forms['frmRegistro'].elements[1].id
```

```
document.forms['frmRegistro'].elements['password'].type
```

- ✓ En Internet Explorer, Firefox y Opera, el elemento 0 del array hace referencia al propio formulario; en Chrome y Safari el elemento 0 sería el primer elemento de la interfaz.
  - Es mejor referenciarlos mediante el identificador.

- ❑ Los elementos de un formulario presentan las propiedades:
  - `type`. Devuelve el atributo `type` del elemento `input`, "textarea", "option-one" u "option-multiple" para el elemento `select`, o "submit", "reset" o "button" para el elemento `button`.
  - `form`. Hace referencia al formulario dónde está el elemento.
  - `value`. Devuelve o establece el valor del elemento.

# Formularios

## Acceso al contenido de los campos

- ❑ Al enviar un formulario se envía una cadena con el URL, una interrogación y parejas formadas por *nombreControl=valor*.
  - Los espacios en blanco se sustituyen por el signo +
  - Los caracteres no estándar aparecen codificados con un signo % y un código.
- ❑ Cuadros de texto.
  - Para los elementos `input` de tipo `text`, como a los de tipo `password` o a los elementos `textarea` la propiedad `value` permite obtener o establecer el valor del cuadro de texto. campos.
    - ✓ Por ejemplo, `document.getElementById("email").value`, devolvería el valor del campo `email`.
- ❑ Botones de radio.
  - La propiedad `value` se utiliza para indicar el valor que se envía al procesar el formulario.
  - Para detectar si está marcada una acción habría que utilizar la propiedad `checked`.

# Formularios

## Acceso al contenido de los campos (II)

### ❑ Botones de radio.

- Con el siguiente grupo botones de radio, se podría detectar la opción pulsada con...

<input type="radio"/> Hombre
<input checked="" type="radio"/> Mujer

```
<label for="hombre" class="descripcion">
  <input type="radio" name="sexo"
    id="hombre" checked="true" value="hombre"/>
  Hombre
</label>
<label for="mujer" class="descripcion">
  <input type="radio" name="sexo"
    id="mujer" value="mujer" />
  Mujer
</label>
```

```
//Comprobar el estado de las opciones
if(document.getElementById("hombre").checked){
  //Hacer algo con los hombres}
else{
  //Hacer algo con las mujeres
}
```

# Formularios

## Acceso al contenido de los campos (III)

### Casillas de verificación.

- Funcionan de forma similar a los botones de radio.
- A la hora de hacer la comprobación hay que tener en cuenta que no son excluyentes.
  - ✓ Hay que comprobar todas las opciones.

### Listas desplegables.

- Cada elemento `select` guarda sus opciones en el array `options`.
- La propiedad `selectedIndex` guarda el índice del elemento seleccionado.
- Ejemplo: determinar el día seleccionado:

```
var dias = document.getElementById("dia").options;  
var diaSeleccionado = dias[dias.selectedIndex].value;
```



# Formularios

## Métodos de los elementos de un formulario

---

### Control del foco.

- Método `focus()`.
  - ✓ Hace que el elemento del formulario entre en foco.
- Método `blur()`.
  - ✓ Hace que el elemento del formulario pierda el foco.

### Método `click()`.

- Lanza el evento click del elemento.

### Método `select()`.

- En los campos de texto selecciona el texto del elemento.

# Formularios

## Ejemplos

- ❑ Comprobar si un campo tiene una longitud mínima:

```
function longitudMinima(texto,longitud){
    return texto.length >= longitud;
}

...
<label for="campo">Campo:
<input type="text" id="campo"
    onblur=
        "if(longitudMinima(this.value,5))alert(
            'Debe tener 5 caracteres')"/>
</label>
```

# Formularios

## Ejemplos (II)

### ❑ Comprobar si un campo está vacío:

```
function esCampoVacio(texto){
    if(texto != null && texto.length != 0){
        //Comprobar si todos los campos son blancos
        for(var i=0; i < texto.length;i++){
            if(texto.charAt(i) != " "){
                return false;}
        }
    }
    return true;
}
...
<input type="text" id="campo"
onblur="if(esCampoVacio(this.value))alert('Está vacío') />
```

# Formularios

## Ejemplos (III)

### ❑ Comprobar si un campo es numérico:

```
function esNumerico(texto){
  var numero = parseFloat(texto);
  //Para ver si el campo es numérico tiene que ser un número
  //y todos los caracteres válidos
  if(!isNaN(texto) || numero.length == texto.length){
    return true;}
  else{
    return false;
  }
}
```

...

```
<label for="precio">Precio:
<input type="text" id="precio"
  onblur="if(esNumerico(this.value))alert('Es numérico')"/>
</label>
```

# Formularios

## Ejemplos (IV)

---

- ❑ No dejar que se introduzcan valores alfabéticos:
  - Hay que detectar la tecla pulsada.
    - ✓ La referencia al evento en Internet Explorer se hace mediante `window.event`, en Mozilla mediante el argumento `event`.
    - ✓ La tecla pulsada se detecta en Internet Explorer mediante `keyCode`, en Mozilla mediante `charCode`.
    - ✓ Se comprueba si la tecla es un número si está contenida en la cadena "0123456789".
  - En una propiedad de evento, si se devuelve `false`, se anula el evento (es como si no se hubiera producido).
    - ✓ Si la propiedad `onkeypress` devuelve `false`, se anula la pulsación de la tecla.

# Formularios

## Ejemplos (V)

```
function soloNumeros(event){
    var evento = (event || window.event);
    var codigoTecla = (evento.charCode || evento.keyCode);
    var caracter= String.fromCharCode(codigoTecla);
    var nums = "0123456789";

    if(nums.indexOf(caracter) == -1){
        return false;}
    else{
        return true;
    }
}
...
<input type="text" id="campo"
        onkeypress="return soloNumeros(event)"/>
```

# Formularios

## Ejemplos (VI)

### ❑ Comprobar si se trata de un e-mail:

- Mediante expresiones regulares se puede comprobar si un dato tiene un formato determinado.
  - ✓ El método `test`, permite comprobar una expresión regular con un patrón.

```
function esEmailValido(texto){
    //Comprueba que tenga caracteres alfanuméricos, una arroba,
    //caracteres alfanuméricos, un punto y más caracteres alfanuméricos
    if(/\w+@\w+\.\w+/.test(texto)){
        return true;}
    else{
        return false;
    }
}
```

```
...
<label for="email">Correo electrónico:
<input type="text" id="email"
    onblur="if(!esEmailValido(this.value))alert('No es un e-mail')"/>
</label>
```

# Formularios

## Validar formularios

---

- ❑ Una de las funciones más comunes de JavaScript en los formularios sería la validación de los campos en el lado del cliente.
- ❑ Al pulsar el botón de envío (elemento `input` con el atributo `type="submit"`) se produce el evento `submit` del formulario.
  - Se debería responder al evento `submit` con la ejecución de una función de validación que compruebe todos los campos necesarios del formulario.
  - En JavaScript, si la función devuelve `true`, se realiza la instrucción que aparece en la propiedad `action` del formulario.
  - Si la función devuelve `false`, JavaScript no ejecuta dicha instrucción



# Formularios

## Validar formularios (II)

```
<script type="text/javascript">
//
window.onload = function(){
    document.getElementById("miFormulario").onsubmit = "validar";
}

function validar(){
    //Comprobación campo1
    if(condición_error_campo1){
        //Mensaje de error campo1
        return false;}
    //Comprobación campo2
    if(condición_error_campo2){
        //Mensaje de error campo2
        return false;}

    ...
    //Comprobación campoN
    if(condición_error_campoN){
        //Mensaje de error campoN
        return false;}
    //Si no se ha dado ninguna condición de error la función devuelve
    //true para ejecutar la instrucción del atributo action del formulario
    return true;
}
//]]&gt;
&lt;/script&gt;
...
&lt;form action="http://www.miservidor.com/procesar.php" method="get" id="miFormulario"&gt;
...
&lt;/form&gt;</pre></div><div data-bbox="3 947 467 991" data-label="Page-Footer"><p>Universidad Pontificia de Salamanca. Escuela Superior de Ingeniería y Arquitectura<br/>(CC) Luis Rodríguez Baena, 2013</p></div><div data-bbox="967 959 993 980" data-label="Page-Footer"><p>113</p></div>
```

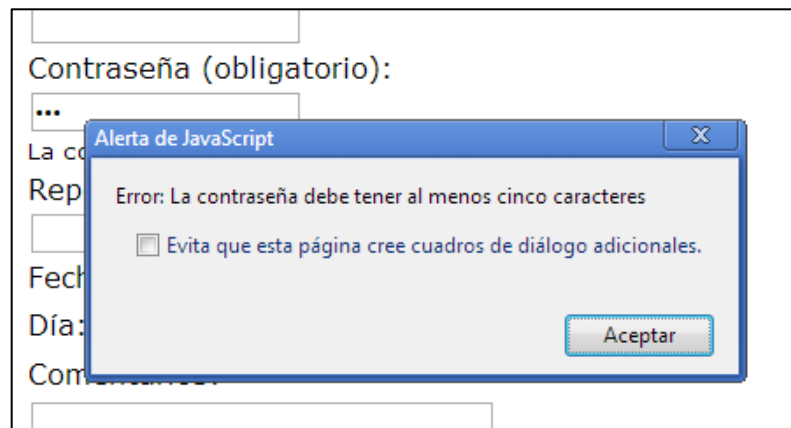
# Formularios

## Validar formularios (III)

### ❑ Mensajes de error.

- En general, existen dos formas de mandar mensajes de error desde el lado del cliente.
  - ✓ Mediante ventanas de alerta.

```
//Comprobar contraseña
var nodo = document.getElementById("password");
if(!longitudMinima(nodo.value,5)){
    alert("Error: La contraseña debe tener al menos cinco caracteres");
    hayError=true;
}
```



# Formularios

## Validar formularios (IV)

### ❑ Mensajes de error (*continuación*).

- Añadiendo elementos de aviso en el árbol DOM.

```
var nodo = document.getElementById("password");
if(!longitudMinima(nodo.value,5)){
    var nodoError = document.createElement("div");
    nodoError.appendChild(document.createTextNode(
        "La contraseña debe tener al menos cinco caracteres"));
    nodoError.className = "error";
    nodoError.id ="errorPassword";
    nodo.parentNode.insertBefore(nodoError,nodo);
    nodo.style.backgroundColor = "#FFFF66";
    hayError=true;
}
```

