Tabla de contenido

¿Qué es el DOM?.

Las siglas DOM significan Document Object Model, o bien, Modelo de objetos del Documento. El DOM es un estándar que nos permitirá incrementar nuestro nivel de interactividad en las páginas programadas en el cliente, considerando el documento como un árbol donde todos sus nodos están relacionados de forma jerárquica.

Durante algunos años las diferencias del DOM entre los distintos navegadores fueron un problema para producir aplicaciones cliente capaces de ser ejecutadas en cualquier navegador comercial (el término empleado para estas aplicaciones es cross-browser).

Por ese motivo el W3C, actualmente responsable del DOM, se ha esforzado en la definición de un estándar o marco común. Desafortunadamente todavía la implementación completa del DOM conforme al estándar puede variar entre un navegador u otro.

En general el DOM se encarga de indicarnos cómo analizar y acceder a cada parte de un documento estructurado, que no será necesariamente una página web sino puede ser también un documento XML.

En definitiva el DOM se comporta como un API (Application Interface) que está compuesto por tres partes principales:

- El núcleo (DOM Core).
- El modelo estándar para documentos XML (DOM-Xml).
- El modelo estándar para documentos HTML (DOM-Html).

En cada una de estas partes se definen un conjunto de objetos (**con sus propiedades y métodos específicos**) para poder acceder a cualquier elemento del documento.

Nosotros, en los temas anteriores ya hemos estado utilizando objetos del DOM-HTML: el objeto select, document, input type=text, window, …, etc. Son objetos del DOM-HTML. Y por otra parte el DOM-XML incluye un conjunto de objetos para acceder y tratar cada uno de los elementos posibles en un documento XML.

Sin embargo el DOM no es sólo una biblioteca de objetos. Nos permitirá, por ejemplo, agregar o eliminar nuevos elementos, aplicarles o eliminarles determinados atributos, incluir nuevos nodos de texto, obtener referencias al elemento padre o a los elementos hijo, crear y destruir nuevos elementos, etc.

¿Existe un único DOM?.

Ya hemos indicado que el responsable de la estandarización es el W3C. Los navegadores de la familia WebKit, como Chrome y Safari se han esforzado bastante por ajustarse a este estándar. El navegador Ópera incorpora también nuevas funcionalidades del W3C en cada versión. Los navegadores de la familia de Mozilla,

como es el caso de FireFox, están aún implementando el nivel 3. Y por último Internet Explorer es sin duda el que más se aleja del DOM, llegando incluso a crear su propio DOM no estandarizado, aunque para las últimas versiones se anuncia un ajuste paulatino al estándar del W3C.

Los primeros navegadores que implementaron un DOM lo hacían de forma individual y no estandarizada. Estas primeras veriones fueron llamadas DOM de nivel 0. Ejemplos fueron los DOM de Netscape Comunicator 2.0 y de Internet Explorer 3.0, totalmente incompatibles entre sí.

Después de esta guerra de navegadores el W3C intervino publicando su DOM de nivel 1 en 1998. Por primera vez había una recomendación que consideraba todos los elementos presentes en una página web y muchos programadores apoyaron esta propuesta, cansados de tener que programar dos versiones incompatibles de sus páginas.

Sin embargo cuando se publicó este DOM de nivel 1 ya estaban en el mercado Netscape Communicator 4.0 e Internet Explorer 4.0.

El DOM de nivel 1 incluía una referencia de todos los objetos HTML-DOM, pero carecía de estandarización para el modelo de eventos. El DOM de nivel 2 fue publicado a finales del 2000. Incorpora por primera vez tratamiento para las hojas de estilo (todas las propiedades que has utilizado en JavaScript para modificar los estilos fueron incorporadas en este DOM) y también incorpora el tratamiento de eventos.

En el 2004 se publica una recomendación del W3C llamada DOM de nivel 3. Esta versión incorpora mucha de la terminología de XML. Se agregaron muchos métodos para el trabajo con documentos XML y el uso de los namespaces. Se habla por primera vez de validación y se permite el acceso a DTDs (Documentos de Definición de Tipos).

El árbol del documento

El objeto document, que ya has utilizado muchas veces, actúa como la raíz del árbol (representa todo el fichero HTML). De esta raíz cuelgan dos hijos: el primero es un objeto **documentType** que será el que represente a la DTD empleada. En nuestro caso este objeto almacenará la información del DOCTYPE. El segundo hijo es un elemento que representa la etiqueta **<html>…</html>**.

El elemento HTML tiene también dos hijos: el elemento **HEAD** y el elemento **BODY**. Y así sucesivamente. Ya tienes experiencia en este tipo de representación en árbol porque es la misma que utilizamos el curso pasado con los documentos XML. Un párrafo como este:

```
<p>Segundo curso</p>
```

En el árbol existe un último nodo de elemento <p>. Y este nodo tiene un hijo, que es un nodo de texto donde se almacena el texto “Segundo curso”. Es decir, los textos se almacenan en nodos hoja del árbol, y estos nodos son de un tipo especial Text.

Por otra *parte los atributos también son nodos independientes y están asociados al elemento que los contiene*. Recuerda que los **atributos NO SON considerados nodos hijo sino por el contrario se consideran nodos asociados a un elemento**.

Este árbol se crea conforme el navegador va realizando una lectura del fichero que incluye el código de marcado (HTML o XML). Cuando el árbol del DOM está cargado

y listo para usarse se dispara el evento **onload** y ya podemos programar acciones que lo manipulen.

Tipo de nodos en el árbol.

Ya hemos avanzado que en el árbol podrá haber diferentes tipos de nodos. En total son doce los tipos de nodos existentes y cada uno de estos tipos tendrá un objeto asociado con propiedades y métodos específicos.

De estos doce tipos algunos son exclusivos para documentos XML y se definen en el DOM-XML. Es el caso de los tipos ProcessingInstruction, CDataSection, Entity, EntityReference, DocumentFragment y Notation. Nosotros no vamos a entrar en este tema en el tratamiento del DOM-XML pero ten en cuenta que accediendo a una guía de referencia de JavaScript para estos objetos podrás manipular documentos XML con facilidad. Pero vamos a centrarnos en el DOM-HTML.

- Objetos **document** y **documentType**: ya hemos hablado de ellos. Representan la raíz y el doctype.
- Objeto **Element**: es, quizás, la parte central del DOM. Un elemento equivale a una etiqueta HTML o XML y a todo su contenido. El nodo Element guarda información de la etiqueta. Es un tipo de nodo especial puesto que puede contener nodos hijo que podrán ser nodos de texto o bien otros elementos. El nodo element también tiene una colección de nodos atributos asociados. Por ejemplo: un nodo element correspondiente a la etiqueta <table>…</table> tendrá como hijos a todos los elementos <tr>…</tr> que contenga la tabla. Éstos a su vez tendrán como hijos a todos los elementos <td>…</td>.
- Objeto **Attr**: este nodo representa a un atributo individual. Los atributos están obligatoriamente asociados a un elemento. Este nodo guarda el nombre del atributo (ejemplo "*href*") y el valor del mismo (ejemplo *www.daw.es*). Los atributos se asocian a los elementos pero no son sus hijos.
- Objeto **Text**: los textos están contenidos su propio nodo, que siempre será hijo de un Element. Al igual que los atributos los nodos de texto tienen un valor asociado, que es la cadena con el texto en cuestión.
- Por último los comentarios se almacenan en nodos especiales dentro de la jerarquía llamados nodos **Comment**.

Por último y antes de terminar este apartado recuerda que la especificación HTML-DOM incluye objetos para casi cualquier etiqueta. Ya hemos utilizando algunos. Buscando una guía de referencia podrás observar los métodos más apropiados. Citemos cuáles son estos objetos:

Document, HTMLElement, Anchor, Area, Base, Body, Button, Event, Form, Frame/IFrame, Frameset, Image, Input Button, Input Checkbox, Input File, Input Hidden, Input Password, Input Radio, Input Reset, Input Submit, Input Text, Link, Meta, Object, Option, Select, Style, Table, TableCell, TableRow, Textarea

Accesos desde la raíz.

En temas anteriores hemos utilizado el objeto document, raíz de nuestro árbol. Este objeto dispone de algunos métodos que nos permiten hacer búsquedas directas desde la raíz del DOM. El más popular es el método **getElementById(id)**.

Habitualmente debíamos esperar a que el DOM fuera cargado y estuviera listo para utilizarse. Esto lo conseguíamos con el evento onload. Una vez que el DOM está cargado y listo para su uso podemos acceder a elementos desde la raíz utilizando los métodos más habituales del objeto document, recordemos

- **getElementByID**(id): localiza un elemento por su atributo id.
- **getElementsByName**(name): este método localiza elementos mediante su atributo name. Como puede haber varios elementos con el mismo name (por ejemplo los input type="radio") se retorna siempre una colección.
- **getElementsByTagName**(tag): este método retorna una colección con las referencias a todos los elementos cuya etiqueta sea el tag descrito.

Estos tres métodos no son exclusivos de document. Cualquier nodo de tipo elemento también dispone de estos métodos. De esta forma podemos tener seleccionado un elemento (por ejemplo un div con `id="mybox"`) y acceder a todos sus párrafos. Observa el ejemplo:

```
…
<div id="mybox">
<p>Un primer párrafo de texto</p>
<p>Un segundo párrafo de texto</p>
<p>Un tercer párrafo de texto</p>
</div>
…
```

```
// código JS
      var elemento = document.getElementById("mybox");
      var coleccion = elemento.getElementsByTagName("p"); // colección con los párrafos
      // tenemos un array con cada elemento <p> pero el texto de cada párrafo será un nodo hijo.
      alert("Último párrafo: " + coleccion[2].childNodes[0].nodeValue);
// alerta con el tercer párrafo
```

Propiedad **childNodes**: retorna una colección con todos los hijos

Actividad 1

Carga los ficheros adjuntos, un html (objetos_dom.html) y un css (objetos_dom.css). La información contenida en los mismos NO se debe modificar, para realizar las prácticas cree un fichero objetos_dom.js.

Se encargará de aplicar correctamente las clases de estilo (utilizando el método className a los tr de la tabla). Para ello deberás acceder a las filas de la tabla utilizando el DOM. Primero obtén una referencia a la tabla mediante getElementById. Utilizando esta referencia podrás acceder a las filas localizando todas las etiquetas <tr>. Obtendrás una colección: recórrela asignando las clases convenientemente con className. La clase cabecera a la primera fila, la clase filaImpar para las filas impares y la clase filaPar para las pares. Observe la imagen.

Los métodos utilizados para los accesos getElementById y getElementsByTagName no son exclusivos del objeto document, se pueden aplicar también a los elementos.

Elecciones

Id	Junta Provincial	Responsable	Cargo	Email	Número de votantes	Votos en blanco
1	Almería	María Mercedes Benz Seguro	Presidente	mmbseg.es	452579	3507
2	Cádiz	Rinder Kate Proud	Presidente	rkp@gmail.es	993327	9194
3	Córdoba	Jesús Manuel Pérez Lindl	presidente	jmperlin@hotmail.es	653926	6946
4	Granada	Jack Adrian Lang	Suplente		74777	5251
5	Huelva	Soledad Aburrida Díaz	Policía	??	398492	3456
6	Jaén	Curly Team Papa	PreSidente	papateam	535847	4226
7	Málaga	Paterquito Otoño Helado	Suplente	pater@.es	1153047	8424
8	Sevilla	Jaqueline Hernández García	Suplente		1529822	13713
9	Mérida	Brenda Trans Delte	policia	btrdan@cse.es	2536	329

Modifica la primer columna para que tenga la clase primeraColumna

Id	Jun
1	Aln
2	Cád
3	Cór
4	Gra

Node: interfaz con aspectos comunes.

En los apartados anteriores vimos que el DOM utiliza objetos como document, documentType, Element, Attr, Text o Comment. Cada uno de estos objetos representa un tipo concreto de nodo en el árbol del DOM y más adelante en este mismo tema los abordaremos.

Pero vamos a comenzar con la interfaz Node, que provee un acceso a bajo nivel sobre el DOM. Todos los nodos del árbol necesitan unas propiedades y métodos comunes. Por ejemplo si nos encontramos en un nodo concreto, sea cual sea, nos puede interesar averiguar cuál es el nodo padre que lo contiene.

Para hacer esto posible se creó la interfaz Node de forma que todos los nodos del árbol, sean del tipo que sean, implementan esta interfaz. Eso sí, dependiendo del tipo de nodo en que estemos tendrá sentido o no utilizar un método concreto. Por ejemplo un nodo de texto no tiene hijos, no tendrá sentido por tanto utilizar la propiedad firstChild, que almacena una referencia al primer hijo.

Las propiedades de la interfaz node

attributes[]: Retorna una colección con los atributos del nodo. Serán objetos de clase Attr.
baseURI: Devuelve la URI base absoluta del nodo
childNodes: Devuelve una colección con todos los nodos hijo de este nodo. Esta colección es un objeto de la clase NodeList. Recuerda que los atributos no son hijos
firstChild: Devuelve el primer hijo de un nodo.
lastChild: Devuelve el último hijo de un nodo.
localName: Devuelve la parte local del nombre de un nodo.
namespaceURI: Si estamos utilizando espacios de nombre, este método devolverá el URI con el namespace del nodo.
nextSibling: Devuelve el siguiente nodo hermano (está en el mismo nivel de jerarquía en el árbol y tiene el mismo padre)

nodeName: Devuelve el nombre del nodo dependiendo de su tipo.
nodeType: Devuelve el tipo del nodo (elemento, atributo, comentario, etc). Estos tipos de datos se almacenan en la interfaz mediante un conjunto de constantes estáticas (hay 12, una para cada tipo de nodo). Y podemos utilizar el valor numérico o el nombre de la constante para averiguar de qué tipo de nodo se trata:

Node.ELEMENT_NODE=1
Node.ATTRIBUTE_NODE=2
Node.TEXT_NODE=3
Node.CDATA_SECTION_NODE=4
Node.ENTITY_REFERENCE_NODE=5
Node.ENTITY_NODE=6
Node.PROCESSING_INSTRUCTION_NODE=7
Node.COMMENT_NODE=8
Node.DOCUMENT_NODE=9
Node.DOCUMENT_TYPE_NODE=10
Node.DOCUMENT_FRAGMENT_NODE=11
Node.NOTATION_NODE=12

nodeValue: Permite leer/modificar el valor del nodo, pero dependiendo de su tipo.
ownerDocument: Devuelve una referencia a la raíz del árbol (objeto document) en el que se encuentra este nodo
parentNode: Devuelve una referencia al padre de este nodo.
prefix: Permite leer/modificar el prefijo del espacio de nombres.
previousSibling: Retorna una referencia al hermano anterior (opuesto a nextSibling).
textContent: Permite leer/modificar el contenido textual de un nodo y sus descencientes.

Una propiedad muy utilizada es **childNodes**: devuelve una colección con todos los nodos hijo del nodo actual. Y prestemos atención a las propiedades **nodeName** y **nodeValue**. Estas dos propiedades se comportan de forma diferente dependiendo del tipo de nodo. Por ejemplo sólo los atributos, los textos y los comentarios tienen valor:

Si el nodo es:	nodeName devuelve:	nodeValue devuelve:
Element	Nombre del elemento mayúsculas (ej:A).	null
Attr	Nombre del atributo minúsculas (ej: href).	El valor del atributo. (ej: www.google.es)
Text	#text	El contenido del texto
Comment	#comment	El texto del comentario
Document	#document	null
DocumentType	El nombre del doctype	null

Y por otra parte si queremos averiguar el tipo de nodo para una referencia utilizaremos la propiedad nodeType. Recuerda que había doce tipos diferentes de nodos en el árbol. Para trabajar con los tipos de nodos la interfaz Node define una serie de constantes que podemos utilizar en nuestros códigos:

De esta forma para saber si un nodo es un elemento podríamos hacer:

```
// código JS
var objeto = document.getElementById("mybox");
if (objeto.nodeType == Node.ELEMENT_NODE) { …. Es un elemento …}
// o bien utilizar el valor numérico
if (objeto.nodeType == 1) { …. Es un elemento …}
```

Node.ELEMENT_NODE=1	**Node.PROCESSING_INSTRUCTION_NODE=7**
Node.ATTRIBUTE_NODE=2	Node.COMMENT_NODE=8
Node.TEXT_NODE=3	Node.DOCUMENT_NODE=9
Node.CDATA_SECTION_NODE=4	Node.DOCUMENT_TYPE_NODE=10
Node.ENTITY_REFERENCE_NODE=5	Node.DOCUMENT_FRAGMENT_NODE=11
Node.ENTITY_NODE=6	Node.NOTATION_NODE=12

Actividad 2

Cargo
Presidente
Presidente
presidente
Suplente
Policía
PreSidente
Suplente
Suplente
policia

Modifica las celdas donde aparezca el cargo de presidente para que tenga la clase "presidente".

PISTA: accede a la colección de todos los <td>. Busca en sus hijos: si el primer hijo es de tipo texto y su valor es "Presidente" cambia la clase del objeto td por la clase "presidente".

Realiza lo mismo para el cargo policía, para que se aplique la clase "policia" en la fila.

Id	Junta Provincial	Responsable	Cargo	Email	Número de votantes	Votos en blanco
1	Almería	María Mercedes Benz Seguro	Presidente	mmbseg.es	452579	3507
2	Cádiz	Rinder Kate Proud	Presidente	rkp@gmail.es	993327	9194
3	Córdoba	Jesús Manuel Pérez Lindl	presidente	jmperlin@hotmail.es	653926	6946
4	Granada	Jack Adrian Lang	Suplente		74777	5251
5	Huelva	Soledad Aburrida Díaz	Policía	??	398492	3456
6	Jaén	Curly Team Papa	PreSidente	papateam	535847	4226
7	Málaga	Paterquito Otoño Helado	Suplente	pater@.es	1153047	8424
8	Sevilla	Jaqueline Hernández García	Suplente		1529822	13713
9	Mérida	Brenda Trans Delte	policía	btrdan@cse.es	2536	329

PISTA: una vez que hayas localizado el <td> necesario, recuerda que la fila de la tabla, elemento <tr>, es su padre. Para eso basta con utilizar la propiedad parentNode.

El problema de los espacios y saltos de línea

En los navegadores actuales la gestión de los espacios en blanco y saltos de línea dificulta mucho el trabajo con la interfaz Node, ya que se crean nodos por cada salto o grupos de espacios en blanco detectados. IE por su parte suele ignorar estos nodos (¡¡este aspecto es positivo!!). Esto complica aún más las cosas, ya que cada navegador puede mostrar un número de hijos distinto para un elemento.

Actividad 3

Muestra un alert con el número de hijos de la primera fila, deberían ser 7 filas, pero al hacerlo ocurre que se muestran 15 ¿será culpa de los espacios en blanco?, ocurre que al crear las columnas hay retornos de carro y espacios en blanco que se conforman como si se trataran de un nodo de texto.

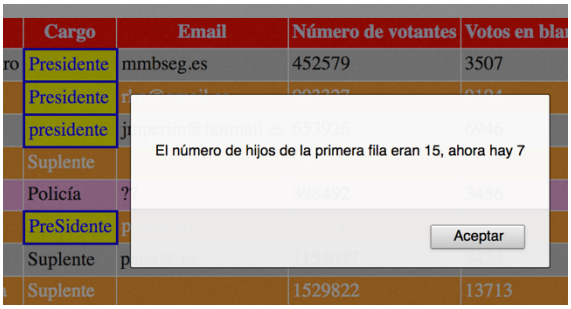
Este problema afecta mucho a propiedades de la interfaz Node, como con firstChild (primer hijo), lastChild (último hijo), nextSibling (siguiente hermano) y previousSibling (hermano anterior).

En el fichero html elimina todos los espacios en blanco y saltos de línea de la tabla y muestra ahora un alert con el número de hijos que contiene, deberían ser 7.

Programa la función limpiar_blancos(fila). Esta función recibe una fila de la tabla y elimina todos los nodos hijo de tipo texto que contienen saltos de línea o blancos. Para ello utiliza una expresión regular como esta /^[\n\s]+$/.

La idea es recorrer todos los nodos hijo (childNodes) de la fila recibida: Si el nodo es de tipo texto y cumple la expresión regular se elimina con el método removeChild(nodo) sobre la propia fila. Observa por tanto cómo para eliminar un nodo debemos contar con una referencia al elemento padre.

Para comprobar que ya no existen nodos vacíos realiza un recorrido sobre las filas de la tabla imprimiendo el número de hijos. El resultado ya no debe ser 15 sino 7.



Una vez hecho este trabajo sí que podremos utilizar libremente y sin errores las propiedades firstChild, lastChild, nextSibling y previousSibling.

Busca las columnas donde aparecen los responsables, sólo en el caso de aquellos cuyo nombre empiece por "J" realiza las siguientes acciones:

- El primer y último hijo de esa fila obtendrán la clase "borde_rojo".
- El propio texto del responsable se transformará en mayúsculas.
- Utilizando nextSibling y previousSibling añade la clase "destaca" a la Junta Provincial y al Email

La tabla quedará:

Id	Junta Provincial	Responsable	Cargo	Email	Número de votantes	Votos en blanco
1	Almería	María Mercedes Benz Seguro	Presidente	mmbseg.es	452579	3507
2	Cádiz	Rinder Kate Proud	Presidente	rkp@gmail.es	993327	9194
3	Córdoba	JESÚS MANUEL PÉREZ LINDL	presidente	jmperlin@hotmail.es	653926	6946
4	Granada	JACK ADRIAN LANG	Suplente		74777	5251
5	Huelva	Soledad Aburrida Díaz	Policía	??	398492	3456
6	Jaén	Curly Team Papa	PreSidente	papateam	535847	4226
7	Málaga	Paterquito Otoño Helado	Suplente	pater@.es	1153047	8424
8	Sevilla	JAQUELINE HERNÁNDEZ GARCÍA	Suplente		1529822	13713
9	Mérida	Brenda Trans Delte	policía	btrdan@cse.es	2536	329

Los atributos

Para terminar de analizar las principales propiedades de la interfaz node nos fijaremos ahora en la propiedad attributes. Cuando el nodo es de tipo elemento esta propiedad es una colección y en cada posición dispondremos de un nodo de la clase Attr. En los atributos también podemos utilizar las propiedades nodeName (nombre del atributo) y nodeValue (valor del atributo).

Actividad 4

Acceda a los atributos de la imagen con id “escudo” para que src se igual a "img/objeto_dom.gif" y el atributo alt sea "Logotipo del Gobierno".

9 Merida Brenda Trans Delt

Los métodos de la interfaz node

appendChild(new_node): Agrega un nuevo nodo hijo n. Lo agrega por el final (será el último hijo).
cloneNode(profundidad): Realiza una copia de un nodo con todos sus atributos y valores. El parámetro profundidad es de tipo booleano. Si utilizas true se copiará no sólo el elemento sino también todos sus descendientes.
compareDocumentPosition(n): Compara la posición en el documento del nodo n con la del nodo actual.
getFeature(feature,version): Devuelve un objeto DOC que implementa una API especializada.
getUserData(key): Devuelve el objeto asociado con una clave “key” en este nodo. Para poder utilizarlo se ha debido asignar previamente la clave con una llamada a setUserData.
hasAttributes(): Método booleano que devuelve si el nodo contiene atributos o no.
hasChildNodes(): Método booleano que devuelve si el nodo contiene hijos o no.
insertBefore(new_node, ref): Inserta un nuevo nodo hijo antes de un hijo especificado.
isDefaultNamespace(): Devuelve true si el nodo utiliza el namespace por defecto. En otro caso false
isEqualNode(node): Comprueba si dos nodos son iguales.
isSameNode(node): Comprueba si dos nodos son el mismo (esto es: si son iguales y la referencia en memoria es exactamente la misma).
isSupported(feature, version): Método booleano que devuelve true si una característica especial es soportada en un nodo.
lookupNamespaceURI(prefix): Devuelve el namespace URI que se corresponde con un prefijo especificado.
lookupPrefix(namespace): Devuelve el prefijo que se corresponde con un namespace URI especificado.
normalize(): Unifica nodos de texto adyacentes y elimina los nodos de texto vacíos.

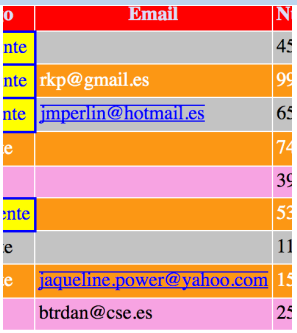
removeChild(child): Elimina un nodo hijo.
replaceChild(new_child, old): Reemplaza un nodo hijo.
setUserData(key,data,handler): Asocia un objeto a una clave en un nodo

Si por ejemplo queremos crear un párrafo nuevo haremos:

```
var parrafo = document.createElement("p");
var texto = document.createTextNode("Este es un nuevo párrafo");
parrafo.appendChild(texto); // agregamos el nodo de texto al párrafo
var cuerpo = document.getElementsByTagName("body")[0]; //acceso a body
cuerpo.appendChild(parrafo); // agregamos el elemento párrafo al final
de <body>
```

Actividad 5

1. Añade un correo electrónico al registro con id igual a 8 para que Jaqueline Hernández García tenga el correo "jaqueline.power@yahoo.com".
2. Utilizando expresiones regulares verifique que los correos electrónicos son correctos, aquellos que no lo sean serán eliminados. NOTA: elimine el nodo texto, no el <td>
3. Clona la primera fila y añádela al <tfoot>



o	Email	N
nte		45
nte	rkp@gmail.es	99
nte	jmperlin@hotmail.es	65
e		74
		39
nte		53
e		11
e	jaqueline.power@yahoo.com	15
	btrdan@cse.es	25

Id	Junta Provincial	Responsable	Cargo	Email	Número de votantes	Votos en blanco
1	Almería	María Mercedes Benz Seguro	Presidente		452579	3507
2	Cádiz	Rinder Kate Proud	Presidente	rkp@gmail.es	993327	9194
3	Córdoba	JESÚS MANUEL PÉREZ LINDL	presidente	jmperlin@hotmail.es	653926	6946
4	Granada	JACK ADRIAN LANG	Suplente		74777	5251
5	Huelva	Soledad Aburrida Díaz	Policía		398492	3456
6	Jaén	Curly Team Papa	PreSidente		535847	4226
7	Málaga	Paterquito Otoño Helado	Suplente		1153047	8424
8	Sevilla	JAQUELINE HERNÁNDEZ GARCÍA	Suplente	jaqueline.power@yahoo.com	1529822	13713
9	Mérida	Brenda Trans Delte	policía	btrdan@cse.es	2536	329
Id	Junta Provincial	Responsable	Cargo	Email	Número de votantes	Votos en blanco

NodeList: listas ordenadas de nodos

En las prácticas anteriores habrás observado que a menudo accedemos a los hijos de un elemento y obtenemos una lista ordenada de nodos.

Esto lo hacemos cada vez que utilizamos la propiedad **childNodes**, o incluso cuando utilizamos **getElementsByTagName**(). En ese momento se crean colecciones ordenadas de nodos donde sus elementos pueden ser accedidos por el índice.

Estas colecciones son en realidad objetos de la clase **NodeList**. Esta clase define únicamente una propiedad: length que guarda el tamaño de la colección, y un único método llamado **item(pos)** que nos permite obtener cualquier elemento de la colección mediante su índice **(pos)**, aunque nosotros solemos emplear los accesos de tipo array, **[pos]**.

Los siguientes códigos serían por tanto equivalentes:

```
var tabla = document.getElementById("tblUsuarios");
var filas = tabla.getElementsByTagName("tr");
// copiar la primera fila
var copia = filas[0].cloneNode(true);
tabla.appendChild(copia);
```

```
var tabla = document.getElementById("tblUsuarios");
var filas = tabla.getElementsByTagName("tr");
// copiar la primera fila
var copia = filas.item(0).cloneNode(true);
tabla.appendChild(copia);
```

NamedNodeMap: listas no ordenadas de nodos.

Los objetos de la clase **NamedNodeMap** representan colecciones no ordenadas. Y como segunda particularidad tenemos que estas colecciones son del tipo vector asociativo, con lo que podemos acceder a los elementos mediante su nombre. Este tipo de objetos se utiliza, por ejemplo, cuando obtenemos una colección con los atributos de un nodo (empleando la propiedad **attributes**).

La única propiedad de este objeto es **length**, aunque dispone de varios métodos para soportar el acceso por nombre y el acceso por índices:
getNamedItem(name): Devuelve un nodo de la colección especificando su nombre
getNamedItemNS(name): Devuelve un nodo de la colección especificando su nombre y su namespace
item(index): Devuelve un nodo de la colección especificando su índice
removeNamedItem(name): Elimina un nodo de la colección especificando su nombre
removeNamedItemNS(name): Elimina un nodo de la colección especificando su nombre y su namespace
setNamedItem(name): Modifica un nodo de la colección especificando su nombre
setNamedItemNS(name): Modifica un nodo de la colección especificando su nombre y su namespace

```
var btn = document.getElementsByTagName("H1")[0];
var typ = document.createAttribute("class");
typ.value = "democlass";
btn.attributes.setNamedItem(typ);
```

DOM a alto nivel.

En el apartado anterior hemos visto manipulación del DOM a bajo nivel, utilizando la interfaz Node que representa un comportamiento común para todos los tipos de nodos del DOM.

Sin embargo cada tipo de nodo (elementos, atributos, nodos de texto, comentarios, …) tiene una serie de funcionalidades diferentes. Por ese motivo para cada tipo de nodo se han definido una serie de métodos específicos. Así las inserciones, borrados, modificaciones y otras operaciones serán más sencillas.

Por ejemplo: hemos visto cómo acceder a los atributos mediante NamedNodeMap. Pero también hemos visto qué fácil es modificarlos con el método setAttribute de la clase Element.

En este apartado veremos los tipos de nodos que nos interesan para manipular el DOM-HTML, con detalle de sus propiedades y métodos.

Objeto Document.

Este objeto representa la raíz del DOM. Es decir, todo el contenido de nuestra página web estará bajo este objeto. Además de los conocidos métodos para acceder elementos (getElementById(), getElementsByName() y getElementsByTagName()) dispondremos de métodos para la creación de elementos nuevos.

Cualquier nodo del árbol dispone de la propiedad **ownerDocument** (descrita en las propiedades de la interfaz Node), que almacena una referencia al objeto document donde este nodo fue creado. Esta propiedad puede ser útil al trabajar con ventanas o marcos para conseguir siempre un acceso rápido a la raíz del DOM que nos interese.

Antes de presentar el objeto document recordemos que en una página web este objeto es la raíz del árbol, y está situado encima del primer elemento.

Para acceder al primer elemento, que es la etiqueta <html>…</html> tendremos que hacerlo mediante:

```
var primero = document.documentElement; // etiqueta <html>
```

Conviene no crear elementos nuevos como hijos de document, sino siempre como hijos de documentElement pues en otro caso quedarían fuera de la etiqueta <html>…</html>.

Veamos la definición de la clase Document:

Muchos de sus propiedades y métodos sirven para el trabajo con el XML-DOM. Otros muchos métodos son exclusivos para trabajo con documentos XML que además soportan espacios de nombres, pudiendo así filtrar por ejemplo elementos que tengan un prefijo determinado.

PROPIEDADES:

doctype: Devuelve el objeto DocumentType asociado con este documento.
documentElement: Devuelve el primer elemento de este documento. Este elemento es el correspondiente a <html>…</html>. Se trata de un objeto de la clase HTMLElement.
documentURI: Propiedad de lectura/escritura. Almacena la localización del documento.
domConfig: Devuelve la configuración empleada cuando se invocó a normalizeDocument().
implementation: Devuelve el objeto de la clase DOMImplementation, encargado de gestionar este documento.
inputEncoding: Almacena el sistema de codificicación (charset) empleado en el documento.
strictErrorChecking: Propiedad booleana. Indica si se está forzando el el chequeo de errores o no.
xmlEncoding: Retorna el sistema de codificación XML (para documentos XML)

xmlStandalone: Propiedad de lectura/escritura. Indica cuando un documento XML es de tipo standalone o no (para documentos XML).
xmlVersion: Establece/devuelve la versión XML (para documentos XML)

MÉTODOS:
adoptNode(node): Adopta un nodo de otro documento al documento actual. Se retorna el nodo.
createAttribute(): Permite crear un nodo de atributo.
createAttributeNS(URI,name): Permite crear un nodo de atributo especificando namespace URI y nombre.
createCDATASection(text): Sólo para XML: permite crear una sección CDATA con el texto especificado.
createComment(text): Permite crear un nodo comentario con el texto especificado.
createDocumentFragment(): Permite crear un nodo vacío de tipo DocumentFragment.
createElement(etiqueta): Permite crear un nodo de elemento.
createElementNS(URI): Permite crear un nodo de elemento con el namespace URI especificado.
createEntityReference(): Sólo para XML. Permite crear un nodo de tipo EntityReference.
createProcessingInstruction(): Sólo para XML. Permite crear un nodo de tipo ProcessingInstruction.
createTextNode(text); Permite crear un nodo de tipo texto.
getElementById(): Devuelve una referencia al elemento con la ID especificada.
getElementsByTagName(): Devuelve un NodeList con todos los elementos de la etiqueta especificada.
getElementsByTagNameNS(): Devuelve un NodeList con los elementos del namespace y etiqueta especificados.
importNode(): Importa un nodo de otro documento.
normalizeDocument(): Elimina los nodos de Texto vacíos y unifica los nodos de este tipo adyacentes.
renameNode(): Renombra el nodo especificado.

Creando elementos.

Para crear un elemento nuevo basta con utilizar el método createElement() que recibe el nombre de la etiqueta. Ya lo hemos utilizado varias veces. Por ejemplo para agregar un <option> dentro de un <select>:

```
var opcion = document.createElement("option");
```

Por otra parte cuando utilizamos objetos recuerda que existen objetos HTML-DOM adecuados para cada elemento. Por ejemplo en el caso de los objetos option disponíamos de las propiedades value y text para fijar el valor y el texto respectivamente.
Veamos un ejemplo de creación de elementos. Partiremos de este extracto de código HTML:

```
<div id="micampo"></div>
```

Y vamos a crear un párrafo con un texto. Tendremos que crear primero un elemento p, y después un nodo de texto. Observa:

```
// creamos el objeto Element con la etiqueta <p>
var miparrafo = document.createElement("p");
// creamos un objeto Text con un texto
var mitexto = document.createTextNode("Desarrollo web");
// insertamos el texto en el párrafo
miparrafo.appendChild(mitexto);
// accedemos al div y le agregamos un nuevo hijo:
document.getElementById("micampo").appendChild(miparrafo);
```

Y mediante la manipulación del DOM hemos provocado que el div quede:

```
<div id="micampo">
<p>Desarrollo Web</p>
</div>
```

Lógicamente esto podríamos hacerlo sin utilizar los métodos del DOM, mediante el uso de innerHTML:

```
document.getElementById("micampo").innerHTML = "<p>Desarrollo
web</p>";
```

A pesar de la facilidad de innerHTML la manipulación del DOM es muy útil cuando nos encontramos con contenidos totalmente dinámicos o con creaciones e inserciones complejas: por ejemplo en páginas ya creadas podemos, en el cliente, modificar un elemento insertando o borrando elementos hijo sin alterar el resto del contenido.
Y otro ejemplo en el que **el uso del DOM es más eficiente que el innerHTML**: la manipulación de tablas.

En la siguiente página tienes una guía de referencia sobre el objeto HTML DOM Table: http://www.w3schools.com/jsref/dom_obj_table.asp. Este objeto representa un elemento <table>. Tiene propiedades interesantes como:
rows: propiedad de un nodo tabla que contiene todas sus filas (<tr>)
Y métodos para insertar o eliminar fácilmente:
insertRow(pos): insertar nueva fila vacía (nodo tr). La primera está en posición 0.
deleteRow(pos): borrar la fila determinada por la posición pos.

Por su parte si un nodo del árbol se corresponde con el elemento <tr> también dispone de un objeto asociado con propiedades y métodos específicos. El objeto HTML DOM TR está descrito en: http://www.w3schools.com/jsref/dom_obj_tablerow.asp

Este objeto HTML DOM TR dispone de la propiedad cells que retorna una colección de las celdas de esta fila (sean <td> o <th>). Y dispone de métodos como
insertCell(pos): insertar nueva celda vacía (nodo td) en la posición pos.
deleteCell(pos): borrar la celda en la posición pos.

Conocer bien cada objeto Javascript puede producir que los códigos sean más sencillos y estructurados que el mero uso de la interfaz Node.

Actividad 6

Crear una columna dinámicamente. Crearemos en la tabla una columna con la edad, estará entre el nombre del representante y su cargo. Para rellenar los datos se los pediremos dinámicamente al usuario (utilice prompt).

Id	Junta Provincial	Responsable	Edad	Cargo	Email	Número de votantes	Votos en blanco
1	Almería	María Mercedes Benz Seguro	18	Presidente		452579	3507
2	Cádiz	Rinder Kate Proud	23	Presidente	rkp@gmail.es	993327	9194
3	Córdoba	JESÚS MANUEL PÉREZ LINDL	55	presidente	jmperlin@hotmail.es	653926	6946
4	Granada	JACK ADRIAN LANG	34	Suplente		74777	5251
5	Huelva	Soledad Aburrida Díaz	22	Policía		398492	3456
6	Jaén	Curly Team Papa	25	PreSidente		535847	4226
7	Málaga	Paterquito Otoño Helado	30	Suplente		1153047	8424
8	Sevilla	JAQUELINE HERNÁNDEZ GARCÍA	48	Suplente	jaqueline.power@yahoo.com	1529822	13713
9	Mérida	Brenda Trans Delte	33	policía	btrdan@cse.es	2536	329
Id	Junta Provincial	Responsable	Edad	Cargo	Email	Número de votantes	Votos en blanco

Utiliza los métodos de los objetos HTML-DOM Table y HTML-DOM Tr. PISTA: puede utilizar los createTextNode, insertCell, rows, cells…

Objeto Element.

Este objeto representa una etiqueta de un documento HTML o bien XML. Puede contener hijos y también nodos de atributos asociados. Los hijos pueden ser otros elementos, textos, comentarios, y en el caso de documentos XML pueden ser instrucciones de procesamiento, secciones CDATA o entidades de tipo referencia.

Este objeto dispone también de métodos para el acceso y creación rápida de atributos. En alguna práctica anterior utilizamos el método setAttribute(nombre, valor) que nos permite crear atributos nuevos o en el caso de que el atributo exista modificar su valor.
Veamos la definición de esta clase Element:
PROPIEDADES:
schemaTypeInfo: Devuelve la información de tipo (según esquemas XML).
tagName: Devuelve una cadena con la etiqueta del elemento.

MÉTODOS:
getAttribute(name): Devuelve el valor del atributo especificado.
getAttributeNS(name): Solo para XML. Devueve el valor del atributo con el espacio de nombres.
getAttributeNode(name): Devuelve el nodo atributo especificado.
getAttributeNodeNS(name): Sólo para XML. Devuelve el nodo atributo especificado con espacio de nombres.
getElementsByTagName(): Devuelve una colección de nodos hijo con un nombre de etiqueta.
getElementsByTagNameNS(): Sólo para XML devuelve la colección de hijos con etiqueta especificada.
hasAttribute(name): True si el elemento disponde de ese atributo. False en caso contrario.
hasAttributeNS(name): Idem sólo para XML. Consulta si el elemento dispone de ese atributo.
removeAttribute(name): Elimina el atributo especificado

removeAttributeNS(name): Idem sólo para XML. Elimina un atributo (utiliza namespaces).
removeAttributeNode(name): Se elimina el atributo del elemento pero se devuelve el nodo eliminado.
setAttribute(name, value): Crea o modifica el atributo con un valor especificado.
setAttributeNS(name, value): Idem sólo para XML. Crea o si existe modifica el atributo con el valor.
setAttributeNode(name, value): Crea o modifica el nodo del atributo con el nombre y el valor.
setAttributeNodeNS(): Idem sólo para XML con namespaces. Crea o modifica un nodo atributo

Actividad 7

En la tabla provoca que las celdas de la columna 'Junta Provincial' cuyo valor comience por "M" o por "G" tengan alineación a la derecha.

Objeto Attr.

Por último este objeto representa cada uno de los atributos de un elemento. Recuerda que el conjunto de atributos de un elemento se conforma con una colección de tipo **NamedNodeMap**, ya que son un conjunto de nodos no ordenados. El orden en que se escriben los atributos de un elemento no tiene trascendencia.

El objeto atributo dispone de pocas propiedades: para acceder al nombre, al elemento dueño del atributo y para leer o escribir el valor.

PROPIEDADES:

isId: True si el atributo es de tipo ID. False en caso contrario.
name: Devuelve el nombre del atributo.
ownerElement: Devuelve una referencia al elemento que contiene este atributo.
schemaTypeInfo: Devuelve la información de tipos (esquemas XML) del atributo.
specified: Devuelve true si el atributo ha sido especificado (tiene un valor asignado). False en caso contrario.
value: Propiedad de lectura/escritura. Se corresponde con el valor del atributo

Este objeto es poco utilizado por la sencilla razón de que los métodos del objeto Element permiten trabajar con los atributos de forma más cómoda.

Actividad 8

Utilice deleteRow para eliminar última fila.

Utiliza los siguientes elementos para crear una última fila en tfoot, para que muestre los totales, fíjate que hay varias celdas combinadas.

Id	Junta Provincial	Responsable	Edad	Cargo	Email	Número de votantes	Votos en blanco
1	Almería	María Mercedes Benz Seguro	20	Presidente		452579	3507
2	Cádiz	Rinder Kate Proud	20	Presidente	rkp@gmail.es	993327	9194
3	Córdoba	JESÚS MANUEL PÉREZ LINDL	20	presidente	jmperlin@hotmail.es	653926	6946
4	Granada	JACK ADRIAN LANG	20	Suplente		74777	5251
5	Huelva	Soledad Aburrida Díaz	20	Policía		398492	3456
6	Jaén	Curly Team Papa	20	PreSidente		535847	4226
7	Málaga	Paterquito Otoño Helado	20	Suplente		1153047	8424
8	Sevilla	JAQUELINE HERNÁNDEZ GARCÍA	20	Suplente	jaqueline.power@yahoo.com	1529822	13713
9	Mérida	Brenda Trans Delte	20	policía	btrdan@cse.es	2536	329
TOTAL:						5794353	55046

PISTA: utiliza los siguientes elementos: tFoot, insertRow, setAttribute, insertCell, createTextNode, appendChild, tBodies, rows

Fuentes

http://www.maestrosdelweb.com/que-es-javascript/
http://librosweb.es/libro/javascript/
http://librosweb.es/
http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=788:tipos-de-datos-javascript-tipos-primitivos-y-objeto-significado-de-undefined-null-nan-ejemplos-cu01112e-&catid=78:tutorial-basico-programador-web-javascript-desde-&Itemid=206
https://www.todojs.com/ref/javascript/global/
https://jherax.wordpress.com/2014/07/08/javascript-poo-1/