Tema 5

DOM

DOM

Para W3C DOM (Modelo de objetos del documento) es una interfaz de programación de aplicaciones (API) para documentos HTML y XML. El DOM define la estructura lógica de los documentos y el modo en que se accede y manipula un documento.

Con el DOM los programadores pueden construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido.

Todos los navegadores deben contener y contienen, este interfaz para poder trabajar con documentos HTML y XML.

Al trabajar con DOM nos proporciona un control muy preciso sobre la estructura del documento HTML. DOM crea una estructura de árbol del documento que se va a manipular.

Veamos uno de los ejemplos empleados en el tema

```
<!DOCTYPE html>
<html>
<head>
<script>
window.onload=function(){
 alert("la página se ha cargado correctamente");
}
</script>
</head>
<body>
<p>HOLA MUNDO</p>
</body>
</html>
```

Cada elemento se denomina nodo. Existen diferentes tipos de nodos. Los nodos que a nosotros nos puede interesar son los siguientes:

- **Document**: es el nodo raíz de todos los documentos HTML y XML. Todos los demás nodos derivan de él.
- **DocumentType**: es el nodo que contiene la representación del DTD empleado en la página (indicado mediante el DOCTYPE).
- **Element**: representa el contenido definido por un par de etiquetas de apertura y cierre (<etiqueta>...</etiqueta>) o de una etiqueta *abreviada* que se abre y se cierra a la vez (<etiqueta/>). Es el único nodo que puede tener tanto nodos hijos como atributos.
- **Attr**: representa el par nombre-de-atributo/valor.
- **Text**: almacena el contenido del texto que se encuentra entre una etiqueta de apertura y una de cierre. También almacena el contenido de una sección de tipo CDATA.

- **CDataSection**: es el nodo que representa una sección de tipo <![CDATA[ ]]>.
- **Comment**: representa un comentario de XML.

Como hemos comentado DOM crea un árbol de la página web y cada elemento es un nodo y los nodos son de un tipo determinado. Aquí tienes la lista de tipos de nodos. Están definidas como constantes.

- 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

Ahora podemos ver las propiedades y métodos de Node

Propiedad/Método	Valor devuelto	Descripción
nodeName	String	El nombre del nodo (no está definido para algunos tipos de nodo)
nodeValue	String	El valor del nodo (no está definido para algunos tipos de nodo)
nodeType	Number	Una de las 12 constantes definidas anteriormente
ownerDocument	Document	Referencia del documento al que pertenece el nodo
firstChild	Node	Referencia del primer nodo de la lista childNodes
lastChild	Node	Referencia del último nodo de la lista childNodes
childNodes	NodeList	Lista de todos los nodos y elementos hijo del nodo actual
children	Element	Lista todos los elementos hijos de un elemento.
previousSibling	Node	Referencia del nodo hermano anterior o null si este nodo es el primer hermano
nextSibling	Node	Referencia del nodo hermano siguiente o null si este nodo es el último hermano
hasChildNodes()	Bolean	Devuelve true si el nodo actual tiene uno o más nodos hijo

Propiedad/Método	Valor devuelto	Descripción
attributes	NamedNodeMap	Se emplea con nodos de tipo Element. Contiene objetos de tipo Attr que definen todos los atributos del elemento
appendChild(nodo)	Node	Añade un nuevo nodo al final de la lista childNodes
removeChild(nodo)	Node	Elimina un nodo de la lista childNodes
replaceChild(nuevoNodo, anteriorNodo)	Node	Reemplaza el nodo anteriorNodo por el nodo nuevoNodo
insertBefore(nuevoNodo, anteriorNodo)	Node	Inserta el nodo nuevoNodo antes que la posición del nodoanteriorNodo dentro de la lista childNodes

Para empezar a estudiar tenemos que saber diferentes propiedades y métodos de los nodos de tipo Document y Element.

En el tema de *Formularios* ya explicamos los siguientes métodos *getElementById(), getElementsByName(), getElementsByTagName(), getElementsByClassName()*. Los tres últimos métodos devuelve un HTMLCollection.

HTMLCollection es un interface que representa una colección genérica (objeto como un array, similar a arguments) de elementos y ofrece unos métodos y propiedades para seleccionar desde la lista. (https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection // https://dom.spec.whatwg.org/#htmlcollection).

También podemos utilizar ***querySelectorAll()*** para devolver un array de elementos. A diferencia de los métodos vistos anteriormente, este método devuelve un NodeList.

El objeto NodeList es una colección de nodos. (Como argumento le pasamos un selector de css, nos devuelve elementos). El objeto NodeList se puede recorrer utilizando el método *forEach()*. También tenemos que saber que hay algunos navegadores que no reconocen este método.

Otro método similar es ***querySelector()***. Este método devuelve el primer elemento que coincida con el selector pasado por parámetro.

Son dos métodos de objetos contenedor. (document, Element....)

DOM Element posee las siguientes tres propiedades:

- style → Propiedad que nos permite asignar u obtener el atributo style de un elemento. Esta propiedad a su vez tiene propiedades que son las propiedades de CSS. Por tanto, para poder acceder a una propiedad de CSS de una etiqueta hay que utilizar la siguiente sintaxis:

```
variable = document.getElementById("id").style.propiedad;
document.getElementById("id").style.propiedad = "valor";
```

Hay que tener en cuenta la siguiente lista para poder acceder a las propiedades de CSS.

Propiedad CSS	Sintaxis JavaScript	Propiedad CSS	Sintaxis JavaScript
background	background	background-attachment	backgroundAttachment
background-color	backgroundColor	background-image	backgroundImage

Propiedad CSS	Sintaxis JavaScript	Propiedad CSS	Sintaxis JavaScript
background-position	backgroundPosition	background-repeat	backgroundRepeat
border	border	border-bottom	borderBottom
border-bottom-color	borderBottomColor	border-bottom-style	borderBottomStyle
border-bottom-width	borderBottomWidth	border-color	borderColor
border-left	borderLeft	border-left-color	borderLeftColor
border-left-style	borderLeftStyle	border-left-width	borderLeftWidth
border-right	borderRight	border-right-color	borderRightColor
border-right-style	borderRightStyle	border-right-width	borderRightWidth
border-style	borderStyle	border-top	borderTop
border-top-color	borderTopColor	border-top-style	borderTopStyle
border-top-width	borderTopWidth	border-width	borderWidth
clear	clear	clip	clip
color	color	cursor	cursor
display	display	filter	filter
float	cssFloat	font	font
font-family	fontFamily	font-size	fontSize
font-variant	fontVariant	font-weight	fontWeight
height	height	left	left
letter-spacing	letterSpacing	line-height	lineHeight
list-style	listStyle	list-style-image	listStyleImage
list-style-position	listStylePosition	list-style-type	listStyleType
margin	margin	margin-bottom	marginBottom
margin-left	marginLeft	margin-right	marginRight
margin-top	marginTop	overflow	overflow
padding	padding	padding-bottom	paddingBottom
padding-left	paddingLeft	padding-right	paddingRight
padding-top	paddingTop	page-break-after	pageBreakAfter
page-break-before	pageBreakBefore	position	position

Propiedad CSS	Sintaxis JavaScript	Propiedad CSS	Sintaxis JavaScript
text-align	textAlign	text-decoration	textDecoration
text-indent	textIndent	text-transform	textTransform
top	top	vertical-align	verticalAlign
visibility	visibility	width	width
z-index	zIndex		

http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=807:cambiar-css-con-javascript-lista-o-tabla-de-equivalencias-de-propiedades-css-js-camelcase-cu01129e&catid=78:tutorial-basico-programador-web-javascript-desde-&Itemid=206

- value → Si estamos trabajando con un formulario nos devuelve el valor del campo.
- className → Propiedad que nos permite asignar y obtener una clase definida en la hoja de estilos.

Para poder acceder a los diferentes atributos que posee un elemento (un tag) simplemente tenemos que acceder utilizando la sintaxis del punto.

```
document.getElementById("parrafo1").className = "clase";
variable = document.getElementById("parrafo1").className;
console.log(document.getElementById("parrafo1").id);
```

Creación de elementos en tiempo de ejecución. DOM

Para crear elementos de un documento html en tiempo de ejecución podemos utilizar el método ***createElement("elemento")***.

Tenemos que ir creando desde el elemento más interno y hay que ir añadiendo a los elementos subiendo en el árbol de DOM. El texto que aparece en la página web es un nodo del tipo Text, por tanto, tenemos que crear este primer nodo con el método ***createTextNode("texto")***.

Para añadir un elemento a otro hay que utilizar el método ***appendChild(elemento)***.

Veamos ahora un ejemplo para que todo quede un poquito más claro.

```
<script>
   window.addEventListener("load",function(){
      var x = document.createElement("p"); // creamos el objeto párrafo
      var y = document.createTextNode("esto es un texto"); // creamos el texto
      x.appendChild(y); // añadimos el texto al párrafo
      //document.getElementsByTagName("body")[0].appendChild(x);
      document.body.appendChild(x); //añadimos el párrafo al documento.
   });
</script>
```

El método ***padre.insertBefore(este_elemento,antes_de_este)***, se utiliza para introducir un elemento antes que otro elemento dentro de otro elemento .

Otros métodos que podemos utilizar son ***padre.append(x)*** añade el elemento *x* en la última posición del contenedor *padre.* Y ***padre.prepend(x)*** añade el elemento x como primer hijo del contenedor *padre*.

Estos métodos se estudiarán más adelante en este tema.

Ahora, nos puede interesar poder colocar un mismo elemento en diversas partes de un mismo documento web. Para conseguir esto sólo tenemos que clonar el elemento, ***cloneNode(boolean)***. *Boolean* será *true* o *false*. Si colocamos *true*, clonará todos los elementos hijos del objeto,si pasamos el valor *false* sólo clona el objeto.

```
<script>
window.addEventListener("load",function(){
   var x = document.createElement("p");
   var y = document.createTextNode(" esto es un texto ");
   document.getElementById("prueba").appendChild(y.cloneNode(true));
   x.appendChild(y);
   document.body.appendChild(x.cloneNode(true));
   document.body.appendChild(x.cloneNode(false));
});

</script>
</head>
<body>
<p id="prueba">esto es un ejemplo para ver que sucede</p>
<div id="nuevo">
</div>
</body>
```

El resultado es el siguiente:

```
<body>
<p id="prueba">esto es un ejemplo para ver que sucede esto es un texto </p>
<div id="nuevo">
</div>

<p>esto es un texto </p>
<p></p>
</body>
```

Si intentamos colocar el mismo elemento en diferentes lugares de la página y no clonamos el elemento este se colocará en la última asignación.

Código

```
window.addEventListener("load",function(){
  var x = document.createElement("p");
  var y = document.createTextNode(" esto es un texto ");
    x.appendChild(y);
  document.body.appendChild(x);
  document.getElementById("nuevo").appendChild(x);
});
```

Resultado

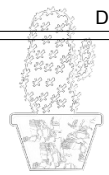
```
<body>
<p id="prueba">esto es un ejemplo para ver que sucede</p>
<div id="nuevo">
<p> esto es un texto </p></div>

</body>
```

Código

```
window.addEventListener("load",function(){
  var x = document.createElement("p");
  var y = document.createTextNode(" esto es un texto ");
```

```
    x.appendChild(y);
  document.body.appendChild(x);
  document.getElementById("nuevo").appendChild(x.cloneNode(true));
});
```



Resultado

```
<body>
<p id="prueba">esto es un ejemplo para ver que sucede</p>
<div id="nuevo">
<p> esto es un texto </p></div>

<p> esto es un texto </p>
</body>
```

Creados los nuevos nodos ahora tenemos que asignar los atributos que deseamos.

Para asignar un valor a un atributo

```
document.getElementById("capa").setAttribute("style","background-color:#951");
```

Para acceder al contenido de un atributo

```
document.getElementById("capa").getAttribute("style");
```

Para crea los atributos utilizamos el método *createAttribute(“atributo”)*.

var att=document.createAttribute(“style”);

att.value=”background-color:#951”;

Un vez hemos creado el atributo y le hemos asignado el valor deseado lo asignamos al elemento.

document.getElementById(“capa”).setAttributeNode(att);

Como vimos en temas anteriores también podemos hacer:

document.getElementById(“capa”).style.backgroundColor=”#999”;

De este modo sólo podemos acceder a las propiedades definidas en el atributo style de la etiqueta.

Ahora queremos acceder a las propiedades de estilo definidas en un fichero .css o en la etiqueta style.

Tenemos que saber que ***document.defaultView*** devuelve el objeto *window* asociado a document, por tanto, es lo mismo poner *document.defaultView* que *window*.

getComputedStyle() devuelve el estilo del elemento. Los estilos representan los valores finales de las propiedades CSS del elemento. Devuelve un objeto del tipo CSSStyleDeclaration.

Este objeto contiene propiedades numéricas que nos permiten acceder a los nombres de todas las propiedades css de este objeto y propiedades con el nombre de las propiedades css para acceder a los valores de las mismas

Probar esta línea en vuestro depurador del navegador y todo estará más claro

```
window.getComputedStyle(document.getElementsByTagName("body")[0],null)
```

Ahora para acceder al valor de la propiedad podemos acceder como si fuera un array asociativo

```
variable = window.getComputedStyle(jsparrafo1,null)["background-color"]
```

o utilizando el método **getPropertyValue()**

```
variable = window.getComputedStyle(jsparrafo1,null).getPropertyValue("font-size")
```

Escribir texto entre dos etiquetas

Primero veremos como facilitar la acción de escribir texto entre dos etiquetas. Es decir, ya hemos explicamos el método ***createTextNode***, ahora vamos a ver como introducir ese texto de un modo más sencillo.

Utilizaremos dos propiedades ***innerHTML*** y ***innerText***. (innerText no es accesible desde firefox).

innerHTML → devuelve un texto del tipo text/html. Me devuelve el contenido de ese elemento con los elementos hijos. Nos devuelve el contenido del nodo con las etiquetas de los hijos.

En google, realiza la búsqueda “Diferencia entre innerText y innerHTML”, entra en el depurador del navegador y en la consola teclea el siguiente código

```
document.getElementsByClassName("r")[0].innerHTML
```

devolverá algo parecido a esto

```
<a href="http://stackoverflow.com/questions/24427621/innertext-vs-innerhtml-vs-label-vs-text-vs-textcontent-vs-outertext" onmousedown="return rwt(this,'','','','1','AFQjCNFnixTzJHNBi01Eo1ggKD1kR8ZSOw','','0CCMQFjAA','','',event)">javascript - innerText vs innerHtml vs label vs text vs ...</a>
```

innerText → devuelve un texto del tipo text/plain. Nos devuelve el contenido sin las etiquetas de los hijos. Nos devuelve un texto formateado.

Repitamos el ejemplo anterior, pero, ahora pulsamos lo siguientes

```
document.getElementsByClassName("r")[0].innerText
```

devolverá algo parecido a esto

"Difference between innerText and innerHTML in javascript ..."

Inserción de etiquetas en la posición deseada

Tenemos que saber, que por ahora, sólo disponemos de la función insertBefore(). Digo por ahora, ya que en DOM **no** existe el método insertAfter() ni nada que se parezca.

insertBefore(obj_añadir,obj_ref) → Partimos desde el contenedor de los objetos. Elegimos el objeto delante del cual queremos insertar el nuevo objeto.

Veamos el ejemplo. En este ejemplo movemos el último elemento a la primera posición.

```
<html>
<head>
```

```
</head>
<body>
<div id="capa">
<p>parrafo01</p>
<p>parrafo02</p>
<p>parrafo03</p>
<p>parrafo04</p>
<p>parrafo05</p>
<p>parrafo06</p>
<p>parrafo07</p>
</div>
<div id="botones">
<input type="button" value="pulsar" id="boton"/>
<script>
boton.addEventListener("mouseup",function(e){
     var capa=document.getElementById("capa");
     var parrafos=capa.getElementsByTagName("p");
     var ultnodo=parrafos[parrafos.length-1]
     capa.insertBefore(ultnodo,parrafos[0]);
   });

</script>
</body>
</html>
```

Como hemos vista anteriormente existen los métodos **append** y **prepend**.

append(elemento) → Tomando como punto de partida el elemento contenedor, añade *elemento* como último hijo del contenedor.

```
<html>
<head>
</head>
<body>
<div id="capa">
<p>parrafo01</p>
<p>parrafo02</p>
<p>parrafo03</p>
<p>parrafo04</p>
<p>parrafo05</p>
<p>parrafo06</p>
<p>parrafo07</p>
</div>
<div id="botones">
<input type="button" value="pulsar" id="boton"/>
<script>
   boton.addEventListener("mouseup",function(e){
     var capa=document.getElementById("capa");
     var parrafos=capa.getElementsByTagName("p");
     var ultnodo=parrafos[0]
     capa.append(ultnodo);
   });

</script>
</body>
</html>
```

prepend → Tomando como punto de partida el elemento contenedor, añade *elemento* como primer hijo del contenedor.

```
<html>
```

```
<head>

</head>
<body>
<div id="capa">
<p>parrafo01</p>
<p>parrafo02</p>
<p>parrafo03</p>
<p>parrafo04</p>
<p>parrafo05</p>
<p>parrafo06</p>
<p>parrafo07</p>
</div>
<div id="botones">
<input type="button" value="pulsar" id="boton"/>
<script>
boton.addEventListener("mouseup",function(e){
    var capa=document.getElementById("capa");
     var parrafos=capa.getElementsByTagName("p");
     var ultnodo=parrafos[parrafos.length-1]
     capa.prepend(ultnodo);
   });

</script>
</body>
</html>
```

Ahora vamos a ver un par de propiedades que nos permitirán desplazarnos por el árbol de elementos de la página web.

nextSibling // nextElementSibling → Estas dos propiedades nos permite acceder al siguiente nodo hermano del árbol o al siguiente nodo que sea un elemento.

```
<div id="capa">
<p>parrafo01</p><p>parrafo02</p><p>parrafo03</p><p>parrafo04</p><p>parrafo05</
p><p>parrafo06</p><p>parrafo07</p>
</div>

parr=document.getElementById("capa").firstElementChild;
parr=parr.nextSibling;// <p>parrafo02</p>
parr=parr.nextSibling; // <p>parrafo03</p>
```

```
<div id="capa">
<p>parrafo01</p>
<p>parrafo02</p>
<p>parrafo03</p>
<p>parrafo04</p>
<p>parrafo05</p>
<p>parrafo06</p>
<p>parrafo07</p>
</div>

parr=document.getElementById("capa").firstElementChild;
parr=parr.nextSibling; // #text
parr=parr.nextSibling; // <p>parrafo02</p>
```

Esta diferencia es debido a que el retorno de carro es un nodo en el árbol

Tanto uno como otro al llegar al último elemento y no poder acceder a otro nodo devuelve null.

previousSibling // previousElementSibling → Nos permite desplazarnos por los nodos hermanos. Entre nodos hermanos o entre nodos elementos hermano.

lastChild → Devuelve el último hijo de un elemento.

lastElementChild → Devuelve el último elemento hijo de un elemento.

firstChild → Devuelve el primer hijo de un elemento.

firstElementChild → Devuelve el primer elemento hijo de un elemento.

hasAttribute() → devuelve verdadero si el elemento tiene algún atributo y falso si dicho elemento no tiene elementos.

```
elemento.hasAttribute();  // true o false
```

hasAttribute(“atributo”) → devuelve verdadero si el elemento tiene el elemento “atributo” y false si no lo tiene dicho “atributo”.

Para terminar veremos otra propiedad que se utiliza para asignar un valor a una propiedad de CSS.

setProperty(“propiedad”,”valor”, prioridad) → “propiedad” es el nombre de la propiedad en CSS, “valor”, uno de los valores que se pueden asignar a dicha propiedad, “prioridad” es para asignarle la palabra reservda “important” en la declaración de estilos.

```
document.getElementById("pp").style.setProperty("background-color","green")
```

o

```
document.getElementById("pp").style.backgroundColor="green";
```

getPropertyValue(“propiedad”) → Devuelve el valor de la propiedad que recibe como parámetro.

```
document.getElementById("pp").style.getPropertyValue("background-color");
// green
```

getPropertyPriority(“propiedad”) → Devuelve si la propiedad que recibe como parámetro tiene la propiedad de prioritario (!important).

removeProperty(“propiedad”) → Elimina la propiedad css que recibe como parámetro.

```
document.getElementById("pp").style.removeProperty("background-color");
```