# DOCUMENT OBJECT MODEL ( DOM )

## 1. INTRODUCCIÓN

El llamado DOM (Document Object Model) es un modelo que permite tratar un documento Web XHTML como si fuera XML, navegando por los nodos existentes que forman la página, pudiendo manipular sus atributos e incluso crear nuevos elementos.

Usando Javascript para navegar en el DOM podemos acceder a todos los elementos XHTML de una página. Esto nos permite cambiar dinámicamente el aspecto de nuestras páginas Web.

En este tema vamos a estudiar las principales funciones de Javascript para modificar el DOM.

## 2. MODIFICANDO ATRIBUTOS

Cuando obtengamos algún elemento con las funciones que estudiaremos más adelante, podemos manipular los atributos de dicho elemento de la siguiente forma:

```
let elemento=document.getElementById("miElemento");
elemento.innerHTML="El html interno a cambiar de ese elemento";
```

Los atributos disponibles a modificar pueden depender de cada elemento.

## 3. FUNCIONES DE JAVASCRIPT PARA LOCALIZAR ELEMENTOS EN EL DOM

Las funciones aquí estudiadas normalmente se usan sobre el elemento "document", ya que así se aplican a todo el documento.

Aun así, pueden usarse en cualquier nodo XHTML, entonces la búsqueda se realizaría no en todo en el documento, sino en el sub-árbol formado por el elemento en sí y sus hijos.

### 3.1 GetElementById (identificador)

Esta función devuelve un elemento DOM del subárbol cuya identificador sea el indicado en la cadena "identificador".

http://www.w3schools.com/jsref/met_document_getelementbyid.asp

Ejemplo:

```
let myDiv = document.getElementById("miDiv");
console.log("El html de miDiv es "+myDiv.innerHTML);
```

### 3.2 GetElementsByTagName (etiqueta)

Esta función devuelve una array con todos los elementos DOM del subárbol cuya etiqueta XHTML sea la indicada en la cadena "etiqueta".

http://www.w3schools.com/jsref/met_document_getelementsbytagname.asp

Ejemplo:

```
let myDiv = document.getElementById("miDiv")
let losP = myDiv.getElementsByTagName("p");
let num = losP.length;
console.log("Hay " + num + " <p> elementos en el elemento miDiv");
console.log("En el primer P el HTML asociado es
"+losP[0].innerHTML);
```

### 3.3 GetElementsByName (nombre)

Esta función devuelve una array con todos los elementos DOM del subárbol cuya atributo name sea el indicado en la cadena “nombre”.

http://www.w3schools.com/jsref/met_doc_getelementsbyname.asp

**Ejemplo**:

```
let elementos = document.getElementsByName("name");
let i;
// Todos los textbox que tengan de name alumnos, los marcamos
for (i = 0; i < elementos.length; i++) {
     if (elementos[i].type === "checkbox") {
           elementos[i].checked = true;
     }
}
```

## 4. FUNCIONES PARA CREAR/ELIMINAR NODOS

En esta parte veremos las funciones básicas para crear y eliminar nodos XHTML.

### 4.1 RemoveChild (nodo)

Esta función se aplica a un nodo padre. La función recibe un nodo hijo suyo y lo borra. Es útil usarlo con el atributo "parentnode", que devuelve el nodo padre del elemento que estamos manejando.

http://www.w3schools.com/jsref/met_node_removechild.asp

Ejemplo:

```
let parrafo=document.getElementById("miParrafo");
// Obtiene la referencia del padre, y al padre le aplica la función removeChild
parrafo.parentnode.removeChild(parrafo);
```

### 4.2 AppendChild (nodo)

Esta función se aplica a un nodo padre. La función recibe un nodo y lo incluye como nodo hijo del padre. Se puede combinar con funciones como "createElement", que permiten crear elementos XHTML.

http://www.w3schools.com/jsref/met_node_appendchild.asp

Ejemplo:

```
// Creo un nodo de tipo LI
let nuevoNodo = document.createElement("LI");
// Al nodo LI le asocio un texto (también podría asociarse XHTML con innerHTML)
let nodoTexto = document.createTextNode("Agua");
nuevoNodo.appendChild(nodoTexto);
// A miLista, lista ya existente, le añado el elemento creado
document.getElementById("miLista").appendChild(nuevoNodo);
```