JavaScript 04 – DOM - Document Object Model

Introducción

La mayoría de las veces que se programa con Javascript es para que se ejecute en una página web mostrada por el navegador. En este contexto, y para facilitar el desarrollo de la aplicación, tenemos acceso a ciertos objetos que nos permiten interactuar con la página (DOM) y con el navegador (Browser Object Model, BOM).

El **DOM** es una estructura en árbol que representa todos los elementos HTML de la página y sus atributos.

Todo lo que contiene la página se representa como nodos del árbol y mediante el DOM podemos acceder a cada nodo, modificarlo, eliminarlo o añadir nuevos nodos de forma que cambiamos dinámicamente la página mostrada al usuario.

La raíz del árbol DOM es **document** y de este nodo cuelgan el resto de elementos HTML.

Cada elemento constituye su propio nodo y tiene subnodos con sus *atributos*, *estilos* y otros elementos HTML que contiene.

Nota: Un **elemento** HTMl consiste en:

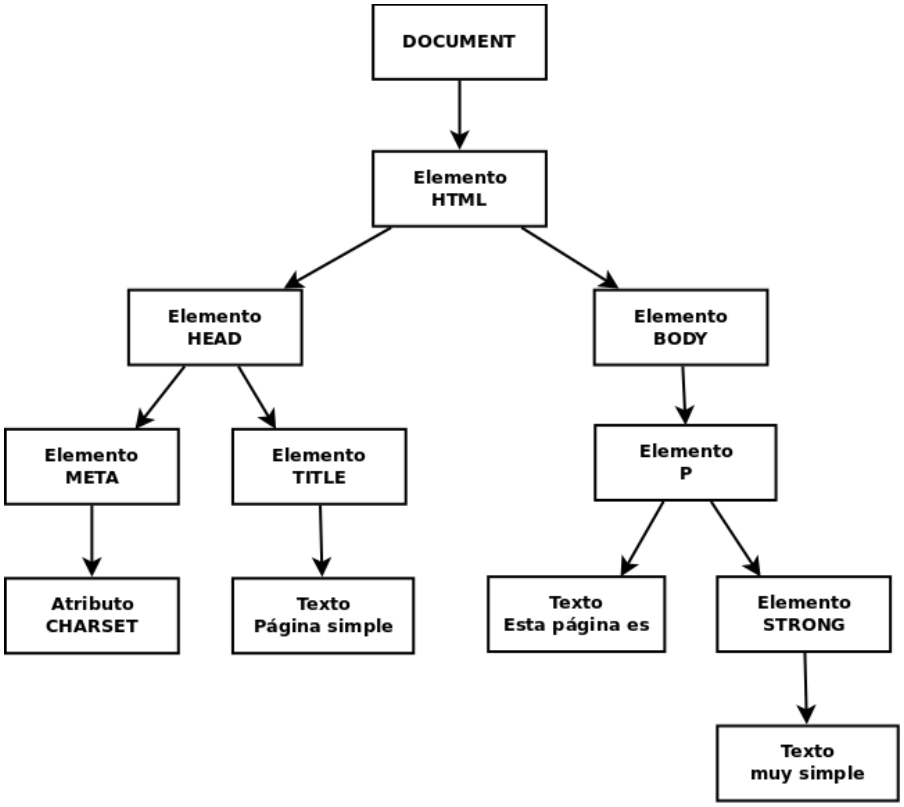
- Una **etiqueta inicial**. Opcionalmente contiene pares "atributo: valor".
- **Contenido** del elemento (no siempre aparece).
- **Etiqueta final** o de cierre (no siempre aparece).

Es frecuente que alguien se refiera a un 'elemento HTML' como 'etiqueta HTML'. Por lo que debes interpretar el significado de **etiqueta** en cada momento según el contexto.

Por ejemplo, la página HTML:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Página simple</title>
</head>
<body>
  <p>Esta página es <strong>muy simple</strong></p>
</body>
</html>
```

se convierte en el siguiente árbol DOM:



Cada elemento HTML suele originar 2 nodos:

- **Element**: correspondiente al elemento.
- **Text**: correspondiente a su contenido (lo que hay entre la etiqueta inicial y su par de cierre)

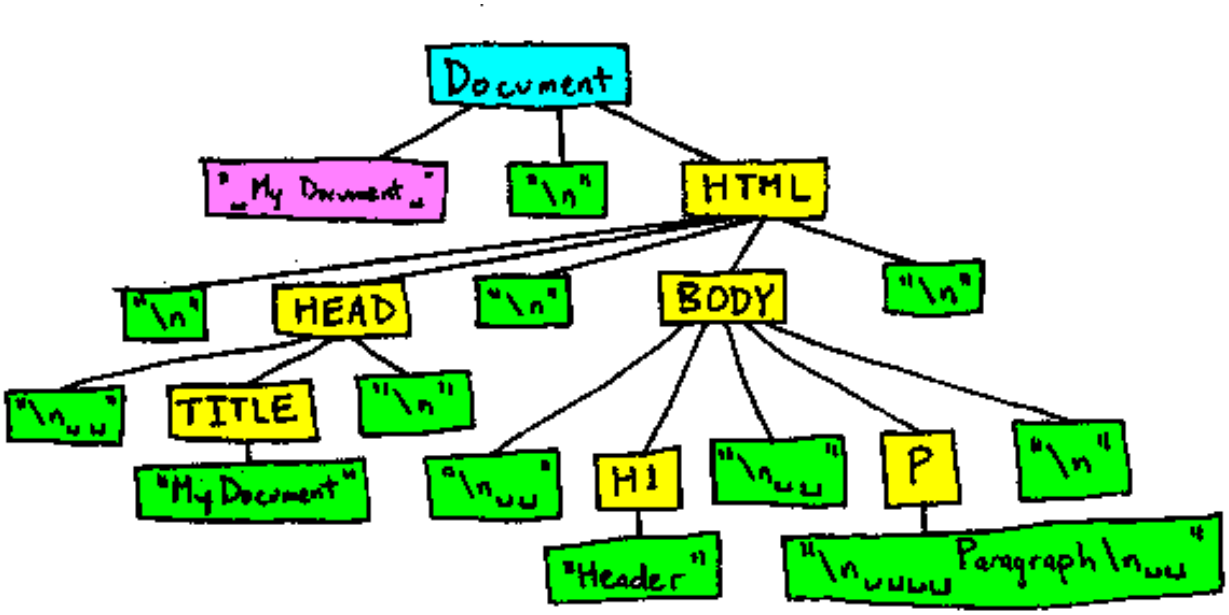
Cada nodo es un objeto con sus propiedades y métodos.

El ejemplo anterior está simplificado porque sólo aparecen los nodos de tipo ***elemento,*** pero en realidad también generan nodos los saltos de línea, tabuladores, espacios, comentarios, etc.

En el siguiente ejemplo podemos ver TODOS los nodos que realmente se generan. La página:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Document</title>
</head>
<body>
  <h1>Header</h1>
  <p>
    Paragraph
  </p>
</body>
</html>
```

se convierte en el siguiente árbol DOM:



Acceso a los nodos

Los principales métodos para acceder a los diferentes nodos son:

getElementById(id)

.getElementById(id): devuelve el nodo con la *id* pasada. Ej.:

```
let nodo = document.getElementById('main');    // nodo contendrá el nodo cuya id es _main_
```

Cuidado de no confundir con el método **.getElementsByName(*nombre*)**, que devuelve los elementos con el atributo *name* especificado.

getElementsByClassName(clase)

.getElementsByClassName(clase): devuelve una colección (similar a un array) con todos los nodos de la *clase* indicada. Ej.:

```
let nodos = document.getElementsByClassName('error');    // nodos contendrá todos los nodos cuya clase es _error_
```

NOTA: las colecciones son similares a arrays (se accede a sus elementos con *[indice]*) pero no se les pueden aplicar los métodos *filter*, *map*, … a menos que se conviertan a arrays con *Array.from()*

getElementsByTagName(elemento)

.getElementsByTagName(elemento): devuelve una colección con todos los nodos del tipo *elemento* HTML indicado. Ej.:

```
let nodos = document.getElementsByTagName('p');    // nodos contendrá todos los nodos de tipo _<p>_
```

querySelector(selector)

.querySelector(selector): devuelve el primer nodo seleccionado por el *selector* CSS indicado. Ej.:

```
let nodo = document.querySelector('p.error');    // nodo contendrá el primer párrafo de clase _error_
```

querySelectorAll(selector)

.querySelectorAll(selector): devuelve una NodeList con todos los nodos seleccionados por el *selector* CSS indicado. Ej.:

```
let nodos = document.querySelectorAll('p.error');    // nodos contendrá todos los párrafos de clase _error_

let nodo = document.querySelectorAll('#text');    // nodo contendrá el elemento con ID=text
```

Más información en: https://developer.mozilla.org/es/docs/Web/API/Document/querySelectorAll

NOTA: Los objetos **`NodeList`** son colecciones de nodos como los devueltos por propiedades como `Node.childNodes` y el método `document.querySelectorAll()`..

NOTA: al aplicar estos métodos sobre *document* se seleccionarán sobre la página (objeto document). Pero podrían también aplicarse a cualquier nodo, en ese caso la búsqueda se realizaría sólo entre los descendientes de dicho nodo.

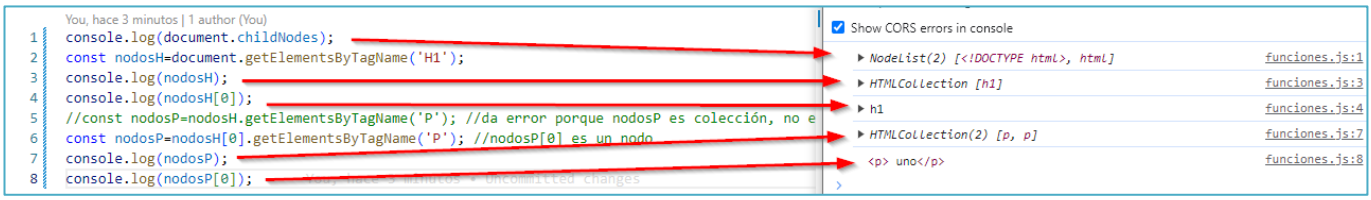
Ejemplo:

Partiendo de este .html:

```
<body>
    <h1>El DOM
        <p> uno</p>
        <p> dos</p>
    </h1>
    <script src="js/funciones.js"></script>
</body>
```

Aplicamos el siguiente código:

```
console.log(document.childNodes);
const nodosH=document.getElementsByTagName('H1');
console.log(nodosH);
console.log(nodosH[0]);
//const nodosP=nodosH.getElementsByTagName('P'); //da error porque nodosP es colección, no es nodo
const nodosP=nodosH[0].getElementsByTagName('P'); //nodosP[0] es un nodo
console.log(nodosP);
console.log(nodosP[0]);
```



Atajos

También tenemos 'atajos' para obtener algunos elementos comunes:

- `document.documentElement`: devuelve el nodo del elemento *<html>*
- `document.head`: devuelve el nodo del elemento *<head>*
- `document.body`: devuelve el nodo del elemento *<body>*
- `document.title`: devuelve el nodo del elemento *<title>*
- `document.link`: devuelve una colección con todos los hiperenlaces del documento
- `document.anchor`: devuelve una colección con todas las anclas del documento
- `document.forms`: devuelve una colección con todos los formularios del documento
- `document.images`: devuelve una colección con todas las imágenes del documento
- `document.scripts`: devuelve una colección con todos los scripts del documento

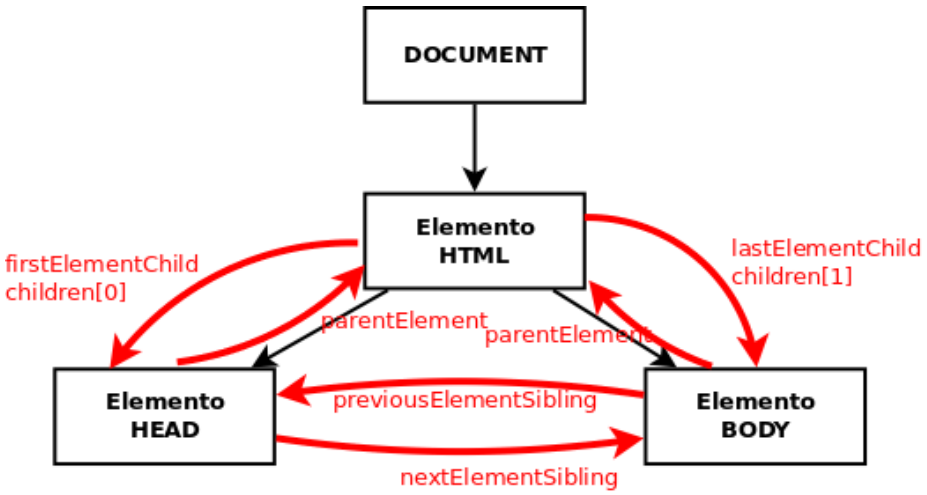
EJERCICIO: Para hacer los ejercicios de este tema descárgate esta página de ejemplo y ábrela en tu navegador. Obtén por consola, al menos de 2 formas diferentes lo que se pide:

1. El elemento con id 'input2'
2. La colección de párrafos
3. Lo mismo pero sólo de los párrafos que hay dentro del div 'lipsum'
4. El formulario (ojo, no la colección con el formulario sino sólo el formulario)

5. Todos los inputs
6. Sólo los inputs con nombre 'sexo'
7. Los items de lista de la clase 'important' (sólo los LI)

Acceso a nodos a partir de otros

En muchas ocasiones queremos acceder a cierto nodo a partir de uno dado. Para ello tenemos los siguientes métodos que se aplican sobre un elemento del árbol DOM:



- `elemento.parentElement`: devuelve el elemento padre de *elemento*
- `elemento.children`: devuelve la colección con todos los elementos hijo de *elemento* (sólo elementos HTML, no comentarios ni nodos de tipo texto)
- `elemento.childNodes`: devuelve la colección con todos los hijos de *elemento*, incluyendo comentarios y nodos de tipo texto por lo que no suele utilizarse
- `elemento.firstElementChild`: devuelve el elemento HTML que es el primer hijo de *elemento*
- `elemento.firstChild`: devuelve el nodo que es el primer hijo de *elemento* (incluyendo nodos de tipo texto o comentarios)
- `elemento.lastElementChild`, `elemento.lastChild`: igual pero con el último hijo
- `elemento.nextElementSibling`: devuelve el elemento HTML que es el siguiente hermano de *elemento*
- `elemento.nextSibling`: devuelve el nodo que es el siguiente hermano de *elemento* (incluyendo nodos de tipo texto o comentarios)
- `elemento.previousElementSibling`, `elemento.previousSibling`: igual pero con el hermano anterior
- `elemento.hasChildNodes`: indica si *elemento* tiene o no nodos hijos
- `elemento.childElementCount`: devuelve el nº de nodos hijo de *elemento*

IMPORTANTE: a menos que interesen comentarios, saltos de página, etc …, **siempre** hay que usar los métodos que sólo devuelven elementos HTML, no todos los nodos.

EJERCICIO: Siguiendo con la página de ejemplo obtén desde la consola, al menos de 2 formas diferentes:

1. El primér párrafo que hay dentro del div 'lipsum'
2. El segundo párrafo de 'lipsum'
3. El último item de la lista
4. El elemento *label* de 'Escoge sexo'

Propiedades de un nodo

Las principales propiedades de un nodo son:

innerHTML

`elemento.innerHTML`: todo lo que hay entre la etiqueta que abre *elemento* y la que lo cierra, incluyendo otras etiquetas HTML. Por ejemplo, si *elemento* es el nodo:

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

```
let contenido = elemento.innerHTML;  // contenido='Esta página es <strong>muy simple</strong>'
```

textContent

`elemento.textContent`: todo lo que hay entre la etiqueta que abre *elemento* y la que lo cierra, pero ignorando otras etiquetas HTML. Siguiendo con el ejemplo anterior:

```
let contenido = elemento.textContent;   // contenido='Esta página es muy simple'
```

value

`elemento.value`: devuelve la propiedad 'value' de un <input> (en el caso de un <input> de tipo text devuelve lo que hay escrito en él).

Como los <inputs> no tienen etiqueta de cierre (</input>) no podemos usar *.innerHTML* ni *.textContent*.
Por ejemplo:
si *elem1* es el nodo `<input name="nombre">` y *elem2* es el nodo `<input type="radio" value="H"> Hombre`

```
let cont1 = elem1.value;   // cont1 valdría lo que haya escrito en el <input> en ese momento
let cont2 = elem2.value;   // cont2="H"
```

Otras propiedades:

- `elemento.innerText`: igual que *textContent*
- `elemento.focus`: pone (sitúa) el foco en *elemento* (para inputs, etc). Para quitarle el foco `elemento.blur`
- `elemento.clientHeight` / `elemento.clientWidth`: devuelve el alto / ancho visible del *elemento*
- `elemento.offsetHeight` / `elemento.offsetWidth`: devuelve el alto / ancho total del *elemento*
- `elemento.clientLeft` / `elemento.clientTop`: devuelve la distancia de *elemento* al borde izquierdo / borde superior
- `elemento.offsetLeft` / `elemento.offsetTop`: devuelve los *píxels* que hemos desplazado *elemento* a la izquierda / abajo

EJERCICIO: Obtén desde la consola, al menos de 2 formas:

1. El innerHTML de la etiqueta de 'Escoge sexo'
2. El textContent de esa etiqueta
3. El valor del primer input de sexo

4. El valor del sexo que esté seleccionado (difícil, búscalo por Internet)

Manipular el árbol DOM

Vamos a ver qué métodos nos permiten cambiar el árbol DOM, y por tanto modificar la página:

createElement

`document.createElement('etiqueta')`: crea un nuevo elemento HTML con la etiqueta indicada, pero aún no se añade a la página. Ej.:

```
let nuevoLi = document.createElement('li');
```

createTextNode

`document.createTextNode('texto')`: crea un nuevo nodo de texto con el texto indicado, que luego tendremos que añadir a un nodo HTML. Ej.:

```
let textoLi = document.createTextNode('Nuevo elemento de lista');
```

appendChild

`elemento.appendChild(nuevoNodo)`: añade *nuevoNodo* como último hijo de *elemento*. Ahora ya se ha añadido a la página. Ej.:

```
nuevoLi.appendChild(textoLi);     // añade el texto creado al elemento LI creado
let miPrimeraLista = document.getElementsByTagName('ul')[0];  // selecciona el 1º UL de la página
miPrimeraLista.appendChild(nuevoLi);    // añade LI como último hijo de UL, es decir al final de la lista
```

insertBefore

`elemento.insertBefore(nuevoNodo, nodo)`: añade *nuevoNodo* como hijo de *elemento* antes del hijo *nodo*. Ej.:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];   // selecciona el 1º UL de la página
let primerElementoDeLista = miPrimeraLista.getElementsByTagName('li')[0]; // selecciona el 1º LI de miPrimeraLista
miPrimeraLista.insertBefore(nuevoLi, primerElementoDeLista);  // añade LI al principio de la lista
```

removeChild

`elemento.removeChild(nodo)`: borra *nodo* de *elemento* y por tanto se elimina de la página. Ej.:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];  // selecciona el 1º UL de la página
let primerElementoDeLista = miPrimeraLista.getElementsByTagName('li')[0];  // selecciona el 1º LI de miPrimeraLista
```

```
miPrimeraLista.removeChild(primerElementoDeLista);    // borra el primer elemento de la lista
// También podríamos haberlo borrado sin tener el padre con:
primerElementoDeLista.parentElement.removeChild(primerElementoDeLista);
```

replaceChild

`elemento.replaceChild(nuevoNodo, viejoNodo)`: reemplaza *viejoNodo* con *nuevoNodo* como hijo de *elemento*. Ej.:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];  // selecciona el 1º UL de la página
let primerElementoDeLista = miPrimeraLista.getElementsByTagName('li')[0];  // selecciona el 1º LI de miPrimeraLista
miPrimeraLista.replaceChild(nuevoLi, primerElementoDeLista);    // reemplaza el 1º elemento de la lista con nuevoLi
```

cloneNode

`elementoAClonar`.cloneNode(boolean): devuelve un clon de elementoAClonar o de elementoAClonar con todos sus descendientes según le pasemos como parámetro false o true. Luego podremos insertarlo donde queramos.

OJO: Si añado con el método `appendChild` un nodo que estaba en otro sitio **se elimina de donde estaba** para añadirse a su nueva posición. Si quiero que esté en los 2 sitios deberé clonar el nodo y luego añadir el clon y no el nodo original.

Ejemplo de creación de nuevos nodos:

Tenemos un código HTML con un DIV que contiene 3 párrafos y vamos a añadir un nuevo párrafo al final del div con el texto 'Párrafo añadido al final' y otro que sea el 2º del div con el texto 'Este es el **nuevo** segundo párrafo':

```
<div id="articulos">
        <p>Este es el primer párrafo que tiene <strong>algo en negrita</strong>.</p>
        <p>Este era el segundo párrafo pero será desplazado hacia abajo.</p>
        <p>Y este es el último párrafo pero luego añadiremos otro después</p>
</div>
```

```
let miDiv=document.getElementById('articulos');

miDiv.innerHTML+='<p>Párrafo añadido al final</p>';

let nuevoSegundoParrafo=document.createElement('p');
nuevoSegundoParrafo.innerHTML='Este es el <strong>nuevo</strong> segundo párrafo';

let segundoParrafo=miDiv.children[1];
miDiv.insertBefore(nuevoSegundoParrafo, segundoParrafo);
```

Resultado:

Este es el primer párrafo que tiene **algo en negrita**.

Este es el **nuevo** segundo párrafo

Este era el segundo párrafo pero será desplazado hacia abajo.

Y este es el último párrafo pero luego añadiremos otro después

Párrafo añadido al final

Si utilizamos la propiedad **innerHTML** el código a usar es mucho más simple:

```
let ultimoParrafo = document.createElement('p');
ultimoParrafo.innerHTML = 'Párrafo añadido al final';
miDiv.appendChild(ultimoParrafo);
```

OJO: La forma de añadir el último párrafo (línea #3: `miDiv.innerHTML+='<p>Párrafo añadido al final</p>';`) aunque es válida no es muy eficiente ya que obliga al navegador a volver a pintar TODO el contenido de miDIV. La forma correcta de hacerlo sería:

```
let ultimoParrafo = document.createElement('p');
ultimoParrafo.innerHTML = 'Párrafo añadido al final';
miDiv.appendChild(ultimoParrafo);
```

Así sólo debe repintar el párrafo añadido, conservando todo lo demás que tenga *miDiv*.

Podemos ver más ejemplos de creación y eliminación de nodos en W3Schools.

EJERCICIO: Añade a la página:

1. Un nuevo párrafo al final del DIV *'lipsum'* con el texto "Nuevo párrafo **añadido** por javascript" (fíjate que una palabra está en negrita)
2. Un nuevo elemento al formulario tras el *'Dato 1'* con la etiqueta *'Dato 1 bis'* y el INPUT con id *'input1bis'* que al cargar la página tendrá escrito "Hola"

Modificar el DOM con ChildNode

Childnode es una interfaz que permite manipular del DOM de forma más sencilla pero no está soportada en los navegadores Safari de IOS. Incluye los métodos:

- `elemento.before(nuevoNodo)`: añade el *nuevoNodo* pasado antes del nodo *elemento*
- `elemento.after(nuevoNodo)`: añade el *nuevoNodo* pasado después del nodo *elemento*
- `elemento.replaceWith(nuevoNodo)`: reemplaza el nodo *elemento* con el *nuevoNodo* pasado
- `elemento.remove()`: elimina el nodo *elemento*

Atributos de los nodos

Podemos ver y modificar los valores de los atributos de cada elemento HTML y también añadir o eliminar atributos:

- `elemento.attributes`: devuelve un array con todos los atributos de *elemento*

- `elemento.hasAttribute('nombreAtributo')`: indica si *elemento* tiene o no definido el atributo *nombreAtributo*
- `elemento.getAttribute('nombreAtributo')`: devuelve el valor del atributo *nombreAtributo* de *elemento*. Para muchos elementos este valor puede directamente con `elemento.atributo`.
- `elemento.setAttribute('nombreAtributo', 'valor')`: establece *valor* como nuevo valor del atributo *nombreAtributo* de *elemento*. También puede cambiarse el valor directamente con `elemento.atributo=valor`.
- `elemento.removeAttribute('nombreAtributo')`: elimina el atributo *nombreAtributo* de *elemento*

A algunos atributos comunes como `id`, `title` o `className` (para el atributo **class**) se puede acceder y cambiar como si fueran una propiedad del elemento (`elemento.atributo`). Ejemplos:

```
let miPrimeraLista = document.getElementsByTagName('ul')[0];  // selecciona el 1º UL de la página
miPrimeraLista.id = 'primera-lista';
// es equivalente ha hacer:
miPrimeraLista.setAttribute('id', 'primera-lista');
```

Estilos de los nodos

Los estilos están accesibles como el atributo **style**. Cualquier estilo es una propiedad de dicho atributo, pero con la sintaxis *camelCase* en vez de *kebab-case*.

Por ejemplo, para cambiar el color de fondo (propiedad `background-color`) y ponerle el color *rojo* al elemento *miPrimeraLista* haremos:

```
miPrimeraLista.style.backgroundColor = 'red';
```

De todas formas, normalmente **NO CAMBIAREMOS ESTILOS** a los elementos, sino que les pondremos o quitaremos clases que harán que se le apliquen o no los estilos definidos para ellas en el CSS.

Atributos de clase

Ya sabemos que el aspecto de la página debe configurarse en el CSS por lo que no debemos aplicar atributos *style* al HTML. En lugar de ello les ponemos clases a los elementos que harán que se les aplique el estilo definido para dicha clase.

Como es algo muy común en lugar de utilizar las instrucciones de `elemento.setAttribute('className', 'destacado')` o directamente `elemento.className='destacado'` podemos usar la propiedad ***classList*** que devuelve la colección de todas las clases que tiene el elemento.

Por ejemplo, si *elemento* es `<p class="destacado direccion">...`:

```
let clases=elemento.classList;   // clases=['destacado', 'direccion'], OJO es una colección, no un Array
```

Además, dispone de los métodos:

add

.add(clase): añade al elemento la clase pasada (si ya la tiene no hace nada). Ej.:

```
elemento.classList.add('primero');   // ahora elemento será <p class="destacado
direccion primero">...
```

remove

.remove(clase): elimina del elemento la clase pasada (si no la tiene no hace nada). Ej.:

```
elemento.classList.remove('direccion');   // ahora elemento será <p class="destacado
primero">...
```

toogle

.toogle(clase): añade la clase pasada si no la tiene o la elimina si la tiene ya. Ej.:

```
elemento.classList.toogle('destacado');   // ahora elemento será <p class="primero">...
elemento.classList.toogle('direccion');   // ahora elemento será <p class="primero
direccion">...
```

contains

.contains(clase): dice si el elemento tiene o no la clase pasada. Ej.:

```
elemento.classList.contains('direccion');   // devuelve true
```

replace

.replace(oldClase, newClase): reemplaza del elemento una clase existente por una nueva. Ej.:

```
elemento.classList.replace('primero', 'ultimo');   // ahora elemento será <p
class="ultimo direccion">...
```

Ten en cuenta que NO todos los navegadores soportan *classList* por lo que si queremos añadir o quitar clases en navegadores que no lo soportan debemos hacerlo con los métodos estándar, por ejemplo para añadir la clase 'rojo':

```
let clases = elemento.className.split(" ");
if (clases.indexOf('rojo') == -1) {
  elemento.className += ' ' + 'rojo';
}
```