Introducción
En este momento acabamos de recibir mucha información referente al manejo de los elementos en un documento web y necesitamos asentarla.

La mejor forma de asentar contenidos es poniéndolos en práctica, y en este Lab vamos a hacer un ejercicio guiado donde utilizaremos todo lo visto hasta ahora relativo al DOM.

Resumen
En este punto sabemos que el DOM representa los documentos HTML como un árbol de Nodos de diferentes tipos que tienen propiedades, algunas son comunes a todos y otras son específicas de cada tipo de Nodo.

- nodeType: Indica el tipo de nodo que es:
  - Document: 9, Element: 1, Text: 3, Comment: 8
- nodeValue: Es el contenido textual de un nodo de texto o comentario.
- nodeName: Es el tag name

Conocemos el API para seleccionar elementos del árbol.

- document.getElementById('id'): Selecciona el único nodo que tiene el id especificado.
- document.getElementsByName('name'): Selecciona todos los nodos cuyo atributo name tenga el valor especificado.
- document.getElementsByTagName('tag'): Selecciona todos los nodos cuyo tag sea el especificado
- document.getElementsByClassName('class'): Selecciona todos los nodos cuyo atributo class contenga el valor especificado.
- document.querySelector('sel'): Selecciona el primer nodo que cumpla con el selector CSS proporcionado.
- document.querySelectorAll('sel'): Selecciona todos los nodos que cumplan con el selector CSS proporcionado.

Conocemos el API para movernos por el árbol de Nodos a partir de un Nodo determinado. Recordamos que existen dos APIs, uno muy sensible a los espacios y otro que se centra en los elementos. El API sensible a los espacios es el estándar, pero debido a la comodidad que supone el uso del otro y a que su uso está soportado por la mayoría de los navegadores, será éste el que usemos.

- nodo.children: Nos devuelve un array con todos sus hijos
- nodo.firstElementChild: Nos devuelve el primero de sus hijos
- nodo.lastElementChild: Nos devuelve el último de sus hijos
- nodo.nextElementSibling: Nos devuelve el siguiente hermano.
- nodo.previousElementSibling: Nos devuelve el hermano anterior.
- nodo.childElementCount: Nos indica el número de hijos que contiene.

Conocemos el API para manejar los atributos de los elementos HTML, que simplemente son propiedades de los objetos Node, accesibles con notación de punto y de corchetes. La única particularidad que debemos recordar es que para eliminar atributos estándares HTML no podemos usar el operador JavaScript delete. Para este caso debemos usar la función nodo.removeAttribute('atributo').

Y finalmente conocemos el API para modificar la estructura del árbol creando, insertando, reemplazando y eliminando nodos.

- document.createElement('tipo'): Crea un nuevo nodo del tipo especificado
- elemento.appendChild(nodo): Añade el nodo especificado como el último de sus hijos.
- elemento.insertBefore(nodo_nuevo, nodo): Añade nodo_nuevo como hijo suyo, pero lo coloca delante de nodo.
- elemento.removeChild(nodo): Elimina nodo de sus hijos.
- elemento.replaceChild(nodo_nuevo, nodo): Reemplaza nodo por nodo_nuevo.

Y para terminar el repaso sólo queda nombrar el tipo especial DocumentFragment que representa un conjunto de nodos que podemos manejar como si de uno solo se tratase.

- document.createDocumentFragment(): Nos devuelve el nuevo fragmento y a partir de ese momento se maneja exactamente igual que si se tratase de un simple Nodo.

Ejercicio guiado

En este Lab vamos a hacer un ejercicio guiado que nos servirá de repaso y aplicación práctica de todo lo visto hasta ahora.

El objetivo del ejercicio es crear una utilidad JavaScript que, cuando la apliquemos a cualquier página web bien estructurada nos incluya, de forma dinámica, una tabla de contenidos de dicha página web.

Paso previo

Abrir y entender los documentos base. Los documentos base son dos archivos, un documento HTML que contiene una estructura de ejemplo de una página web y un documento JavaScript prácticamente vacío donde escribiremos el ejercicio.

Lo primero que debéis hacer es abrir y leer los documentos y aseguraros de entender todo.

Como aspectos importantes de estos documentos hay que fijarse en la declaración del body del documento HTML, que define una función que se debe invocar cuando el documento esté cargado y listo para su uso. Esta función está definida en el archivo js/toc.js que, como podéis comprobar, se incluye en la cabecera del documento HTML.

El otro aspecto importante es el contenido HTML. Está compuesto por headers de diferente importancia. HTML nos proporciona 6 tags para definir headers dependiendo de su importancia, siendo h1 el más importante y h6 el menos importante. El objetivo de los headers en HTML es servir como encabezados de las diferentes secciones y subsecciones en que se divide el contenido del documento.

Además del header, cada sección incluye un párrafo.

Si abrimos este documento HTML en nuestro navegador, podemos ver como el navegador, por defecto, aplica estilos de letra diferentes a cada header dependiendo de su importancia asociada.

Una vez que tenemos nuestro documento base ya podemos empezar a escribir el código de la función buildTOC.

Paso primero
La función buildTOC creará un nuevo elemento de tipo , que colocará al comienzo de la página y será dentro de él donde construya la tabla de contenidos.

Añadir este código al comienzo de la función buildTOC.

```
var toc = document.createElement('div');
document.body.insertBefore(toc, document.body.firstElementChild);
```

Este código crea un nuevo div y lo añade, como hijo del body, y además, por delante del primer hijo que tuviera. Con esto nos aseguramos que la tabla se situará al comienzo de la página.

A continuación debemos recopilar todos los elementos relevantes para nuestra tabla de contenidos, es decir, todos los elementos header. Esto podemos hacerlo de diferentes maneras, pero la más sencilla es como veremos a continuación.

```
var headings = document.querySelectorAll("h1, h2, h3, h4, h5, h6");
```

Esta sentencia retornará todos los elementos h1, h2, ... h6 presentes en el documento y ordenados en orden de aparición. A continuación recorreremos todos los elementos contenidos en headings para determinar su nivel de importancia y así saber el grado de indentación que tendrán.

```
for (var i = 0 ; i < headings.length ; ++i) {
        var heading = headings[i];
        var level = parseInt(heading.tagName.charAt(1));
        var tocEntry = document.createElement('p');
        var indentation = "";
        for (var j = 0 ; j < level ; ++j) {
            indentation += "+";
        }
        tocEntry.innerHTML = indentation + heading.innerHTML;
        toc.appendChild(tocEntry);
    }
```

Con este código, para cada elemento contenido en el array de headings, determinamos su nivel de importancia obteniendo el número que acompaña a la 'h' en el tag y lo guardamos en level.
Para calcular la indentación, simplemente añadimos tantos símbolos '+' como sea su nivel de importancia, y componemos una cadena con los símbolos '+' y el texto del heading.
Finalmente añadimos la cadena resultante como un párrafo nuevo de la tabla.

Si guardamos el archivo y recargamos la página, podremos ver como se añade la tabla dinámicamente al comienzo del documento. Es cierto que no es muy atractiva visualmente, pero lo importante es la funcionalidad.

Este Lab no finaliza cuando hayáis conseguido escribir el ejercicio y que funcione, sino cuando hayáis entendido minuciosamente todo el código y todo lo que hace para conseguir el resultado. Preguntad todo lo que sea necesario.