# Modelo de Objetos del Documento - DOM

1. Introducción

- Modelo De Objetos del Documento – Es un árbol de nodos
- Nodos de distintos tipos (12 tipos)
- Los más importantes son:
  - Nodo document
  - Nodo element (cualquier etiqueta html: h1, form, etc)
  - Nodo attr (atributos de las etiquetas)
  - Nodo texto

2. Nomenclatura de árboles

- Raíz - Nodo superior de la estructura. Equivale al objeto document.
- Nodo(s) hijo(s) - Los nodos que están anidados (o que cuelgan) de uno dado
- Nodo padre - Es el nodo del que cuelga o en el que está anidado.
- Nodo hermano - Son nodos que comparten nodo padre (sibling)
- Nodo hoja - Es un nodo que no tiene hijos. Por ejemplo, todos los nodos de texto son nodos hoja.

3. Buscar en el árbol

- Métodos específicos
  - `document.getElementById("id"); //  Por valor del atributo id`
  - `document.getElementsByName("txtNombre"); // Campos de formulario [ARRAY]`
  - `document.getElementsByTagName("h1"); // Por nombre de etiqueta [ARRAY]`
  - `document.getElementsByClassName("azul"); // Por nombre de clase [ARRAY]`

- Métodos que utilizan selector CSS
  - `document.querySelector("selector css")`
  - `document.querySelectorAll("selector css");`

- Ejemplos:

```
let oInput = document.querySelector("#txtNombre"); // por ID
let oH1 = document.querySelectorAll("h1"); // por etiqueta
let oSeleccionados = document.querySelectorAll(".seleccionado"); // por clase CSS
let oTxtNombre = document.querySelector("[name='txtNombre']"); // por valor atributo
```

4. Propiedades de los nodos

- nodeType - Número que identifica el tipo de nodo
- nodeValue - Valor asociado al nodo y depende del tipo de nodo.
- nodeName - Nombre del nodo, por ejemplo en elementos HTML es la etiqueta en mayusculas "DIV" o "UL". Si el nodo es de texto "#text".

5. Propiedades que devuelven cualquier tipo de Nodo

   - firstChild - Referencia al primer nodo hijo
   - lastChild - Referencia al último nodo hijo
   - parentNode - Referencia al nodo padre
   - nextSibling - Referencia al siguiente hermano
   - previousSibling - Referencia al hermano anterior
   - childNodes - Array con los nodos hijo

6. Propiedades que devuelven Nodos de tipo elemento (etiquetas HTML)

   - firstElementChild
   - lastElementChild
   - parentElement
   - nextElementSibling
   - previousElementSibling
   - children

7. Creación, eliminación e incorporación de nodos al documento

   - document.createElement(“ETIQUETA”) - Crea un nodo para la etiqueta indicada en forma de cadena de texto. El nodo no se añade al documento, se debe recoger en una variable para posteriormente agregarlo.

   - document.createTextNode(“cadena de texto”); - Crea un nodo de texto con el texto pasado como parámetro. El nodo se debe recoger en una variable para posteriormente agregarlo.

   - elemento.appendChild(oNodo) - Agrega un nodo como último hijo del elemento HTML.

   - elemento.insertBefore(oNodoNuevo, oNodoHijo) - Agrega el nodo nuevo como hermano y antes del nodo hijo que se indica en el segundo parámetro.

   - elemento.insertAdjacentElement("posicion",oNodo) - Inserta en la posición que indica el primer parámetro. Puede insertar como "hermano" del "elemento" o como "hijo".

```
"beforebegin"   // antes del <ul>, como nodo "hermano" previo
<ul>
"afterbegin" // después del <ul>, como primer nodo "hijo"
...
"beforeend" // antes del </ul>, como último nodo "hijo"
</ul>
"afterend"  // después del </ul>, como nodo "hermano" posterior
```

   - elemento.cloneNode(bClonarDescendientes) - Obtiene un clon de un elemento y si el parámetro de entrada es true, también clona los descendientes.

- elemento.remove() - Elimina el elemento y sus descendientes del documento
- elemento.removeChild(oNodoHijo) - Elimina el nodo hijo especificado como parámetro
- elemento.replaceChild(oNodoNuevo, oNodoAntiguo) - Reemplaza el nodo antiguo por el nuevo.

8. Contenido de las etiquetas HTML y nodos de texto

- innerHTML - Contenido interno de las etiquetas HTML tratado como una cadena de texto al leerlo, pero que cuando inserta texto en la página es procesado por el navegador de forma que genera los nodos (elementos, nodos de texto, atributos,...) necesarios.

```
// La capa (un <div>) incluirá como hija una etiqueta <h1>
oCapa.innerHTML = '<h1 id="encabezado">Introducción a HTML</h1>';
```

- outerHTML - Recupera o sustituye, utilizando cadenas de texto, el propio HTML de un elemento.

```
// Se sustituye una  capa (un <div>) por un encabezado h1
oCapa.outerHTML = '<h1 id="encabezado">Introducción a HTML</h1>';
```

- textContent - Contenido textual de un elemento HTML, incluyendo también el contenido de texto de todos los elementos descendientes.

```
<ul id="lista">
    <li>uno</li>
    <li>dos</li>
    <li>tres</li>
</ul>

let oLista = document.querySelector("#lista");
let sTexto = oLista.textContent; // uno dos tres
oLista.textContent = "borra todo el HTML interno";
// <ul id="lista"> borra todo el HTML interno </ul>
```

9. Trabajar con atributos de etiquetas HTML

- Podemos acceder a un atributo estándar directamente por su nombre.

```
oHipervinculo.href = "https:\\google.es";
```

- Cuidado, porque si el atributo no es estándar para esa etiqueta, por ejemplo, href para un , lo que se estará creando es un atributo del objeto (nodo) .
    - Atributos estándar: https://www.w3schools.com/tags/ref_attributes.asp

- Métodos para atributos
    - `oCapa.setAttribute("nombre","valor"); // Define <div nombre="valor"></div>`
    - `oCapa.getAttribute("id"); // Valor del atributo de etiqueta id`
    - `oCapa.removeAttribute("id"); // Borra el atributo id`
    - `oCapa.hasAttribute("id"); // True si id existe como atributo`

- También podemos manipular los atributos accediendo a los nodos de tipo atributo. Los nodos de tipo elemento (etiquetas) tiene la propiedad attributes, que es un array de nodos de atributos.

```
let oNodoAtributo = oCapa.attributes[0];
console.log(oNodoAtributo.nodeName);
console.log(oNodoAtributo.nodeValue);
```

10. Atributos personalizados

- Los atributos personalizados permiten almacenar datos en las propias etiquetas HTML.

- Un ejemplo podría ser almacenar un código de producto en la etiqueta con la imagen de un producto. Es un dato que permanecerá oculto al usuario, pero que podremos utilizar para hacer operaciones con él.

- Estos atributos aparecen en las etiquetas con el prefijo “data-”, seguido del nombre específico que lo identifique.

```
<div id="user" data-id="1234567890" data-user="johndoe" data-date-of-birth>
John Doe
</div>
```

- Después en JS podemos acceder al contenido de los atributos, modificarlos o crear nuevos atributos mediante la propiedad dataset del elemento HTML.

- Por ejemplo, podemos escribir el siguiente código para la anterior etiqueta :

```
let oCapa = document.querySelector("#user");

console.log(oCapa.dataset.id); // aparece en consola "1234567890"
console.log(oCapa.dataset.user); // aparece en consola "johndoe"
console.log(oCapa.dataset.dateOfBirth); // aparece en consola "", una cadena vacía

oCapa.dataset.dateOfBirth="22-10-1999"; // Se actualiza el atributo

delete oCapa.dataset.id; // eliminamos el atributo personalizado data-id
oCapa.removeAttribute("data-id"); // así también

console.log(oCapa.dataset.id); // aparece en consola “undefined”

oCapa.dataset.nuevoAtributo = "Más datos";
console.log(oCapa.dataset.nuevoAtributo); // aparece en consola "Más datos"
```

- Quedando la etiqueta tras estas instrucciones:

```
<div id="user" data-user="johndoe" data-date-of-birth="22-10-1999"
data-nuevo-atributo="Más datos">
    John Doe
</div>
```

- Hay que tener en cuenta las siguientes reglas de nombrado de los atributos personalizados:

  - El nombre de un atributo personalizado comienza en HTML con data-. Sólo puede estar formado por letras minúsculas, números y los caracteres: guión (-), punto (.), dos puntos (:) y guión bajo (_). NUNCA una letra mayúscula (A a Z).

  - El nombre del atributo en JavaScript será el del correspondiente atributo HTML pero en camelCase, sin guiones, puntos, etc.

## 11. Manejo de estilos con el DOM

### 11.1. Mediante la propiedad style

- Esta modificación implica modificar el estilo en la etiqueta HTML:

```
elemento.style.propiedad = "valor propiedad";
```

Por ejemplo:

```
oCapa.style.backgroundColor = "yellow";
```

El resultado en la etiqueta es el siguiente:

```
<div class="fondoAzul" style="background-color : yellow;"></div>
```

- Hacerlo de la siguiente manera no es estándar, no funciona en todos los navegadores, así que no se recomienda:

```
// NO FUNCIONABA (ahora según el navegador sí lo hace)
oCapa.style = "background-color : yellow;";
```

- Por aplicación de la cascada CSS, la propiedad definida en la propia etiqueta HTML tiene más prioridad que las definidas a otro nivel.

- Para eliminar un estilo aplicado (reset de la propiedad), igualamos la propiedad a la cadena vacía:

```
oCapa.style.backgroundColor = ""; // Se borra el estilo definido
```

- Hay que utilizar las unidades en las propiedades que lo requieran:

```
oCapa.style.marginTop = "10px";
```

- Los nombres de las propiedades CSS se transforman a CamelCase para obtener el nombre de la propiedad en JS.

```
background-color → backgroundColor
margin-top → marginTop
```

- Aunque hay alguna excepción:

```
float → cssFloat
text → cssText
```

### 11.2. Utilizando classList y className.

- La utilización de clases suele ser la opción más cómoda y flexible para manipular la apariencia de la página. El uso de frameworks como Bootstrap hacen el uso de clases sea intensivo.

- El objeto classList representa la lista de clases del elemento HTML.

```
elemento.classList.add("nombre_clase");
elemento.classList.remove("nombre_clase");
elemento.classList.toggle("nombre_clase"); // Alternar la clase
elemento.classList.contains("nombre_clase"); // true - false
```

- También podemos recorrer las clases que se aplican a un elemento:

```
for ( clase of elemento.classList ) {
  console.log(clase);
}
```

- También podemos utilizar la propiedad className, que es menos cómoda de utilizar. Por ejemplo, para añadir una clase:

```
elemento.className += " otraClase";
```

  Pero para eliminar una clase existente se tendría que hacer un tratamiento de la cadena de texto mucho más complejo que utilizar el método remove de classList.

### 11.3. Estilos computados

- Los estilos computados son los estilos que calcula el navegador tras aplicar el conjunto de reglas CSS que afectan a un elemento HTML.
- Los estilos computados son de solo lectura, así que realmente su uso no tiene demasiado interés, ya que no podemos modificarlos directamente.
- La propiedad style no nos permite conocer los estilos computados.
- Podemos obtener esos estilos utilizando el método: `getComputedStyles(elemento)`

```
let oEstilo = getComputedStyle(oCapa);
console.log(oEstilo.marginTop);
```

- Solo se pueden recuperar propiedades no compuestas:

```
console.log(oEstilo.margin); // --> ""
console.log(oEstilo.marginTop); // --> "4px"
```

### 11.4. Objetos que describen ficheros y bloques de estilo

- document.styleSheets contiene un array de los ficheros CSS y los bloques de estilo incluidos en nuestra página.
    - `document.styleSheets[index]  // Es una hoja de estilos concreta`
    - `document.styleSheets[index].cssRules  // Array de reglas CSS`

- El array de reglas contiene objetos cssRule con las siguientes propiedades:
    - cssText - texto completo de la regla
    - selectorText - texto del selector
    - objeto Style - que nos permite conocer los estilos aplicados

- Podemos utilizar el método insertRule para crear dinámicamente reglas nuevas:

```
// El 0 indica el índice, en este caso se inserta al principio
document.styleSheets[index].insertRule("#blanco { color: white }", 0);
```

- El método deleteRule(index) borraría la regla apuntada por el índice.

## 12. Proceso de documentos XML

- Podemos procesar documentos XML en el navegador cargándolos desde un fichero del servidor.
- El fichero XML estará almacenado en el servidor o bien se genera dinámicamente por la ejecución de un script del servidor (escrito en PHP, por ejemplo).
- En el cliente obtendremos un objeto de tipo documento XML, que es análogo al objeto document de la página HTML.
- Las propiedades y métodos de manejo de nodos son iguales que en un documento HTML.

```
function loadXMLDoc(filename) {
        let http = new XMLHttpRequest();
        xhttp.open("GET", filename, false); // Petición GET síncronaç

        xhttp.send();

        return xhttp.responseXML;
}
```