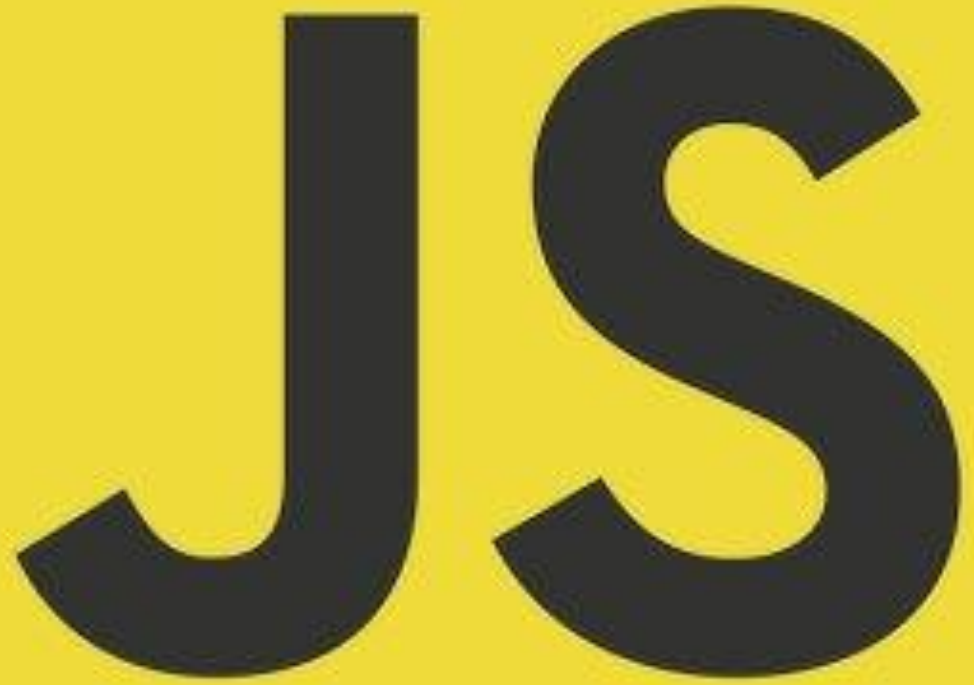
UT 5 - Utilización del modelo de objetos del documento (DOM)



Marina Navarro Pina

BOM

El Browser Object Model (**BOM**) es un conjunto de objetos que permiten interactuar con el *navegador web*.

A través del BOM se pueden realizar diversas acciones como:

abrir y cerrar ventanas, ejecutar código en momentos específicos (timers), obtener información sobre el navegador y la pantalla, y gestionar cookies, entre otras funcionalidades.

BOM: Objetos

El BOM incluye varios objetos importantes:

- **Objeto window**: Representa **la ventana del navegador** y es el objeto principal. Permite acceder a propiedades como `.name`, `.status`, y métodos como `.open()`, que abre nuevas ventanas. Puede omitirse al llamar a sus propiedades y métodos *(setTimeout()===window.setTimeout())*

  El objeto `window` tiene métodos para mostrar diálogos al usuario:

  alert, confirm, prompt

  Más información:
  - https://www.w3schools.com/jsref/obj_window.asp
  - https://developer.mozilla.org/es/docs/Web/API/Window

Ejemplo 1

BOM: Objetos

- **Objeto location**: Proporciona información sobre la URL actual y permite modificarla. Incluye propiedades como `.href`, `.protocol`, y métodos como `.reload()`.

  Más información:
  - https://www.w3schools.com/jsref/obj_location.asp
  - https://developer.mozilla.org/es/docs/Web/API/Location

- **Objeto history**: Permite acceder al historial de navegación, con métodos para navegar hacia atrás y adelante en las páginas visitadas.

  Más información:
  - https://www.w3schools.com/jsref/obj_history.asp
  - https://developer.mozilla.org/es/docs/Web/API/History

Ejemplo 1

BOM: Objetos

- **Objeto navigator**: Proporciona información sobre el navegador y el sistema operativo, incluyendo propiedades como `.userAgent` y `.language`.

  Más información:
  - https://www.w3schools.com/jsref/obj_navigator.asp
  - https://developer.mozilla.org/es/docs/Web/API/Navigator

- **Otros objetos**: Incluyen `screen`, que proporciona información sobre la pantalla, y otros objetos que permiten interactuar con características del navegador.

  Más información:
  - https://www.w3schools.com/jsref/obj_screen.asp
  - https://developer.mozilla.org/es/docs/Web/API/Screen

Ejemplo 1

DOM

El Document Object Model (DOM) es **una representación estructurada de un documento HTML en forma de árbol**.

Cada elemento del HTML, como etiquetas, atributos y texto, se convierte en un **nodo dentro de este árbol**. Esto permite que se pueda interactuar con el contenido y la estructura del documento de manera programática, lo que significa que pueden *añadir, eliminar o modificar nodos según sea necesario*.

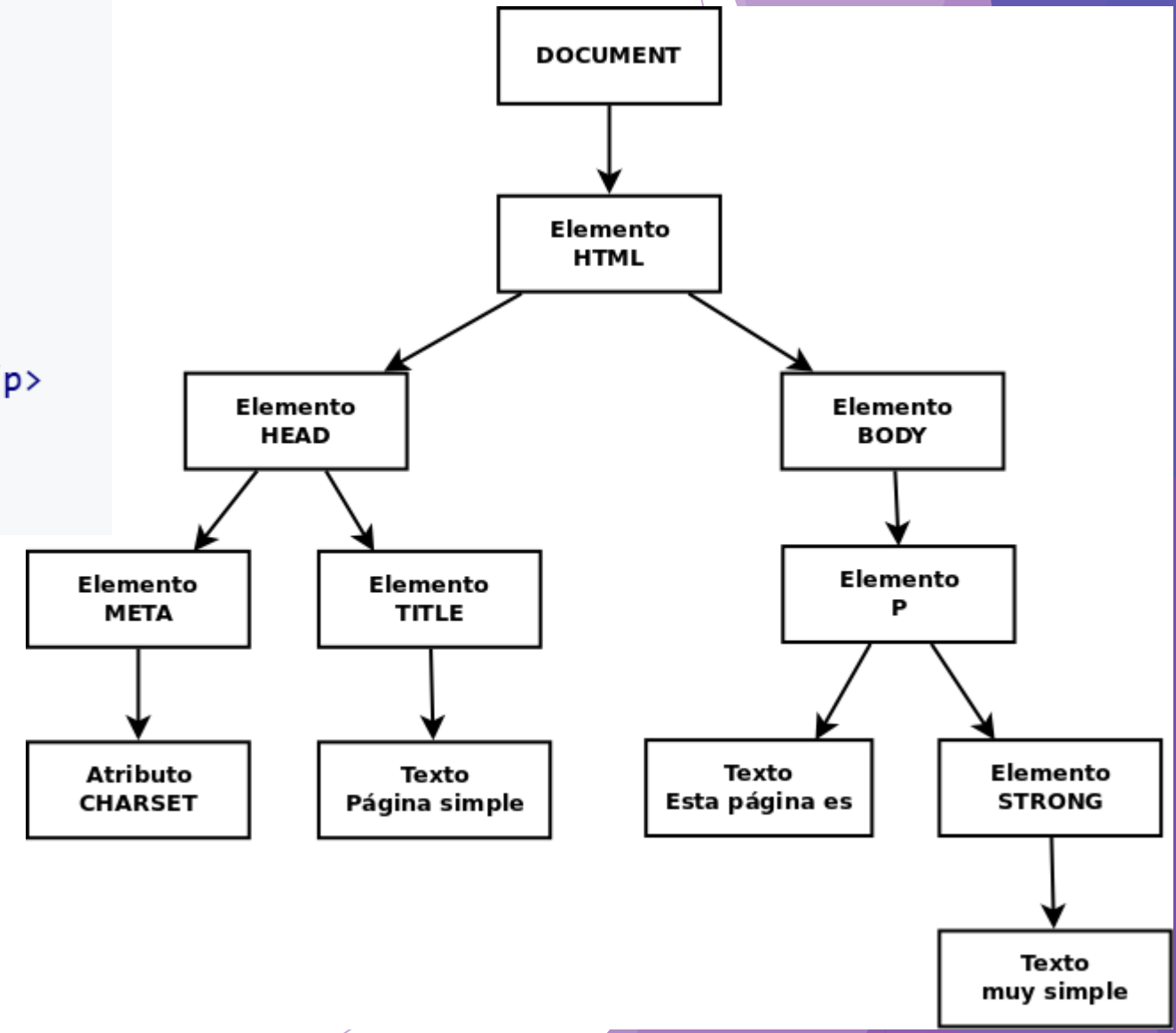
DOM

El objeto principal del DOM es **document**, que actúa como la *entrada global* para acceder y manipular el contenido del HTML.

Cada *nodo HTML* se representa como un objeto de tipo **element**, que puede contener: *otros nodos (hijos), atributos (como id, class, etc.) y estilos (CSS).*

DOM - Ejemplo

```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>
```



DOM - Navegando a través del DOM

Manipular el DOM directamente con JavaScript puede ser más complicado en comparación con el uso de librerías como jQuery o frameworks como Angular.

Estas herramientas proporcionan métodos y funciones más simplificados y eficientes para interactuar con el DOM, lo que facilita tareas comunes como la selección de elementos, la manipulación de atributos y la gestión de eventos.

DOM - Navegando a través del DOM: Métodos

Los principales métodos para acceder a los diferentes nodos son:

- **document.getElementById(“id”)** → Devuelve el elemento que tiene el id especificado, o null si no existe.
- **document.getElementsByClassName(“class”)** → Devuelve un array de elementos que tengan la clase especificada. Al llamar a este método desde un nodo (en lugar de document), buscará los elementos a partir de dicho nodo.
- **document.getElementsByTagName(“HTML tag”)** → Devuelve un array con los elementos con la etiqueta HTML especificada. Por ejemplo “p” (párrafos).

DOM - Navegando a través del DOM: Métodos

Los principales '*atajos*' para obtener algunos elementos comunes son:

- **document.documentElement** → Devuelve el elemento <html>
- **document.head** → Devuelve el elemento <head>
- **document.body** → Devuelve el elemento <body>

DOM - Navegando a través del DOM: Métodos

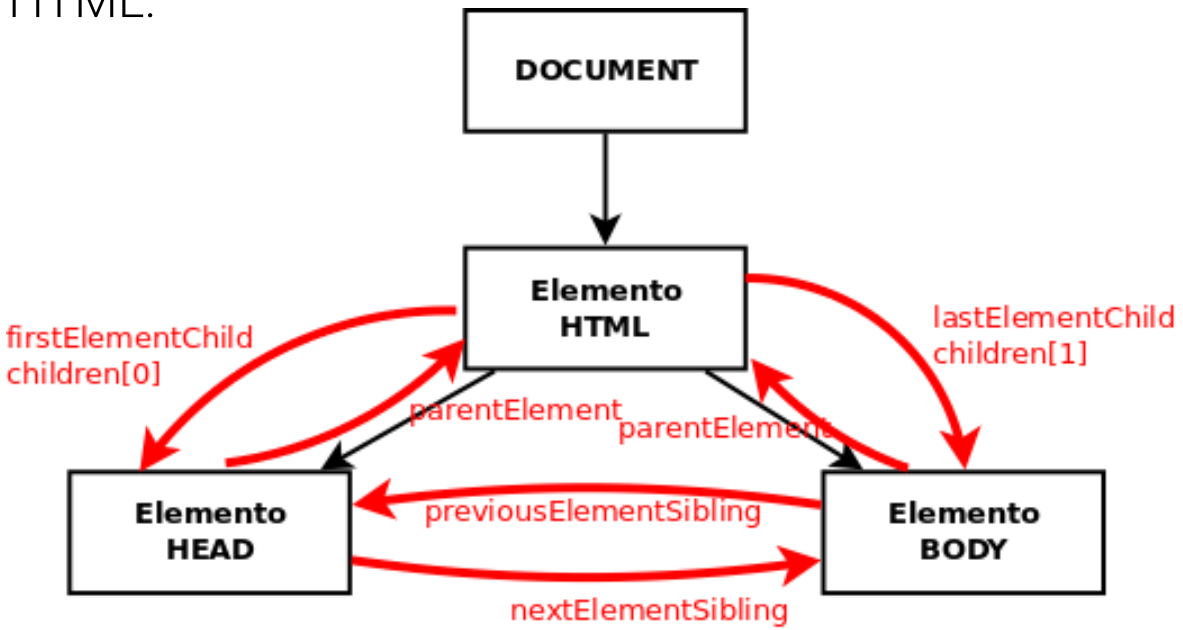
Los principales métodos para acceder a los diferentes nodos *a partir de uno dado* son:

- **element.childNodes** → Devuelve un array con los descendientes (hijos) del nodo. Esto incluye los nodos de tipo texto y comentarios.
- **element.chidren** → Igual que arriba pero excluye los comentarios y las etiquetas de texto (sólo nodos HTML). Normalmente es el recomendado.
- **element.parentNode** → Devuelve el nodo padre de un elemento.

DOM - Navegando a través del DOM: Métodos

Los principales métodos para acceder a los diferentes nodos *a partir de uno dado* son:

- **element.nextSibling** → Devuelve el siguiente nodo del mismo nivel (el hermano). El método **previousSibling** hace justo lo opuesto. Es recomendable usar **nextElementSibling** o **previousElementSibling** si queremos obtener sólo los elementos HTML.



Ejemplo 2

DOM - Navegando a través del DOM: Propiedades nodos

Los principales propiedades de un nodo son:

- **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 Esta página es muy simple *
  let contenido = elemento.innerHTML; // contenido='Esta página es muy simple'

- **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'

- **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 type="text" 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"

DOM - Navegando a través del DOM: querySelector

El concepto de "Selector Query" se refiere a la capacidad de seleccionar elementos del DOM **utilizando selectores CSS**, lo que permite acceder y manipular elementos HTML de manera más intuitiva y eficiente.

Esta característica fue popularizada por jQuery cuando se lanzó en 2006, ya que ofrecía una forma sencilla de seleccionar elementos basándose en sus clases, IDs, atributos y otros selectores CSS.

Los navegadores han adoptado esta funcionalidad de forma nativa, lo que significa que ahora se puede utilizar directamente en JavaScript sin necesidad de depender de jQuery.

DOM - Navegando a través del DOM: querySelector

Los principales métodos son:

- **document.querySelector(selector)**: Devuelve el primer elemento que coincide con el selector CSS proporcionado.
- **document.querySelectorAll(selector)**: Devuelve una lista de todos los elementos que coinciden con el selector CSS.

Ejemplo 3

DOM - Manipulando el DOM: crear elementos

Los principales métodos que nos permiten cambiar el árbol DOM, y por tanto modificar la página, son:

- **document.createElement("tag")**: Este método se utiliza para **crear** un nuevo **elemento** HTML con la etiqueta especificada. Sin embargo, este nuevo elemento aún no se añadirá al DOM hasta que se inserte en un elemento existente.

  Por ejemplo, si se desea crear un nuevo elemento <li>, se puede hacer así:
  let nuevoElemento = document.createElement("li");

DOM - Manipulando el DOM: añadir elementos

- **elemento.append(elementos o texto)**: Este método **añade** uno o más nodos o texto como *últimos hijos del elemento especificado*. Se pueden pasar múltiples parámetros.

  Por ejemplo:
  ```
  nuevoLi.append('Nuevo elemento de lista');
  let miPrimeraLista = document.getElementsByTagName('ul')[0];
  miPrimeraLista.append(nuevoLi); // Añade nuevoLi al final de la lista
  ```

- **elemento.prepend(elementos o texto)**: Similar a append, pero añade los elementos *antes del primer hijo del elemento*.

  Por ejemplo:
  ```
  miPrimeraLista.prepend(nuevoLi); // Añade nuevoLi al principio de la lista
  ```

- **elemento.after(elementos o texto)**: Añade los elementos como *hermanos siguientes del elemento* especificado.

- **elemento.before(elementos o texto)**: Añade los elementos como hermanos anteriores del elemento especificado.

DOM - Manipulando el DOM: eliminar elementos

- **elemento.remove()**: Este método elimina el nodo del documento.
- elemento.removeChild(nodo): Borra un nodo hijo específico del elemento.

  Por ejemplo:

```
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
```

DOM - Manipulando el DOM: reemplazar elementos

- **elemento.replaceWith(nuevoNodo)**: Reemplaza el nodo elemento con el nuevoNodo pasado.

  Por ejemplo:

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

- elemento.replaceChild(nuevoNodo, viejoNodo): Reemplaza un nodo hijo específico con otro nodo.

  Por ejemplo:

  ```
  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
  ```

DOM - Manipulando el DOM: clonar elementos

- **elementoAClonar.cloneNode(boolean)**: Clona un nodo. Si se pasa true, se clonan también todos sus descendientes. Esto es útil si se desea mantener el nodo original en su lugar y crear una copia para añadir en otro lugar.

Y muchos métodos menos usados como son:

document.createTextNode('texto'), elemento.appendChild(nuevoNodo), elemento.insertBefore(nuevoNodo, nodo) o los expuestos removeChild y replaceChild

Cuando se añade un nodo que ya existe en el DOM a una nueva ubicación, se elimina de su posición original. Para mantenerlo en ambas ubicaciones, se debe clonar el nodo antes de añadirlo.

DOM - Manipulando el DOM: innerHTML

Añadir nuevos nodos a un elemento del DOM utilizando **innerHTML** es una forma rápida y sencilla de modificar el contenido HTML de un elemento. Sin embargo, hay consideraciones importantes sobre su uso, especialmente en *términos de eficiencia, seguridad y rendimiento*.

Supongamos que tenemos un con el id myDiv y queremos añadir dos párrafos al final de este div. El primer párrafo contendrá un texto simple, y el segundo párrafo incluirá un texto en negrita. Aquí hay dos formas de hacerlo:

- **Usando Métodos de Creación de Nodos** (RECOMENDABLE):

```
let miDiv = document.getElementById('myDiv');
let nuevoParrafo = document.createElement('p');
nuevoParrafo.textContent = 'Párrafo añadido al final';

let ultimoParrafo = document.createElement('p');
const textoNegrita = document.createElement('strong');
textoNegrita.textContent = 'con texto en negrita';
ultimoParrafo.append('Último párrafo ', textoNegrita);

miDiv.append(nuevoParrafo, ultimoParrafo);
```

- **Usando innerHTML**:

```
let miDiv = document.getElementById('myDiv');
miDiv.innerHTML += '<p>Párrafo añadido al final</p><p>Último párrafo
<strong>con texto en negrita</strong></p>';
```

Aunque es válida **no es muy eficiente** ya que obliga al navegador a volver a pintar TODO el contenido de miDIV.

Ejemplo 4

DOM - Atributos de los nodos. Acceso

Los atributos son propiedades que se pueden *ver, modificar, añadir o eliminar* en los elementos HTML. Las principales operaciones que se pueden realizar con los atributos de los nodos:

- **elemento.attributes:** Devuelve un array con todos los atributos del elemento, permitiendo ver qué atributos están presentes.
- **elemento.hasAttribute('nombreAtributo'):** Indica si el elemento tiene definido un atributo específico.
- **elemento.getAttribute('nombreAtributo'):** Devuelve el valor del atributo especificado. Para muchos elementos, este valor también se puede acceder directamente usando **elemento.atributo**.

DOM - Atributos de los nodos. Modificación

- **elemento.setAttribute('nombreAtributo', 'valor'):** Establece un nuevo valor para el atributo especificado. También se puede cambiar el valor directamente usando **elemento.atributo = valor**.

- **elemento.removeAttribute('nombreAtributo'):** Elimina el atributo especificado del elemento.

Algunos atributos como *id, title o className* se pueden acceder y modificar como si fueran propiedades del elemento. Por ejemplo:

```
elemento.id = 'primera-lista';
// es equivalente ha hacer:
elemento.setAttribute('id', 'primera-lista');
```

DOM - Estilos de los nodos

Los estilos se acceden a través del atributo **style**, donde cada estilo se representa como una propiedad en notación camelCase.

Por ejemplo, para cambiar el color de fondo:

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

Sin embargo, **es recomendable usar clases CSS para aplicar estilos** en lugar de modificar directamente los estilos en línea.

DOM - Atributos de clase

Para manejar clases, se utiliza la propiedad **classList**, que proporciona métodos para añadir, eliminar y verificar clases:

- **.add(clase):** Añade una clase al elemento.
- **.remove(clase):** Elimina una clase del elemento.
- **.toggle(clase):** Añade la clase si no está presente, o la elimina si ya está.
- **.contains(clase):** Verifica si el elemento tiene la clase especificada.
- **.replace(oldClase, newClase):** Reemplaza una clase existente por una nueva.

Ejemplo 5

DOM - Referencias

- https://www.w3schools.com/jsref/dom_obj_document.asp
- https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model