Crear elementos del DOM

El método .createElement()

Permite crear un Elemento HTML **en memoria**. En ese momento **¡no estará insertado AÚN en nuestro documento HTML!**. Habitualmente, se almacena en una variable o constante, permitiéndonos modificar sus atributos o contenido, para **posteriormente** insertarlo en una posición determinada del DOM o documento HTML. Se le debe pasar el nombre de la etiqueta tag a utilizar:

```
const div = document.createElement("div"); // Creamos un <div></div>
const span = document.createElement("span"); // Creamos un <span></span>
const img = document.createElement("img"); // Creamos un <img>
```

El método createElement() tiene un segundo parámetro opcional denominado options, pero va a sociado a los **Web Components**.

La propiedad .isConnected

La propiedad isConnected nos indica si el elemento en cuestión está insertado en el DOM (*conectado*), o por el contrario, sólo lo tenemos guardado en una variable (*desconectado*)_ en el primer caso devuelve True y en el segundo False.
En el ejemplo anterior están desconectadas

Crear elementos del DOM

El método .createComment()

Crea comentarios HTML, e insertarlos en el DOM. No es algo muy práctico. Tampoco lo conecta.

```
const comment = document.createComment("Comentario"); // <!--Comentario-->
```

El método createTextNode()

Permite crear un nodo de texto con el texto indicado por parámetro. Nos permite crear fragmentos de texto sin usar una etiqueta HTML contenedora. No es muy común:

```
const text = document.createTextNode("Hola"); // Nodo de texto: 'hola'
```

El método .cloneNode()

Clona y devuelve el nuevo elemento duplicado del original. Si sólo asigno un elemento a otro, el nuevo haría referencia a la posición del anterior.

```
const div = document.createElement("div");
div.textContent = "Elemento 1";
const div2 = div; // NO se está haciendo una copia
div2.textContent = "Elemento 2";
div.textContent; // 'Elemento 2' (se modifica el original)
```

Para evitar, clonar:

```
div = document.createElement("div");
div.textContent = "Elemento 1";
const div2 = div.cloneNode(); // Ahora SÍ estamos duplicando
div2.textContent = "Elemento 2";  div.textContent; // 'Elemento 1' (no se modifica el original)
```

Crear elementos del DOM

El parámetro Deep

El método cloneNode(deep) acepta un parámetro deep opcional, a false por defecto. Indica el tipo de clonación que se realizará:

- Si es true, clonará también elementos hijos. Se conoce como **clonación profunda** (*Deep Clone*).
- Si es false, no clonará elementos hijos. Se conoce como **clonación superficial** (*Shallow Clone*).

API de Nodos

La **API de Nodos** de Javascript es la más antigua y la que trabaja a más bajo nivel (*a nivel de detalle muy profundo*). Se basa en trabajar con ítems llamados **nodos**, que son una unidad aún más básica que los **elementos HTML**.

```
const container = document.createElement("div");
container.innerHTML = `Hola <strong>Mundo.</strong>Adios`;
container.children; // HTMLCollection [strong]
container.childNodes; // NodeList(3) [text, strong, text]
container.childNodes[0].textContent // 'Hola '
```

Mediante .children obtenemos una lista de elementos , y sólo obtenemos el . Mediante .childNodes que nos devuelve NODE en lugar de ELEMENT, nos devuelve 3 nodos:

Como puedes ver, la **API de nodos** tiene un nivel más detallado, pero muchas veces es demasiado tediosa.

El método .appendChild()

Es el método más extendido de la API de Nodos. Añade o inserta un nuevo elemento, como si fuera un hijo al final de todos los hijos del elemento sobre el que se realiza

```
const container = document.querySelector(".container");
const img = document.createElement("img");
img.src = "https://lenguajejs.com/assets/logo.svg";
img.alt = "Logo Javascript";
container.appendChild(img);
```

Insertará una imagen como último hijo del elemento <body>.

API de Nodos

El método .removeChild()

Permite eliminar un nodo hijo de un elemento,pasado por argumento.

```
const container = document.querySelector(".container");
const secondItem = container.querySelector(".item:nth-child(2)");
container.removeChild(secondItem); // Desconecta el segundo .item
```

El método .replaceChild()

El método replaceChild(new, old) nos permite cambiar un nodo hijo old por un nuevo nodo hijo new. Devuelve el nodo reemplazado:

```
const container = document.querySelector(".container");
const secondItem = container.querySelector(".item:nth-child(2)");
const newNode = document.createElement("div");
newNode.textContent = "DOS";
container.replaceChild(newNode, secondItem);
```

API de Nodos

El método .insertBefore()

InsertBefore(newnode, node) permite especificar exactamente el lugar a insertar un nodo. El primer parámetro es el nodo a insertar, mientras que el **segundo parámetro** puede ser:

- NULL: inserta newnode después del último nodo hijo. Equivale a .appendChild().
- ELEMENT: inserta newnode antes de dicho node de referencia.

```
const container = document.querySelector(".container");
const secondItem = container.querySelector(".item:nth-child(2)");
const newNode = document.createElement("div");
newNode.textContent = "Nuevo elemento";
container.insertBefore(newNode, secondItem);//el nuevo elemento aparecerá justo antes del segundo item
```

API de Elementos HTML

Los métodos anteriores sirven en muchos casos, pero son poco específicos y puede que no cubran todas las posibles situaciones. Existe otra familia de métodos para añadir e insertar elementos más completa, los de la API Elementos.
Todos los métodos permiten pasar por parámetro un elemento o una lista de elementos. También puedes pasar un String (*para insertar un fragmento de texto*).

El método .before() y .after()

Con .before() podemos insertar uno o varios elementos antes del elemento que llama al before (*en el ejemplo, container*). Con el método .after() ocurre exactamente lo mismo, pero **después del elemento** en lugar de antes:

```
const element = document.createElement("div");
element.textContent = "Item insertado";
// A) Inserta antes de .container (elemento del ejemplo anterior) :
container.before(element);
// B) Inserta después de .container:
container.after(element);
```

API de Elementos HTML

El método .prepend() y .append()

El método .prepend() permite insertar uno o varios elementos antes del primer elemento hijo de nuestro elemento base. En el caso de append() ocurre lo mismo, pero **después del último elemento hijo**:

```
const element = document.createElement("div");
element.textContent = "Item insertado";
container.prepend(element); // A) Inserta antes del primer hijo de .container
container.append(element); // B) Inserta después del último hijo de .container
```

El método .append() es equivalente al .appendChild() visto anteriormente.

El método .replaceChildren() y .replaceWith()

replaceChildren() permite eliminar todos los elementos hijos del elemento base, y sustituirlos por uno o varios que se indiquen por parámetro.

replaceWith(), permite reemplazar **el propio elemento base** con uno o varios elementos que pasemos por parámetro, por lo que se realiza un reemplazo completo:

```
const element = document.createElement("div");
element.textContent = "Nuevo Item hijo";
container.replaceChildren(element); // Reemplaza por todos sus hijos
container.replaceWith(element); // El container es reemplazado por el nuevo item hijo
```

API de Elementos HTML

El método .remove()

Al «eliminar» un nodo o elemento HTML, lo que hacemos realmente no es borrarlo, sino **desconectarlo del DOM o documento HTML**, de modo que no están conectados, pero siguen existiendo.

El método .remove() se encarga de desconectarse del DOM a sí mismo,

```
const div = document.querySelector(".deleteme");
div.isConnected; // true
div.remove();
div.isConnected; // false
```

API de Inyección Adyacente

Esta API tiene 3 modalidades diferentes enmarcadas en 3 métodos:

- .insertAdjacentElement() para insertar elementos (*etiquetas HTML*)
- .insertAdjacentHTML() para insertar **código HTML** directamente (*similar a innerHTML*)
- .insertAdjacentText() para insertar **textos** directamente (*similar a textContent*)

El parámetro position

Este parámetro, que tienen los 3 métodos, es un String que indica la posición donde se va a insertar el nuevo elemento, código HTML o texto. Debe indicarse uno de los siguientes valores:

Valor	Descripción	Equivalente a...
beforebegin	Inserta el elemento **antes** de la etiqueta HTML de apertura.	before()
afterbegin	Inserta el elemento dentro, **antes de su primer hijo**.	preprend()
beforeend	Inserta el elemento dentro, **justo antes de la etiqueta HTML de cierre**.	append() o appendChild()
afterend	Inserta el elemento **después** de la etiqueta HTML de cierre.	after()

API de Inyección Adyacente

El método .insertAdjacentElement

Le pasamos la posición y un ELEMENT:

```
const container = document.querySelector(".container");
const div = document.createElement("div");
div.textContent = "Ejemplo";

container.insertAdjacentElement("beforebegin",  div);
// A) <div>Ejemplo</div> <div class="container">container</div>

container.insertAdjacentElement("afterbegin", div);
// B) <div class="container"> <div>Ejemplo</div> container</div>

container.insertAdjacentElement("beforeend",  div);
// C) <div class="container">container <div>Ejemplo</div> </div>

container.insertAdjacentElement("afterend", div);
// D) <div class="container">App</div> <div>Ejemplo</div>
```

A),B),C) y D) son **opciones alternativas**, no lo que ocurriría tras ejecutar una tras otra.

API de Inyección Adyacente

El método .insertAdjacentHTML()

Igual que el anterior, sólo que en lugar de pasarle un elemento HTML , le pasamos un con un **código HTML**.

```
const container = document.querySelector(".container");
const htmlCode = `<div class="text">¡Hola, amigo!</div>`;
container.insertAdjacentHTML("beforebegin", htmlCode); // Antes
container.insertAdjacentHTML("afterbegin", htmlCode); // Como hijo, al principio
container.insertAdjacentHTML("beforeend", htmlCode); // Como hijo, al final
container.insertAdjacentHTML("afterend", htmlCode); // Después
```

El método .insertAdjacentText()

Igual paro le pasamos texto, que se insertará como si estuvieramos utilizando .textContent:

```
const container = document.querySelector(".container");
const htmlCode = `<div class="text">Hola a </div>`; container.insertAdjacentText("beforeend", “ todos");
container.textContent; // Hola a todos
```

Con estos tres métodos, podemos realizar prácticamente cualquier tarea en el DOM de una página.

Selectores CSS de queryselectorall()

El método **querySelectorAll(SelectoresCSS)** devuelve una NodeList que representa una lista de elementos del documento que coinciden con el grupo de selectores indicados.

Selectores básicos:

1. Selector de tipo

Selecciona todos los elementos que coinciden con el nombre del elemento especificado.

Sintaxis: eltname

Ejemplo: input se aplicará a cualquier elemento <input>.

2. Selector de clase

Selecciona todos los elementos que tienen el atributo de class especificado.

Sintaxis: .classname

Ejemplo: .index seleccionará cualquier elemento que tenga la clase "*index*".

3. Selector de ID

Selecciona un elemento basándose en el valor de su atributo id. Solo puede haber un elemento con un determinado ID dentro de un documento.

Sintaxis: #idname

Ejemplo: #toc se aplicará a cualquier elemento que tenga el ID "*toc*".

Selectores CSS de queryselectorall()

4. Selector universal

Selecciona todos los elementos. Opcionalmente, puede estar restringido a un espacio de nombre específico o a todos los espacios de nombres.

Sintaxis: * ns|* *|*

Ejemplo: * se aplicará a todos los elementos del documento.

5. Selector de atributo

Selecciona elementos basándose en el valor de un determinado atributo.

Sintaxis: [attr] [attr=value] [attr~=value] [attr|=value] [attr^=value] [attr$=value] [attr*=value]

Ejemplo: [autoplay] seleccionará todos los elementos que tengan el atributo "autoplay" establecido (a cualquier valor).

1. **Presencia de un atributo:**

elemento[atributo]

Selecciona todos los elementos del tipo especificado que tengan el atributo indicado, independientemente de su valor.

Selectores CSS de queryselectorall()

2. Valor exacto de un atributo:

elemento[atributo="valor"]

Selecciona los elementos cuyo atributo tenga exactamente el valor especificado.

3. Valor que contiene una subcadena:

elemento[atributo*="valor"]

Selecciona los elementos cuyo atributo tenga exactamente el valor especificado.

4. Valor que comienza con una subcadena:

elemento[atributo^="valor"]

Selecciona los elementos cuyo atributo comience con la subcadena especificada.

Selectores CSS de queryselectorall()

5. Valor que termina con una subcadena:

elemento[atributo$="valor"]

Selecciona los elementos cuyo atributo termine con la subcadena especificada.

6. Valor que coincide con un patrón:

elemento[atributo~="valor"]

Selecciona elementos cuyo atributo contiene una lista de palabras separadas por espacios, y al menos una de esas palabras coincide con el valor especificado.

7. Valor que comienza con un prefijo definido:

elemento[atributo|="valor"]

Selecciona los elementos cuyo atributo comienza con el valor especificado, seguido opcionalmente de un guion.

Selectores CSS de queryselectorall()

Ejemplo:

```
JavaScript
// Selecciona todos los enlaces que apuntan a un archivo PDF
const enlacesPDF = document.querySelectorAll('a[href$=".pdf"]');

// Selecciona todos los elementos con el atributo "data-role" igual a "button"
const botones = document.querySelectorAll('[data-role="button"]');

// Selecciona todos los elementos <input> de tipo "text"
const inputsTexto = document.querySelectorAll('input[type="text"]');
```

Selectores CSS de queryselectorall()

Combinadores

1. Combinador de hermanos adyacentes

El combinador + selecciona hermanos adyacentes. Esto quiere decir que el segundo elemento sigue directamente al primero y ambos comparten el mismo elemento padre.

Sintaxis: A + B
Ejemplo: La regla h2 + p se aplicará a todos los elementos <p> que siguen directamente a un elemento <h2>.

2. Combinador general de hermanos

El combinador ~ selecciona hermanos. Esto quiere decir que el segundo elemento sigue al primero (no necesariamente de forma inmediata) y ambos comparten el mismo elemento padre.

Sintaxis: A ~ B
Ejemplo: La regla p ~ span se aplicará a todos los elementos que siguen un elemento <p>.

Selectores CSS de queryselectorall()

3. Combinador de hijo

El combinador > selecciona los elementos que son hijos directos del primer elemento.

Sintaxis: A > B

Ejemplo: La regla ul > li se aplicará a todos los elementos <li> que son hijos directos de un elemento <ul>.

4. Combinador de descendientes

El combinador (espacio) selecciona los elementos que son descendientes del primer elemento.

Sintaxis: A B

Ejemplo: La regla div span se aplicará a todos los elementos que están dentro de un elemento .

Selectores CSS de queryselectorall()

Pseudoclases

Las pseudoclases permiten la selección de elementos, basada en información de estado que no está contenida en el árbol de documentos.

Ejemplo: La regla a:visited se aplicará a todos los elementos <a> que hayan sido visitados por el usuario.

Ejemplos de uso:

- **Seleccionar elementos en estados específicos:** seleccionar elementos cuando el cursor está sobre ellos (:hover), cuando son clicados (:active), o cuando están en un formulario y son inválidos (:invalid).
- **Seleccionar elementos basados en su posición:** seleccionar el primer elemento de un grupo (:first-child), el último (:last-child), o elementos pares o impares (:nth-child(even), :nth-child(odd)).
- **Crear estilos condicionales:** Puedes aplicar estilos diferentes a elementos según su estado o posición.

Selectores CSS de queryselectorall()

Ejemplo:

```
// Selecciona todos los enlaces que están siendo apuntados por el cursor
const enlacesSobrevolados = document.querySelectorAll('a:hover');
// Selecciona todos los elementos de párrafo que son el primer hijo de su contenedor
const primerosParrafos = document.querySelectorAll('p:first-child');
// Selecciona todos los elementos de entrada que son inválidos
const inputsInvalidos = document.querySelectorAll('input:invalid');
// Selecciona todos los elementos de lista que son el segundo hijo de su contenedor
const segundosElementosLista = document.querySelectorAll('li:nth-child(2)');
```

Selectores CSS de queryselectorall()

Pseudoelementos

Los pseudoelementos son abstracciones del árbol que representan entidades más allá de los elementos HTML. Por ejemplo, HTML no tiene un elemento que describa la primera letra de un párrafo ni los marcadores de una lista. Los pseudoelementos representan estas entidades y nos permiten asignarles reglas CSS. De este modo podemos diseñar estas entidades de forma independiente.

Ejemplo: La regla p::first-line se aplicará a la primera línea de texto de todos los elementos <p>.

Atributos data (atributos de datos)

Son atributos personalizados que se pueden añadir a cualquier elemento HTML para almacenar datos adicionales. La parte "data-" indica que es un atributo de datos, y lo que lo acompaña es un nombre que tú le asignas. Este nombre puede ser cualquier cosa que tenga sentido para tu código y te ayude a identificar el propósito de esos datos.

¿Para qué sirve?

- **Almacenar información personalizada:** Puedes utilizar este atributo para guardar cualquier tipo de información que necesites asociar a un elemento, como un identificador único, una configuración específica, o cualquier otro dato relevante para tu aplicación.
- **Facilitar la selección y manipulación de elementos:** Al tener un identificador único en cada elemento, puedes seleccionar fácilmente esos elementos con JavaScript utilizando querySelectorAll o querySelector y realizar acciones sobre ellos.
- **Crear estructuras de datos personalizadas:** Puedes crear estructuras de datos jerárquicas o relacionales utilizando los atributos data- de datos.

Atributos data (atributos de datos)

Ejemplo:

HTML

```
<div data-id="producto-1" class="producto">
  <h2>Producto 1</h2>
  <p>Descripción del producto 1</p>
  <button data-action="agregar-al-carrito">Agregar al carrito</button>
</div>
```

JavaScript

```
const producto = document.querySelector('[data-id="producto-1"]');
const idProducto = producto.dataset.id;
console.log(idProducto); // Imprime "producto-1"
```

Atributos data (atributos de datos)

Propiedad dataset

.dataset es una propiedad del DOM (Document Object Model) que nos permite acceder y manipular los atributos de datos personalizados (aquellos que comienzan con data-) de un elemento HTML. En otras palabras, es una forma sencilla de leer y escribir los valores almacenados en los atributos data-* de un elemento.

Ejemplo:

HTML

```
<div data-producto="manzana" data-precio="2.5">Manzana</div>
```

Javascript

```
const elemento = document.querySelector('div');
console.log(elemento.dataset.producto); // Imprime "manzana"
console.log(elemento.dataset.precio); // Imprime "1.50"
```