



Somos un **ecosistema** de desarrolladores de software

# Java Script DOM (Document Object Model)



```
<!--  
?>  
BEGIN NAVIGATION  
"  
>Home</a></li>  
.html">Home Events</a></li>  
nu.html">Multiple Column Men  
<a href="#" class="current":  
button-header.html">Tall But  
ogo.html">Image Logo</a></  
mber="tall-logo.html">Ta  
f="#">Carousels</a>  
ith-slider.html">Variat  
ider.html">Testimoni
```

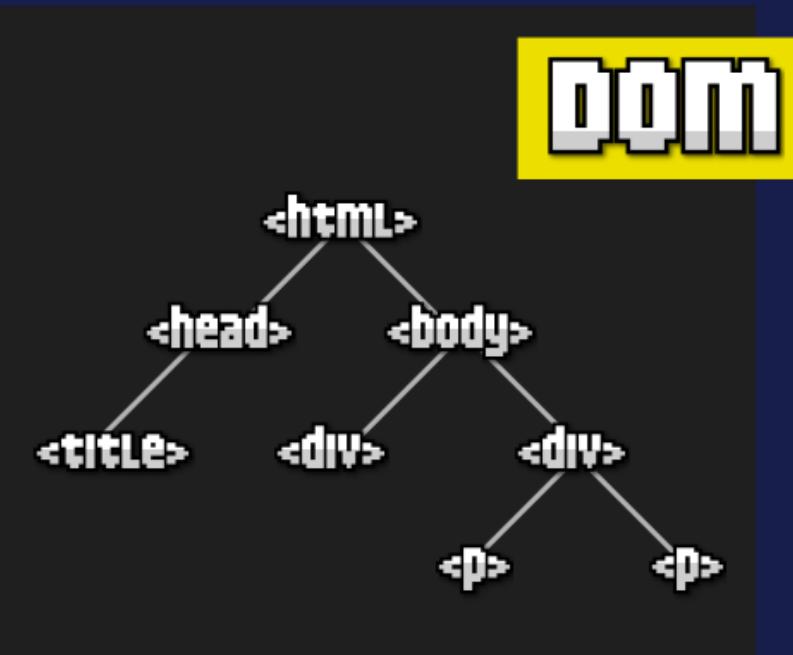
# DOM

¿Qué es DOM?

Las siglas DOM significan Document Object Model, o lo que es lo mismo, la estructura del documento HTML. Una página HTML está formada por múltiples etiquetas HTML, anidadas una dentro de otra, formando un árbol de etiquetas relacionadas entre sí, que se denomina árbol DOM (o simplemente DOM).

```
<html>
  <head>
    <title>Title</title>
  </head>
  <body>
    <div></div>
    <div>
      <p>Párrafo 1</p>
      <p>Párrafo 2</p>
    </div>
  </body>
</html>
```

**HTML**



# DOM

## El objeto Document

Tipo de dato genérico	Tipo específico	Etiqueta	Descripción	+ info
HTMLElement	HTMLDivElement	<div>	Etiqueta divisoria (en bloque).	<a href="#">Elemento &lt;div&gt;</a>
HTMLElement	HTMLSpanElement	<span>	Etiqueta divisoria (en línea).	<a href="#">Elemento &lt;span&gt;</a>
HTMLElement	HTMLImageElement	<img>	Imagen.	<a href="#">Elemento &lt;img&gt;</a>
HTMLElement	HTMLAudioElement	<audio>	Contenedor de audio.	<a href="#">Elemento &lt;audio&gt;</a>

# DOM

API nativa de Javascript

Capítulo del DOM	Descripción
 <a href="#">Buscar etiquetas</a>	Métodos para buscar elementos en el DOM como .querySelector().
 <a href="#">Crear etiquetas</a>	Métodos y consejos para crear elementos en el DOM y trabajar con ellos.
 <a href="#">Gestionar atributos</a>	Formas de gestionar y modificar atributos HTML de elementos del DOM.
 <a href="#">Gestión de CSS</a>	Uso de la API .classList para manipular clases CSS desde Javascript.
 <a href="#">Contenido etiquetas</a>	Acceder y modificar el contenido de una etiqueta HTML del DOM.
 <a href="#">Insertar etiquetas</a>	Formas de añadir elementos en el DOM, como .appendChild() u otros.
 <a href="#">Navegar por etiquetas</a>	Métodos para «navegar» a través de la jerarquía del DOM.
 <a href="#">Animar elementos del DOM</a>	Aplicar animaciones CSS a elementos del DOM desde Javascript con animate().

# DOM

## Seleccionar elementos de DOM – Métodos tradicionales

Métodos de búsqueda	Descripción	Si no lo encuentra...
ELEMENT.getElementById(id)	Busca el elemento HTML por su id.	Devuelve .NULL
ARRAY.getElementsByClassName(class)	Busca elementos con la clase class.	Devuelve [].
ARRAY.getElementsByName(value)	Busca elementos con el atributo name a value.	Devuelve [].
ARRAY.getElementsByTagName(tag)	Busca etiquetas HTML tag.	Devuelve [].

# DOM

## getElementById()

El primer método, `.getElementById(id)` busca un elemento HTML con el id especificado. En principio, un documento HTML bien construído no debería tener más de un elemento con el mismo id, por lo tanto, este método devolverá siempre un solo elemento



```
const page = document.getElementById("page");
// <div id="page"></div>
```

# DOM

## getElementByClassName()

Por otro lado, el método `.getElementsByClassName(class)` permite buscar los elementos que tengan la clase especificada en `class`. Es importante darse cuenta del matiz de que el método tiene `getElements` en plural, es decir, puede devolver varias clases ya que al contrario que los `id`, pueden existir varios elementos con la misma clase

- □ ×

```
const items = document.getElementsByClassName("item"); // [div, div, div]
console.log(items[0]); // Primer item encontrado: <div class="item"></div>
console.log(items.length); // 3
```

# DOM

getElementsByName (name) – getElementsByTagName(tag)

– □ ×

```
// Obtiene todos los elementos con atributo  
name="nickname"  
const nicknames = document.getElementsByName("nickname");  
// [input]
```

```
// Obtiene todos los elementos <div> de la página  
const divs = document.getElementsByTagName("div");
```

# DOM

## Métodos Modernos

Método de búsqueda	Descripción	Si no lo encuentra...
<code>.querySelector(sel)</code>	Busca el primer elemento que coincide con el selector CSS sel.	Devuelve <code>.</code>
<code>.querySelectorAll(sel)</code>	Busca todos los elementos que coinciden con el selector CSS sel.	Devuelve <code>[]</code> .

# DOM

## QuerySelector()

- □ ×

```
const page = document.querySelector("#page");
// <div id="page"></div>
const info = document.querySelector(".main .info");
// <div class="info"></div>
```

# DOM

## QuerySelectorAll()

- □ ×

```
// Obtiene todos los elementos con clase "info"  
const infos = document.querySelectorAll(".info");
```

```
// Obtiene todos los elementos con atributo  
name="nickname"
```

```
const nicknames =  
document.querySelectorAll('[name="nickname"]');
```

```
// Obtiene todos los elementos <div> de la página  
HTML
```

```
const divs = document.querySelectorAll("div");
```

# DOM

## Busquedas acotadas

- □ ×

```
const menu = document.querySelector("#menu");
const links = menu.querySelectorAll("a");
//si controlamos un poco de CSS, esto puede
ser más sencillo
const links = document.querySelectorAll("#menu
a");
```

# DOM

NodeList o HTMLCollection

— □ ×

```
const elements =  
document.querySelectorAll("div");  
elements.map // undefined  
  
const elements =  
[ ... document.querySelectorAll("div")];  
elements.map // f map() { [native code] }
```

# DOM

Crear elementos en el DOM –  
Crear elementos HTML

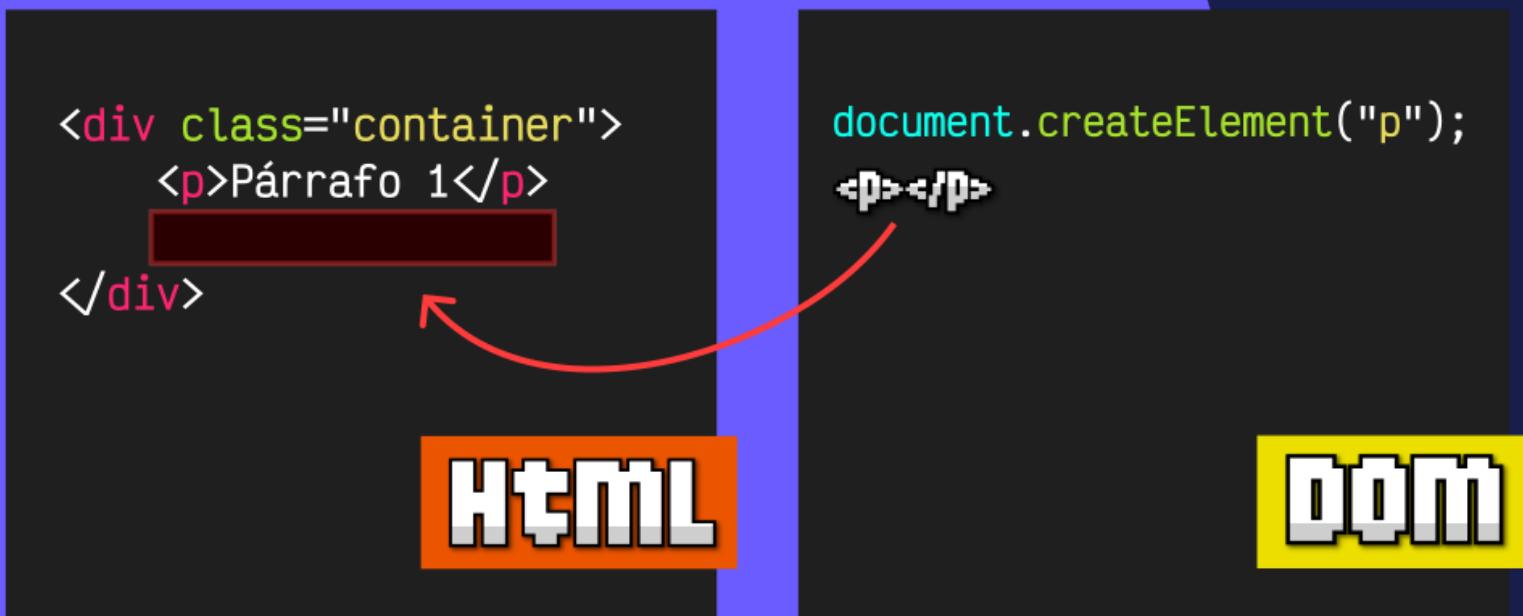
Métodos	Descripción
ELEMENT.createElement(tag, options)	Crea y devuelve el elemento HTML definido por el STRING tag.
NODE.createComment(text)	Crea y devuelve un nodo de comentarios HTML <!-- text -->.
NODE.createTextNode(text)	Crea y devuelve un nodo HTML con el texto text.
NODE.cloneNode(deep)	Clona el nodo HTML y devuelve una copia. deep es <b>false</b> por defecto.
BOOLEAN.isConnected	Indica si el nodo HTML está insertado en el documento HTML.

# DOM

</ >

## Crear elementos con createElement()

Mediante el método `.createElement()` podemos crear un HTML en memoria (¡no estará insertado aún en nuestro documento HTML!). Con dicho elemento almacenado en una variable o constante, podremos modificar sus características o contenido, para posteriormente insertarlo en una posición determinada del DOM o documento HTML.



# DOM

</ >

## Crear elementos con createElement()

Vamos a centrarnos en el proceso de creación del elemento, y más adelante veremos diferentes formas de insertarlo en el DOM. El funcionamiento de `.createElement()` es muy sencillo: se trata de pasarle 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>
```

# DOM

</ >

## Crear elementos con createElement()

Aunque menos frecuente, de la misma forma, podríamos crear comentarios HTML con `.createComment()` o fragmentos de texto sin etiqueta HTML con `.createTextNode()`, pasándole a ambos un STRING con el texto en cuestión. En ambos, se devuelve un NODE que podremos utilizar luego para insertar en el documento HTML:

```
const comment = document.createComment("Comentario"); // <!--Comentario-->
const text = document.createTextNode("Hola"); // Nodo de texto: 'hola'
```

# DOM

## El Método .cloneNode()

```
- □ ×  
  
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'  
  
const div = document.createElement("div");  
div.textContent = "Elemento 1";  
  
const div2 = div.cloneNode(); // Ahora SÍ estamos duplicando  
div2.textContent = "Elemento 2";  
  
div.textContent; // 'Elemento 1'
```

## La propiedad .isConnected

La propiedad `isConnected` nos indica si el elemento en cuestión está conectado al DOM, es decir, si está insertado en el documento HTML:

Si devuelve `true`, significa que el elemento está conectado al DOM.  
Si devuelve `false`, significa que el elemento no está conectado al DOM.

## Usando Fragmentos

En algunas ocasiones, nos puede resultar muy interesante utilizar fragmentos. Los fragmentos son una especie de documento paralelo, aislado de la página con la que estamos trabajando, que tiene varias características:

No tiene elemento padre. Está aislado de la página o documento.

Es mucho más simple y ligero (mejor rendimiento).

Si necesitamos hacer cambios consecutivos, no afecta al reflow (repintado de un documento).

Métodos	Descripción
<code>document.createDocumentFragment()</code>	Crea un fragmento aislado (sin padre).

## Usando Fragmentos

Así pues, el que devuelve el método `document.createDocumentFragment()` es un fragmento que podremos utilizar para almacenar en su interior un pequeño DOM temporal, que luego añadiremos en nuestro DOM principal.

```
const fragment = document.createDocumentFragment();

for (let i = 0; i < 5000; i++) {
  const div = document.createElement("div");
  div.textContent = `Item número ${i}`;
  fragment.appendChild(div);
}

document.body.appendChild(fragment);
```

## Usando Fragmentos

```
const fragment = document.createDocumentFragment();
```

Se crea un fragmento de documento. Un fragmento es un nodo del DOM que no está vinculado al árbol principal del documento. Puedes realizar operaciones en él sin afectar directamente la interfaz del usuario.

```
for (let i = 0; i < 5000; i++) { ... };
```

Se inicia un bucle for que se ejecutará 5000 veces.

```
const div = document.createElement("div");
```

En cada iteración del bucle, se crea un nuevo elemento div utilizando `document.createElement("div")`. Este es un nuevo nodo div en el DOM.

```
div.textContent = `Item número ${i}`;
```

Se establece el contenido de texto del nuevo div con el formato "Item número [i]".

```
fragment.appendChild(div);
```

Se agrega cada nuevo div al fragmento de documento utilizando `appendChild()`. Esto construye el contenido del fragmento sin afectar directamente la interfaz del usuario en cada iteración.

```
document.body.appendChild(fragment);
```

Después de que se completan las 5000 iteraciones, se agrega el fragmento completo al cuerpo del documento utilizando `appendChild()`.

En este punto, se realiza una única manipulación en el árbol del documento, lo que es más eficiente que agregar cada elemento individualmente en cada iteración del bucle.

</Be a  
coder>