Contenido

1 SELECCIÓN DE ELEMENTOS DEL DOM

Si nos encontramos en nuestro código Javascript y queremos hacer modificaciones en un elemento de la página HTML, lo primero que debemos hacer es buscar dicho elemento. Para ello, se suele intentar identificar el elemento a través de alguno de sus atributos más utilizados, generalmente el **id** o la **clase**.

1.1 MÉTODOS TRADICIONALES

Existen varios métodos, los más clásicos y tradicionales para realizar búsquedas de elementos en el documento. Observa que si lo que buscas es un elemento específico, lo mejor sería utilizar **getElementById()**, en caso contrario, si utilizamos uno de los 3 siguientes métodos, nos devolverá un **ARRAY** donde tendremos que elegir el elemento en cuestión posteriormente:

Métodos de búsqueda	Descripción
.getElementById(id)	Busca el elemento HTML con el id **id**. Si no, devuelve NULL .
.getElementsByClassName(class)	Busca elementos con la clase **class**. Si no, devuelve**[]**.
.getElementsByName(name)	Busca elementos con atributo name **name**. Si no, devuelve **[]**.
.getElementsByTagName(tag)	Busca elementos **tag**. Si no encuentra ninguno, devuelve **[]**.

Estos son los **4 métodos tradicionales** de Javascript para manipular el DOM. Se denominan tradicionales porque son los que existen en Javascript desde versiones más antiguas. Dichos métodos te permiten buscar elementos en la página dependiendo de los atributos **id**, **class**, **name** o de la propia **etiqueta**, respectivamente.

1.1.1 getElementById()

El primer método, **.getElementById(id)** busca un elemento HTML con el **id** especificado en **id** por parámetro. 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>
```

Recuerda que en el caso de no encontrar el elemento indicado, devolverá **NULL**.

1.1.2 getElementsByClassName()

Por otro lado, el método **.getElementsByClassName(class)** permite buscar los elementos con la **clase** especificada en **class**. Es importante darse cuenta del matiz de que el metodo tiene **getElements** en plural, y esto es porque al devolver **clases** (*al contrario que los id*) se pueden repetir, y por lo tanto, devolvernos varios elementos, no sólo uno.

```
const items = document.getElementsByClassName("item");  // [div, div, div]

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

Estos métodos devuelven siempre un **ARRAY** con todos los elementos encontrados que encajen con el criterio. En el caso de no encontrar ninguno, devolverán un **ARRAY** vacío: **[]**.

Exactamente igual funcionan los métodos **getElementsByName(name)** y **getElementsByTagName(tag)**, salvo que se encargan de buscar elementos HTML por su atributo **name** o por su propia **etiqueta** de elemento HTML, respectivamente:

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

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

OJO: Aunque en esta documentación se hace referencia a **ARRAY**, realmente los métodos de búsqueda generalmente devuelven un tipo de dato **HTMLCollection** o **NodeList**, que aunque actúan de forma muy similar a un **ARRAY**, no son arrays, y por lo tanto pueden carecer de algunos métodos, como por ejemplo: **.map().**

No obstante, si deseamos disponer de esos métodos y de otros, tendremos que convertir este tipo de objetos en un array, lo cual es posible mediante el operador de propagación:

```
let arrayElementosDiv=[…divs]
```

O también:

```
let arrayElementosDiv=Array.from(divs);
```

Recuerda que el primer método tiene **getElement** en singular y el resto **getElements** en plural. Ten en cuenta ese detalle para no olvidarte que uno devuelve un sólo elemento y el resto una lista de ellos.

1.2 MÉTODOS MODERNOS (POR SELECTOR CSS)

Aunque podemos utilizar los métodos tradicionales que acabamos de ver, actualmente tenemos a nuestra disposición dos nuevos métodos de búsqueda de elementos que son mucho más cómodos y prácticos si conocemos y dominamos los selectores CSS. Es el caso de los métodos **.querySelector()** y **.querySelectorAll()**:

Método de búsqueda	Descripción
.querySelector(sel)	Busca el primer elemento que coincide con el selector CSS **sel**. Si no, NULL.
.querySelectorAll(sel)	Busca todos los elementos que coinciden con el selector CSS **sel**. Si no, **[]**.

Con estos dos métodos podemos realizar todo lo que hacíamos con los **métodos tradicionales** mencionados anteriormente e incluso muchas más cosas (*en menos código*), ya que son muy flexibles y potentes gracias a los **selectores CSS**.

1.2.1 querySelector()

El primero, **.querySelector(selector)** devuelve el primer elemento que encuentra que encaja con el selector CSS suministrado en **selector**. Al igual que su «equivalente» **.getElementById()**, devuelve un solo elemento y en caso de no coincidir con ninguno, devuelve :

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

Lo interesante de este método, es que al permitir suministrarle un selector CSS básico o incluso un selector CSS avanzado, se vuelve un sistema mucho más potente.

El primer ejemplo es equivalente a utilizar un **.getElementById()**, sólo que en la versión de **.querySelector()** indicamos por parámetro un **selector** , y en el primero le pasamos un simple **string** . Observa que estamos indicando un **#** porque se trata de un **id**.

En el segundo ejemplo, estamos recuperando el primer elemento con clase **info** que se encuentre dentro de otro elemento con clase **main**. Eso podría realizarse con los métodos tradicionales, pero sería menos directo ya que tendríamos que realizar varias llamadas, con **.querySelector()** se hace directamente con sólo una.

1.2.2 querySelectorAll()

Por otro lado, el método **.querySelectorAll()** realiza una búsqueda de elementos como lo hace el anterior, sólo que como podremos intuir por ese **All()**, devuelve un **array** con todos los elementos que coinciden con el **selector** CSS:

```
// 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");
```

En este caso, recuerda que **.querySelectorAll()** siempre nos devolverá un **array** de elementos. Depende de los elementos que encuentre mediante el **selector**, nos devolverá un **array** de **0** elementos o de **1**, **2** o más elementos.

Al realizar una búsqueda de elementos y guardarlos en una variable, podemos realizar la búsqueda posteriormente sobre esa variable en lugar de hacerla sobre **document**. Esto permite realizar búsquedas acotadas por zonas, en lugar de realizarlo siempre sobre **document**, que buscará en todo el documento HTML.