educación IT
CODE YOUR FUTURE

JavaScript desde cero

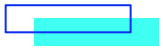
Módulo 4

Métodos de acceso a HTML

Métodos de acceso a HTML

Los tres métodos convencionales y más antiguos que nos **permiten acceder a uno o más *tags* HTML** son:

- `getElementById("id")`.
- `getElementsByTagName("tag")`.
- `getElementsByClassName("tag")`.



Salvo algunos casos muy puntuales, el acceso a elementos HTML desde JavaScript se realiza, generalmente, **de forma individual, directa**. Por esta razón, **el primero de los métodos** es el que utilizaremos o veremos aplicado con mayor frecuencia.

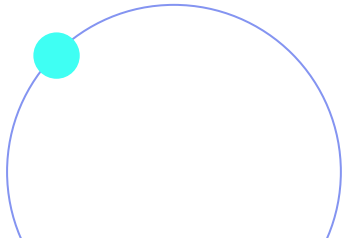
Veamos la tabla en la siguiente diapositiva, con el detalle de cada uno de los métodos.

Etiqueta	Descripción
`getElementById("id")`	Permite acceder a un **elemento HTML** desde JS. Este elemento HTML debe tener definido un ***tag* ID**, **y un nombre**, como valor, aplicado a este.
`getElementsByTagName("tag")`	Permite acceder a un **conjunto de elementos HTML del mismo tipo** desde JS, utilizando el nombre del ***tag* HTML**. Este método genera una colección de elementos HTML, que deberemos recorrer para trabajar con ella.
`getElementsByClassName("tag")`	Permite acceder a un **conjunto de elementos HTML de diferente o igual tipo** desde JS. Estos elementos deben contener una **clase CSS** específica, común a todos los elementos HTML. El método genera una colección de elementos HTML, que deberemos recorrer para trabajar con ella.

Pasos a seguir

1. Se agrega el atributo **`id`** con un valor descriptivo a uno o más *tags* HTML.

```
<h1 id="titulo">Título principal</h1>
<p id="parrafo">Lorem ipsum dolor, sit amet consectetur adipisicing elit.</p>
```



2. En el documento JavaScript, referenciamos los `id` a **dos constantes**, utilizando el objeto **`document`**, el método **`getElementById`**, y definimos **en los paréntesis** del método, el valor del **`id`** aplicado en los elementos HTML.

```
const titulo = document.getElementById("titulo");
const parrafo = document.getElementById("parrafo");
```

Para poder trabajar, desde JavaScript, con el texto definido en estos elementos HTML, existen **propiedades** que podremos utilizar referenciándolas junto a las constantes que definimos.

educación IT
CODE YOUR FUTURE

Propiedades JS para el acceso al contenido HTML

Propiedades de acceso a HTML

Las **tres propiedades más utilizadas para manipular contenido de elementos HTML** son:

- `innerText`.
- `textContent`.
- `innerHTML`.

Cada una aporta lo suyo y se utiliza para determinados casos específicos.

Veamos la tabla en la siguiente diapositiva, con el detalle de estas tres propiedades. Y en los posteriores *slides*, veremos ejemplos enfocados en cómo se utiliza cada una de ellas.

Propiedad	Descripción
`innerText`	Permite acceder, desde JS, al **texto definido dentro del elemento HTML**. Podemos leerlo y/o modificarlo, según nuestro interés.
`textContent`	Aporta el mismo resultado que la propiedad anterior, pero `textContent` es **más moderno** (*llegó en 2015*). Su aporte es poder ser **más claro**, en cuanto al nombre de la propiedad refiere, **a diferencia de `innerText`**, el cual solía confundirse su funcionalidad con el nombre de la siguiente propiedad que veremos.
`innerHTML`	Esta propiedad también permite leer el texto de un elemento HTML pero, como su nombre lo indica, realmente está enfocada a que podamos **leer/escribir uno o más bloques HTML en etiquetas del tipo *contenedor*** (como por ejemplo, el *tag* **`<div>`**).

innerText

Aquí vemos en acción a la propiedad **`innerText`**. Primero leemos el contenido del elemento HTML **`<h1>`**, y luego cambiamos el texto definido en éste, por un nuevo texto.

```
const titulo = document.getElementById("titulo");

//muestra en la consola JS el texto del elemento HTML 'Título principal'
console.log(titulo.innerText);

//cambia en el documento HTML el texto de este elemento
titulo.innerText = "El nuevo título principal";
```

textContent

Ejemplo de la propiedad **`textContent`**. Llegamos al mismo resultado que con la propiedad `innerText`, aunque se recomienda en la actualidad utilizar `textContent`. Si invertimos el uso de las propiedades en los dos elementos HTML intervenidos, llegaremos al mismo resultado.

```
const parrafo = document.getElementById("parrafo");

//muestra en la consola JS el texto del elemento HTML 'Lorem ipsum...'
console.log(parrafo.textContent)

//cambia en el documento HTML el texto de este elemento
parrafo.textContent = "Agregamos un nuevo párrafo, que no sea Lorem Ipsum.";
```

innerHTML

Imaginemos que tenemos un ** **, el cual oficia como *tag* contenedor de un elemento HTML del tipo **<h2>** (subtítulo), y otro elemento **<p>**.

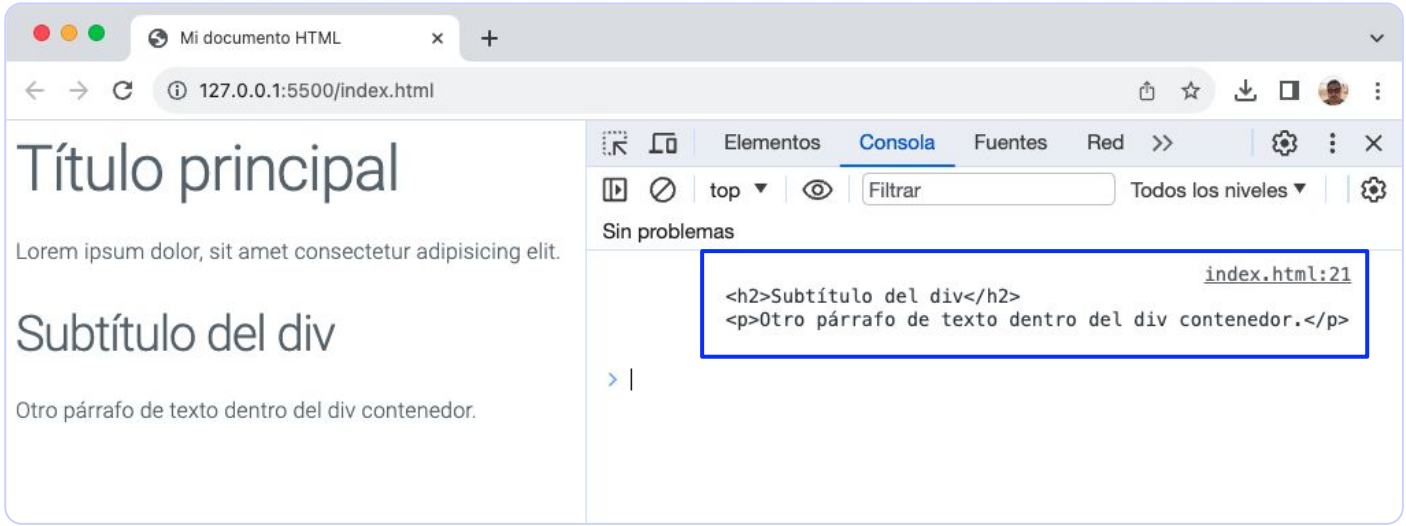
En el ** **, definimos el atributo ***ID***, con un valor para luego enlazarnos desde JS.

```
<div id="contenedor">
    <h2>Subtítulo del div</h2>
    <p>Otro párrafo de texto dentro del div contenedor.</p>
</div>
```

```
const divContenedor = document.getElementById("contenedor");

console.log(divContenedor.innerHTML);
```

En la **consola JS** vemos el resultado de la propiedad **`innerHTML`**:



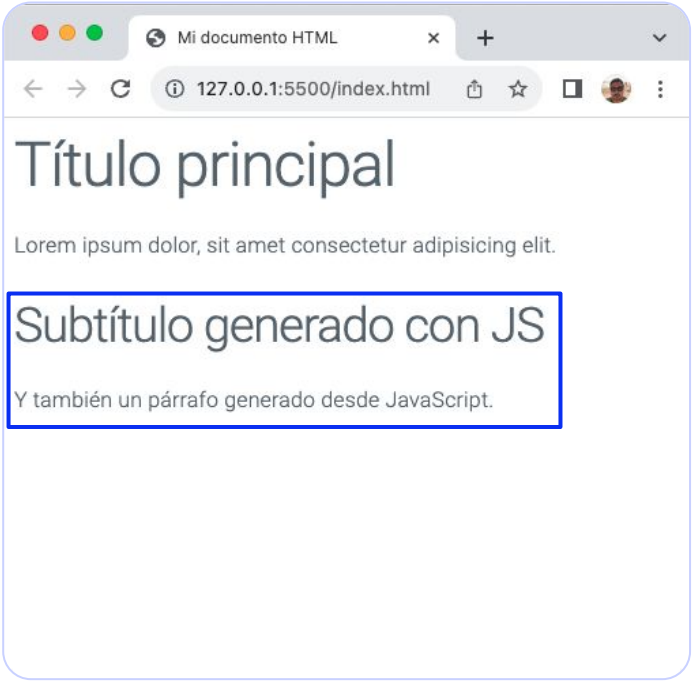
`innerHTML`, a diferencia de las otras propiedades, está pensada para **escribir bloques de código HTML desde JS, de forma dinámica**. De esta forma, se pueden generar uno o más elementos HTML en bloque, de forma dinámica, mediante el atributo `innerHTML` para escribirlos en el documento HTML.

Además, **podremos concatenar** (en el HTML que vamos a generar) variables, constantes o *arrays* JS, para así poder mostrar su valor en el documento HTML. Esto es lo que se denomina: ***la Web Dinámica***.

```
divContenedor.innerHTML = "<h2>Subtítulo generado con JS</h2>" +
                          "<p>Y también un párrafo generado desde JavaScript.</p>"
```

Aquí vemos **el resultado** de nuestra “intervención artística”, creada íntegramente con JS.

Tal como escribimos dos elementos HTML simples, podremos escribir *Cards* HTML completas, múltiples filas en tablas HTML, generar imágenes dinámicas, insertar videos dinámicos en el HTML, de acuerdo a la selección del usuario, y un sinfín de acciones más; comportamientos propios de la web moderna.



educación IT
CODE YOUR FUTURE

El método *querySelector*

Selectores

CSS utiliza selectores para **personalizar los estilos gráficos de los elementos HTML**. Estos selectores se dividen en tres tipos principales:

- ***Elemento***.
- ***ID***.
- ***Clase genérica***.

Veamos la tabla en la siguiente diapositiva, con el detalle de cada uno de ellos.

```css
h2 {
    color: blue;
}

#titulo {
    color: orangered;
}

.clase-generica {
    border-radius: 10px;
    box-shadow: 10px 6px 6px darkslategray;
    padding: 20px;
    width: 440px;
    background-color: lightgray;
}
```

Tipo de selector	Descripción
Elemento	El selector CSS del tipo **elemento**, permite definir estilos gráficos sobre un elemento HTML. Todos los *tags* HTML agregados, serán representados gráficamente con los estilos CSS definidos sobre éstos.
ID	El selector CSS **ID** permite configurar estilos gráficos sobre el elemento HTML que contenga el ID específico. En CSS los debemos referenciar al selector anteponiendo un carácter *hash* '**#**' al nombre.
Clase genérica	El selector CSS **clase genérica**, permite definir una clase genérica, *o agnóstica*, con determinados estilos CSS. Luego, esta clase genérica puede aplicarse en todos los elementos HTML que consideremos estilizar, indistintamente sean del mismo tipo, o no. Se debe referenciar a las clases genéricas anteponiendo un punto '**.**' a su nombre.

```css
#titulo {
    color: orangered;
}
```

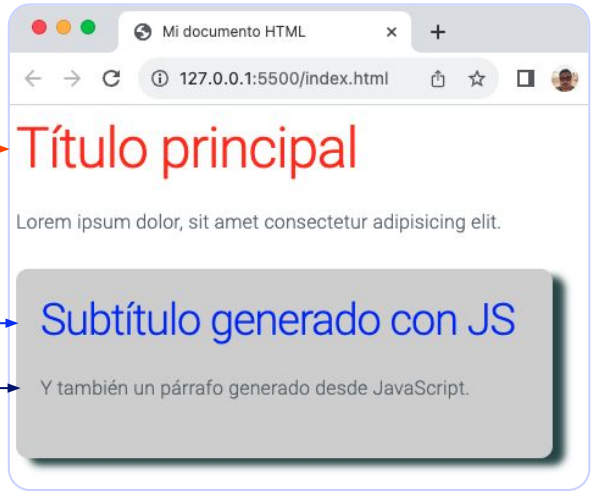
El selector **#titulo**, aplica el color en el título principal.

```css
h2 {
    color: blue;
}
```

El selector **h2**, aplica el color en el subtítulo.

```css
.clase-generica {
    border-radius: 10px;
    box-shadow: 10px 6px 6px darkslategray;
    padding: 20px;
    width: 440px;
    background-color: lightgray;
}
```

.clase-genérica aplica todos los estilos CSS sobre el **DIV** contenedor.



Ejemplos

Aquí tenemos tres ejemplos donde se implementa el método **`querySelector()`**, para enlazar con elementos HTML a través de sus diferentes variantes.

```
const subtitulo = document.querySelector("h2");
const titulo = document.querySelector("#titulo");
const divContenedor = document.querySelector(".clase-generica");


console.log(subtitulo.textContent);
//retorna el texto del H2

console.log(titulo.textContent);
//retorna el texto del H1

console.log(divContenedor.innerHTML);
//retorna el contenido del DIV, en formato HTML.
```

Si bien este método cumple un rol similar al de `getElementById()`, `querySelector()` **da mucha más flexibilidad para enlazar con elementos HTML desde JS**, a través de diferentes posibles opciones.

Así, se evita tener que estar agregando un ID unívoco sobre cada elemento HTML con el que se necesita trabajar desde JS.

Veamos otros ejemplos a continuación:

```
<h2 class="color-naranja">Subtítulo del div</h2>
<p class="color-naranja">Otro párrafo de texto dentro del div contenedor.</p>
```

```
const parrafo = document.querySelector("p.color-naranja");
```

Como se muestra en los códigos de la diapositiva anterior, tenemos dos elementos HTML; **`<h2>`** y **`<p>`**, que utilizan una misma clase CSS, y queremos enlazar con el elemento del tipo **`<p>`** que posee la clase CSS **`color-naranja`**, el método **`.querySelector()`** permite lograrlo. Se especifica, en sus paréntesis, que deseamos enlazar con un *tag* HTML del tipo **`p`**, que contiene la clase CSS **`.color-naranja`**.

Esta **precisión** hace que **`.querySelector()` sea más efectivo que `.getElementById()`** y por ello, sea el recomendado actualmente.

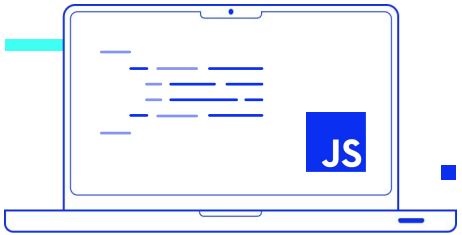
educación IT
CODE YOUR FUTURE

Trabajar con múltiples elementos HTML desde JS

Trabajar con múltiples elementos HTML desde JS

En las siguientes diapositivas, veremos ejemplos de implementación de los siguientes métodos:

- `.getElementsByTagName()`.
- `.getElementsByClassName()`.
- `.querySelectorAll()`.



.getElementsByTagName()

El ***tag* HTML `<ul>`** *(unordered list)*, es un ***tag* contenedor que permite generar una lista con viñetas en HTML**.

Cada ítem que compone la lista, debe estar representado por el *tag* **`<li>`** (*list item*) y definido dentro del *tag* contenedor `<ul>`.

Tenemos la necesidad de enlazar, desde JS, con todos los `<li>`, para transformar el texto, de cada uno de los ítems, a mayúsculas.

```
<ul>
    <li>Notebook</li>
    <li>Macbook</li>
    <li>Mouse bluetooth</li>
    <li>Teclado bluetooth</li>
    <li>Monitor LED UHD</li>
    <li>Base para portátiles con Cooler</li>
</ul>
```

Pasos a seguir:

1. El método **`.getElementsByTagName()`** genera una colección HTML de elementos del tipo definido entre los paréntesis del método.

```
const listItems = document.getElementsByTagName("li");
// creamos una Colección HTML de elementos <li>, iterable.
```

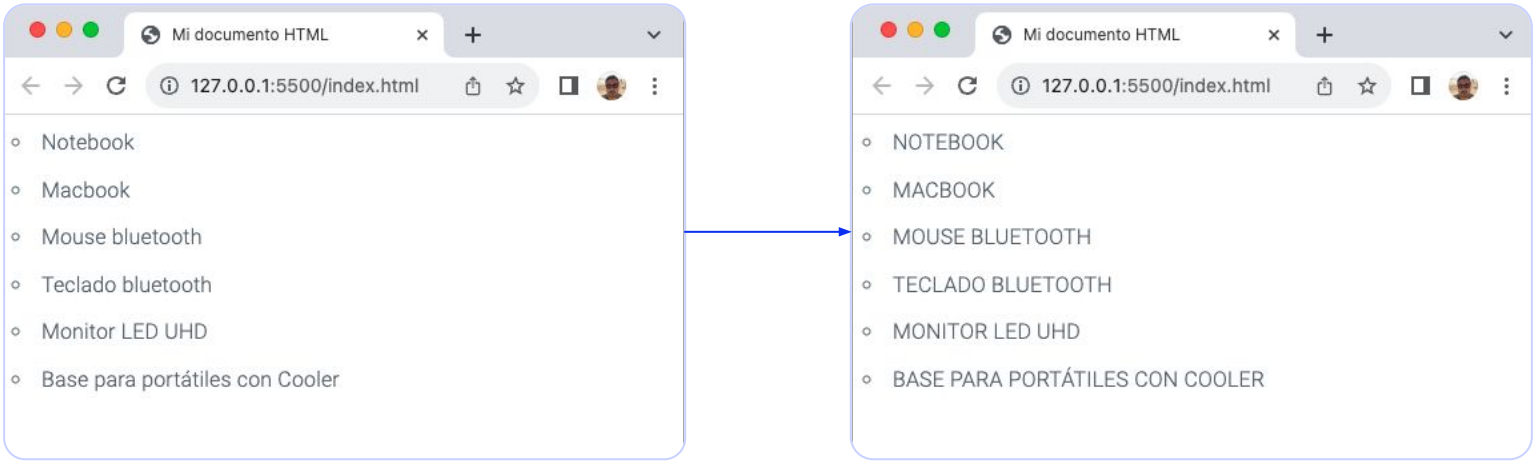
2. Luego de enlazar con los elementos `<li>`, recurrimos al ciclo **`for...of`**, **para recorrer la colección**.

3. La variable definida dentro del **`for()`** tomará el valor del elemento HTML en cada iteración. De esta forma, se podrá acceder al texto, e indicarle cuál es el nuevo valor del texto que deseamos que tenga.

```
const listItems = document.getElementsByTagName("li");

for (let elemento of listItems) {
    elemento.textContent = elemento.textContent.toUpperCase();
}
```

Así de rápido pasamos de una lista con texto en formato *Capital*, a una lista de elementos con **texto en mayúsculas**.



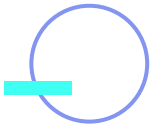
.getElementsByClassName()

En este otro ejemplo, nos encontramos con la necesidad de **enlazar con todo elemento HTML** que posea una **clase genérica CSS** denominada '**`texto-azul`**', indistintamente del tipo de elemento HTML.

Para ello, podemos utilizar el método **`.getElementsByClassName()`**, el cual nos retornará una colección de los elementos HTML que utilicen dicha clase CSS.

```
const claseTextoAzul = document.getElementsByClassName("texto-azul");
```

En este ejemplo, dicha colección contendrá un elemento **<h1>**, un elemento **<h2>**, un elemento **<p>** y el segundo elemento **<li>**.



```
<h1 class="texto-azul" id="titulo">Título principal</h1>
<p id="parrafo">Lorem ipsum dolor, sit amet consectetur
adipisicing elit.</p>

<div id="contenedor" class="clase-generica">
    <h2 class="texto-azul">Subtítulo del div</h2>
    <p class="texto-azul">Otro párrafo de texto
                          dentro del div contenedor.</p>
</div>
<br>
<div>
    <ul>
        <li>Notebook</li>
        <li class="texto-azul">Macbook</li>
        <li>Mouse bluetooth</li>
        <li>Teclado bluetooth</li>
        <li>Monitor LED UHD</li>
        <li>Base para portátiles con Cooler</li>
    </ul>
</div>
```

.querySelectorAll()

El método **`.querySelectorAll()`** permite lograr exactamente lo mismo que los otros dos, con la diferencia de que podemos **aprovechar su mayor precisión**.

Si, por ejemplo, tenemos dos *tag* **`<ul>`** diferentes en el documento HTML, y necesitamos enlazarnos con los *tags* **`<li>`** de solo uno de ellos, podemos agregarle un atributo ***ID*** al *tag* `<ul>` en cuestión y sumarlo a la referencia del método `.querySelectorAll()`, escribiendo:

```
document.querySelectorAll(“ul#computacion li”);
```

Así nos enlazará con todos los *tags* **`<li>`**, que estén contenidos en un *tag* **`<ul>`** y que a su vez, este tenga el atributo **`id=”computacion”`**.

Veamos el código de la siguiente diapositiva.

```
// UTILIZANDO QUERY SELECTOR ALL()
const listItems = document.querySelectorAll("ul li");

const claseTextoAzul = document.querySelectorAll("texto-azul");
```

La misma lógica podemos aplicar si nos queremos enlazar con todos los elementos que usen la clase '`texto-azul`', pero que solamente sean *tags* HTML del tipo **<p>**:

```
document.querySelectorAll(“p.texto-azul”);
```

Resumen

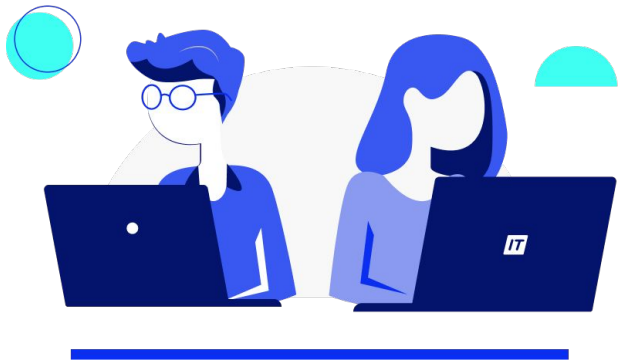
Resumen: el DOM HTML desde JavaScript

En resumen; **el DOM HTML es totalmente manipulable desde JS, para generar contenido dinámico**, de acuerdo a nuestra necesidad.

- Los archivos *scripting* de JS que deben interactuar con el DOM, deben tener el *tag* **`defer`** y ser referenciados dentro del apartado **`<head>`** de un documento HTML.

- El objeto **`document`** cuenta con el método **`.getElementById()`** para enlazarnos con un elemento HTML por su atributo ***id*** y, **`.querySelector()`**, para enlazar con un elemento HTML aprovechando múltiples posibles combinaciones de selectores.
- Las propiedades **`.innerText`** y **`.textContent`**, permiten interactuar con el texto de elementos HTML desde JS y la propiedad **`.innerHTML`** permite leer y/o generar bloques de código HTML dinámico.

- Finalmente, **`.getElementsByTagName()`** - **`.getElementsByClassName()`** y **`.querySelectorAll()`**, son métodos que nos permiten enlazar con múltiples elementos HTML, creando una colección, o *array*, para iterarlo y aplicar múltiples modificaciones de forma simultánea.



Revisión

- Repasar el concepto de ***objeto*** en JavaScript.
- Implementar **`getElementById()`** y **`querySelector()`** para acceder al DOM e interactuar con elementos HTML, desde JS.
- Avanzar con el *output* a través de las propiedades **`innerText`**, **`textContent`**, e **`innerHTML`**.
- Trabajar en el ***Proyecto integrador***.
- Realizar las preguntas necesarias a la/el docente antes de continuar.

¡Sigamos trabajando!