

7. BOM y DOM

G. Modificar el DOM

1. MANIPULACIÓN DE ATRIBUTOS

OBTENER EL VALOR DE UN ATRIBUTO

- Los elementos del DOM disponen del método **getAttribute** para obtener el valor de un atributo concreto.
- Por ejemplo:

```
<ul>
  <li class="hardware">Teclado</li>
  <li class="software">Windows</li>
  <li class="hardware">Raton</li>
  <li class="hardware">CPU</li>
  <li class="software">Office</li>
</ul>
```

HTML

JS

```
var lista=document.querySelectorAll("li");
for (let elemento of lista ) {
  console.log(elemento.getAttribute("class"));
}
```

- Teclado
- Windows
- Raton
- CPU
- Office

	Elements	Console
	hardware	
	software	
2	hardware	
	software	

1. MANIPULACIÓN DE ATRIBUTOS

MODIFICAR VALOR DE UN ATRIBUTO

- El método **setAttribute** es el que permite esta operación.
- El primer parámetro será el nombre del elemento y el segundo el nuevo valor.

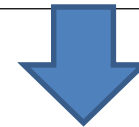
```
<ul>
  <li class="hardware">Teclado</li>
  <li class="software">Windows</li>
  <li class="hardware">Raton</li>
  <li class="hardware">CPU</li>
  <li class="software">Office</li>
</ul>
```

HTML



JS

```
var lista=document.querySelectorAll("li");
for (let elemento of lista ) {
  elemento.setAttribute("class","informática");
  console.log(elemento.getAttribute("class"));
}
```



• Teclado
• Windows
• Raton
• CPU
• Office

Elements Console

top

5 informática

1. MANIPULACIÓN DE ATRIBUTOS

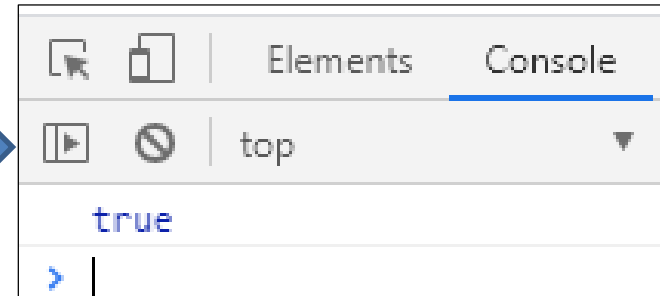
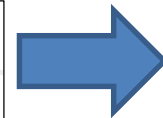
SABER SI UN ELEMENTO TIENE ATRIBUTOS

- El método **hasAttributes** devuelve verdadero si el elemento tiene el atributo cuyo nombre se indica y falso si no los tiene.

```
<li class="hardware" id="teclado">Teclado</li>
```



```
var teclado=document.querySelector("#teclado");  
console.log(teclado.hasAttributes());
```



1. MANIPULACIÓN DE ATRIBUTOS

OBTENER TODOS LOS ATRIBUTOS DE UN ELEMENTO

- La propiedad `attributes` de un elemento permite obtener todos los atributos de un elemento.
- Podemos acceder mediante el bucle `"for ... of"` o usar la propiedad `length`.

```
<input type="number" name="edad" id="edad" min="0" max="199" placeholder="Edad">
```



```
var edad=document.querySelector("#edad").attributes;  
  
for (let elemento of edad) {  
    document.write("<strong>Atributo:</strong>" + elemento.name);  
    document.write(", <strong>Valor:</strong>" + elemento.value + "<br>");  
}
```



Edad

Atributo:type, Valor:number
Atributo:name, Valor:edad
Atributo:id, Valor:edad
Atributo:min, Valor:0
Atributo:max, Valor:199
Atributo:placeholder, Valor:Edad

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

2.1. PROPIEDAD TEXTCONTENT

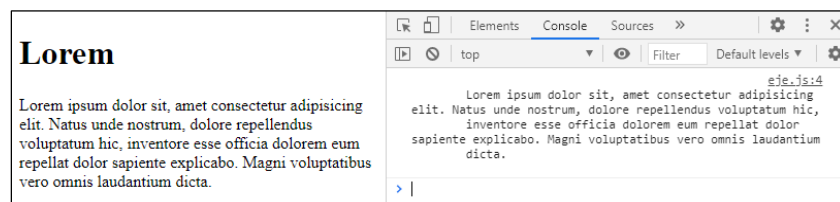
- La propiedad `textContent` de un elemento permite obtener y modificar el texto que contiene ese elemento. Hay que tener en cuenta que un elemento puede tener dentro más elementos y que esta propiedad obtendrá el texto de todos ellos. Por ejemplo, si observamos este código:

```
<h1>Lorem</h1>

<p id="lorem">
  Lorem ipsum dolor sit, amet consectetur adipisicing elit. Natus unde nostrum, dolore
  repellendus voluptatum hic,
  inventore esse officia dolorem eum repellat dolor sapiente explicabo. Magni voluptatibus vero
  omnis laudantium
  dicta.
</p>
```



```
var lorem=document.querySelector("#lorem").textContent;
console.log(lorem);
```



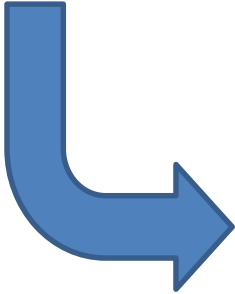
2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

- **Textcontent** no soporta etiquetas dentro, solo recoge el texto interior.
- En el siguiente ejemplo lo vamos a ver:

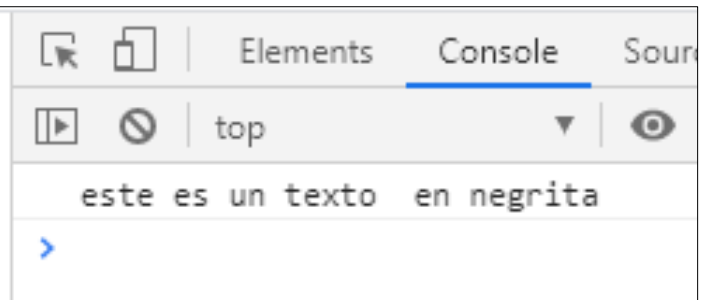
```
<div id="texto">este es un texto <strong> en negrita</strong></div>
```



```
var div=document.querySelector("#texto").textContent;  
console.log(div);
```



este es un texto **en negrita**



2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

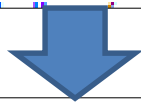
- Que es el texto del párrafo indicado. Sin embargo, dentro de ese elemento hay otro de tipo strong .
- Su texto aparece también en el resultado a la vez que la propia etiqueta strong desaparece del resultado.
- Es decir, solo recoge el texto interior (todo el texto) y no las etiquetas

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

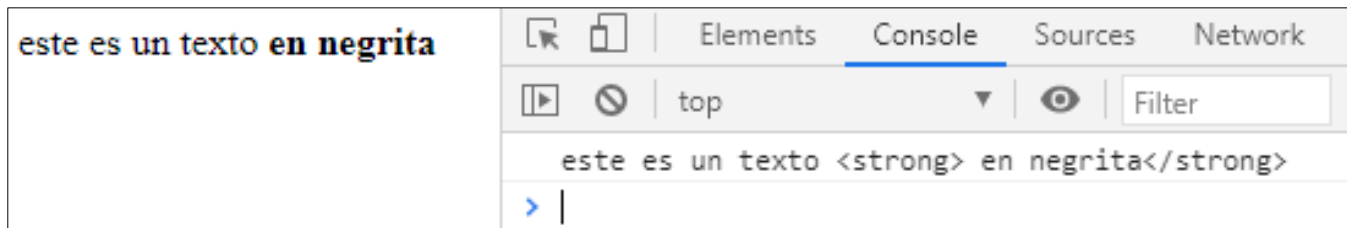
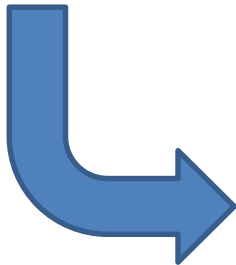
2.2. PROPIEDAD INNERHTML

- Se trata de una propiedad similar a la anterior, pero esta si lee y manipula las etiquetas HTML .
- Por ejemplo:

```
<div id="texto">este es un texto <strong> en negrita</strong></div>
```



```
var div=document.querySelector("#texto").innerHTML;  
console.log(div);
```



2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

2.3. MODIFICAR CSS

- Es posible modificar el CSS de los elementos a través de los atributos `style` o `class` mediante el método `setAttribute` visto anteriormente.
- Pero al ser un elemento tan importante, JavaScript proporciona métodos especiales para manipular las clases CSS de un elemento.

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

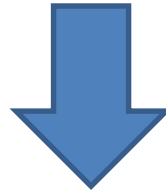
2.3.1. PROPIEDAD STYLE

- Los elementos poseen una propiedad llamada **style** que permite acceder a las propiedades CSS de un elemento. Lo que realmente hace es modificar el atributo **style** del elemento, modifican los valores de la misma.
- Veámoslo con ejemplos.

```
<p> Este es el primer párrafo</p>  
<p> Este es el segundo párrafo</p>  
<p> Este es el tercer párrafo</p>
```

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

```
var losp = document.querySelectorAll("p"); // array de elementos p
for (let elemento of losp) {
  elemento.style.color = "red";
  elemento.style.border = "1px solid black";
}
```



Este es el primer párrafo

Este es el segundo párrafo

Este es el tercer párrafo

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

- Y así con todas las propiedades. El problema viene si hubiéramos querido modificar solo el borde inferior. La propiedad CSS para hacer esa modificación se llama **border-bottom**, es un nombre problemático para la sintaxis anterior, porque hay un guión en él.
- En JavaScript los identificadores de propiedades, funciones, métodos, etc., no pueden tener guiones (se confunde con la resta).
- Lo mismo ocurre la propiedad que modifica el color de fondo: **background-color**.

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

- **background-color** se convierte en **backgroundColor**,
- **border-bottom** en **borderBottom**,
- **text-align** en **textAlign**
- y así con todas las propiedades. A esto se le conoce como notación en **CamelCase**.
- Por lo que modificar el color de fondo se hace así:

```
elemento.style.backgroundColor="yellow";|
```



Este es el primer párrafo

Este es el segundo párrafo

Este es el tercer párrafo

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

- No obstante, la mayoría de navegadores actuales acepta también el formato idéntico a CSS a través de corchetes:

```
elemento.style["background-color"]="yellow";
```



Este es el primer párrafo

Este es el segundo párrafo

Este es el tercer párrafo

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

- No obstante, hay varios problemas a tener en cuenta a la hora de usar `style` para modificar el CSS:
 - Es más fácil de mantener el código si se usan clases para aplicar CSS. Especialmente, cuando queremos modificar varias propiedades CSS a la vez.
 - CSS avanza muy rápido por lo que muchas propiedades CSS ya disponibles, en algunos navegadores no lo están a través de la propiedad `style` de JavaScript. Aunque la mayoría de navegadores se ponen al día rápido.
 - No podemos consultar a través de esta propiedad las propiedades CSS que tiene un elemento al cual se le ha dado formato a través de hojas de estilo externas.

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

2.4. OBTENER ATRIBUTOS DATA

- En los documentos HTML existe la posibilidad de crear atributos **data**. Se trata de un tipo de atributos creados por los propios desarrolladores a voluntad.
- Se usan para almacenar información que puede ser manipulada desde JavaScript.
- Los atributos data empiezan por ese mismo término (data) seguidos de un guion y luego un nombre en sí que queramos data al atributo. Por ejemplo:

2. MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

```
<article id="electriccars" data-columns="3" data-index-number="12314" data-parent="cars">

</article>
```

- La propiedad **dataset** es la encargada de conseguirlo. Podemos recuperar los valores data de HTML. Al igual que con la propiedad style se utiliza una notación **camelCase**.

```
var article = document.querySelector('#electriccars');

console.log(article.dataset.columns); // "3"
console.log(article.dataset.indexNumber); // "12314"
console.log(article.dataset.parent); // "cars"
```



3
12314
cars

3. AÑADIR ELEMENTOS

3.1. CREAR ELEMENTOS

- El método **createElement** del objeto document es el que permite crear elementos. Requiere que indiquemos el nombre de la etiqueta de dicho elemento:

```
let capa1=document.createElement("div");
```

- La variable `capa1` es la referencia al nuevo elemento. Este elemento no se mostrará todavía en el documento, porque realmente no forma parte de los nodos visibles al no haber indicado su posición en document.
- Tiene disponibles las propiedades habituales de los elementos.

```
capa1.innerHTML("<p> Estoy dentro de la nueva capa<p>");  
capa1.setAttribute("id","capa");
```

3. AÑADIR ELEMENTOS

3.2. AÑADIR UN NODO HIJO

- El método **appendChild** permite colocar un nodo como hijo de otro elemento. Es un método que poseen todos los nodos del DOM y que tiene como único parámetro el nodo que deseamos colocar como hijo.

```
<div id="principal">  
  <h1>Estoy dentro del div principal</h1>  
</div>
```



```
let capa1=document.createElement("div");  
capa1.innerHTML="<p> Estoy dentro de la nueva capa</p>";  
capa1.setAttribute("id","capa");  
  
let principal=document.querySelector("#principal");  
principal.appendChild(capa1);
```

Estoy dentro del div principal

Estoy dentro de la nueva capa



```
▼ <div id="principal"> == $0  
  <h1>Estoy dentro del div principal</h1>  
  ▼ <div id="capa">  
    <p> Estoy dentro de la nueva capa</p>  
  </div>  
</div>
```

3. AÑADIR ELEMENTOS

3.3. INSERCIÓN DE NODOS EN POSICIONES CONCRETAS

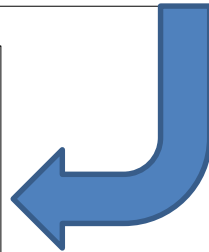
- El método anterior tiene como inconveniente, que el elemento que deseamos añadir irá siempre colocado al final.
- Por ello disponemos de otro método llamado **insertBefore** que permite indicar la posición en la que queremos colocar el elemento. Para ello el nodo que será padre del que queremos añadir, debe de tener nodos hijo. De otro modo deberíamos usar **appendChild**.
- El método **insertBefore** tiene dos parámetros: el primero es el nodo que deseamos añadir y el segundo es el nodo hijo delante del cual quedará colocado el que deseamos añadir.

3. AÑADIR ELEMENTOS

```
<div id="padre">  
  <p>Un párrafo.</p>  
  <p>Otro párrafo.</p>  
</div>
```



```
// Creamos el nuevo párrafo  
var nuevo_parrafo = document.createElement('p');  
nuevo_parrafo.innerHTML="<p>Este parrafo ha sido invitado</p>";  
  
// Recogemos en una variable el segundo párrafo  
var segundo_p = document.querySelectorAll('#padre p')[1]; //segundo parrafo  
  
// Y ahora lo insertamos  
document.querySelector('#padre').insertBefore(nuevo_parrafo,segundo_p);  
|
```



Un párrafo.
Este parrafo ha sido invitado
Otro párrafo.

4. REEMPLAZAR ELEMENTOS

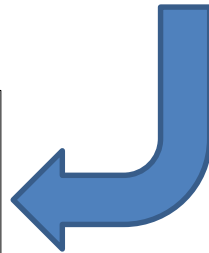
- El método **replaceChild** es agresivo. Permite cambiar un nodo por otro.
- Necesita dos parámetros: primero el nuevo nodo que se desea colocar y después el nodo que se desea reemplazar.
- El nuevo nodo reemplaza el antiguo.

4. REEMPLAZAR ELEMENTOS

```
<div id="padre">  
  <p>Un párrafo.</p>  
  <p>Otro párrafo.</p>  
</div>
```



```
// Creamos el nuevo párrafo  
var nuevo_parrafo = document.createElement('p');  
nuevo_parrafo.innerHTML="<p>Este parrafo ha sido invitado</p>";  
  
// Recogemos en una variable el segundo párrafo  
var segundo_p = document.querySelectorAll('#padre p')[1]; //segundo parrafo  
  
// Y ahora lo modificamos  
document.querySelector('#padre').replaceChild(nuevo_parrafo,segundo_p);
```



Un párrafo.

Este parrafo ha sido invitado

5. ELIMINAR ELEMENTOS

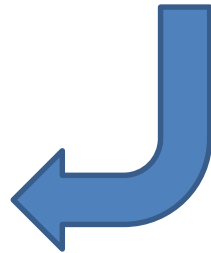
- El método **removeChild** es un método de los nodos que recibe como único parámetro el nodo a eliminar.
- Ese nodo no se elimina realmente de la memoria solo se retira del árbol de elementos, pero permanece en memoria.
- En este ejemplo se quitaría el segundo párrafo y se colocaría al final con ayuda de **appendChild**.

5. ELIMINAR ELEMENTOS

```
<div id="padre">  
  <p>Primera línea</p>  
  <p>Segunda línea</p>  
  <p>Tercera línea</p>  
</div>
```



```
let capa=document.querySelector("#padre");  
let segundoparrafo=document.querySelectorAll("#padre p")[1];  
  
capa.removeChild(segundoparrafo);  
capa.appendChild(segundoparrafo);
```



Primera línea

Tercera línea

Segunda línea