

Desarrollo Web en Entorno Cliente

Tema 9





[styleSheets]

Para obtener las propiedades de estilo de una página web, se puede usar la propiedad `document.styleSheets`

`document.styleSheets` devuelve un array de tipo `StyleSheetList` que contiene como elementos las diferentes hojas de estilo que están aplicadas a la página web.

Así, para acceder a la primera hoja de estilo definida en el documento se realiza a través del índice 0, `document.styleSheets[0]`.

```
const estilo=document.styleSheets;  
console.log(estilo);
```

```
StyleSheetList {0: CSSStyleSheet, 1: CSSStyleSheet, 2: CSSStyleSheet,  
▼ 3: CSSStyleSheet, 4: CSSStyleSheet, 5: CSSStyleSheet, 6: CSSStyleShee  
t, 7: CSSStyleSheet, length: 8} ⓘ  
▼ 0: CSSStyleSheet  
  ▶ cssRules: CSSRuleList {0: CSSStyleRule, 1: CSSStyleRule, 2: CSSStyl  
    disabled: false  
    href: "https://handmadegames.es/dwec/css/normalize.css"  
  ▶ media: MediaList {length: 0, mediaText: ''}  
  ▶ ownerNode: link  
    ownerRule: null  
    parentStyleSheet: null  
  ▶ rules: CSSRuleList {0: CSSStyleRule, 1: CSSStyleRule, 2: CSSStyleRu  
    title: null  
    type: "text/css"  
  ▶ [[Prototype]]: CSSStyleSheet  
  ▶ 1: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF  
  ▶ 2: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF  
  ▶ 3: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF  
  ▶ 4: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF  
  ▶ 5: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF  
  ▶ 6: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF  
  ▶ 7: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF  
    length: 8  
  ▶ [[Prototype]]: StyleSheetList
```

[styleSheets]

Cada elemento de *StyleSheetList* contiene a su vez otros elementos, entre ellos otros arrays, como *cssRules* que se corresponde con los selectores en la hoja de estilo.

```
StyleSheetList {0: CSSStyleSheet, 1: CSSStyleSheet, 2: CSSStyleSheet,
▼ 3: CSSStyleSheet, 4: CSSStyleSheet, 5: CSSStyleSheet, 6: CSSStyleShee
t, 7: CSSStyleSheet, Length: 8} ⓘ
  ► 0: CSSStyleSheet {ownerRule: null, cssRules: CSSRuleList, rules: CSSF
  ▼ 1: CSSStyleSheet
    ▼ cssRules: CSSRuleList
      ► 0: CSSStyleRule {selectorText: 'header', style: CSSStyleDeclarati
      ► 1: CSSStyleRule {selectorText: 'section', style: CSSStyleDeclarat
      ► 2: CSSStyleRule {selectorText: 'article', style: CSSStyleDeclarat
      ► 3: CSSStyleRule {selectorText: 'footer', style: CSSStyleDeclarati
      ▼ 4: CSSStyleRule
        ► cssRules: CSSRuleList {length: 0}
        ► cssText: "body { background-color: rgb(224, 215, 81); font-size
          parentRule: null
        ► parentStyleSheet: CSSStyleSheet {ownerRule: null, cssRules: CSS
          selectorText: "body"
        ► style: CSSStyleDeclaration {0: 'background-color', 1: 'font-siz
        ► styleMap: StylePropertyMap {size: 2}
          type: 1
        ► [[Prototype]]: CSSStyleRule
      ► 5: CSSStyleRule {selectorText: 'h1, h2', style: CSSStyleDeclarati
      ► 6: CSSStyleRule {selectorText: 'nav ul', style: CSSStyleDeclarati
      ► 7: CSSStyleRule {selectorText: 'nav li', style: CSSStyleDeclarati
      ► 8: CSSStyleRule {selectorText: 'nav li a', style: CSSStyleDeclara
      ► 9: CSSStyleRule {selectorText: 'figcaption', style: CSSStyleDecla
```

```
1 header{
2   border:1px solid grey;
3   border-radius:2rem;
4   padding:2rem;
5   padding-top:2rem;;
6   background-color:white;
7   margin:2rem;
8 }
9
10 section{
11   border:1px solid grey;
12   border-radius:2rem;
13   padding:2rem;
14   padding-top:2rem;;
15   background-color:white;
16   margin:2rem;
17 }
18
19 article{
20   margin-top: -1rem;
21   padding-top: 2rem;
22 }
23
24 footer{
25   border:1px solid grey;
26   border-radius:2rem;
27   padding:2rem;
28   padding-top:2rem;;
29   background-color:white;
30   margin:2rem;
31 }
32
33 body{
34   background-color:rgb(224, 215, 81);
35   font-size: 2rem;
36 }
```

[styleSheets]

Se pueden modificar los elementos de *StyleSheetList* para cambiar el estilo de la página web la hoja de estilo en la hoja de estilo.

```
▼ 1: CSSStyleSheet
  ▼ cssRules: CSSRuleList
    ▶ 0: CSSStyleRule {selectorText: 'header', style: CSSStyleDeclarati
    ▶ 1: CSSStyleRule {selectorText: 'section', style: CSSStyleDeclarat
    ▶ 2: CSSStyleRule {selectorText: 'article', style: CSSStyleDeclarat
    ▶ 3: CSSStyleRule {selectorText: 'footer', style: CSSStyleDeclarati
    ▼ 4: CSSStyleRule
      ▶ cssRules: CSSRuleList {length: 0}
      ▶ cssText: "body" { background-color: rgb(224, 215, 81); font-size
      ▶ parentRule: null
      ▶ parentStyleSheet: CSSStyleSheet {ownerRule: null, cssRules: CSS
      ▶ selectorText: "body"
      ▶ style: CSSStyleDeclaration {0: 'background-color', 1: 'font-siz
      ▶ styleMap: StylePropertyMap {size: 2}
      ▶ type: 1
      ▶ [[Prototype]]: CSSStyleRule
```

```
const estilo=document.styleSheets[1];
estilo.cssRules[4].style.backgroundColor="blue";
```

**Página del Módulo Desarrollo
Web en Entorno Cliente**

[setAttribute]

setAttribute es un método de *Element* que establece el valor de un atributo en el elemento indicado. El primer argumento de *setAttribute* restablece el nombre del atributo y el segundo el valor del mismo.

Si el atributo ya existe, el valor es actualizado, en caso contrario, el nuevo atributo es añadido con el nombre y valor indicado.

```
const elemento =document.querySelector("#Profesor p + p");  
console.log(elemento);
```

```
elemento.setAttribute("id", "NombreProfesor");  
console.log(elemento);
```

[getAttribute]

getAttribute es otro método de *Element* que devuelve el valor de un atributo que se le pasa como argumento.

Si el atributo no existe devuelve *null*.

```
const elemento = document.querySelector("#Profesor p + p");  
console.log(elemento.getAttribute("id"));
```

null

```
elemento.setAttribute("id", "NombreProfesor");  
console.log(elemento.getAttribute("id"));
```

NombreProfesor

[removeAttribute]

removeAttribute es un método de *Element* que elimina el atributo cuyo nombre se le pasa como argumento.

Si el atributo no existe no se genera ningún error.

```
const elemento =document.querySelector("#Profesor p + p");  
console.log(elemento.getAttribute("id"));  
elemento.setAttribute("id", "NombreProfesor");  
console.log(elemento.getAttribute("id"));  
elemento.removeAttribute("id");  
console.log(elemento.getAttribute("id"));
```

null
NombreProfesor
null

(traversing)

Se llama *traversing* (*atravesando*) a recorrer el árbol DOM, accediendo a diferentes elementos o nodos partiendo desde uno de ellos, esto es, una vez que se ha seleccionado un elemento, acceder a sus hijos, nietos... o a sus ancestros, padre, abuelo...; entendiendo como hijos los nodos que están contenidos dentro de dicho elemento y como padre el nodo que contiene al elemento, etc.

Existen muchas librerías de JS que simplifican el *traversing*, una de las más conocidas es *jQuery*, de código abierto, simplifica la tarea de programar en JavaScript y permite agregar interactividad a un sitio web sin tener conocimientos del lenguaje.



[closest]

`closest()` es un método de la interfaz *Element* que busca el elemento HTML padre o ancestro más cercano que coincide con un selector CSS válido que se pasa como argumento. El propio elemento también se incluye en la búsqueda.

Este método sube del elemento y comprueba cada uno de los padres. Si coincide con el selector, entonces la búsqueda se detiene y devuelve dicho elemento, en el caso de no encontrar ningún elemento que coincida con la búsqueda devuelve *null*.

```
<main>
  <section id="Profesor">
    <p class="cabecera">Profesor</p>
    <p>Me llamo Jorge García Flores,...
```



```
const elemento = document.querySelector("#Profesor .cabecera");
console.log(elemento.closest("main"));
```

[childNodes]

childNodes es una propiedad de solo lectura del elemento *Node*, esta propiedad inalterable contiene un *NodeList*, con la colección de nodos hijos de dicho elemento, comenzando la asignación del índice 0 al primer hijo.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento= document.querySelector('ul');
console.log(elemento);
console.log(elemento.childNodes);
```

```
▼ <ul>
  ▶ <li>...</li>
  ▶ <li>...</li>
  ▶ <li>...</li>
  </ul>
```

```
▼ NodeList(7) [text, li, text, li, text, li, text]
  ▶ 0: text
  ▶ 1: li
  ▶ 2: text
  ▶ 3: li
  ▶ 4: text
  ▶ 5: li
  ▶ 6: text
  length: 7
  ▶ [[Prototype]]: NodeList
```

[childNodes]

Sin embargo, *childNodes*, al ser nodos, como ya se explicó en el tema anterior, considera los “retorno de carro” y otros caracteres como hijos, por eso, en el ejemplo, a pesar de que se aprecian solo 3 hijos (no nietos) que se corresponden con los tres `` *childNodes* se indica que hay 7 elementos en el *NodeList*.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento= document.querySelector('ul');
console.log(elemento);
console.log(elemento.childNodes);
```

```
<ul>
  > <li>...</li>
  > <li>...</li>
  > <li>...</li>
</ul>
```

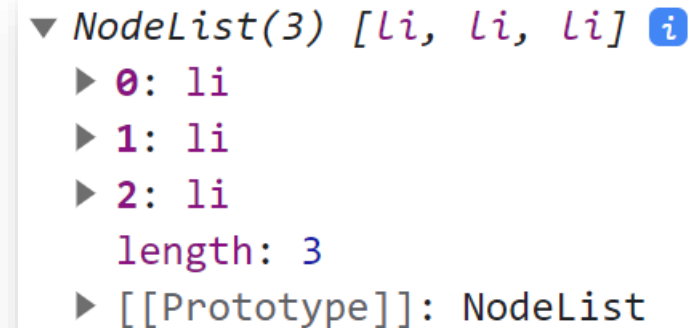
```
▼ NodeList(7) [text, li, text, li, text, li, text]
  ► 0: text
  ► 1: li
  ► 2: text
  ► 3: li
  ► 4: text
  ► 5: li
  ► 6: text
  length: 7
  ► [[Prototype]]: NodeList
```

[childNodes]

Si se elimina, en el ejemplo, los “retorno de carro”, empeorando la legibilidad del código, *childNodes* devolverá los 3 hijos (no nietos) que existen.

```
<ul><li><a href="#Profesor">Profesor</a></li><li><a href="#Contenidos">Contenidos</a></li><li><a href="#Recursos">Recursos</a></li></ul>
```

```
const elemento= document.querySelector('ul');  
console.log(elemento);  
console.log(elemento.childNodes);
```



```
▼ NodeList(3) [li, li, li] ⓘ  
  ▶ 0: li  
  ▶ 1: li  
  ▶ 2: li  
    length: 3  
  ▶ [[Prototype]]: NodeList
```

Este motivo es por lo que esta propiedad es menos usada que *children* (se verá a continuación) para hacer *traversing* DOM hacia los descendientes de un elemento.

[childNodes]

En el siguiente ejemplo se hace uso de la propiedad `childNodes` para hacer *traversing* y acceder al destino del primer `<a>` partiendo del elemento ``.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento= document.querySelector('ul');
const hijo= elemento.childNodes[1];
console.log(hijo);
console.log(hijo.childNodes[0].href);
```

```
▼ NodeList(7) [text, li, text, li, text, li, text]
  ► 0: text
  ► 1: li
  ► 2: text
  ► 3: li
  ► 4: text
  ► 5: li
  ► 6: text
    length: 7
  ► [[Prototype]]: NodeList
```

```
▼ <li>
  |   <a href="#Profesor">Profesor</a>
  |   </li>
  |
  | https://handmadegames.es/dwec/#Profesor
```

[children]

children es una propiedad de solo lectura, de *Element*, que contiene un *HTMLCollection* con todos los elementos hijos de dicho elemento.

children incluye solo nodos de elementos, a diferencia de *childNodes* que incluye todos los nodos, como los nodos de texto y nodos de comentarios.

```
const elemento= document.querySelector('ul');  
console.log(elemento.children);
```

```
▼ HTMLCollection(3) [li, li, li] ⓘ  
  ▶ 0: li  
  ▶ 1: li  
  ▶ 2: li  
    length: 3  
  ▶ [[Prototype]]: HTMLCollection
```

[children]

En el siguiente ejemplo se hace uso de la propiedad *children* para hacer *traversing* y acceder al destino del primer `<a>` partiendo del elemento ``.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento= document.querySelector('ul');
const hijo= elemento.children[0];
console.log(hijo);
console.log(hijo.children[0].href);
```

```
▼ HTMLCollection(3) [li, li, li] ⓘ
  ► 0: li
  ► 1: li
  ► 2: li
    length: 3
  ► [[Prototype]]: HTMLCollection
```

```
▼ <li>
  |   <a href="#Profesor">Profesor</a>
  | </li>
  https://handmadegames.es/dwec/#Profesor
```


[children]

Este ejemplo es parecido al anterior, no obstante, se hace uso de *chaining methods* para hacer *traversing* y acceder al texto del primer `<a>` partiendo del elemento ``.

```
<ul>  
  <li><a href="#Profesor">Profesor</a></li>  
  <li><a href="#Contenidos">Contenidos</a></li>  
  <li><a href="#Recursos">Recursos</a></li>  
</ul>
```

```
▼ HTMLCollection(3) [li, li, li] ⓘ  
  ► 0: li  
  ► 1: li  
  ► 2: li  
    length: 3  
  ► [[Prototype]]: HTMLCollection
```

```
console.log(document.querySelector('ul').children[0].children[0].textContent);
```

Profesor

(parentNode)

parentNode es una propiedad de solo lectura, de *Node*, que contiene el nodo padre del elemento actual.

El padre de un elemento puede ser un nodo de tipo *Element*, un nodo *Document*, o un nodo *DocumentFragment*.

```
const elemento= document.querySelector('h1');  
console.log(elemento.parentNode);
```

DocumentFragment representa un modelo de documento de objetos mínimo (un DOM mínimo), que no tiene padre. Se utiliza como una “versión ligera” de *Document* para almacenar una parte de la estructura del documento.

```
▼ <hgroup>  
  <h1 class="text-center">Página del Módulo Desarrollo Web en Entorno  
  Cliente</h1>  
  <h2 class="text-center">Ciclo Formativo DAW</h2>  
</hgroup>
```

[parentElement]

parentElement es una propiedad de sólo lectura, de *Node*, que contiene el elemento padre del elemento actual.

El nodo padre puede ser *Element* o, si el nodo no tiene padre o si el padre no es un elemento del DOM, *null*.

```
const elemento= document.querySelector('h1');  
console.log(elemento. parentElement);
```

```
▼ <hgroup>  
  <h1 class="text-center">Página del Módulo Desarrollo Web en Entorno  
  Cliente</h1>  
  <h2 class="text-center">Ciclo Formativo DAW</h2>  
</hgroup>
```

(parentElement)

Se puede ir encadenando *parentElement* hasta llegar al elemento que sea de interés, con cada encadenamiento, se accede al elemento padre del anterior.

```
<body>
<header>
  <hgroup>
    <h1 class="text-center">Página del Módulo Desarrollo Web en Entorno Cliente</h1>
    <h2 class="text-center">Ciclo Formativo DAW</h2>
  </hgroup>
```

```
const elemento= document.querySelector('h1');
console.log(elemento.parentElement);
console.log(elemento.parentElement.parentElement);
console.log(elemento.parentElement.parentElement.parentElement);
```

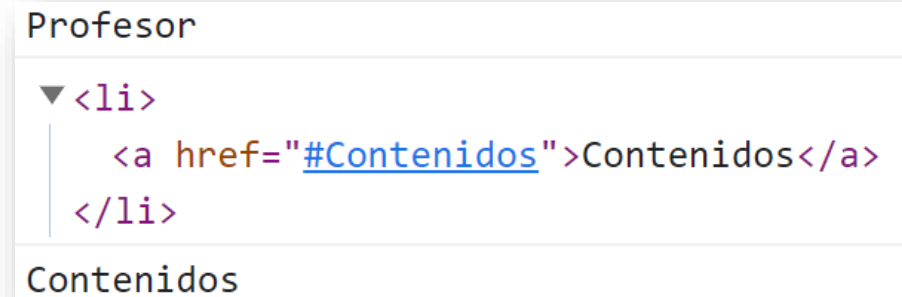
```
▶ <hgroup> ... </hgroup>
▶ <header> ... </header>
▶ <body> ... </body>
```

[nextElementSibling]

Además, se puede acceder a los hermanos de un elemento, a través de la propiedad *nextElementSibling*.

Esta propiedad, de solo lectura, devuelve el elemento inmediatamente posterior al especificado, dentro de la lista de elementos hijos de su padre, o *null* si el elemento especificado es el último en dicha lista.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```



```
const elemento= document.querySelector('li');
console.log(elemento.textContent);
console.log(elemento.nextElementSibling);
console.log(elemento.nextElementSibling.textContent);
```

[previousElementSibling]

Con *previousElementSibling* se accede al elemento inmediatamente anterior o se obtiene *null* si el elemento especificado es el primero en dicha lista.

Esta propiedad es también de solo lectura.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento= document.querySelector('li:nth-child(3)');
console.log(elemento.textContent);
console.log(elemento.previousElementSibling);
console.log(elemento.previousElementSibling.textContent);
```

Recursos

```
▼ <li>
  <a href="#Contenidos">Contenidos</a>
</li>
```

Contenidos

[firstElementChild]

La propiedad de solo lectura *firstElementChild* permite acceder al primer elemento hijo de un elemento o *null* si el elemento especificado no tiene elementos hijos.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento= document.querySelector('ul');
console.log(elemento.firstElementChild.textContent);
```

```
const elemento= document.querySelector('ul');
console.log(elemento.firstElementChild);
```

```
const elemento= document.querySelector('li');
console.log(elemento.firstElementChild);
console.log(elemento.firstElementChild.textContent);
```

Profesor

```
▼ <li>
  <a href="#Profesor">Profesor</a>
</li>
```

```
<a href="#Profesor">Profesor</a>
Profesor
```

[lastElementChild]

La propiedad de solo lectura *lastElementChild* permite acceder al último elemento hijo de un elemento o *null* si el elemento especificado no tiene elementos hijos.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento= document.querySelector('ul');
console.log(elemento.lastElementChild.textContent);
console.log(elemento.lastElementChild);
```

```
const elemento= document.querySelector('li');
console.log(elemento.lastElementChild.textContent);
console.log(elemento.lastElementChild);
```

Recursos

```
<li>
  <a href="#Recursos">Recursos</a>
</li>
```

Profesor

```
<a href="#Profesor">Profesor</a>
```


[remove]

remove es un método de *Element* que permite eliminar un elemento del DOM.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
let elemento = document.querySelector('li');
console.log("Eliminado:", elemento.textContent);
elemento.remove();
elemento = document.querySelectorAll('li');
for (el of elemento)
  console.log(el.textContent);
```

Eliminado: Profesor
Contenidos
Recursos

[removeChild]

También se pueden eliminar elementos, desde el elemento padre, con el método *removeChild*.

Este método borrará la referencia al elemento hijo pasada como argumento.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento = document.querySelector('ul');
let i=0;
for (hijo of elemento.children)
  console.log(i++,hijo.textContent);
console.log("Eliminado: ", elemento.children[1].textContent);
elemento.removeChild(elemento.children[1]);
i=0;
for (hijo of elemento.children)
  console.log(i++,hijo.textContent);
```

0	'Profesor'
1	'Contenidos'
2	'Recursos'
Eliminado: Contenidos	
0	'Profesor'
1	'Recursos'

[removeChild]

Otro ejemplo:

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
```

```
const elemento = document.querySelector('ul');
const elemento2 = document.querySelector('li');
let i=0;
for (hijo of elemento.children)
  console.log(i++,hijo.textContent);
i=0;
elemento.removeChild(elemento2);
for (hijo of elemento.children)
  console.log(i++,hijo.textContent);
```

0	'Profesor'
1	'Contenidos'
2	'Recursos'
0	'Contenidos'
1	'Recursos'

[removeChild]

En el siguiente ejemplo se borrarán todos los elementos hijos de *ul*. El bucle *while* se ejecuta mientras haya elementos dentro del cuerpo de *ul*:

```
<ul>  
  <li><a href="#Profesor">Profesor</a></li>  
  <li><a href="#Contenidos">Contenidos</a></li>  
  <li><a href="#Recursos">Recursos</a></li>  
</ul>
```

```
const elemento = document.querySelector('ul');  
while(elemento.firstChild)  
  elemento.removeChild(elemento.firstChild);
```

Después de ejecutar el código el resultado es el siguiente:

" "

```
<ul>  
</ul>
```

[createElement]

createElement es un método de *Document* que permite crear nuevos nodos de tipo Elemento HTML en una página web.

Recibe como parámetro la etiqueta HTML del elemento que se desea crear, no es *case sensitive* por lo que no influyen las mayúsculas o minúsculas, pero debe ir entre comillas, como, por ejemplo, "*div*", "*a*", "*img*"... Devuelve el nuevo nodo de tipo *Element*.

El elemento creado de momento no es visible, está ubicado en la memoria, pero no está asignado en un nodo del árbol DOM, por lo que no está presente todavía en la página web, así, hasta que no se le asigne una posición en el árbol no se mostrará al usuario.

```
const elemento = document.createElement('a');  
// Aunque no sea visible en la página web se puede ver su estructura en la consola  
console.log(elemento);
```

```
<a></a>
```

[createElement]

A continuación, se puede ir añadiendo los atributos que se consideren apropiados, a través de sus propiedades o a través del método `setAttribute()`, como `"href"`, `"src"`, e incluso `"textContent"`....

```
const elemento = document.createElement("a");  
elemento.setAttribute("href", "#Contenidos");  
elemento.textContent="Nuevo Enlace";  
console.log(elemento);  
elemento.target="_blank";
```

```
<a href="#Contenidos" target="_blank">Nuevo Enlace</a>
```

[append]

Para que el nuevo elemento HTML se muestre en la página web y sea visible al usuario hay que asignarlo a un nodo del árbol DOM. Para ello, se selecciona el elemento padre que va a contener el nuevo elemento y se invoca el método *append()* con el nuevo elemento creado como argumento.

Este método añade al final del elemento padre el elemento que se le pasa como argumento.

```
<main>
  <section id="Profesor">
    <p class="cabecera">Profesor</p>
    <p>Me llamo Jorge García Flores,...
```

```
const elementoPadre = document.querySelector("#Profesor");
const elemento = document.createElement("a");
elemento.textContent="Nuevo Enlace";
elemento.href="#Contenidos";
elementoPadre.append(elemento);
```

En utilizar una *PIGA* para
sintetizar las máquinas y
consolas antiguas.

Puedes visitar mi portfolio [aquí](#).

[Nuevo Enlace](#)

[append]

`append()` es un método de *Element* que inserta un conjunto de objetos *Nodo* u objetos *DOMString* después del último hijo del elemento y no devuelve nada.

Los objetos *DOMString*, se pueden considerar como literales de cadenas de texto de JS, se insertan con `append()` como nodos de tipo texto.

```
<main>
  <section id="Profesor">
    <p class="cabecera">Profesor</p>
    <p>Me llamo Jorge García Flores,
      ...
      aquí</a>.
    </p>
```

Puedes visitar mi portfolio [aquí](#).

Por último, me gustaría añadir...Y eso es todo.

```
const elementoPadre = document.querySelector("#Profesor");
elementoPadre.append("Por último, me gustaría añadir...", "Y eso es todo.");
```


[append]

Si el elemento que se le pasa a *append()* ya está ubicado en el DOM simplemente se mueve de un nodo a otro nodo del árbol DOM.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
...
<main>
  <section id="Profesor">
    <p class="cabecera">Profesor</p>
    <p>Me llamo Jorge García Flores,...
```

consolas antiguas.

Puedes visitar mi portfolio [aquí](#).

- [Profesor](#)
- [Contenidos](#)
- [Recursos](#)

```
const elementoPadre = document.querySelector("#Profesor");
const elemento = document.querySelector("ul");
elementoPadre.append(elemento);
```

[appendChild]

Otra forma para mostrar el nuevo el elemento HTML es mediante el método *appendChild()*, muy parecido a *append()* pero con ciertas diferencias; *appendChild()* es un método de *Node* que solo acepta como argumentos tipo *Node* no cadenas de texto (*DOMStrings*).

Igual que con *append()*, se selecciona el elemento padre que va a contener el nuevo elemento y se invoca al método *appendChild()* con el nuevo elemento creado como argumento.

```
const elementoPadre = document.querySelector("#Profesor");  
const elemento = document.createElement("a");  
elemento.setAttribute("href", "#Contenidos");  
elemento.textContent="Nuevo Enlace";  
elemento.target="_blank";  
elementoPadre.appendChild(elemento);
```

Puedes visitar mi portfolio [aquí](#).

[Nuevo Enlace](#)

[appendChild]

El método *appendChild()*, al igual que *append()*, posiciona al final de todos los elementos hijos el nuevo elemento, y si la referencia al elemento ya existe, entonces *appendChild()* lo mueve de un nodo a otro nodo del árbol DOM.

```
<ul>
  <li><a href="#Profesor">Profesor</a></li>
  <li><a href="#Contenidos">Contenidos</a></li>
  <li><a href="#Recursos">Recursos</a></li>
</ul>
...
<main>
  <section id="Profesor">
    <p class="cabecera">Profesor</p>
    <p>Me llamo Jorge García Flores,...
```

Puedes visitar mi portfolio [aquí](#).

- [Profesor](#)
- [Contenidos](#)
- [Recursos](#)

```
const elementoPadre = document.querySelector("#Profesor");
const elemento = document.querySelector("ul");
elementoPadre.appendChild(elemento);
```

append vs. appendChild

Las principales diferencias entre *appendChild()* y *append()* son:

- ❑ *append()* acepta objetos *Node* y *DOMStrings*, mientras que *appendChild()* solo acepta objetos de *Node*.
- ❑ *append()* no tiene un valor de retorno mientras que *appendChild()* devuelve el objeto *Node* pasado como argumento.
- ❑ *append()* permite agregar varios elementos mientras *appendChild()* solo permite agregar un solo elemento a la vez.

append vs. appendChild

```
const padre = document.querySelector('#Profesor');  
const hijo1 = document.createElement('a');  
const hijo2 = document.createElement('a');  
hijo1.textContent=hijo2.textContent="NUEVO ELEMENTO";  
padre.appendChild(hijo1,hijo2, 'Hola mundo');
```

```
const padre = document.querySelector('#Profesor');  
const hijo1 = document.createElement('a');  
const hijo2 = document.createElement('a');  
hijo1.textContent=hijo2.textContent="NUEVO ELEMENTO";  
padre.append(hijo1,hijo2, 'Hola mundo');
```

Puedes visitar mi portfolio [aquí](#).

NUEVO ELEMENTO

Puedes visitar mi portfolio [aquí](#).

NUEVO ELEMENTONUEVO
ELEMENTOHola mundo

insertBefore

insertBefore() es otro método de *Node* parecido a *appendChild()*, pero permite reposicionar o colocar elementos justo antes de otro elemento indicado como argumento, y no al final como hace *appendChild()*.

Recibe como primer argumento la referencia al elemento que se desea colocar y, como segundo argumento, la referencia a otro elemento existente que indica que el elemento a insertar se posicionará justo delante de este.

```
const elementoPadre = document.querySelector("#Profesor");
const elemento = document.createElement("a");
elemento.setAttribute("href", "#Contenidos");
elemento.textContent="Nuevo Enlace";
elemento.target="_blank";
elementoPadre.insertBefore(elemento, elementoPadre.children[0]);
```

[Nuevo Enlace](#)
Profesor

Me llamo Jorge García Flores,

[replaceChild]

replaceChild() es otro método de *Node* que permite reemplazar un nodo por otro.

Recibe como primer argumento la referencia al elemento *reemplazador* (puede ser uno nuevo o ya existente, en este caso lo reposiciona), y, como segundo argumento, la referencia al elemento a reemplazar; devuelve una referencia al nodo reemplazado.

```
<main>
  <section id="Profesor">
    <p class="cabecera">Profesor</p>
    <p>Me llamo Jorge García Flores,...
```

Nuevo Enlace

Me llamo Jorge García Flores,
soy ingeniero en sistemas

```
const elemento1 = document.querySelector("#Profesor .cabecera");
const elemento2 = document.createElement("a");
elemento2.setAttribute("href", "#Contenidos");
elemento2.textContent="Nuevo Enlace";
elemento1.parentElement.replaceChild(elemento2,elemento1);
```

Bibliografía y recursos online

- <https://developer.mozilla.org/es/docs/Glossary/DOM>
- <https://developer.mozilla.org/en-US/docs/Web/API/Element/closest>
- <https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild>
- <https://developer.mozilla.org/en-US/docs/Web/API/Element/remove>
- <https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement>
- <https://developer.mozilla.org/en-US/docs/Web/API/Element/append>
- <https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild>
- <https://developer.mozilla.org/en-US/docs/Web/API/Node/insertBefore>
- <https://developer.mozilla.org/en-US/docs/Web/API/Node/replaceChild>