

# Desarrollo Web en Entorno Cliente

## Tema 8





# [DOM]

El DOM (*Document Object Model*), Modelo de Objetos del Documento, es una API definida para representar e interactuar con cualquier documento HTML o XML.

El DOM surgió a partir de la implementación de JS en los navegadores. A esta primera versión también se la conoce como *DOM 0* o “*Legacy DOM*”.

En la actualidad el grupo WHATWG (*Web Hypertext Application Technology Working Group*) es el encargado de actualizar el estándar de DOM.

El termino *Legacy* (heredado) hace referencia, en informática, a una tecnología o sistema antiguo o desactualizado que sigue en uso porque sigue desempeñando las funciones para las que fue diseñado.

Por lo general, ya no cuenta con soporte y mantenimiento y está limitado a nivel de crecimiento.

# [DOM]

El DOM se carga en un objeto *Window* del navegador web y representa el documento como un árbol de nodos, donde cada nodo representa una parte del documento.

Estos nodos pueden crearse, moverse o modificarse, se les puede añadir manejadores de eventos (*event listeners*), que se ejecutarán o activarán cuando ocurra un evento indicado en el manejador.

El DOM no es un lenguaje de programación, pero sin él, el lenguaje JS no tiene ninguna noción de acceso a las páginas web, a las páginas XML.

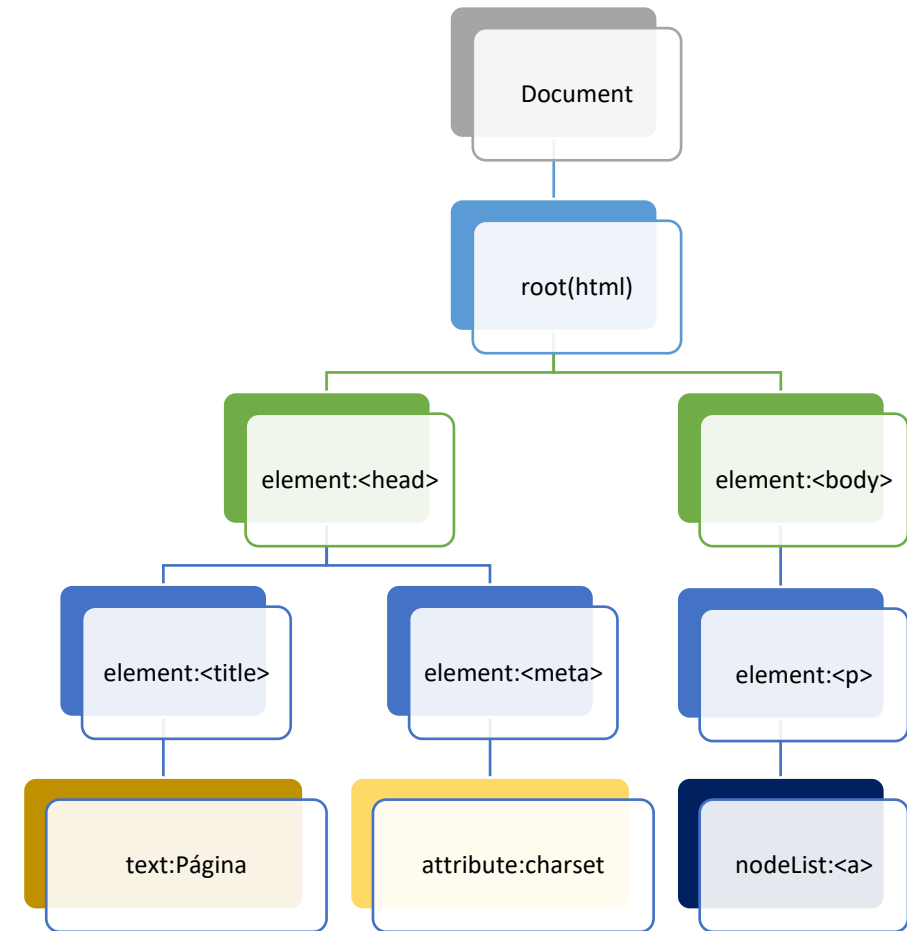
Cada elemento del documento: el documento en su totalidad, el título, las tablas dentro del documento, los títulos de las tablas, el texto dentro de las celdas de las tablas, las imágenes, los enlaces... es parte del modelo de objeto del documento al que se pueden acceder y manipular utilizando el DOM y un lenguaje de programación, como JavaScript.

No se tiene que hacer nada especial para empezar a utilizar el DOM, todos los navegadores web usan el modelo de objeto de documento para hacer accesibles las páginas web a los diferentes lenguajes de programación mediante su API.

# [DOM]

El DOM muestra una representación del documento como un árbol de nodos. La API se refiere a cada nodo como *node*, a cada etiqueta html como *element*, a una lista de nodos como *nodeLists* y a los nodos de atributos como *attribute*.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Página</title>
</head>
<body>
  <p>
    <a href=#>enlace1</a>
    <a href=#>enlace2</a>
  </p>
</body>
</html>
```



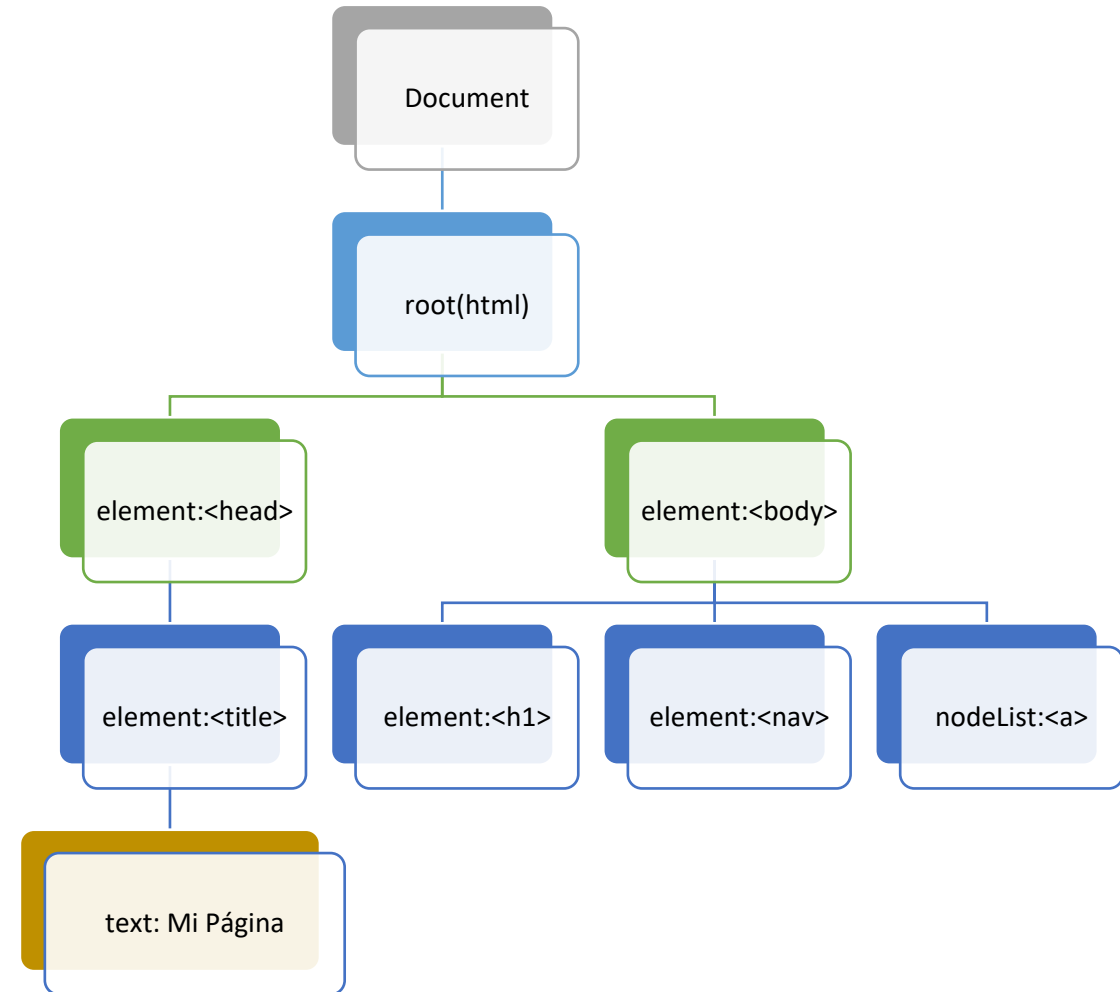
# [DOM]

*node* se refiere a cada uno de los nodos del árbol DOM.

*NodeList* es una lista de nodos.

*element* es un nodo de tipo elemento, y se refiere a cada una de las etiquetas html del objeto documento. Añade, además de la interfaz (propiedades y métodos) que dispone el *node*, la interfaz *element*.

*attribute* es una referencia a una interfaz particular a para acceder a los atributos de un elemento. Los atributos son nodos en el DOM igual que los elementos, pero en JS no suelen usarse así.



# [Node]

Los nodos del DOM se pueden clasificar en diferentes tipos, los principales son:

- ❑ *Document*: es el nodo *document* es el nodo raíz, a partir del cual derivan el resto de nodos.
- ❑ *Element*: son los nodos definidos por etiquetas html. Por ejemplo, una etiqueta *div* genera un nodo. Si dentro de ese *div* hay tres etiquetas *p*, dichas etiquetas definen nodos hijos de la etiqueta *div*.
- ❑ *Text*: es el texto dentro de un nodo *element* se considera un nuevo nodo hijo de tipo *text* (texto). Los navegadores también crean nodos tipo texto sin contenido para representar elementos como saltos de línea o espacios vacíos.
- ❑ *Attribute*: son los atributos de las etiquetas definen nodos, aunque JS no lo presenta como nodos, sino que lo considera información asociada al nodo de tipo *element*.
- ❑ Comentarios y otros: representa a los comentarios y otros elementos, como las declaraciones *doctype* en cabecera.

# [Node]

*Node* proporciona una forma de poder acceder y manipular los nodos del DOM (realmente es una interfaz abstracta que es implementada por otros objetos como *Document* y *Element*). Dispone de muchas propiedades y métodos, por ejemplo:

- ❑ La propiedad *nodeName* devuelve el tipo de nodo: *#text* para nodos de texto, *#comment* para nodos de comentarios, *#document* para nodos de documentos..., si el nodo es un nodo de tipo elemento devuelve su nombre en mayúsculas.
- ❑ La propiedad *nodeValue* devuelve el valor de un nodo. Si es un nodo de tipo elemento, la propiedad *nodeValue* devuelve un valor *null*.
- ❑ El método *appendChild()* permite agregar al nodo otro nodo como un nodo hijo (una ramificación en el árbol que parte de este nodo).

A lo largo de este tema y el que le sigue se presentarán las propiedades y métodos más importantes de *Node*.



# [NodeList]

Un *NodeList* es una lista de nodos del DOM, similar a un array de JS, y cuenta con la propiedad *Length* que indica el número de nodos presentes en el *NodeList*.

Dispone únicamente de los siguientes métodos:

- ❑ *NodeList.item()* Devuelve un elemento por su índice o *null* si el índice está fuera de los límites.
- ❑ *NodeList.entries()* Devuelve un iterador, igual que en los arrays, las claves son números enteros a partir de 0 y los valores son los nodos.
- ❑ *NodeList.forEach()* Es un iterador que permite recorrer la lista de nodos, al igual que los arrays.
- ❑ *NodeList.keys()* Devuelve un iterador, igual que en los arrays, las claves son números enteros a partir de 0 y los valores son los nodos.
- ❑ *NodeList.values()* Devuelve un iterador, igual que en los arrays.

# (DOMTokenList)

*DOMTokenList* es una lista de *tokens* (*palabras*) separados por espacios.

Un *DOMTokenList* se indexa a partir de 0 como un array en JS, y siempre distingue entre mayúsculas y minúsculas.

Tiene las propiedades *Length* que devuelve el número de elementos que contiene y *value* que devuelve los elementos del *DOMTokenList* en una cadena de caracteres.

El método *classList* de un elemento del DOM (que se verá más adelante en este mismo tema) devuelve *DOMTokenList*, será, en ese momento, cuando se presenten los métodos de *DOMTokenList*.

# [NamedNodeMap]

*NamedNodeMap*, representa una colección de objetos *Attr*. Estos objetos *Attr* sirven para acceder y modificar los atributos de los elementos del DOM.

Los objetos dentro de *NamedNodeMap* no están en ningún orden en particular, pero se puede acceder a ellos mediante un índice como en un array.

*NamedNodeMap* tiene la propiedad *Length* que indica la cantidad de objetos que contiene. Además, posee varios métodos como: *getNamedItem()*, *getNamedItemNS()*, *setNamedItem()*, *setNamedItemNS()*, *removeNamedItem()*, *removeNamedItemNS()* e *item()*.

No obstante, tanto *NamedNodeMap* como *Attr* se mencionan a modo de dar a conocer su existencia dentro de la API del DOM, pero no se suelen utilizar para acceder y modificar los atributos de los elementos del DOM mediante JS, como ya se indicó; en su lugar se presentarán otros métodos que resultan más fáciles para este cometido.

# [HTMLCollection]

Un *HTMLCollection* es muy parecido a un array de JS, aunque cuenta con todos sus métodos de utilidad (*Array Methods*).

*HTMLCollection* es una colección genérica de elementos del DOM (un tipo de nodo), que se presentan ordenados por su aparición en el documento, en la página web.

Al ser similar a un array se puede acceder a sus elementos por su índice.

```
const elemento=document.all;  
console.log(elemento.item(4));
```

```
const elemento=document.all;  
console.log(elemento.length);
```

80

```
const elemento=document.all;  
console.log(elemento[4]);
```

```
<title>Página del Módulo Desarrollo Web en Entorno  
Cliente</title>
```

```
<title>Página del Módulo Desarrollo Web en Entorno  
Cliente</title>
```

# [document]

Mediante el (nodo) objeto *document* se tiene acceso a todo el documento, a toda la página web.

Para seleccionar elementos o nodos de la página se hace utilizando las propiedades y métodos que presenta el objeto *document*.

```
const elemento=document;  
console.log(elemento);
```

```
▼ #document  
  <!DOCTYPE html>  
  <html lang="es">  
    ▶ <head> ⋮ </head>  
    ▶ <body> ⋮ </body>  
  </html>
```

```
const elemento=document.all;  
console.log(elemento);
```

```
HTMLAllCollection(81) [html, head, meta,  
meta, title, link, link, link, style, style,  
style, style, style, body, header, hgroup,  
h1.text-center, h2.text-center, nav, ul, li,  
a, li, a, li, a, div, figure,  
picture.centrado, source, source, source,  
img, figcaption, audio, p, a, main,  
section#Profesor, p.cabecera, p, p,  
span.negrita, span.cursiva, p, span.negrita,  
▶ span.cursiva, p, a.cursiva,  
section#Contenidos, p.cabecera, article, p,  
p, article, p, span.cursiva, p, span.cursiva  
, span.cursiva, p, article, p,  
section#Recursos, p.cabecera,  
article.descargas.cursiva, p, a, p, a, p, a,  
footer, div.licencia, a, img, br, a, p, p, p  
, viewport: meta, Profesor: section#Profesor  
, Contenidos: section#Contenidos, Recursos:  
section#Recursos]
```

# [document]

`document.head` devuelve el elemento `head` del DOM que se corresponde con la etiqueta `<head>` de la página web.

```
const elemento=document.head;  
console.log(elemento);
```

`document.domain` devuelve el dominio donde se encuentra la página web.

```
const elemento=document.domain;  
console.log(elemento);
```

handmadegames.es

```
▼ <head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Página del Módulo Desarrollo Web en Entorno Cliente</title>  
  <link href="css/normalize.css" rel="stylesheet">  
  <link href="css/styles.css" rel="stylesheet">  
  <link rel="stylesheet" crossorigin="anonymous" href="https://gc.kis.v2.scr.kaspersky-labs.com/E3E8934C-235A-4B0E-825A-35A08381A191/abn/main.css?attr=aHR0cHM6Ly9oYW5kbWFKZWdhbWVzLmVzL2R3ZWV">  
  ▶ <style type="text/css" nonce="undefined"> ...  
    </style>  
  ▶ <style type="text/css" nonce="undefined"> ...  
    </style>  
  ▶ <style type="text/css" nonce="undefined"> ...  
    </style>  
  ▶ <style type="text/css" nonce="undefined"> ...  
    </style>  
  ▶ <style type="text/css" nonce="undefined"> ...  
    </style>  
</head>
```

# [document]

`document.body` devuelve el elemento `body` del DOM que se corresponde con la etiqueta `<body>` de la página web. `body`.

```
const elemento=document.body;  
console.log(elemento);
```

```
▼ <body>  
  ▶ <header> ... </header>  
  ▶ <main> ... </main>  
  ▶ <footer> ... </footer>  
</body>
```

`document.forms` devuelve los formularios que hay presentes en la página. Devuelve un `HTMLCollection` con todos los formularios presentes en la página web.

```
const elemento=document.forms;  
console.log(elemento);
```

```
▼ HTMLCollection [] ⓘ  
  length: 0  
  ▶ [[Prototype]]: HTMLCollection
```

```
◀ HTMLCollection(3) [form#simpleSearcher, form, form, simpleSearcher:  
  form#simpleSearcher, formSelectChildrenCombo_16852692131395738207:  
  form, formSelectChildrenCombo_18387424202020552920: form] ⓘ
```

# [document]

`document.links` devuelve un *HTMLCollection* con los enlaces presentes en la página.

```
const elemento=document.links;  
console.log(elemento);
```

```
▶ HTMLCollection(10) [a, a, a, a, a.cursiva, a, a, a,  
a, a]
```

`document.images` devuelve un *HTMLCollection* con las imágenes presentes en la página.

```
const elemento=document.images;  
console.log(elemento);
```

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

`document.scripts` devuelve un *HTMLCollection* con los scripts presentes en la página.

```
const elemento=document.scripts;  
console.log(elemento);
```

```
▼ HTMLCollection [] ⓘ  
  length: 0  
  ▶ [[Prototype]]: HTMLCollection
```



# [document]

Existe, además, un objeto *Window* que representa la ventana que contiene un documento DOM. Este objeto proporciona la interfaz, mediante una variedad de funciones, espacios de nombres, objetos y constructores que no están necesariamente asociados directamente con el concepto de una ventana de interfaz de usuario.

Además, en el objeto *Window* se incluyen elementos de JS que necesitan ser accedidos (ámbito) globalmente.

Se puede obtener una referencia a este objeto mediante la propiedad *document.defaultView*.

```
const objWindow= document.defaultView;  
console.log(objWindow);
```

```
▶ Window {window: Window, self: Window, document: document,  
  name: '', location: Location, ...}
```

# [window]

El objeto *Window* es accesible directamente para JS mediante una variable global llamada *window*.

Y, mediante la propiedad *document* de *Window* se hace referencia al documento DOM cargado en esa ventana.

```
console.log(window.document);
```

```
▼ #document
  <!DOCTYPE html>
  <html lang="es">
    ▶ <head> ... </head>
    ▶ <body> ... </body>
  </html>
```

En un navegador, cada pestaña tiene su propio objeto *Window* y el objeto global *window* de JS siempre referencia a la pestaña en la que se ejecuta el código.

El objeto *window* contiene muchas propiedades y métodos, como, por ejemplo, la propiedad *name* que permite obtener o establecer el nombre de la ventana, etc.

## [window]

El ejemplo, mediante el método *open()* de *window*, abre una nueva pestaña del navegador a la URL *https://handmadegames.es/dwec/*:

```
window.open("https://handmadegames.es/dwec/");
```

La propiedad *navigator* del objeto *window* aporta mucha información sobre el navegador desde el que se está ejecutando el script.

En el siguiente ejemplo se visualiza el idioma preferido del usuario, generalmente el idioma de la interfaz de usuario del navegador mediante la propiedad de solo lectura *Language*:

```
console.log(window.navigator.Language);
```

es-ES

# [getElementsByName]

*getElementsByName()* es un método de la interfaz *Element* que devuelve un *NodeList* con los elementos que tengan un atributo llamado *name* que coincida con el argumento de este método.

En el caso de que no haya elementos en la página web que cumplan la condición devolverá un *NodeList* vacío.

```
const elementos = document.getElementsByName('viewport');  
console.log(elementos);
```

```
▼ NodeList [meta] ⓘ  
  ► 0: meta  
    length: 1  
  ► [[Prototype]]: NodeList
```

# [getElementsByTagName]

`getElementsByTagName()` es un método de la interfaz *Element* que devuelve un *HTMLCollection* con los elementos que tengan como etiqueta la proporcionada en el argumento.

La etiqueta se puede proporcionar en mayúsculas como en minúsculas, ya que el método convierte a minúsculas el argumento antes de buscarlo. Este comportamiento a veces no es deseable, por ejemplo, para buscar elementos SVG en Camel Case (como `<LinearGradient>`).

Por eso existe el método `getElementsByTagNameNS()`, que realiza la misma funcionalidad que `getElementsByTagName()`, pero conserva el argumento que recibe, sin convertirlo a minúsculas.

```
const elementos = document.getElementsByTagName('P');  
console.log(elementos);
```

```
HTMLCollection(20) [p, p.cabecera, p, p, p, p, p.cabecera, p, p, p, p, p, p, p.cabecera,  
p, p, p, p, p, p]
```

## [getElementsByClassName]

`getElementsByClassName()` es un método de la interfaz *Element* que devuelve un objeto *HTMLCollection* con los elementos que tengan todos los nombres de clase indicados como argumento de este método.

Cuando se busca por una clase que no aparece en el documento el método devuelve un *HTMLCollection* vacío.

```
const elementos = document.getElementsByClassName('cabecera');  
console.log(elementos);
```

```
▼ HTMLCollection(3) [p.cabecera, p.cabecera, p.cabecera]  
  ► 0: p.cabecera  
  ► 1: p.cabecera  
  ► 2: p.cabecera  
    length: 3  
  ► [[Prototype]]: HTMLCollection
```

# [getElementsByClassName]

El siguiente ejemplo busca los elementos, en todo el documento, que tienen la clase *cabecera*. Se obtiene un *HTMLCollection* con tres elementos.

```
const elementos =  
document.getElementsByClassName('cabecera');  
console.log(elementos);
```

```
▼ HTMLCollection(3) [p.cabecera, p.cabecera, p.cabecera]  
  ► 0: p.cabecera  
  ► 1: p.cabecera  
  ► 2: p.cabecera  
    length: 3  
  ► [[Prototype]]: HTMLCollection
```

También se puede buscar, en todo el documento, los elementos que tienen a la vez la clase *descargas* y la clase *cursiva*.

```
const elementos =  
document.getElementsByClassName('descargas cursiva');  
console.log(elementos);
```

```
▼ HTMLCollection [article.descargas.cursiva]  
  ► 0: article.descargas.cursiva  
    length: 1  
  ► [[Prototype]]: HTMLCollection
```

# [getElementById]

`getElementById()` es un método de la interfaz *Element* que devuelve una referencia a un elemento del DOM por su atributo *id*.

El *id* buscado es pasado como argumento, JS es sensible a mayúsculas y minúsculas (*Case Sensitive*), por lo que `getElementById('carrito')` no es igual a `getElementById('Carrito')`.

Si no existe un elemento con *id* buscado la función devuelve *null*, en otro caso, devuelve un objeto con la referencia al elemento.

Influye el lugar de la página web donde se incluya las etiquetas *script* que ejecutan la llamada a `getElementById()`. Así, si el script es llamado desde las etiquetas *head* es muy probable que no encuentre el elemento debido a que la página no se ha cargado todavía.

En el siguiente ejemplo se busca el elemento que tienen el *id* *Profesor* y se muestra el valor de su propiedad *nodeName*.

```
console.log(document.getElementById("Profesor").nodeName);
```

SECTION



# [getElementById]

Aunque, por su definición, el *id* debe ser único y solo debe haber un elemento con ese mismo *id*, en el caso de que haya varios elementos con el mismo *id*, devolverá el primero que encuentre que tenga el *id* buscado.

En el siguiente ejemplo se busca el elemento que tienen el *id* *Profesor*.

```
const elemento = document.getElementById('Profesor');  
console.log(elemento);
```

```
<section id="Profesor">  
  <p class="cabecera">Profesor</p>  
  <p>...</p>  
  <p>...</p>  
  <p>...</p>  
  <p>...</p>  
</section>
```

También se puede hacer un encadenamiento, como en el siguiente ejemplo, donde se selecciona los elementos que tienen la clase *cabecera* pero son hijos (están dentro) del elemento que tiene el *id* *Profesor*.

```
const elemento = document.getElementById('Profesor').getElementsByClassName('cabecera');  
console.log(elemento);
```

```
▼ HTMLCollection [p.cabecera] ⓘ  
  ▶ 0: p.cabecera  
    length: 1  
  ▶ [[Prototype]]: HTMLCollection
```

# [querySelector]

Otro método de la interfaz *Element* es *querySelector()*, este método es el más utilizado en la actualidad para acceder a algún elemento de la página web.

Recibe como argumento el nombre de un selector, con la sintaxis igual a los selectores de las hojas de estilo CSS, y devuelve el primer elemento del DOM que cumpla dicho selector.

Devuelve *null* si ningún elemento cumple el selector.

```
const elemento = document.querySelector('#profesor');  
console.log(elemento);
```

```
const elemento = document.querySelector('#Profesor');  
console.log(elemento);
```

null

```
<section id="Profesor">  
  <p class="cabecera">Profesor</p>  
  <p>...</p>  
  <p>...</p>  
  <p>...</p>  
  <p>...</p>  
</section>
```

# [querySelector]

En el siguiente ejemplo se selecciona el primer elemento que tenga la clase *cabecera* y que cumpla que es hijo de un elemento con Id *Contenidos*.

```
const elemento = document.querySelector('#Contenidos >.cabecera');  
console.log(elemento);
```

```
<p class="cabecera">Contenidos</p>
```

En el siguiente ejemplo se selecciona el primer elemento de la cabecera que tenga un atributo *charset*.

```
const elemento = document.querySelector('head [charset]');  
console.log(elemento);
```

```
<meta charset="UTF-8">
```

# [querySelector]

En el siguiente ejemplo se selecciona el primer elemento que tenga a la vez la clase *descargas* y la clase *cursiva*.

```
const elemento = document.querySelector('.descargas.cursiva');  
console.log(elemento);
```

```
▶ <article class="descargas cursiva">...</article>
```

En el siguiente ejemplo se selecciona el primer elemento que sea un `<a>` y que tenga un atributo *href* cuyo valor contenga *Profesor* y que cumpla que es hijo de un elemento con `<li>`.

```
const elemento = document.querySelector('li a[href*="Profesor"]');  
console.log(elemento);
```

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

# [querySelector]

También se puede hacer uso de la *pseudoclase* `:nth-child()` de CSS, esta *pseudoclase* selecciona uno o más elementos del DOM en función de su posición, de entre un grupo de elementos hermanos (en el mismo nivel).

Ejemplo: se quiere seleccionar el tercer párrafo de la sección con *id* `Profesor` haciendo uso de la pseudoclase `:nth-child()`; para ello, entre los paréntesis, se le indica la posición del elemento al que se quiere acceder, teniendo en cuenta que comienza en 1 y no en 0.

```
const elemento = document.querySelector('#Profesor p:nth-child(3)');  
console.log(elemento);
```

```
<p>  
  "Entre mis "  
  <span class="negrita">gustos</span>  
  " está Linux, el diseño 3D, los videojuegos, las computadoras retro  
  (crecí con un "  
  <span class="cursiva">ZX Spectrum</span>  
  ") ..."  
</p>
```

# [querySelectorAll]

El método *querySelectorAll()*, también es otro método de la interfaz *Element* utilizado en la actualidad para seleccionar varios elementos de la página web.

A diferencia de *querySelector()*, *querySelectorAll()* selecciona todos los elementos del DOM que cumplan que su selector, con la sintaxis igual a los selectores de las hojas de estilo CSS, coincide con el selector pasado como argumento.

El elemento devuelto es un *NodeList*. En el caso de que no haya elementos en la página web que cumplan el selector, devolverá un *NodeList* vacío.

```
const elementos = document.querySelectorAll('a');  
console.log(elementos);
```

```
► NodeList(10) [a, a, a, a, a.cursiva, a, a, a, a, a]
```

# [querySelectorAll]

En el siguiente ejemplo seleccionamos los elementos que tenga la clase *cabecera* y que cumpla que son hijos de un elemento `<section>`.

```
const elementos = document.querySelectorAll('section .cabecera');  
console.log(elementos);
```

```
▼ NodeList(3) [p.cabecera, p.cabecera, p.cabecera]  
  ► 0: p.cabecera  
  ► 1: p.cabecera  
  ► 2: p.cabecera  
    length: 3  
  ► [[Prototype]]: NodeList
```

```
elementos.forEach(elemento=>console.log(elemento.textContent));
```

Profesor  
Contenidos  
Recursos

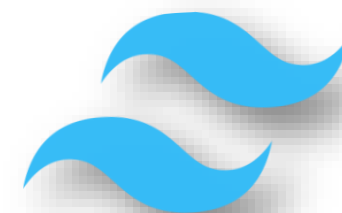
## [cambiando los atributos de los elementos]

Seleccionar los distintos elementos o nodos presentes en la página web, ya sea con *getElementsByClassName*, *getElementById*, *querySelector* o *querySelectorAll*, permite modificar y alterar las propiedades de los mismos, imprimiéndole un carácter dinámico a la página web.

Hay muchos frameworks que facilitan el desarrollo web sirviéndose de este dinamismo, como Bootstrap o Tailwind CSS, utilizados para desarrollar aplicaciones web *responsive*, con la filosofía *mobile first*, que manipulan las hojas de estilo mediante de librerías de JS.

El siguiente ejemplo sustituye la primera imagen que encuentre que tenga la clase *Licencia* por la imagen del logo de la Junta de Castilla y León.

```
const imagen = document.querySelector('.Licencia img');  
imagen.src='https://www.educa.jcyl.es/es/banners/202569-Logo-jcyl.gif';
```





# [innerText]

*innerText* es una propiedad de la interfaz *Element* que representa el contenido de texto “formateado”, es decir, tal y como se mostraría finalmente en la página web, de un nodo y sus descendientes.

```
const elemento = document.querySelector('h2');  
console.log(elemento.innerText);
```

## Ciclo Formativo DAW

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

*innerText* permite modificar el contenido de texto de una etiqueta en la página web de forma dinámica, cambiando el valor que contiene.

```
const elemento = document.querySelector('h2');  
elemento.innerText='Hola';
```

## Hola

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

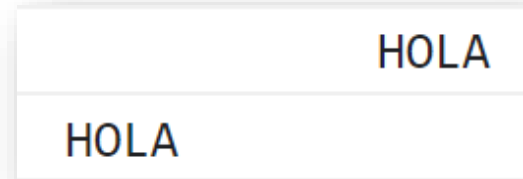
# [textContent]

*textContent* es una propiedad de la interfaz *Node*, y representa el contenido de texto de un nodo y sus descendientes.

*innerText* se confunde habitualmente con la propiedad *textContent* pero existen importantes diferencias entre los dos. Básicamente, *innerText* presenta la apariencia formateada del texto, mientras *textContent* es simplemente el texto sin formato.

En el siguiente ejemplo, se puede apreciar que las dos propiedades muestran el contenido de la etiqueta `<h2>`, pero *textContent* muestra el texto tal cual está en el código fuente de la página web mientras que *innerText* lo muestra formateado, por eso omite los espacios en blanco.

```
const elemento = document.querySelector('h2');
elemento.textContent="          HOLA";
console.log(elemento.textContent);
console.log(elemento.innerText);
```



# [textContent]

Además, como *innerText* muestra el texto formateado, cuando la propiedad de CSS *visibility* no está establecida o lo está a *hidden*, *innerText* devolverá una cadena en blanco, ya que dicho texto está oculto, mientras que *textContent* sí devolverá el texto contenido entre las etiquetas del elemento.

```
h2{  
  visibility: hidden;  
}
```

```
const elemento = document.querySelector('h2');  
console.log(elemento.innerText);  
console.log(elemento.textContent);
```

HOLA

# [innerHTML]

*innerHTML* es una propiedad de la interfaz *Element*, representa todo el texto, incluidas las etiquetas HTML contenidas en el elemento y en los descendientes del elemento.

```
const elemento = document.querySelector('hgroup');  
console.log(elemento.innerHTML);
```

```
<h1 class="text-center">Página del Módulo  
Desarrollo Web en Entorno Cliente</h1>  
<h2 class="text-center">                HOLA</h2>
```

Al igual que con *innerText* y *textContent*, *innerHTML* también permite modificar el contenido de texto de una etiqueta en la página web de forma dinámica, cambiando el valor que contiene dicha propiedad.

# [innerHTML]

Sin embargo, a diferencia de *innerText* y *textContent*, *innerHTML*, permite añadir etiquetas HTML de forma dinámica a la página web.

```
const elemento = document.querySelector('h2');  
elemento.innerHTML='<a href="pago.html">100 &yen;</a>';
```





La propiedad llamada *style* de la interfaz *Element* puede cambiar el estilo de un elemento de forma dinámica en la página web.

*style* es un objeto dentro del elemento que posee todas las propiedades CSS.

```
const elemento= document.querySelector('h2');  
elemento.style.backgroundColor="red";
```



**Ciclo Formativo DAW**



Profesor



Contenidos



Recursos

El nombre de las propiedades de *style* cambia con respecto a cómo son nombradas en las hojas de estilo CSS; en *style* siguen la convención *Camel Case*, y no *Kebab Case* como en CSS. Ejemplos:

CSS

*background-color*  
*font-family*

(style) JS

*backgroundColor*  
*fontFamily*

## [style]

La propiedad *style* contiene también el atributo *cssText* que permite asignar varias propiedades CSS al mismo tiempo, mediante una cadena en formato CSS; no obstante, si no se concatena sobrescribirá cualquier otro estilo preexistente.

```
const elemento= document.querySelector('h2');  
elemento.style.backgroundColor="red";  
console.log(elemento.style.cssText);
```

```
background-color: red;
```

El siguiente ejemplo hace uso del atributo *cssText* para asignar varias propiedades CSS al elemento `<h2>`.

```
elemento.style.cssText = 'background-color: black; color: white;';
```

**Ciclo Formativo DAW**

Profesor

Contenidos

Recursos

## (className)

El método *className* de la interfaz *Element* permite conocer el contenido del atributo *class* de html, devuelve una cadena que contiene el nombre de las clases CSS, separadas por espacios, que contiene el atributo *class*.

Se utiliza *className* en lugar de *class* para evitar conflictos con la palabra reservada *class* de JS, esta palabra reservada crea una clase del paradigma orientado a objetos (se estudiará en temas posteriores).

```
const elemento= document.querySelector('#Contenidos p');  
console.log(elemento.className);
```

*cabecera*

*className*, junto con los métodos de *classList* (que se verá a continuación), permite también aplicar clases a los distintos elementos del DOM.



## [className]

También se puede usar `className` para cambiar la clase que se aplica a un elemento.

```
const elemento= document.querySelector('h2');  
console.log(elemento.className);  
elemento.className='licencia';  
console.log(elemento.className);
```

text-center

licencia

Ciclo Formativo DAW

Profesor

Contenidos

Recursos

En el siguiente ejemplo, se aplica la clase `licencia` al elemento sin eliminar las clases que ya tuviera aplicadas.

```
const elemento= document.querySelector('h2');  
console.log(elemento.className);  
elemento.className= elemento.className + ' licencia';  
console.log(elemento.className);
```

text-center

text-center licencia

## [classList]

También está disponible, en la interfaz *Element*, la propiedad de solo lectura *classList* que devuelve una cadena que contiene el nombre de las clases CSS, separadas por espacios, que contiene el atributo *class*.

Este método devuelve un *DOMTokenList* cuyos elementos son las clases que se aplican al elemento.

```
const elemento= document.querySelector('h2');  
elemento.className= elemento.className + ' licencia';  
console.log(elemento.classList);
```

```
► DOMTokenList(2) ['text-center', 'licencia', value: 'text-center licencia']
```

## `classList.add`

`classList`, al ser una propiedad de solo lectura y un `DOMTokenList`, dispone de métodos útiles, como `add()` que permite añadir y aplicar nuevas clases a un elemento sin perder las clases que ya tuviera dicho elemento.

```
const elemento= document.querySelector('h2');  
console.log(elemento.className);  
elemento.classList.add('licencia');  
console.log(elemento.className);
```

```
text-center  
text-center licencia
```

El método `add()` también permite añadir varias clases a la vez, pasadas como argumentos

```
const elemento= document.querySelector('h2');  
elemento.classList.add('licencia', 'cursiva', 'negrita');  
console.log(elemento.className);
```

```
text-center licencia cursiva negrita
```

# [classList.toggle]

*toggle()* es otro método de *classList* que permite añadir y aplicar una clase si dicha clase no la posee el elemento, si la posee la elimina.

```
const elemento= document.querySelector('h2');  
console.log(elemento.className);  
elemento.classList.add('licencia');  
console.log(elemento.className);  
elemento.classList.toggle('licencia');  
console.log(elemento.className);
```

text-center
text-center licencia
text-center

**Ciclo Formativo DAW**

## [classList.contains]

*contains()* es un método de *classList* que devuelve *true* si el elemento posee la clase pasada como argumento y *false* en caso contrario.

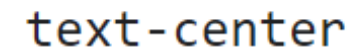
```
const elemento= document.querySelector('h2');  
elemento.classList.add('licencia');  
console.log(elemento.classList.contains('licencia'));
```

true

## `classList.remove`

`remove()` es un método de `classList` que elimina la clase o clases pasadas como argumento.

```
const elemento= document.querySelector('h2');  
console.log(elemento.className);  
elemento.classList.remove('text-center');  
console.log(elemento.className);
```



# Bibliografía y recursos online

- <https://developer.mozilla.org/es/docs/Glossary/DOM>
- [https://developer.mozilla.org/es/docs/Web/API/Document Object Model/Introduction](https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Introduction)
- [https://developer.mozilla.org/es/docs/Web/CSS/Attribute selectors](https://developer.mozilla.org/es/docs/Web/CSS/Attribute_selectors)
- <https://developer.mozilla.org/es/docs/Web/API/Element/innerHTML>
- <https://developer.mozilla.org/en-US/docs/Web/API/DOMTokenList>
- <https://programmerclick.com/article/6152965371/>