



UT04: MODELO DE OBJETOS DEL DOCUMENTO. DOM.

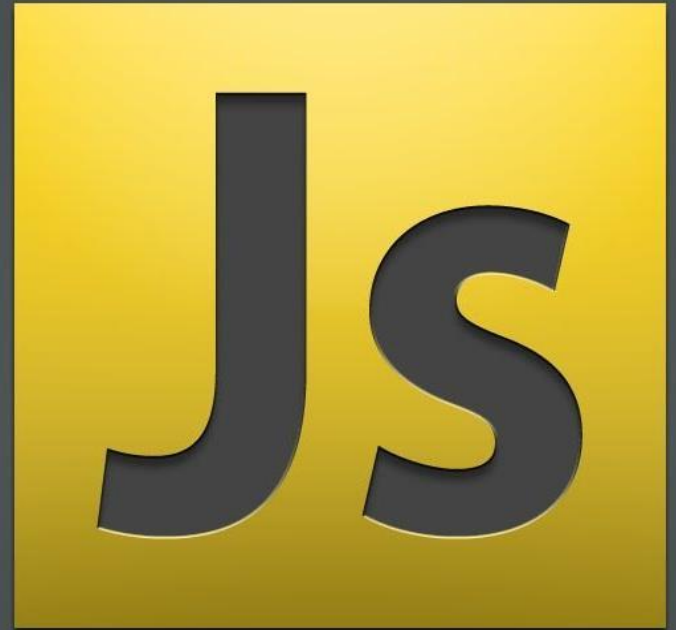
ÍNDICE

- 1.- Introducción al DOM
- 2.- Recorriendo el DOM
- 3.- Búsqueda: getElement y querySelector
- 4.- Atributos y propiedades
- 5.- Modificando el documento. Estilos y clases
- 6.- Tamaño de elementos y desplazamiento
- 7.- Tamaño de ventana y desplazamiento
- 8.- Coordenadas



1

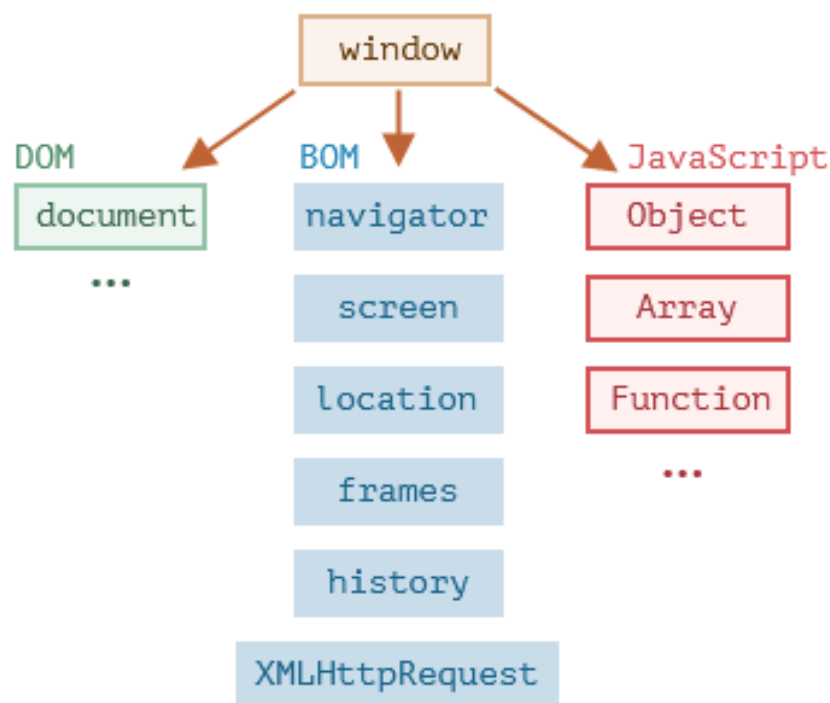
INTRODUCCIÓN AL DOM



JavaScript se puede ejecutar en diferentes plataformas (navegador, servidor web, ...) y cada uno de ellos proporciona una funcionalidad específica denominada **entorno de host**.

El entorno de host añade objetos propios y funciones adicionales al núcleo de lenguaje.

En la imagen se puede ver el **entorno de host en un navegador**.



El objeto raíz se llama **window**, y tiene dos roles:

- Es el objeto global para el código JavaScript
- Representa la ventana del navegador y proporciona métodos para controlarla.

```
function saluda() {  
    alert("Hola Mundo");  
}
```

```
// Las funciones globales son métodos del objeto global  
window.saluda();
```

```
alert(window.innerHeight); // Altura interior de la ventana
```

El **Document Object Model (DOM)** representa todo el contenido de la página como objetos que pueden ser modificados.

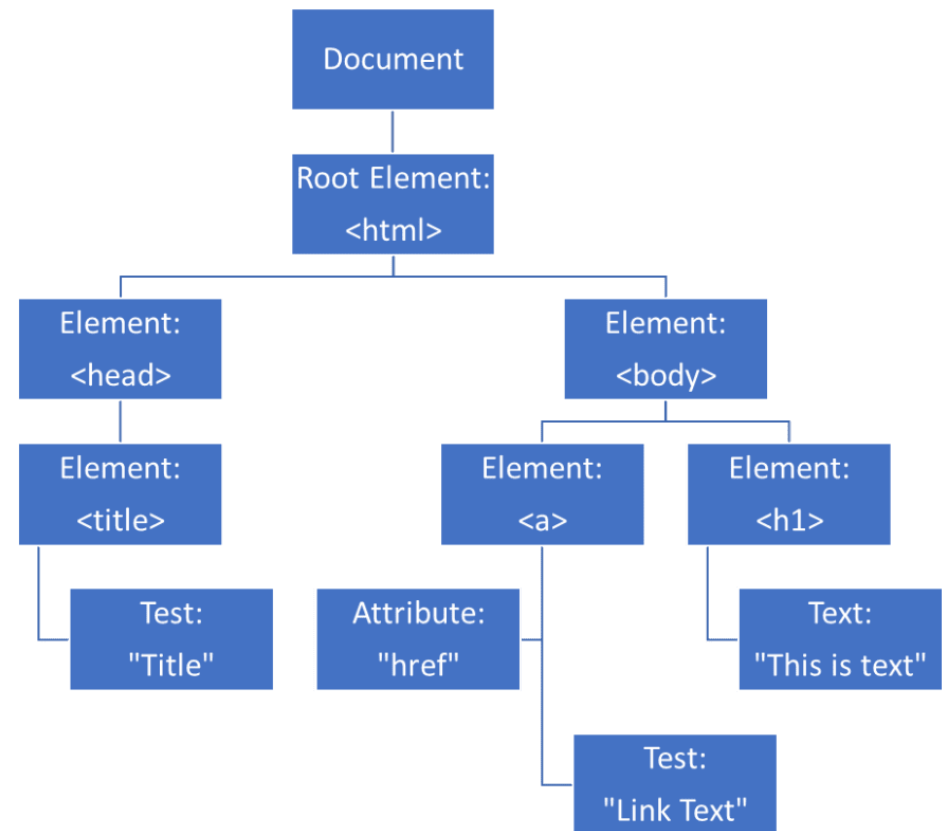
```
document.body.style.background = "red";  
setTimeout( () => document.body.style.background="", 1000 );
```

El **Browser Object Model (BOM)** son objetos adicionales proporcionados por el navegador para trabajar con todo excepto el documento.

```
alert(location.href);  
if ( confirm("Ir a Wikipedia?") ){  
    location.href = "https://wikipedia.org";  
}
```

La estructura de HTML son las etiquetas. Según DOM, **cada etiqueta HTML es un objeto**. El texto dentro de cada etiqueta también es un objeto.

Esto hace que podamos ver una página Web como un árbol de objetos.



En este árbol podemos ver:

- **document:** es el punto de entrada al DOM
- **Nodos de elementos:** que corresponden a las etiquetas HTML y forman la estructura del árbol.
- **Nodos de texto:** que corresponden a texto dentro de los elementos y siempre son nodos hoja.
- **Nodos de comentario:** correspondientes a los comentarios en el HTML

Podemos ver el
DOM en

[http://software.hixie.
ch/utilities/js/live-
dom-viewer/](http://software.hixie.ch/utilities/js/live-dom-viewer/)

Live DOM Viewer

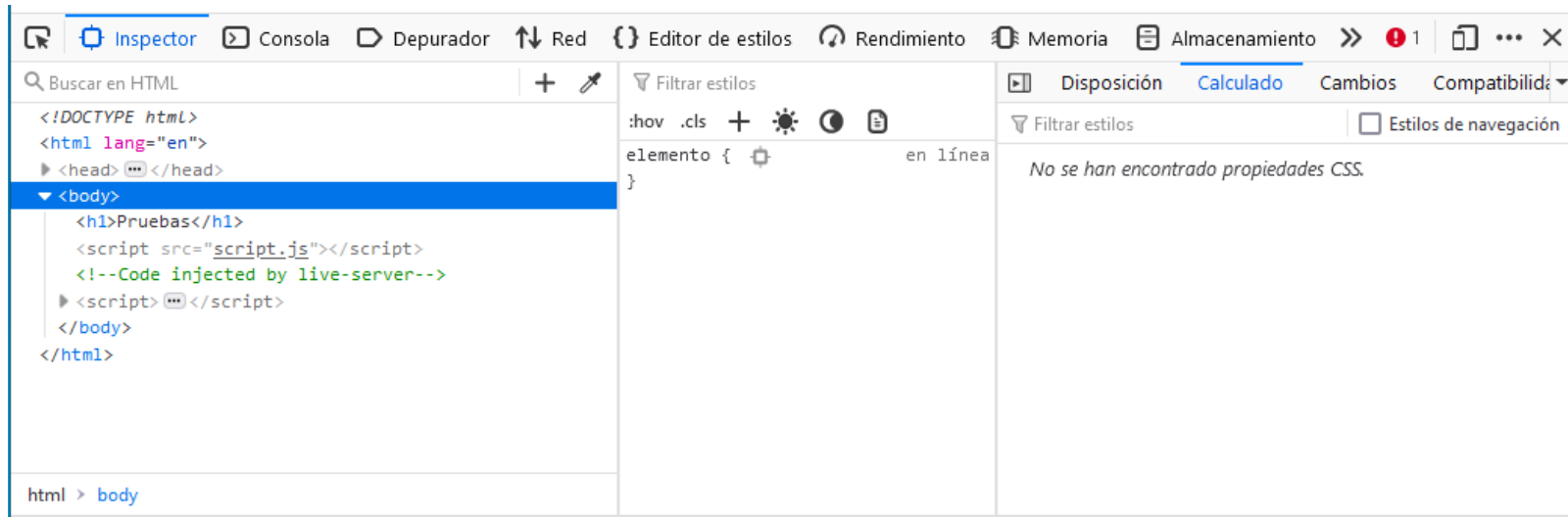
Markup to test ([permalink](#), [save](#), [upload](#), [download](#), [hide](#)):

```
<!DOCTYPE html>
<head>
  <title>Pruebas</title>
</head>
<body>
  <h1>Hola mundo!!!</h1>
</body>
```

DOM view ([hide](#), [refresh](#)):

```
└─ DOCTYPE: html
  └─ HTML
    └─ HEAD
      └─ #text:
        └─ TITLE
          └─ #text: Pruebas
        └─ #text:
      └─ #text:
    └─ BODY
      └─ #text:
        └─ H1
          └─ #text: Hola mundo!!!
        └─ #text:
```

Dentro del navegador, podemos ver el DOM en la pestaña **Inspector** de las Herramientas del desarrollador.



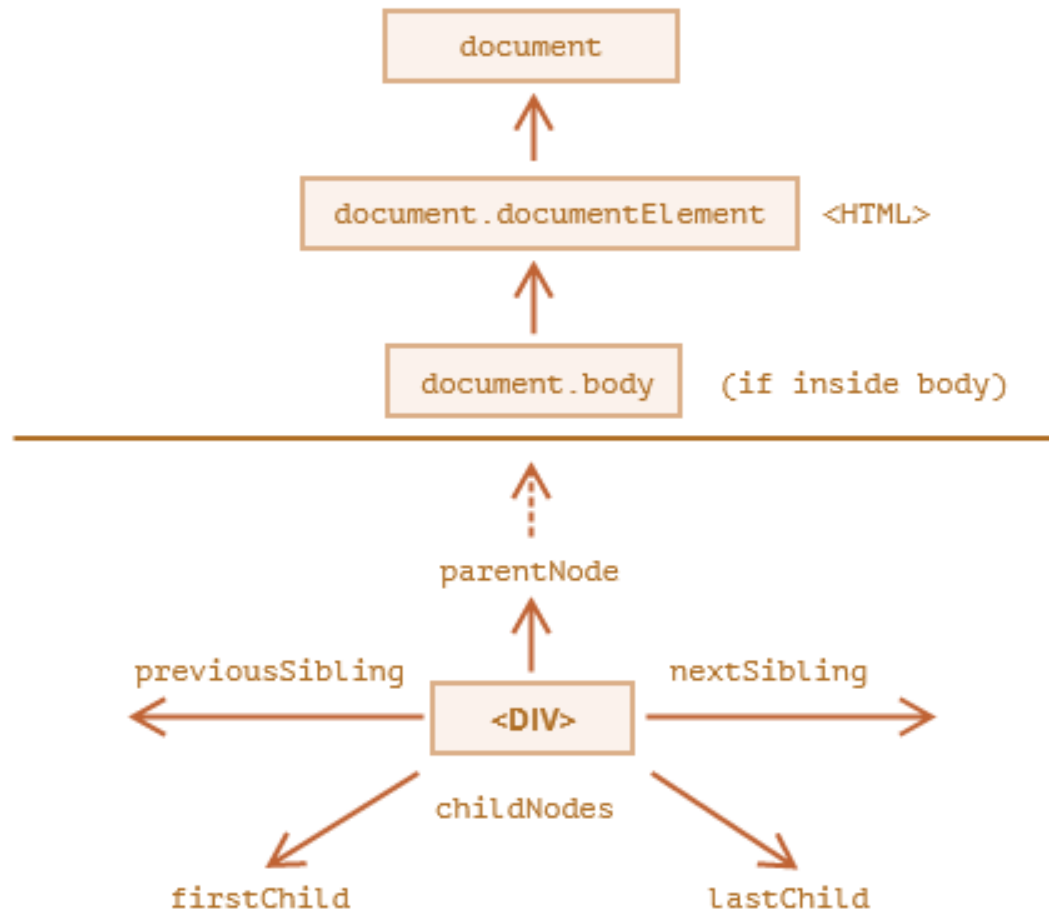
2

RECORRIENDO
EL DOM



Todas las operaciones en el DOM comienzan con el objeto **document**, que es el punto de entrada principal al DOM.

A la derecha se muestran los diferentes enlaces que permiten navegar por el DOM.



document.documentElement

Es el nodo superior del documento y corresponde a la etiqueta `<html>`

document.body

Este nodo es el que corresponde a la etiqueta `<body>`

document.head

Este nodo es el que corresponde a la etiqueta `<head>`

Los elementos anteriores, comunes a todas las páginas Web, pueden ser accedidos directamente.

Para acceder al resto de **nodos** habría dos maneras posibles para hacerlo:

- Recorrer el árbol DOM hasta llegar al nodo deseado
- Acceder directamente al nodo

Para **recorrer el árbol de nodos** disponemos de los siguientes métodos y propiedades:

- `Node.childNodes`
- `Node.firstChild`
- `Node.lastChild`
- `Node.parentNode`
- `Node.nextSibling`
- `Node.previousSibling`

Para **acceder directamente un nodo** las opciones son:

- `document.getElementById`
- `document.getElementsByClassName`
- `document.getElementsByName`
- `document.getElementsByTagName`
- `querySelector`
- `querySelectorAll`

Node.childNodes

Es una propiedad de solo lectura que devuelve una colección de hijos de un elemento dado.

```
console.log( document.body.childNodes );
```

```
▼ NodeList(8) [ #text, h1, #text, <!-- Code injected
  ▶ 0: #text "\n"
  ▶ 1: <h1>
  ▶ 2: #text "\n"
  ▶ 3: <!-- Code injected by live-server -->
  ▶ 4: #text "\n"
  ▶ 5: <script>
  ▶ 6: #text "\n\n"
  ▶ 7: <script src="script.js">
  ▶ 8: #text "\n"
  length: 9
```

Hay que tener cuidado porque, aunque parece que devuelve un array, realmente es una colección, por lo que **los métodos que conocemos para trabajar con arrays no se pueden utilizar directamente.**

Si queremos utilizarlos debemos convertirlo primero a array con **Array.from.**

```
let body = document.body.childNodes
console.log( Array.isArray(body) );           // false

body = Array.from(body);
console.log( Array.isArray(body) );           // true
```

Sin embargo, si queremos acceder directamente a un elemento sí podemos utilizar la notación de corchetes.

```
console.log(document.body.childNodes[1]); // <h1>
```

Node.firstChild y Node.lastChild

Permite un acceso rápido a la primera y a la última propiedad hijo de un elemento.

```
console.log( document.body.firstChild );  
console.log( document.body.lastChild );
```



Node.parentNode

Cada nodo del árbol DOM tiene un único nodo padre. Se puede obtener utilizando la propiedad **parentNode**.

Node.nextSibling y Node.previousSibling

Esta propiedad sirve para acceder a los nodos hermanos de un nodo determinado, considerándose como nodos hermanos aquellos que tienen un mismo padre.

Si un nodo no tiene un nodo hermano en la posición solicitada contendrá *null*

`document.getElementById(id)`

Las propiedades anteriores permiten acceder a un nodo desde otro nodo, pero normalmente es más cómodo acceder a un nodo directamente.

La función **getElementById()** permite obtener un nodo en función de su valor en el atributo de HTML id.

```
<body>
  <h1 id="titulo">Pruebas</h1>
  <h2 id="subtitulo">Este es el H2</h2>
</body>
```

```
let titulo = document.getElementById('titulo');
console.log(titulo.textContent);           // Pruebas
```


`document.getElementsByClassName(className)`

De forma análoga al anterior, este método devuelve una colección con todos los nodos del árbol DOM que tienen el valor indicado en la etiqueta *class* de HTML.

```
<h1 id="titulo">Pruebas</h1>
<ul>
  <li class="item">Primero</li>
  <li class="item">Segundo</li>
  <li class="item">Tercero</li>
</ul>
```

```
let a = document.getElementsByClassName('item');
console.log(a);
```

```
▼ HTMLCollection { 0: li.item , 1: li.item , 2: li.item , length: 3 }
  ► 0: <li class="item"> 
  ► 1: <li class="item"> 
  ► 2: <li class="item"> 
    length: 3
```

Si en lugar de invocar el método desde *document* lo hacemos desde otro nodo nos devolverá los nodos que tengan dicha clase y estén contenidos en el subárbol que parte de dicho nodo.

`document.getElementsByName(name)`

Similar al anterior pero utilizando el atributo *name* de HTML.

Cats:

```
<input name="animal" type="checkbox" value="Cats">
```

Dogs:

```
<input name="animal" type="checkbox" value="Dogs">
```

```
let num = document.getElementsByName("animal").length;  
document.getElementById("demo").innerHTML = num;
```

`document.getElementsByTagName(tagName)`

En este caso el criterio que se utiliza es el nombre de la etiqueta HTML.

Se puede utilizar el asterisco (*) como comodín que representa todos los elementos

```
<h1 id="titulo">Pruebas</h1>
  <ul>
    <li class="item">Primero</li>
    <li class="item">Segundo</li>
    <li class="item">Tercero</li>
  </ul>
```

```
let a = document.getElementsByTagName('li');
console.log(a);
```

```
▶ HTMLCollection { 0: li.item ◻ , 1: li.item ◻ , 2: li.item ◻ , length: 3 }
```

```
• GET http://127.0.0.1:5500/favicon.ico
```

Todos los elementos **getElementsByTagName*** devuelven una **colección viva**. Esto quiere decir que almacena el estado actual del documento, pero si este cambia se actualizan automáticamente.

```
<div>First div</div>

<script>
  let divs = document.getElementsByTagName('div');
  alert(divs.length);           // 1
</script>

<div>Second div</div>

<script>
  alert(divs.length);           // 2
</script>
```

En cambio, **querySelectorAll** devuelve una **colección estática** que refleja siempre el estado del documento en el momento en que se ejecutó el método.

```
<div>First div</div>

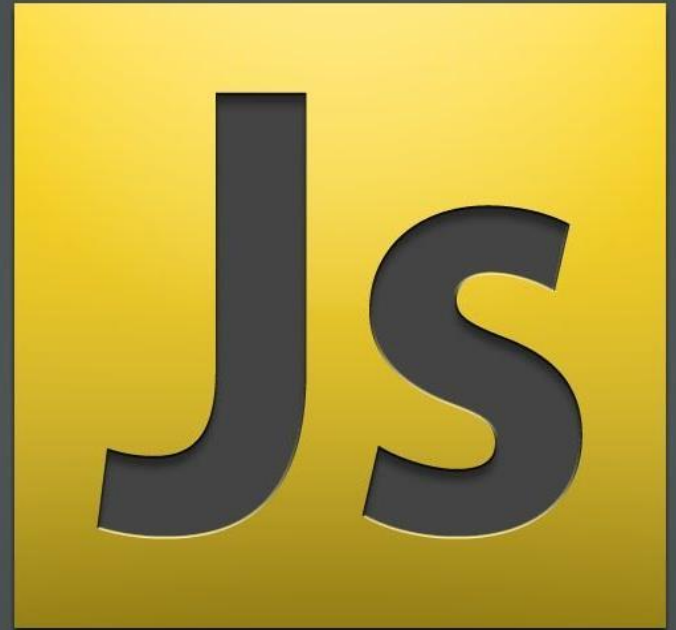
<script>
  let divs = document.querySelectorAll('div');
  alert(divs.length);           // 1
</script>

<div>Second div</div>

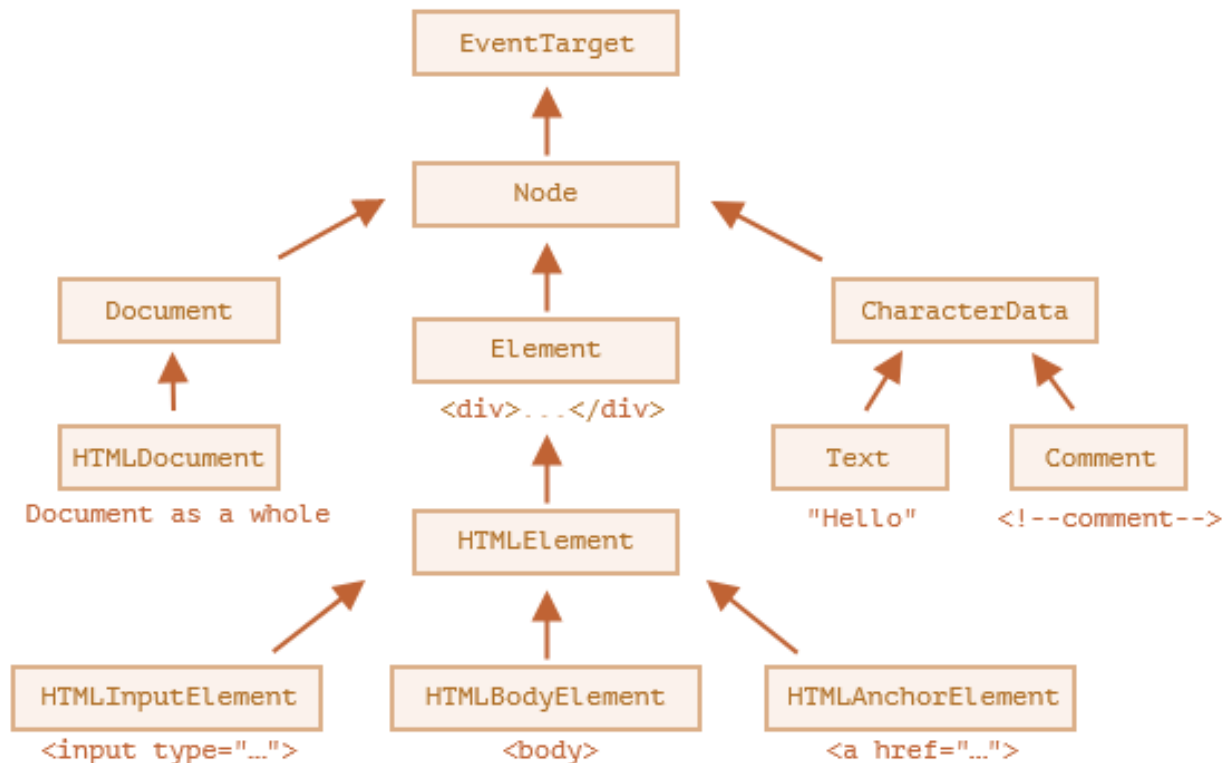
<script>
  alert(divs.length);           // 1
</script>
```

3

ATRIBUTOS Y
PROPIEDADES



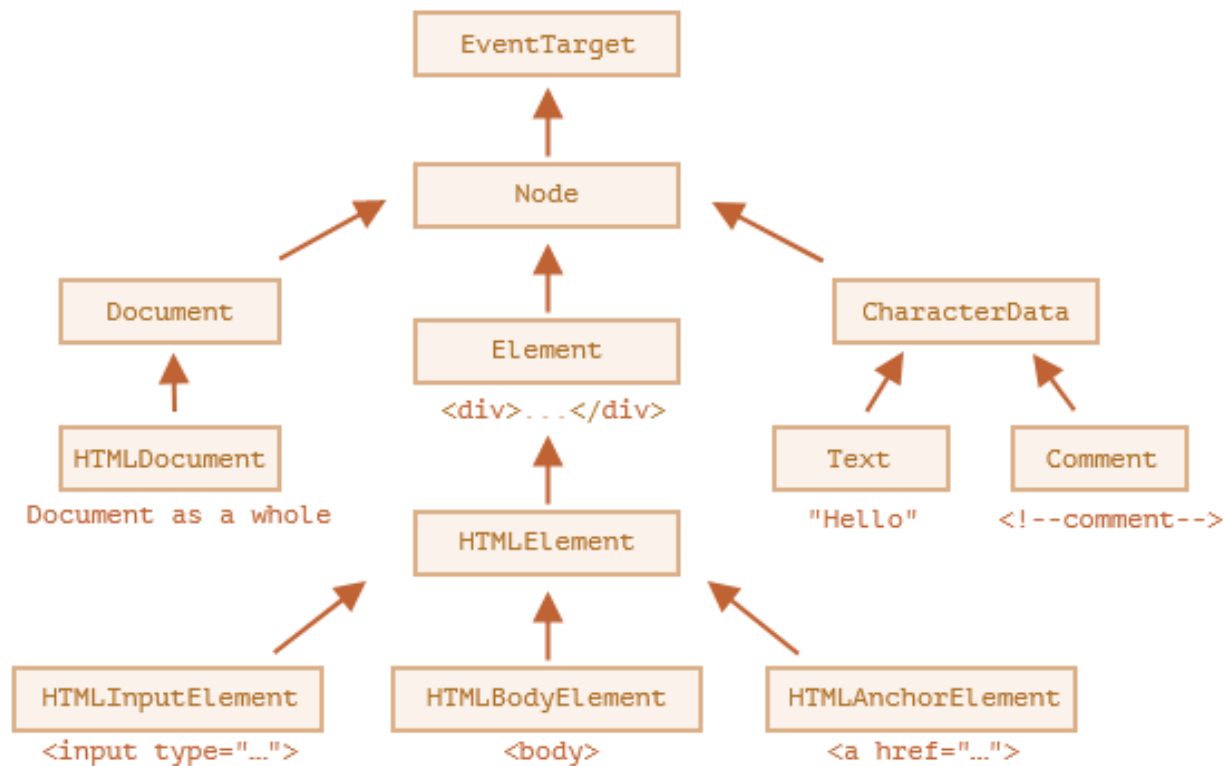
Todos los comandos anteriores devuelven uno o varios nodos de DOM, pero estos pueden ser de varias diferentes, organizadas en la siguiente jerarquía.



- **EventTarget**: clase abstracta que proporciona soporte para eventos al resto de clases.
- **Node**: otra clase abstracta que proporciona funcionalidad básica como las propiedades *parentNode*, *nextSibling*, ...
- **Document**: contiene el documento completo. El objeto global **document** corresponde con esta clase.
- **CharacterData**: clase abstracta de la que heredan las clases:
 - **Text**: los textos que hay dentro de los elementos
 - **Comment**: comentarios

- **Element:** es la clase base para los elementos del DOM. Proporciona métodos de navegación como *nextElementSibling* o *children* o de búsqueda como *getElementsByTagName* o *querySelector*.
- **HTMLElement:** es la clase base para todos los elementos HTML. De esta clase heredan otras como:
 - **HTMLInputElement:** para los elementos `<input>`
 - **HTMLBodyElement:** para los elementos `<body>`
 - **HTMLAnchoElement:** para los elementos `<a>`

De esto se deduce que el conjunto de propiedades y métodos de un nodo **viene como resultado de la cadena de herencia**.



Podemos saber el nombre de la clase de un elemento usando su método `toString` o haciendo referencia a la propiedad `name` de su constructor.

```
let elem = document.getElementById('div');  
console.log( elem.toString() );  
console.log( elem.constructor.name );
```

```
[object HTMLDivElement]
```

```
HTMLDivElement
```

Alternativamente podemos utilizar **instanceOf** para comprobar la clase del objeto.

Además, ten en cuenta que como hereda de múltiples clases será verdadero para todas ellas.

```
let elem = document.getElementById('div');
console.dir( elem instanceof HTMLBodyElement); // false
console.dir( elem instanceof HTMLDivElement); // true
console.dir( elem instanceof HTMLElement);    // true
console.dir( elem instanceof Element);        // true
console.dir( elem instanceof Node);           // true
console.dir( elem instanceof EventTarget);    // true
```

Nombre de la etiqueta: nodeName y tagName

Podemos saber el nombre de la etiqueta que corresponde a un nodo con las propiedades **nodeName** y **tagName**.

Diferencias:

- **tagName** solo existe para los elementos.
- En nodos que no son elementos, **nodeName** contiene una cadena con el tipo de nodo

```
let elem = document.getElementById('div');  
console.log( elem.tagName );           // DIV
```

Contenido interno: innerHTML

La propiedad **innerHTML** permite obtener el HTML dentro de la etiqueta.

```
<h1>Pruebas</h1>

    <div id="div">
        <p>Hola mundo</p>
    </div>
```

```
let elem = document.getElementById('div');
console.log( elem.innerHTML );           // <p>Hola mundo</p>
```

Esta propiedad es de lectura y escritura, por lo que nos **permite modificar el contenido de la página.**

```
<h1>Pruebas</h1>
  <div id="div">
    <p>Hola mundo</p>
  </div>
```

```
let elem = document.getElementById('div');
elem.innerHTML = '<b>Desarrollo de Aplicaciones Web</b>';
```

Pruebas

Desarrollo de Aplicaciones Web

HTML completo del elemento: outerHTML

Básicamente es como el innerHTML pero añadiendo el elemento en sí.

```
<h1>Pruebas</h1>

    <div id="div">
        <p>Hola mundo</p>
    </div>
```

```
let elem = document.getElementById('div');
console.log( elem.outerHTML ); // <div id="div">
                                //<p>Hola mundo</p></div>
```

Contenido del nodo de texto: `nodeValue/data`

Para los nodos que no heredan de *Element* disponemos de las propiedades **`nodeValue`** y **`data`**.

```
<body>
  Hola
  <!-- Comentario -->
  <script>
    let text = document.body.firstChild;
    console.log(text.data);    // Hola

    let comment = text.nextSibling;
    console.log(comment.data); // Comentario
  </script>
</body>
```

Texto puro: `textContent`

La propiedad **`textContent`** devuelve únicamente el texto dentro de un elemento (sin las etiquetas)

```
<body>
  <h1>Pruebas</h1>
  <div id="div">
    <p>Hola mundo</p>
  </div>
  <script src="script.js"></script>
</body>
```

```
console.log( document.body.textContent ); // Pruebas
                                              //Hola mundo
```

Por norma general es preferible utilizar esta propiedad para modificar el contenido de un elemento ya que, si se incluyeran etiquetas HTML *maliciosamente* en el texto no las trataría como tales, sino simplemente como texto.

```
let elem = document.getElementById('div');  
let str = '<h1>XXXXXXXXX</h1>';  
elem.innerHTML = str;
```

```
▼ <body cz-shortcut-listen="true">  
  <h1>Pruebas</h1>  
  ▼ <div id="div">  
    <h1>XXXXXXXXX</h1>  
  </div>
```

```
let elem = document.getElementById('div');  
let str = '<h1>XXXXXXXXX</h1>';  
elem.textContent = str;
```

```
▼ <body cz-shortcut-listen="true">  
  <h1>Pruebas</h1>  
  <div id="div"><h1>XXXXXXXXX</h1></div>  
  <script src="script.js"></script>
```

Ocultar etiquetas: propiedad hidden

El atributo **hidden** de HTML indica si un elemento está visible, por lo que podemos utilizar la propiedad homónima para ocultar un elemento en el DOM.

```
<h1>Pruebas</h1>
<div id="div">
  <p>Hola mundo</p>
</div>
<h1>Fin</h1>
```

```
let elem = document.getElementById('div');
elem.hidden = true;
```

Pruebas

Fin

El contenido de un campo: value

Esta propiedad pertenece a los elementos `<input>`, `<select>` y `<textarea>` y contiene el valor del mismo.

```
<h1>Pruebas</h1>
<label for="txt">Introduce un valor:</label>
<input type="text" name="txt" id="input_text">
```

```
let txt = document.getElementById('input_text');
console.log(txt.value) // 1234
```

Pruebas

Introduce un valor:

Enlaces: href

En el caso de la etiqueta `<a>` podemos obtener el valor del atributo href con la propiedad del mismo nombre.

El atributo id

Otro atributo al que en ocasiones querremos acceder es el **id**, el cual se puede obtener con la propiedad del mismo nombre.

Esto se puede extrapolar a todos los atributos HTML estándar, de forma que siempre se corresponden con una propiedad homónima.

Aparte de una propiedad por atributo, todos los elementos disponen de la propiedad **attributes** que contienen un array con todos los atributos.

```
let txt = document.getElementById('input_text');  
console.log(txt.attributes)
```

```
▼ NamedNodeMap(3) [ type="text", name="txt", id="input_text" ]  
  ▶ 0: type="text"  
  ▶ 1: name="txt"  
  ▶ 2: id="input_text"
```


La mayoría de las propiedades que corresponden a atributos contienen su valor como un *string*, pero hay excepciones:

- La propiedad *input.checked* para casillas de verificación es un booleano.
- El valor de la propiedad *style* se indica como cadena en el atributo, pero la propiedad almacena un objeto de tipo `CSSStyleDeclaration`.

También se pueden manipular los atributos con los siguientes métodos:

`elem.hasAttribute(nombre)` – comprueba si existe.

`elem.getAttribute(nombre)` – obtiene el valor.

`elem.setAttribute(nombre, valor)` – establece el valor.

`elem.removeAttribute(nombre)` – elimina el atributo.

Estas funciones, aparte de servir para manipular los atributos HTML estándar, los podemos utilizar para trabajar con **atributos no estándar**.

Normalmente estos atributos se utilizan para pasar datos personalizados a HTML o para *marcar* elementos HTML para JavaScript.

```
<!-- marque el div para mostrar "nombre" aquí -->
<div show-info="nombre"></div>
<!-- y "edad" aquí -->
<div show-info="edad"></div>

<script>
  // el código encuentra un elemento con la marca y muestra lo
  que se solicita
  let user = {
    nombre: "Pete",
    edad: 25
  };

  for(let div of document.querySelectorAll('[show-info]')) {
    // inserta la información correspondiente en el campo
    let field = div.getAttribute('show-info');
    div.innerHTML = user[field]; // primero Pete en "nombre",
    luego 25 en "edad"
  }
</script>
```

```
<style>
  /* Los estilos se basan en atributo "order-state" */
  .order[order-state="nuevo"] { color: green; }
  .order[order-state="pendiente"] { color: blue; }
  .order[order-state="cancelado"] { color: red; }
</style>

<div class="order" order-state="nuevo">
  Un nuevo pedido.
</div>

<div class="order" order-state="pendiente">
  Un pedido pendiente.
</div>

<div class="order" order-state="cancelado">
  Un pedido cancelado
</div>
```

¿Por qué usar atributos personalizados en lugar de clases?

Porque los atributos son más sencillos de administrar, por ejemplo, se puede cambiar el estado del ejemplo anterior con la siguiente orden

```
// un poco más simple que eliminar/agregar clases  
div.setAttribute('order-state', 'canceled')
```

¿Y qué pasa si creamos un atributo personalizado y luego el estándar introduce uno con el mismo nombre? Para evitar estos posibles conflictos el estándar contempla los **atributos data-***

Estos atributos están reservados para programadores y están disponibles a través de una propiedad del elemento llamada **dataset**.

En esta propiedad se les elimina el prefijo **data-** y si están formados por varias palabras se pasan a formato camelCase.

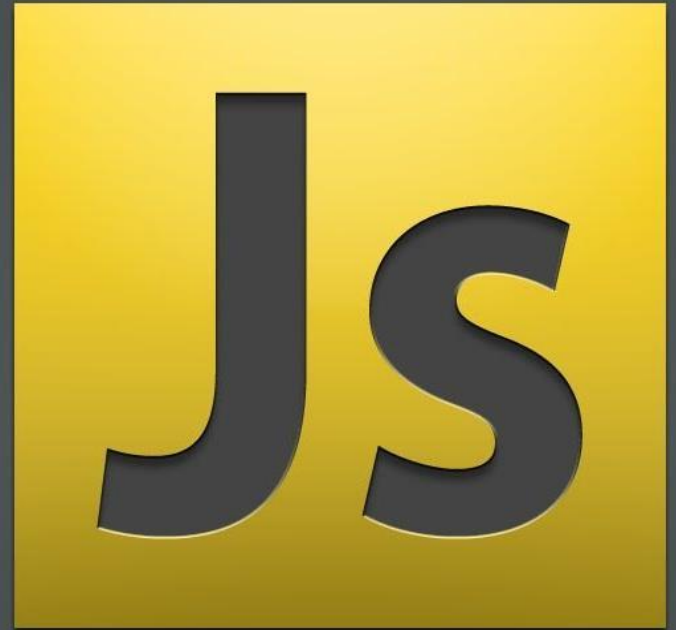
```
<style>
.order[data-order-state="nuevo"] { color: green; }
.order[data-order-state="pendiente"] { color: blue; }
.order[data-order-state="cancelado"] { color: red; }
</style>

<div id="order" class="order" data-order-state="nuevo">
Una nueva orden.
</div>

<script>
// Leer
alert(order.dataset.orderState); // nuevo
// modificar
order.dataset.orderState = "pendiente"; // (*)
</script>
```


4

MODIFICANDO EL
DOCUMENTO:
ESTILOS Y CLASES



Modificar el DOM es el elemento clave para crear páginas Web dinámicas.

Desde JavaScript podemos crear elementos, insertarlos, modificarlos o eliminarlo.

`document.createElement(tag)`

Crea un **nodo elemento** HTML especificado por su nombre de etiqueta que se pasa como parámetro.

El elemento creado está vacío y además no se incluye en el árbol DOM, sino que habrá que hacerlo posteriormente.

Al nodo creado se le pueden modificar las propiedades:

```
// 1. Crear elemento <div>
let div = document.createElement('div');
// 2. Establecer su clase a "alert"
div.className = "alert";
// 3. Agregar el contenido
div.innerHTML = "<strong>¡Hola!</strong> Mundo";
```

document.createTextNode(*text*)

Crea un nuevo **nodo texto** con el texto que se le pasa como parámetro.

Métodos de inserción

Una vez creado un nodo, será necesario insertarlo en algún lugar del documento.

Para ello disponemos de las siguientes funciones:

- **node.append()**: agrega nodos al final de *node*
- **node.prepend()**: agrega nodos al principio de *node*
- **node.before()**: agrega nodos antes de *node*
- **node.after()**: agrega nodos después de *node*
- **node.replaceWith()**: reemplaza *node* con los nodos dados

```
<ol id="ol">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>
```

```
<script>
ol.before('before'); // inserta string "before" antes de <ol>
ol.after('after'); // inserta string "after" después de <ol>

let liFirst = document.createElement('li');
liFirst.innerHTML = 'prepend';
ol.prepend(liFirst); // inserta liFirst al principio de <ol>

let liLast = document.createElement('li');
liLast.innerHTML = 'append';
ol.append(liLast); // inserta liLast al final de <ol>
</script>
```

before

1. prepend
2. 0
3. 1
4. 2
5. append

after

Eliminación de nodos

Para eliminar un nodo disponemos de la función **node.remove()**

Clonando nodos

Se puede clonar un nodo con **node.cloneNode(*deep*)**. El argumento es un booleano que puede ser:

- **true**: realiza una clonación profunda, clonando todo el árbol de nodos dentro de *node*
- **false**: únicamente clona el nodo, no sus nodos hijo.

Uno de los usos más habituales de manipulación del DOM es para aplicar estilos de forma dinámica.

Hay dos formas de hacer esto:

- Crear clases css y agregarlas con JavaScript
- Escribir las propiedades directamente en *style*

Lo más aconsejable es utilizar clases salvo que queramos calcular los valores del CSS en tiempo de ejecución.

La propiedad que corresponde al atributo *class* es **elem.className**.

```
<h1>Desarrollo de Aplicaciones Web</h1>
<h2>Desarrollo Web en Entorno Cliente</h2>
<div id="main" class="container">
  DWECE
</div>
```

```
let container = document.getElementById('main');
console.log(container.className);           // container
container.className = "section";
console.log(container.className);           // section
```

El valor de *className* es una cadena, por lo que puede ser complicado añadir o quitar clases.

Para ello hay otra propiedad llamada **elem.classList** que dispone de una serie de funciones para manipular las clases del elemento.

- **elem.classList.add/remove()**: agrega o elimina una clase
- **elem.classList.toggle()**: agrega la clase si no existe, si no la elimina.
- **elem.classList.contains()**: devuelve *true* si el elemento tiene la clase indicada.

La otra posibilidad para aplicar estilos es manipular directamente la propiedad **elem.style**.

```
let container = document.getElementById('main');  
main.style.color = 'blue';
```

Hay una propiedad dentro de *style* para cada estilo. En el caso de **propiedades de varias palabras** se utiliza la nomenclatura camelCase.

```
background-color => elem.style.backgroundColor  
z-index          => elem.style.zIndex  
border-left-width => elem.style.borderLeftWidth
```

Si queremos eliminar una propiedad podemos asignarle la cadena vacía.

```
document.body.style.backgroundColor = "blue";  
  
setTimeout(() => document.body.style.backgroundColor = "",  
1000);
```

Con las propiedades anteriores podemos modificar propiedades una a una, pero no se pueden asignar todas directamente ya que *style* es de solo lectura.

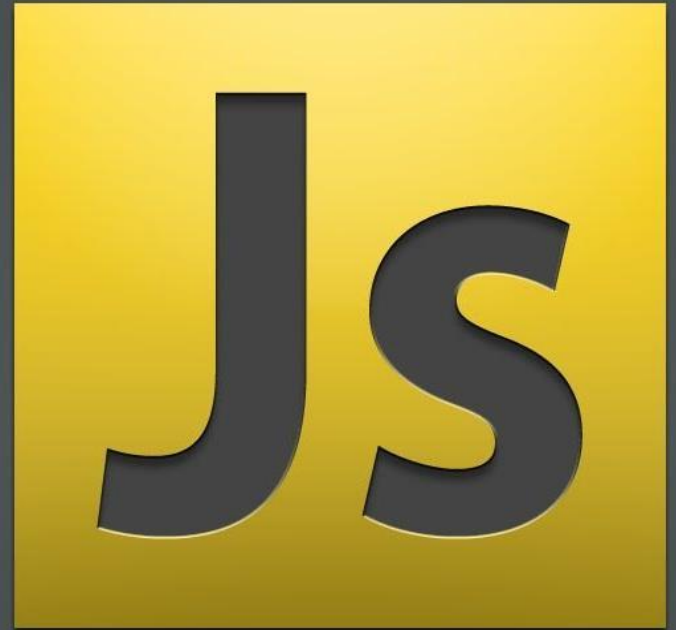
Para añadir múltiples propiedades simultáneamente hay que usar la propiedad especial **style.cssText**.

```
<div id="div">Button</div>

<script>
  div.style.cssText=`color: red !important;
    background-color: yellow;
    width: 100px;
    text-align: center;
  `;
</script>
```

6

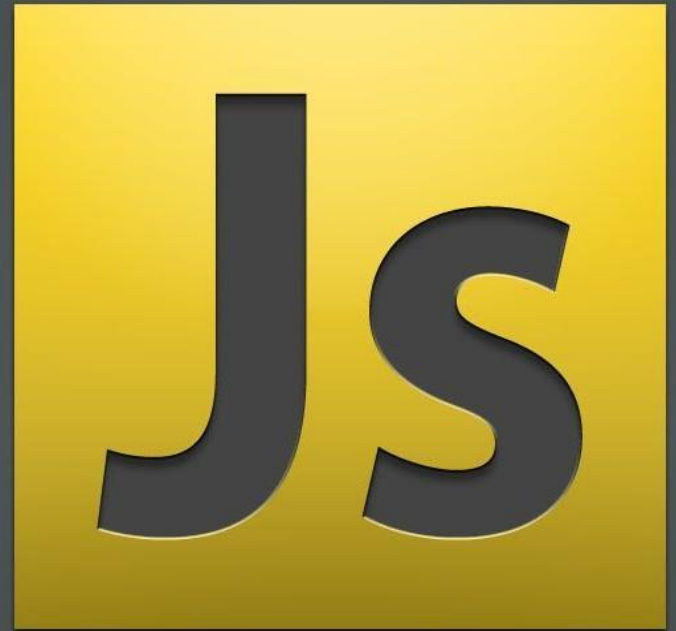
TAMAÑO DE
ELEMENTOS Y
DESPLAZAMIENTO



XXXX

7

TAMAÑO DE
VENTANA Y
DESPLAZAMIENTO



XXXX

8

COORDENADAS



XXXX

