

Unidad 4 - El DOM

Propiedades clave para interactuar con el DOM

Acceso directo a elementos básicos

Permiten acceder a los elementos del documento:

- `document` → Representa todo el documento.
- `document.body` → Cuerpo del documento.
- `document.head` → Cabecera del documento.

Navegación por el DOM

Recorrido por elementos del DOM

| Propiedad | Descripción |
|--|--------------------|
| <code>elemento.parentElement</code> | Elemento padre |
| <code>elemento.previousElementSibling</code> | Hermano anterior |
| <code>elemento.nextElementSibling</code> | Hermano siguiente |
| <code>elemento.children</code> | Colección de hijos |
| <code>elemento.firstElementChild</code> | Primer hijo |
| <code>elemento.lastElementChild</code> | Último hijo |

Recorrido por todos los nodos del DOM

| Propiedad | Descripción |
|---------------------------------------|--------------------------|
| <code>elemento.parentNode</code> | Nodo padre |
| <code>elemento.previousSibling</code> | Nodo hermano anterior |
| <code>elemento.nextSibling</code> | Nodo hermano siguiente |
| <code>elemento.childNodes</code> | Colección de nodos hijos |
| <code>elemento.firstChild</code> | Primer nodo hijo |
| <code>elemento.lastChild</code> | Último nodo hijo |

Colecciones

Las colecciones tienen estas propiedades:

- Iterables (`for...of`).
- Accesibles por índice (`[]`).
- Propiedad `.length` para el tamaño.
- Transformables en arrays (`Array.from()`).
- Solo lectura.
- Algunas son colecciones vivas.

Ejemplo → `elemento.children`

Propiedades específicas para tablas

Los elementos de las tablas tienen propiedades que facilitan acceder a sus elementos.

| Propiedad | Descripción |
|---------------------------------|--|
| <code>tabla.caption</code> | Elemento <code><caption></code> . |
| <code>tabla.thead</code> | Elemento <code><thead></code> . |
| <code>tabla.tBodies</code> | Colección de <code><tbody></code> . |
| <code>tabla.tFoot</code> | Elemento <code><tfoot></code> . |
| <code>tabla.rows</code> | Colección de filas (<code><tr></code>). |
| <code>tbody.rows</code> | Filas (<code><tr></code>) en un <code><tbody></code> . |
| <code>tr.cells</code> | Celdas (<code><td></code> y <code><th></code>) de un <code><tr></code> . |
| <code>tr.sectionRowIndex</code> | Índice de la fila en su sección. |
| <code>tr.rowIndex</code> | Índice de la fila en la tabla. |
| <code>celda.cellIndex</code> | Índice de la celda en su fila. |

Métodos para búsqueda en el DOM

Por ID

- `document.getElementById("id")`
- Exclusivo de `document`.

Por etiqueta, clase o nombre

- Colección viva:
 - `elemento.getElementsByTagName("etiqueta")` → Elementos descendientes con esa etiqueta.
 - `elemento.getElementsByTagName("*")` → Todos los elementos descendientes.
 - `elemento.getElementsByClassName("clase")` → Elementos descendientes de esa clase.
 - `elemento.getElementsByName("nombre")` → Elementos descendientes cuya propiedad `name` tiene ese valor.

Por selectores CSS

| Método | Descripción |
|--|--|
| <code>elemento.querySelector("selector")</code> | Primer descendiente que coincide. |
| <code>elemento.querySelectorAll("selector")</code> | Colección de descendientes coincidentes. |

Recorrido hacia antecesores

| Método | Descripción |
|---|---|
| <code>elemento.closest("selector")</code> | Primer antecesor coincidente. Incluye el propio elemento. |

Validación por selectores

| Método | Descripción |
|---|---|
| <code>elemento.matches("selector")</code> | Booleano: indica si el elemento coincide. |

Tipos y propiedades de nodos del DOM

Tipos de nodos

- `Element`, `Text`, `Comment`, `Document`, etc.
- Jerarquía: Los nodos son subtipos de `Node`.

Propiedades comunes a todos los nodos

| Propiedad | Descripción |
|----------------------------|---|
| <code>nodo.nodeType</code> | Tipo numérico del nodo (<code>1</code> : Element, <code>3</code> : Text, <code>8</code> : Comment, <code>9</code> : Document, etc.). |
| <code>nodo.nodeName</code> | Nombre del nodo, <code>tagName</code> (mayúsculas para elementos, ej.: <code>"DIV"</code>). |

Propiedades de los nodos de tipo texto (`Text`)

| Propiedad | Descripción |
|--|--|
| <code>nodo.nodeValue</code> y <code>nodo.data</code> | Contienen el contenido del nodo de texto. Se utiliza comúnmente la propiedad <code>data</code> . |

Propiedades de los nodos de tipo elemento (`Element`)

| Propiedad | Descripción |
|-----------------------------|--|
| <code>el.tagName</code> | Cadena que indica el tipo de elemento. Equivalente a <code>nodeName</code> y siempre en mayúsculas (por ejemplo, <code>"SPAN"</code>). |
| <code>el.innerHTML</code> | Cadena con el contenido HTML del elemento. Si contiene errores, JavaScript los corrige automáticamente. Permite insertar scripts, pero no ejecutarlos. |
| <code>el.outerHTML</code> | Cadena con el contenido HTML del elemento, incluyendo la etiqueta del propio elemento. |
| <code>el.textContent</code> | Cadena con el contenido en texto puro del elemento (sin formato HTML). |
| <code>el.id</code> | Cadena que representa el atributo <code>id</code> del elemento. |
| <code>el.hidden</code> | Valor booleano que indica si el elemento está visible (<code>false</code>) o está oculto (<code>true</code>). |

Propiedades específicas de los nodos de tipo elemento (`Element`)

Estas propiedades dependen del tipo de elemento:

| Propiedad | Descripción |
|-----------------------|---|
| <code>el.value</code> | Cadena que representa el valor del elemento. Aplicable a etiquetas como <code>INPUT</code> , <code>SELECT</code> o <code>TEXTAREA</code> . |
| <code>el.href</code> | Cadena que representa el enlace en etiquetas como <code>A</code> . |
| <code>el.src</code> | Cadena que representa la ruta de una fuente, imagen o script en etiquetas como <code>IMG</code> , <code>SCRIPT</code> , o <code>IFRAME</code> . |

Atributos de las etiquetas HTML y propiedades del DOM

Este apartado explora la relación entre los atributos de las etiquetas HTML y las propiedades de los nodos del DOM asociados.

Por ejemplo, en el siguiente código HTML, el nodo correspondiente se manipula desde el DOM de esta forma

```
<img id="etiqueta" src="" alt="" />
```

```
let img = document.getElementById("etiqueta");
console.log(img.id);
console.log(img.src);
console.log(img.alt);
```

Creación y características de las propiedades del DOM

1. Generación de propiedades desde los atributos estándar

- Al cargar la página, se crean propiedades para los atributos estándar definidos en el elemento HTML.
- Ejemplo:
 - `.id` → disponible en todos los elementos.
 - `.src` → exclusivo para elementos como `IMG`.
- Los atributos no estándar no generan automáticamente propiedades en el nodo del DOM.

2. Tipos de datos de las propiedades del DOM

- Los atributos HTML son siempre cadenas de texto.
- Las propiedades del DOM pueden tener otros tipos, por ejemplo:
 - `.hidden` → `boolean`
 - `.checked` → `boolean`
 - `.style` → `object`

3. Diferencias entre valores de atributos y propiedades

- El valor de una propiedad no siempre coincide con el valor del atributo correspondiente, incluso si comparten tipo.
- Ejemplo:
 - El atributo `href` (en un elemento `A`) puede contener una URL relativa, completa o un `#hash`.
 - La propiedad `href` siempre contiene una URL absoluta.

4. Sincronización entre atributos y propiedades

- La mayoría de las veces, los cambios en un atributo estándar se reflejan en la propiedad correspondiente, y viceversa.
- Excepción notable:

- `.value` → Si el atributo cambia, la propiedad se actualiza; pero si cambia la propiedad, no se refleja en el atributo.

Métodos para gestionar atributos (estándar o personalizados)

| Método | Descripción |
|---|---|
| <code>el.hasAttribute("nombre")</code> | Devuelve <code>true</code> si el atributo existe. |
| <code>el.getAttribute("nombre")</code> | Obtiene el valor del atributo como cadena. |
| <code>el.setAttribute("nombre", "valor")</code> | |
| <code>el.removeAttribute("nombre")</code> | Elimina el atributo del elemento. |
| <code>el.attributes</code> | Devuelve una colección de todos los atributos del elemento con propiedades <code>name</code> y <code>value</code> . |

Nota: Los nombres de los atributos HTML no distinguen entre mayúsculas y minúsculas. Ejemplo: `"id"`, `"ID"`, `"Id"` son equivalentes.

dataset

JavaScript permite crear y gestionar atributos personalizados fácilmente usando `dataset`.

Condiciones:

- Los nombres de los atributos deben comenzar con `data-`, por ejemplo: `data-about` o `data-user-name`.
- Se puede acceder mediante `element.dataset.nombre` (donde `"nombre"` corresponde al sufijo del atributo).

Ejemplo:

Dado el siguiente HTML → `<p id="p" data-about="Mi párrafo favorito">Texto</p>`

Podemos acceder al atributo personalizado de dos formas:

```
document.getElementById("p").dataset.about; // "Mi párrafo favorito"
document.getElementById("p").getAttribute("data-about"); // "Mi párrafo favorito"
```

Modificar un documento

Creación de elementos o nodos de texto

- Crear un nodo de tipo elemento (`ELEMENT`) → `elemento = document.createElement("etiqueta");`
- Crear un nodo de tipo texto (`TEXT`) → `texto = document.createTextNode("texto");`

Inserción de nodos en el árbol del DOM

Para insertar un nodo, primero debemos localizar el elemento de referencia dentro del DOM y usar los métodos adecuados según la ubicación deseada:

1. Insertar como último hijo

- `elem.append(...nodosOStrings)` → Inserta nodos o cadenas de texto al final del contenido del elemento `elem`.
 - Las cadenas se convierten automáticamente en nodos de texto.
 - Se pueden añadir varios nodos o cadenas a la vez.

Ejemplos:

- Añadir un elemento:

```
let div = document.createElement("div");
let p = document.createElement("p");
div.append(p);
console.log(div.childNodes); // NodeList [ <p> ]
```

- Añadir texto:

```
let div = document.createElement("div");
div.append("Texto adicional");
console.log(div.textContent); // "Texto adicional"
```

- Añadir múltiples nodos y texto:

```
let div = document.createElement("div");
let p = document.createElement("p");
div.append("Texto", p);
console.log(div.childNodes); // NodeList [ #text "Texto", <p> ]
```

2. Insertar como primer hijo

- `elem.prepend(...nodosOStrings)` → Inserta nodos o cadenas al inicio del contenido de `elem`.

3. Insertar antes de un elemento

- `elem.prepend(...nodosOStrings)` → Inserta nodos o cadenas justo antes de `elem`, dentro del mismo padre.

4. Insertar después de un elemento

- `elem.after(...nodosOStrings)` → Inserta nodos o cadenas justo después de `elem`, dentro del mismo padre.

Nota: Estos métodos no solo permiten insertar nuevos nodos, sino que también pueden mover nodos existentes dentro del árbol del DOM.

Reemplazo de nodos

- `elem.replaceWith(...nodosOStrings)` → Reemplaza el elemento `elem` con el conjunto de nodos o cadenas especificadas.
 - Las cadenas se convierten en nodos de texto.

Insertar HTML como texto con `insertAdjacentHTML`

El método `elem.insertAdjacentHTML(posición, texto)` permite insertar contenido HTML o XML directamente en el árbol del DOM, sin alterar los elementos existentes dentro del nodo. A diferencia de `innerHTML`, no reemplaza el contenido del elemento.

| Posición | Descripción |
|--------------------------|---|
| <code>beforebegin</code> | Inserta antes del propio elemento <code>elem</code> . |
| <code>afterbegin</code> | Inserta como el primer hijo dentro de <code>elem</code> . |
| <code>beforeend</code> | Inserta como el último hijo dentro de <code>elem</code> . |
| <code>afterend</code> | Inserta después del propio elemento <code>elem</code> . |

```

<!-- beforebegin --
>
<p>
  <!-- afterbegin -
->
    foo
  <!-- beforeend --
>
</p>
<!-- afterend -->

```

```

// HTML inicial:
// <div id="one">one</div>

let d1 = document.getElementById("one");
d1.insertAdjacentHTML("afterend", '<div id="two">two</div>');

// Resultado en el DOM:
// <div id="one">one</div><div id="two">two</div>

```

Eliminar nodos

`elem.remove()` → Elimina el nodo especificado del DOM, incluyendo sus hijos.

```

let elem = document.getElementById("example");
elem.remove(); // Elimina el elemento con id="example" del DOM.

```

Clonar nodos

El método `elem.cloneNode(clonarHijos)` permite duplicar un nodo. Se puede optar por una **clonación profunda** (incluyendo todos los nodos hijos) o **clonación superficial** (sin incluir los hijos).

```

<div class="alert" id="div">
  <strong>¡Hola!</strong> Este es un mensaje importante.
</div>
<script>
  let divOriginal = document.getElementById("div");

  // Crear una clonación profunda:
  let divClon = divOriginal.cloneNode(true);
  divClon.querySelector("strong").innerHTML = "¡Adiós!"; // Modificar el cl

  // Insertar el clon después del original:

```

```
divOriginal.after(divClon);

// Resultado en el DOM:
// <div class="alert" id="div"><strong>¡Hola!</strong> Este es un mensaje
// <div class="alert"><strong>¡Adiós!</strong> Este es un mensaje importa
</script>
```

Estilos y clases

Métodos para aplicar estilos a un elemento

- **Mediante clases:** utilizando el atributo `class`. Ejemplo: `<div class="mi-clase">`
- **Mediante estilos en línea:** utilizando el atributo `style`. Ejemplo: `<div style="color: red;">`

Recomendación: siempre que sea posible, utiliza clases (`class`) para mantener el código más limpio y reutilizable.

Clases: Propiedades y métodos

`elem.className` → Representa el atributo `class` del elemento. Se puede usar para leer o modificar directamente todas las clases del elemento.

```
<body class="documento importancia1">
  <script>
    alert(document.body.className); // documento importancia1
  </script>
</body>
```

También puedes manipular el atributo `class` utilizando los métodos genéricos `getAttribute` y `setAttribute`:

```
<body class="documento importancia1">
  <script>
    alert(document.body.getAttribute("class")); // documento importancia1
  </script>
</body>
```

`elem.classList` → Proporciona métodos más avanzados para trabajar con clases.

| Método | Descripción |
|--|---|
| <code>el.classList.add("clase", ...)</code> | Añade una o varias clases. |
| <code>el.classList.remove("clase", ...)</code> | Elimina una o varias clases. |
| <code>el.classList.toggle("clase")</code> | Alterna una clase: la añade si no está o la elimina si ya está. |
| <code>el.classList.replace("c1", "c2")</code> | Sustituye una clase existente por otra. |
| <code>el.classList.contains("clase")</code> | Devuelve <code>true</code> si el elemento contiene la clase especificada. |

`classList` es iterable, por lo que puedes recorrer las clases del elemento:


```
for (let clase of document.body.classList) {
    alert(clase); // documento, activo
}
```

Estilos

`elem.style` → Permite manipular estilos en línea definidos en el atributo `style`.

Para asignar un estilo:

```
div.style.color = "red";
div.style.backgroundColor = "blue"; // camelCase para nombres compuestos
```

Para eliminar un estilo → `div.style.color = "";`

`elem.cssText` → Permite asignar varios estilos en una sola operación.

```
let div = document.getElementById("div");
div.style.cssText = "color: red; background: gray";
```

Estilos computados

`getComputedStyle(elem)` → Devuelve un objeto con los estilos finales de un elemento, combinando las hojas de estilo externas, internas y los estilos en línea.

```
let estilos = getComputedStyle(div);
console.log(estilos.backgroundColor); // Estilo final aplicado al fondo
console.log(estilos.width); // Ancho calculado
```

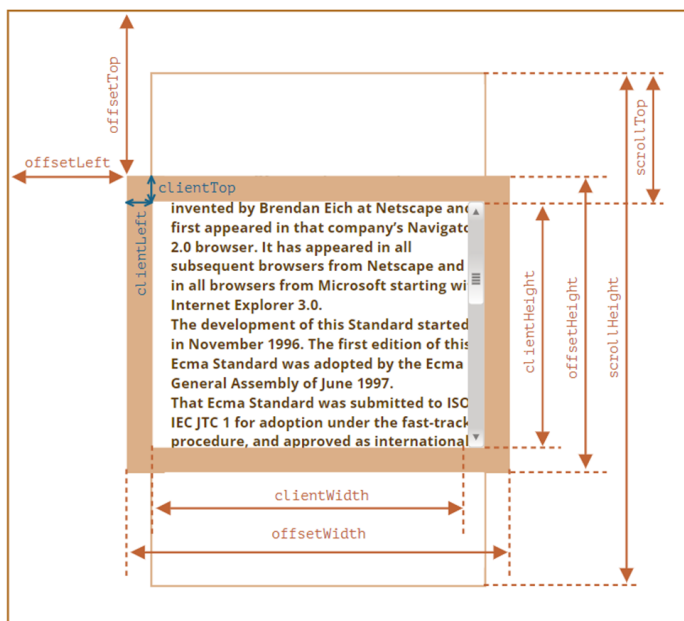
Tamaño de los elementos y scroll

En JavaScript, existen múltiples propiedades para obtener información sobre las dimensiones y características geométricas de un elemento HTML. Estas propiedades son esenciales para mover, posicionar o trabajar con elementos visuales de una página.

| Propiedad | Descripción |
|------------------------------------|--|
| <code>elemento.clientTop</code> | Ancho del borde superior del elemento. |
| <code>elemento.clientLeft</code> | Ancho del borde izquierdo, incluyendo el scroll vertical si está presente. |
| <code>elemento.offsetWidth</code> | Ancho total del elemento, incluyendo padding, bordes y barra de scroll (si existe). |
| <code>elemento.offsetHeight</code> | Altura total del elemento, incluyendo padding, bordes y barra de scroll (si existe). |
| <code>elemento.clientWidth</code> | Ancho del área visible del contenido, incluyendo el padding, pero excluyendo bordes y scroll. |
| <code>elemento.clientHeight</code> | Altura del área visible del contenido, incluyendo el padding, pero excluyendo bordes y scroll. |
| <code>elemento.scrollWidth</code> | Ancho total del contenido, incluyendo el contenido no visible debido al overflow. |
| <code>elemento.scrollHeight</code> | Altura total del contenido, incluyendo el contenido no visible debido al overflow. |
| <code>elemento.scrollLeft</code> | Número de píxeles desplazados horizontalmente hacia la izquierda. Puede leerse o modificarse. |

| Propiedad | Descripción |
|---------------------------------|---|
| <code>elemento.scrollTop</code> | Número de píxeles desplazados verticalmente hacia arriba. Puede leerse o modificarse. |

- `offsetWidth` y `offsetHeight` → Incluyen el ancho/alto del contenido, el `padding`, los bordes y las barras de scroll (si las hay).
- `clientWidth` y `clientHeight` → Miden únicamente el área visible del contenido, incluyendo el `padding`, pero excluyen bordes y barras de scroll.
- `scrollWidth` y `scrollHeight` → Reflejan el tamaño total del contenido, incluyendo cualquier parte que se desborde fuera del área visible.
- `scrollLeft` y `scrollTop` → Indican el desplazamiento horizontal y vertical del contenido respecto a su posición inicial. Son propiedades tanto de lectura como de escritura.



Tamaño de ventana y desplazamiento

JavaScript proporciona propiedades y métodos para obtener información sobre las dimensiones visibles de un documento, el desplazamiento del contenido y para controlar su posición dentro de la ventana.

Tamaño de la parte visible del documento

| Propiedad | Descripción |
|--|------------------------------|
| <code>document.documentElement.clientWidth</code> | Ancho visible del documento. |
| <code>document.documentElement.clientHeight</code> | Alto visible del documento. |

Tamaño completo del documento

El tamaño completo del documento incluye el contenido visible y el no visible debido al overflow.

Para calcularlo, se utiliza el siguiente código para determinar la altura máxima:

```
let scrollHeight = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);
```

Desplazamiento del documento

Desplazamiento actual.

| Propiedad | Descripción |
|---------------------------------|---|
| <code>window.pageXOffset</code> | Número de píxeles desplazados horizontalmente (hacia la izquierda). |
| <code>window.pageYOffset</code> | Número de píxeles desplazados verticalmente (hacia arriba). |

Desplazar el documento respecto a la ventana.

| Método | Descripción |
|--|--|
| <code>window.scrollBy(x, y)</code> | Desplaza la página en relación a su posición actual. Por ejemplo: <code>scrollBy(0, 10)</code> desplaza 10px abajo. |
| <code>window.scrollTo(pageX, pageY)</code> | Desplaza la página a coordenadas absolutas. Ejemplo: <code>scrollTo(0, 0)</code> desplaza al inicio del documento. |
| <code>elem.scrollIntoView(top)</code> | Desplaza la página para hacer visible el elemento <code>elem</code> . Si <code>top = true</code> (valor predeterminado), el borde superior de <code>elem</code> se alinea con la parte superior de la venta. Si <code>top = false</code> , el borde inferior de <code>elem</code> se alinea con la parte inferior de la ventana. |

Control del `scroll`

Deshabilitar el scroll en todo el documento → `document.body.style.overflow = "hidden";`

Restaurar el scroll → `document.body.style.overflow = "";`

Coordenadas

Las coordenadas son esenciales para mover, posicionar o interactuar con elementos en la página.

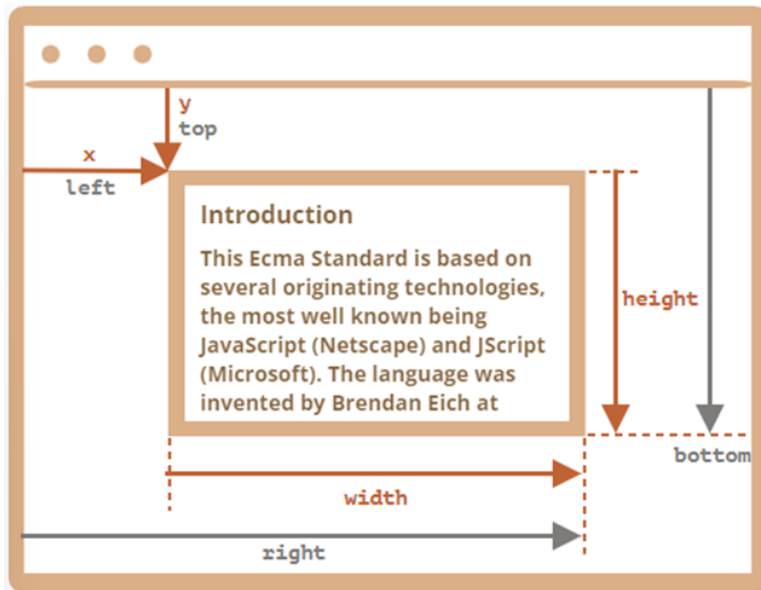
`element.getBoundingClientRect()` → Devuelve un objeto con las coordenadas del elemento respecto a la ventana (viewport).

| Propiedad | Descripción |
|---------------------|---|
| <code>width</code> | Ancho del elemento. |
| <code>height</code> | Alto del elemento. |
| <code>top</code> | Coordenada Y del borde superior del elemento. |

| Propiedad | Descripción |
|---------------------|--|
| <code>bottom</code> | Coordenada Y del borde inferior del elemento. |
| <code>left</code> | Coordenada X del borde izquierdo del elemento. |
| <code>right</code> | Coordenada X del borde derecho del elemento. |

Las coordenadas se expresan en relación con la parte visible de la ventana y pueden incluir valores decimales.

Nota: No es necesario redondear los valores de `top` o `left` para aplicarlos directamente con `style.left` o `style.top`.



`document.elementFromPoint(x, y)` → Permite identificar el elemento que se encuentra en un punto específico de la ventana (coordenadas `x`, `y`).

| Uso | Descripción |
|--|--|
| <code>document.elementFromPoint(x, y)</code> | Devuelve el elemento más profundamente anidado en las coordenadas dadas. |
| Coordenadas fuera de la ventana | Si las coordenadas especificadas están fuera del área visible del navegador (viewport), el método devuelve <code>null</code> . |