TEMA 4: BOM (BROWSER OBJECT MODEL) Y DOM (DOCUMENT OBJECT MODEL)

Módulo: Desarrollo Web en Entorno Cliente
Profesor: Daniel de la Iglesia Rodrigo (daniel.iglrod.1@educa.jcyl.es)
2º DAW

INDICE

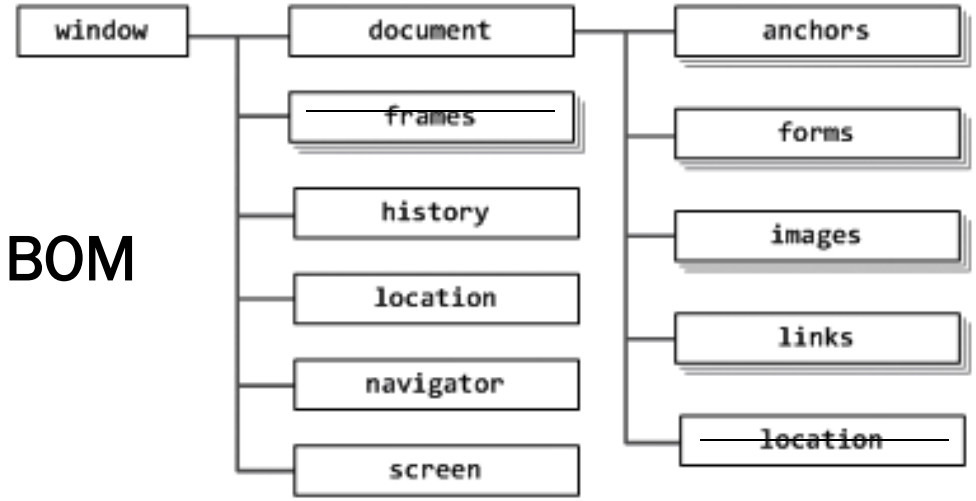
1. Interacción de objetos: JS BOM

BOM (Browser Object Model): Permite acceder y modificar las propiedades de ventanas del propio navegador.

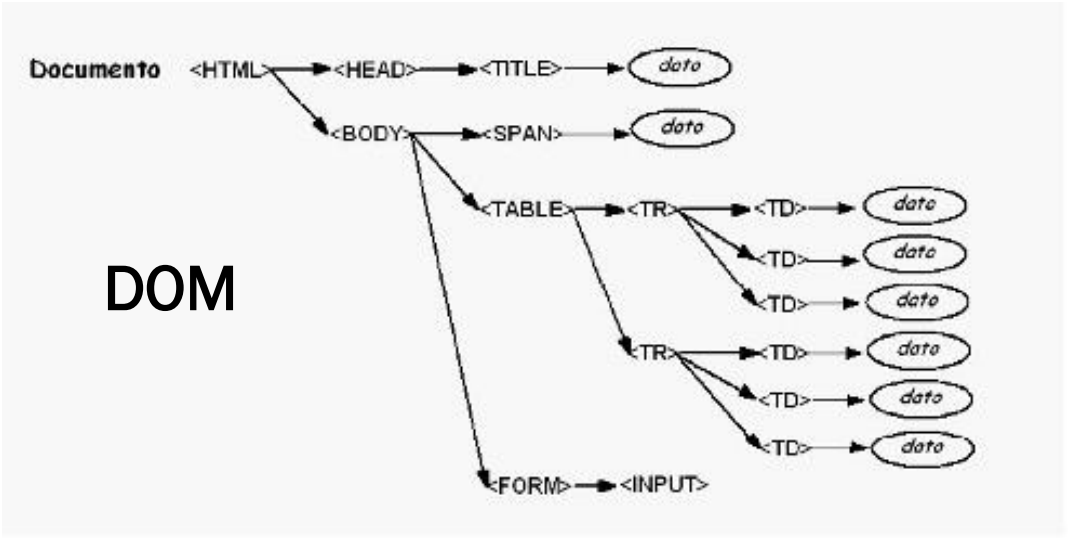
- No está estandarizado ni definidos mínimos, aunque todos los navegadores lo entienden.
- Se centra en acceder a todas las áreas del navegador. Es específico de cada navegador.

BOM Vs DOM:

- BOM se centra en acceder a todas las áreas del navegador. DOM se centra en el documento HTML
- BOM tiene una jerarquía en base al nombre de los objetos (window, document…), DOM se basa en el contenido jerárquico del documento HTML
- BOM no está estandarizado, DOM si. DOM es un subconjunto de BOM



BOM



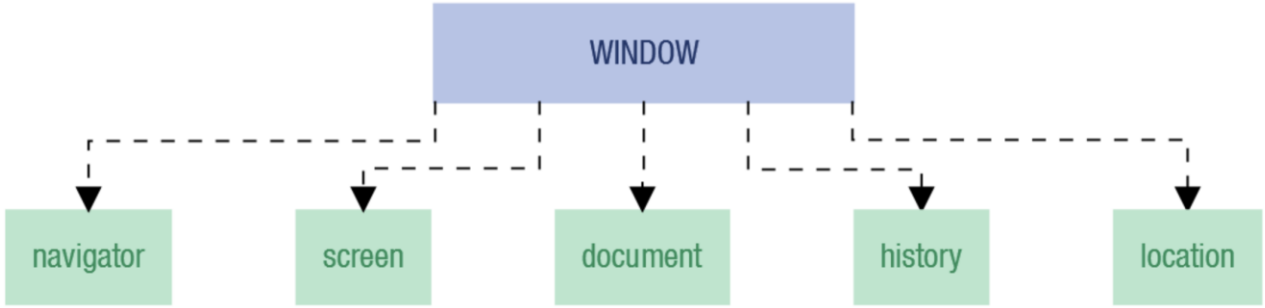
DOM

1.1 JS BOM - Objeto window

- Es el objeto más importante de JavaScript. A partir de él podemos gestionar las ventanas del navegador con sus propiedades y métodos
- Es un objeto implícito, es decir no hace falta nombrarlo, aunque se esté utilizando.
- Abarca, además del contenido propio de la ventana, la barra de estado, desplazamiento, herramientas
- Para JavaScript, ventanas y pestañas son lo mismo. Navegador con varias pestañas = ventanas diferentes
- Se puede hacer referencia a él a través de la palabra `window` o `self` cuando hacemos referencia desde el propio contenido de esa ventana.
- NO existe un estándar público que se aplique al objeto `window`, pero todos los navegadores principales lo admiten.

Gestión de ventanas

Un script nunca abrirá la página principal del navegador, es el usuario quién lo realiza; pero un script que se esté ejecutando en un navegador si podrá abrir sub-ventanas.



1.1 JS BOM - Objeto Window

Métodos "generales"

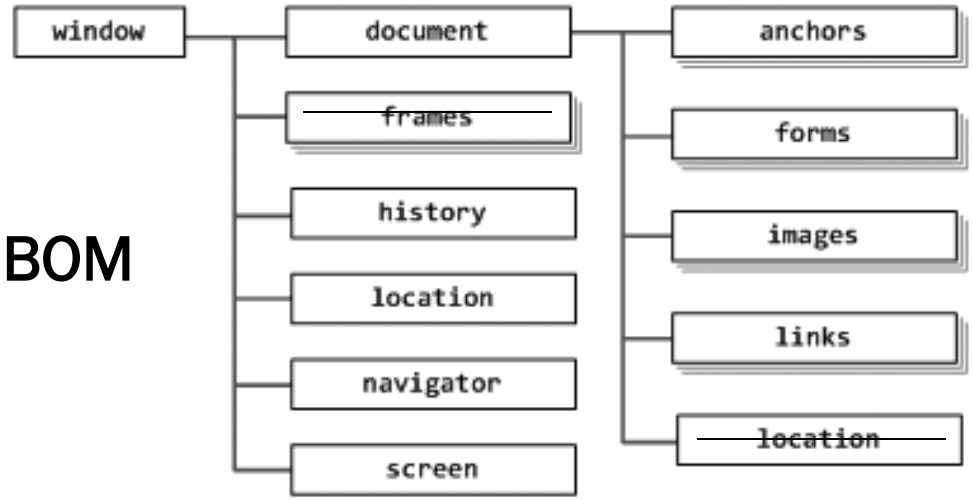
- **alert():** Genera un cuadro de diálogo con un mensaje y botón aceptar
- **prompt():** Genera un cuadro de diálogo con un cuadro de texto para que el usuario introduzca valores
  - Si el usuario introduce algo, devuelve lo introducido (String)
  - Si el usuario no introduce nada, devuelve NULL
- **confirm():** Genera un cuadro de diálogo con los botones "Aceptar" y "Cancelar".
  - Si el usuario pulsa "Aceptar", devuelve true
  - Si el usuario pulsa "Cancelar" o cierra la ventana, devuelve false.
- **open():** Permite crear nuevas ventanas/pestañas (dependiendo de la configuración del navegador).
  Parámetros opcionales
  `window.open(URL, name, specs, replace)`
  - URL: dirección web
  - name: _blank (default), _self o el "name" del objeto ventana.
  - specs: scrollbars, menubar, titlebar toolbar, width, heigth =yes|no|1|0 Separados por comas y sin espacios
- **close():** Cierra la ventana actual
- **focus():** Establece el foco en la ventana independientemente de la configuración del navegador
  ```
  objetoVentana.focus();
  ```
- **blur():** Quita el foco de la ventana. "Contrario" a focus()
  ```
  objetoVentana.blur();
  ```

1.1 JS BOM - Objeto window

Propiedades

- **closed**: Devuelve valor booleano si la ventana está cerrada o no
- **console:** Devuelve información sobre el objeto `console` que a su vez tiene métodos propios
  - `window.console.log ("Entra aquí"); //La famosa línea que utilizamos para depurar`
  - `window.console.clear() //Limpia la consola`
- **name**: Corresponde el nombre de la ventana
- **self**: Devuelve la ventana actual

- **document**: documento actual o ventana
- **history**: Devuelve el objeto `history` que cuelga de window
- **location**: Devuelve el objeto `location` que cuelga de `window`
- **navigator**: Devuelve el objeto `navigator` que cuelga de window
- **screen**: Devuelve el objeto `screen` que cuelga de `window`
  - **document, history, location y navigator** tienen a su vez propiedades y métodos



1.1 JS BOM - Objeto Window

Más Métodos (+ específicos)

- **moveTo()** Mueve la ventana a una posición en concreta que se le pasa por parámetro (width y heigth)

```
objetoVentana.moveTo(600, 50);
```

- **resizeTo():** Cambia el tamaño de la ventana a los valores que se pasan por parámetro (width y height)

```
objetoVentana.moveTo(600, 50);
```

- **scroollTo():** Mueve el scroll de documento a unas coordenadas concretas pasadas por parámetro (horizontal, vertical) e píxeles

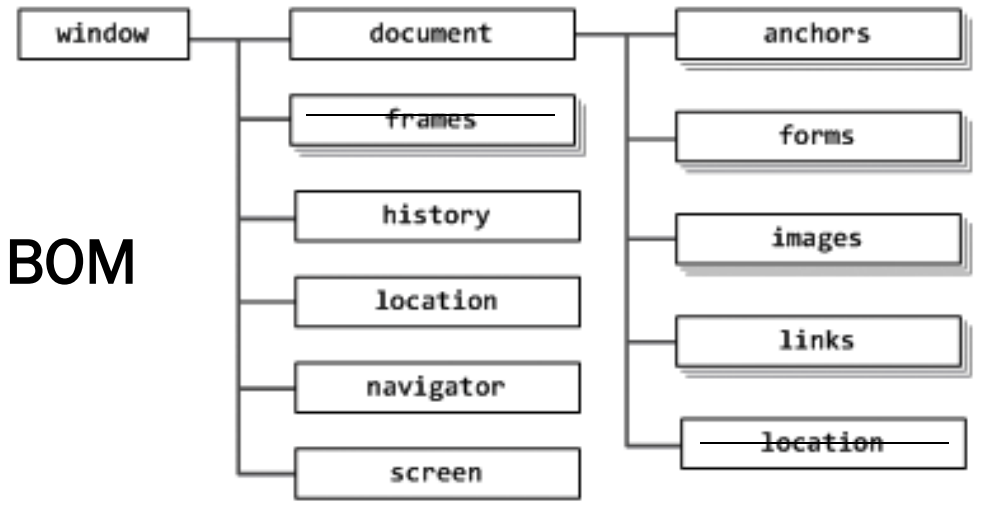
```
obetoVentana.scrollTo(500, 0);
```

- **setInterval():** Llama a una función o evalúa una expresión que especifica un intervalo en milisegundos hasta que el intervalo se "limpie" con clearInterval(). 1000ms = 1s
  - `var temporizador = setInterval(contar(), 2000);`
- **clearInterval():** Para el contador que empieza setInterval(). Se le pasa como parámetro la variable que estableció setInterval() como contador.
  - clearInterval(temporizador);

1.2 JS BOM - Objeto navigator

Propiedades

- **appCodeName**: Devuelve el nombre del navegador. Todos los navegadores modernos devuelven “Mozilla”
- **appName**: Devuelve el nombre del navegador
  - IE11, Firefox, Chrome y Safari devuelven “Netscape”
  - IE10 y anteriores devuelven “Microsoft Internet Explorer”
  - Opera devuelve “Opera”
- **appVersion**: Devuelve la versión del navegador (string)
  - Ejemplo: 5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.111 Safari/537.36
- **cookieEnabled**: Devuelve un valor booleano si las cookies están activadas
- **geolocation**: Devuelve un objeto con la localización (no compatible con todos los navegadores)



Métodos:

- **javaEnabled**(): Devuelve un valor booleano con valor true o false en función de si el navegador tiene Java habilitado o no.

1.3. JS BOM - Objeto screen

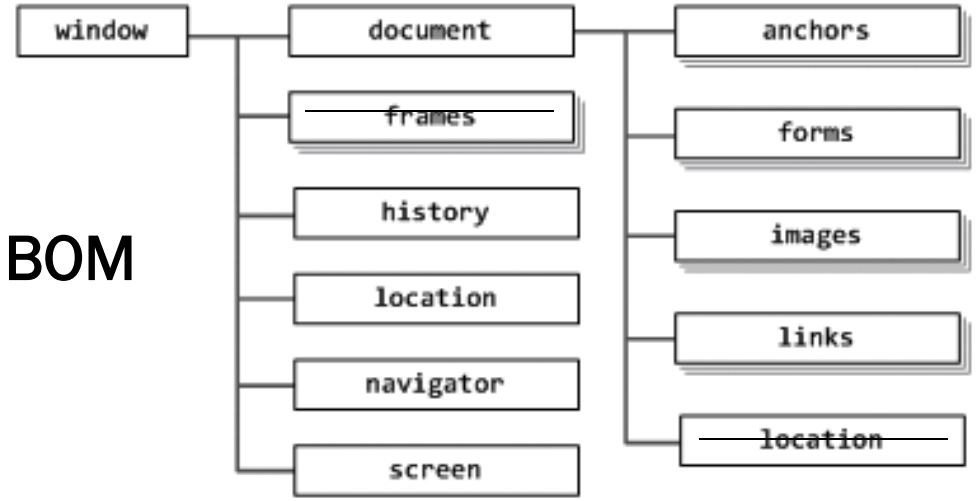
Propiedades

- **availHeigth**: Muestra la altura máxima de la pantalla que se está utilizando en píxeles
  `screen.availHeight`

- **availWidth:** Muestra el ancho máximo de la pantalla que se está utilizando en píxeles (excluyendo la barra de tareas)
  `screen.availWidth`

- **colorDepth:** Muestra los bits por pixel que muestra la pantalla
  `screen.colorDepth`

- **height:** Muestra el alto total de la pantalla.
- **width:** Muestra el ancho total de la pantalla.



1.4. JS BOM - Objeto location

Propiedades

- **hash:** Devuelve la parte ancla de una dirección URL. En una url: http://www.test.com#parte2 (#parte2)

```
location.hash;
```

- **host:** Devuelva el nombre de host y el puerto de la dirección URL actual

```
location.host;
```

- **hostname:** Devuelve el nombre de host de la dirección URL actual

```
location.hostname
```

- **href:** Devuelve la dirección URL completa de la página actual

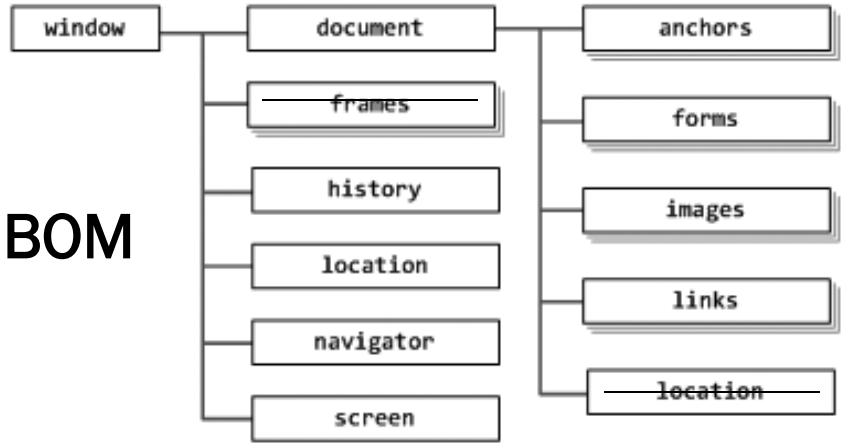
```
location.href;
```

- **orign:** Devuelve el protocolo, el nombre de host y el número de puerto de una dirección URL. Supongamos que la dirección URL actual

```
location.origin;
```

- **pathname**: Devuelve el nombre de la ruta de acceso de la dirección URL actual:

```
location.pathname;
```

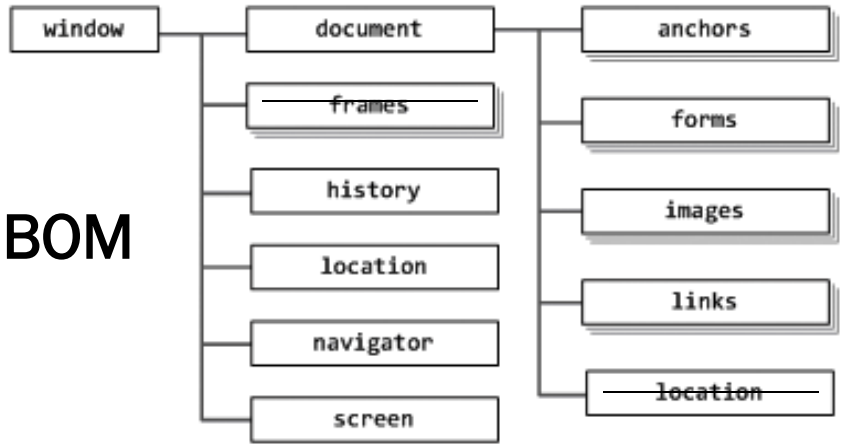


- **port:** Devuelva el número de puerto de la dirección URL actual:
- **protocol**: Devuelve el protocolo de la dirección URL actual
- **search:** Devuelve la parte de cadena de consulta de una dirección URL. Supongamos que la dirección URL actual es https://www.test.com/submit.htm?email-someone@example.com:

1.4. JS BOM - Objeto location

Métodos

- **assign():** Carga un nuevo documento
  ```
  location.assign("https://www.google.es");
  ```
- **reload():** Se utiliza para volver a cargar el documento actual
  ```
  location.reload();
  ```
- **replace():** Remplaza el documento actual por uno nuevo. Elimina historial.
  ```
  location.replace("https://www.google.es");
  ```



1.5. JS BOM - Objeto history

Depende de window, antiguamente se consideraba objeto con entidad propia

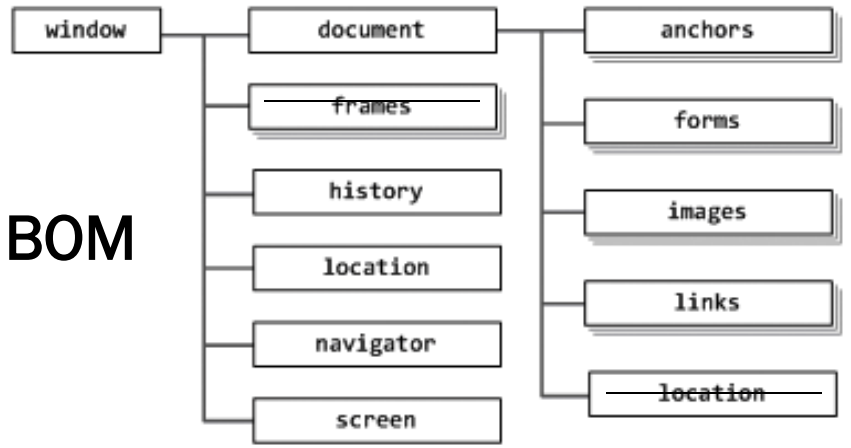
Propiedades

- **length:** Número de páginas que figuran en el historial. Unos navegadores empiezan en 0 y otros en 1 incluyendo la actual

Métodos

- **back():** Vuelve a la página anterior
- **forward():** Carga la siguiente página del navegador (si la hay)
- **go():** Carga una página específica del historial del navegador

```
window.history.go(-2)
```

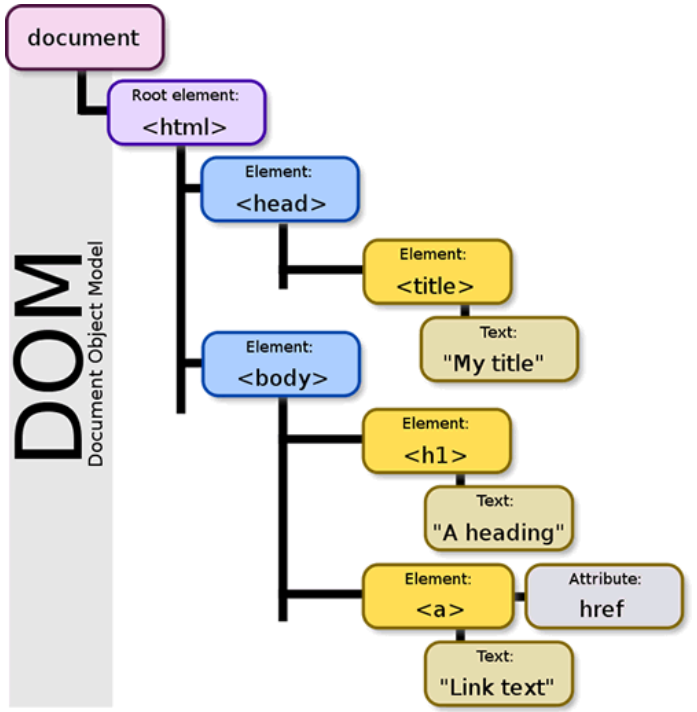


PRÁCTICA 00 – BOM

1.6. JS BOM - Objeto document

DOM: Document Object Model, estructura del documento HTML.

- Una página HTML está formada por múltiples etiquetas HTML, anidadas unas dentro de otras (nodos), formando un árbol de etiquetas relacionadas entre sí => árbol DOM (o simplemente DOM)
- Esta estructura se puede modificar de forma dinámica desde Javascript, añadiendo nuevas etiquetas, modificando o eliminando otras, cambiando atributos HTML, modificando estilos CSS, cambiando contenidos de texto, etc.
- Todo esto se puede automatizar, o ejecutar cuando el usuario realiza una acción determinada, como pulsar un botón, mover el ratón, escribir un texto, etc.



Propiedades

- **title:** Devuelve/Establece el título del documento actual
- **lastModified:** Devuelve la fecha/hora en la que se modificó el documento actual por última vez.
- **domain:** Devuelve el nombre del dominio donde se cargó el documento actual.
- **URL:** Devuelve la url completa del documento
- **images**: devuelve un array con todas las imágenes del documento
  - arrayImages[0].src

1.6. JS BOM - Objeto document

Seleccionar elementos del DOM

Métodos tradicionales

Se denominan así porque existen en Javascript desde las versiones más antiguas

Permiten buscar elementos en la página dependiendo de los atributos id, class, name o la propia etiqueta.

Métodos de búsqueda	Descripción
ELEMENT `.getElementById(id)`	Busca el elemento HTML con el id `id`. Si no, devuelve NULL.
ARRAY `.getElementsByClassName(class)`	Busca elementos con la clase `class`. Si no, devuelve `[]`.
ARRAY `.getElementsByName(name)`	Busca elementos con atributo name `name`. Si no, devuelve `[]`.
ARRAY `.getElementsByTagName(tag)`	Busca elementos `tag`. Si no encuentra ninguno, devuelve `[]`.

getElementById()

Busca el elemento con el id especificado por parámetro. Un documento HTML bien construido no debería tener más de un elemento con el mismo id, por lo que devolvería siempre un solo elemento. (null en caso de no encontrarlo)

```
const pag = document.getElementById("pag");   // <div id="pag"></div>
```

1.6. JS BOM - Objeto document

Seleccionar elementos del DOM

getElementsByClassName(), getElementsByName(), getElementsByTagName()

Buscan el elemento con la clase, nombre o etiqueta especificada por parámetro.
getElements => plural
Puede haber varios elementos con la misma clase, nombre o tag. Siempre devuelve un array (vacio si no encuentra)

```
const elementos = document.getElementsByClassName("elem");  // [div, div, div]

console.log(items[0]);      // Primer elemento encontrado: <div class="elem"></div>
console.log(elementos.length);  // 3

// Obtiene todos los elementos con atributo name="palabras"
const palabras= document.getElementsByName("palabras");

// Obtiene todos los elementos <div> de la página
const divs = document.getElementsByTagName("div");
```

Los métodos devuelven un tipo de dato HTMLCollection o NodeList (similar a array). Pueden no tener algunos métodos como, por ejemplo, .forEach()

1.6. JS BOM - Objeto document

Seleccionar elementos del DOM

Métodos modernos

Se denominan selectores CSS. Son más cómodos y prácticos

Método de búsqueda	Descripción
ELEMENT `.querySelector(sel)`	Busca el primer elemento que coincide con el selector CSS `sel`. Si no, NULL.
ARRAY `.querySelectorAll(sel)`	Busca todos los elementos que coinciden con el selector CSS `sel`. Si no, `[]`.

querySelector(selector)

Devuelve el primer elemento que encuentra atendiendo al selector CSS.

Equivale al getElementById(), devuelve un solo elemento (null si no encuentra)

```
const pag = document.querySelector("#pag");            // <div id="pag"></div>
const info = document.querySelector(".main .info");     // <div class="info"></div>
```

En el segundo ejemplo, recupera el primer elemento con clase **info** que se encuentre dentro de otro elemento con clase **main**.

1.6. JS BOM - Objeto document

Seleccionar elementos del DOM

QuerySelectorAll()

Buscan elementos como querySelector() pero retorna un array con todos los elementos que coincidan con el selector CSS (vacío si no encuentra nada)

```
// Obtiene todos los elementos con clase "info"
const infos = document.querySelectorAll(".info");

// Obtiene todos los elementos con atributo name="sinonimos"
const sinonimos= document.querySelectorAll('[name="sinonimos"]');

// Obtiene todos los elementos <div> de la página HTML
const divs = document.querySelectorAll("div");
```

Al realizar una búsqueda de elementos y guardarlos en una variable, podemos realizar la búsqueda posteriormente sobre esa variable en lugar de hacerla sobre **document**. Esto permite realizar búsquedas acotadas por zonas, en lugar de realizarlo siempre sobre **document**, que buscará en todo el documento HTML.

1.6. JS BOM - Objeto document

Crear elementos en el DOM

Lo normal es programar HTML desde un fichero HTML, pero en la actualidad están en auge las SPA (Single Page Application) y los frameworks Javascript, por lo que es frecuente crear código HTML desde Javascript de forma dinámica.

- Un fichero .html es estático, por lo que siempre es más sencillo y directo
- Un fichero .js es más complejo y menos directo, pero más potente, flexible y con menos limitaciones

Se pueden crear elementos HTML o nodos. Utilizando bucles o estructuras definidas se pueden crear estructuras dinámicas de forma sencilla:

Métodos	Descripción
ELEMENT `.createElement(tag, options)`	Crea y devuelve el elemento HTML definido por el STRING `tag`.
NODE `.createComment(text)`	Crea y devuelve un nodo de comentarios HTML `<!-- text -->`.
NODE `.createTextNode(text)`	Crea y devuelve un nodo HTML con el texto `text`.
NODE `.cloneNode(deep)`	Clona el nodo HTML y devuelve una copia. `deep` es **false** por defecto.
BOOLEAN `.isConnected`	Indica si el nodo HTML está insertado en el documento HTML.

1.6. JS BOM - Objeto document

Crear elementos en el DOM

createElement()

- Crear un HTML **en memoria** (*¡no estará insertado aún en nuestro documento HTML!*). Con dicho elemento almacenado en una variable, podremos modificar sus características o contenido, para **posteriormente** insertarlo en una posición determinada del DOM o documento HTML.

- Pasar la tag del elemento que queremos crear como parámetro:

```
const div = document.createElement("div");      // Se crea un <div></div>
const span = document.createElement("span");    // Se crea un <span></span>
const img = document.createElement("img");      // Se crea un <img>
```

1.6. JS BOM - Objeto document

Crear elementos en el DOM

createComment()

```
const comment = document.createComment("Comentario"); // <!--Comentario-->
```

createTextNode()

```
const text = document.createTextNode("El texto");        // Nodo de texto: 'el texto'
```

cloneNode()

- Cuidado al crear y duplicar elementos HTML. Error: asignar elemento que tenemos en variable porque no crea copia, referencia al original. Si se modifica div2, se modifica tambiń div:

```
const div = document.createElement("div");
div.textContent = "Elemento 1";

const div2 = div;   // NO se está haciendo una copia
div2.textContent = "Elemento 2";

div.textContent;  // 'Elemento 2'
```

1.6. JS BOM - Objeto document

Crear elementos en el DOM

cloneNode()

- Clona un nodo HTML.

- **Parámetro deep**: Indica el tipo de clonación. Es booleano y por defecto es false:
  - True: clonará también los hijos del nodo (clonación profunda – deep clone)
  - False: no clonará los hijos del nodo (clonación superficial – shallow clone)

```
const div = document.createElement("div");
div.textContent = "Elemento 1";

const div2 = div.cloneNode();   // Ahora SÍ clona
div2.textContent = "Elemento 2";

div.textContent;  // 'Elemento 1'
```

Propiedad isConnected

Indica si el nodo está conectado al DOM, es decir, si está insertado en el documento HTML.

- True: insertado en DOM
- False: no insertado en DOM

1.6. JS BOM - Objeto document

Atributos HTML de un elemento

- Hemos creado elementos, pero no hemos visto como modificar los atributos HTML de los mismos.
- Lo más sencillo: asignarle valores como propiedades de objetos:

```
const div = document.createElement("div");          // <div></div>

div.id = "page";          // <div id="page"></div>

div.className = "data";          // <div id="page" class="data"></div>

div.style = "color: red";          // <div id="page" class="data" style="color:
red"></div>
```

- En algunos casos se complica, como en class que es una palabra reservada. No se puede utilizar para crear atributos.
- Para crear una clase a un elemento, tenemos que utilizar la propiedad className.

1.6. JS BOM - Objeto document

Atributos HTML de un elemento

- Métodos para utilizar en un elemento HTML y añadir, modificar o eliminar sus atributos:

Métodos	Descripción
BOOLEAN `hasAttributes()`	Indica si el elemento tiene atributos HTML.
BOOLEAN `hasAttribute(attr)`	Indica si el elemento tiene el atributo HTML `attr`.
ARRAY `getAttributeNames()`	Devuelve un ARRAY con los atributos del elemento.
STRING `getAttribute(attr)`	Devuelve el valor del atributo `attr` del elemento o NULL si no existe.
UNDEFINED `removeAttribute(attr)`	Elimina el atributo `attr` del elemento.
UNDEFINED `setAttribute(attr, value)`	Añade o cambia el atributo `attr` al valor `value`.
NODE `getAttributeNode(attr)`	Idem a `getAttribute()` pero devuelve el atributo como **nodo**.
NODE `removeAttributeNode(attr)`	Idem a `removeAttribute()` pero devuelve el atributo como **nodo**.
NODE `setAttributeNode(attr, value)`	Idem a `setAttribute()` pero devuelve el atributo como **nodo**.

1.6. JS BOM - Objeto document

Atributos HTML de un elemento

```
// Obtenemos <div id="page" class="info data dark" data-number="5"></div>
const div = document.querySelector("#page");

div.hasAttribute("data-number");  // true (data-number existe)
div.hasAttributes();              // true (tiene 3 atributos)

div.getAttributeNames();          // ["id", "data-number", "class"]
div.getAttribute("id");           // "page"

div.removeAttribute("id");        // class="info data dark" y data-number="5"
div.setAttribute("id", "page");   // Vuelve a añadir id="page"
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

- Hemos creado elementos, pero se guardaban en memoria, almacenados en una variable, pero no se insertaban en el dom.

REMPLAZAR CONTENIDO:

Propiedades	Descripción
STRING `.textContent`	Devuelve el contenido de texto del elemento. Se puede asignar para modificar.
STRING `.innerHTML`	Devuelve el contenido HTML del elemento. Se puede usar asignar para modificar.
STRING `.outerHTML`	Idem a `.innerHTML` pero incluyendo el HTML del propio elemento HTML.

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.textContent:

- Devuelve el contenido de texto de un elemento HTML.
- Nos permite reemplazar. Si el elemento está vacío inserta, si tiene contenido y reemplazamos, borra el anterior.

```
const div = document.querySelector("div");        // <div></div>

div.textContent = "Hola a todos";       // <div>Hola a todos</div>
div.textContent;  // "Hola a todos"


// Obtenemos <div class="info">Hola <strong>amigos</strong></div>
const div = document.querySelector(".info");

div.textContent;  // "Hola amigos"
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.innerHTML:

- Devuelve el contenido de un elemento HTML pero **interpreta el HTML**
- Nos permite reemplazar. Si el elemento está vacío inserta, si tiene contenido y reemplazamos, borra el anterior.

```
const div = document.querySelector(".info");          // <div class="info"></div>

div.innerHTML = "<strong>Importante</strong>"; // Interpreta el HTML

div.innerHTML;     // "<strong>Importante</strong>"

div.textContent;   // "Importante"

div.textContent = "<strong>Importante</strong>";  // No interpreta el HTML
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.outerHTML:

- Devuelve el contenido de un elemento HTML y el propio elemento. **Interpreta el HTML**

```
const data = document.querySelector(".data");    //  "<div class="data"><h1>Tema 1</h1></div>"
data.innerHTML = "<h1>Tema 1</h1>";

data.textContent;   // "Tema 1"
data.innerHTML;     // "<h1>Tema 1</h1>"
data.outerHTML;     // "<div class="data"><h1>Tema 1</h1></div>"
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

INSERTAR ELEMENTOS:

Métodos	Descripción
NODE `.appendChild(node)`	Añade como hijo el nodo **node**. Devuelve el nodo insertado.
ELEMENT `.insertAdjacentElement(pos, elem)`	Inserta el elemento **elem** en la posición **pos**. Si falla, NULL.
UNDEFINED `.insertAdjacentHTML(pos, str)`	Inserta el código HTML **str** en la posición **pos**.
UNDEFINED `.insertAdjacentText(pos, text)`	Inserta el texto **text** en la posición **pos**.
NODE `.insertBefore(new, node)`	Inserta el nodo **new** antes de **node** y como hijo del nodo actual.

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.appendChild()

- Inserta un elemento como hijo al final de todos los elementos hijos que existan.
- No siempre queremos insertar el elemento al final

```
const img = document.createElement("img");
img.src = "https://test.com/images/logo.svg";
img.alt = "Logo";

document.body.appendChild(img);   // añade img al final del body
```

```
const div = document.createElement("div");
div.textContent = "Esto es un div insertado con JS.";

const app = document.createElement("div"); // <div></div>
app.id = "app"; // <div id="app"></div>
app.appendChild(div); // <div id="app"><div>Esto es un div insertado con JS</div></div>
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.insertAdjacent()
Son varios métodos. Son más versátiles que .appendChild().

.insertAdjacentElement(): insertamos un objeto element
.insertAdjacentHTML(): insertamos **código HTML** directamente (*similar a innerHTML*)
.insertAdjacentText(): insertamos un nodo como texto

Para todos tenemos el parámetro pos (string) que indica la posición donde vamos a insertar el contenido.
Posiciones:

- **beforebegin**: El elemento se inserta **antes** de la etiqueta HTML de apertura.
- **afterbegin**: El elemento se inserta **dentro** de la etiqueta HTML, **antes de su primer hijo**.
- **beforeend**: El elemento se inserta **dentro** de la etiqueta HTML, **después de su último hijo**. Es el equivalente a usar el método **.appendChild()**.
- **afterend**: El elemento se inserta **después** de la etiqueta HTML de cierre.

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.insertAdjacentElement(), .insertAdjacentHTML(), .insertAdjacentText()

```
const div = document.createElement("div");  // <div></div>
div.textContent = "Ejemplo";                // <div>Ejemplo</div>

const app = document.querySelector("#app"); // <div id="app">App</div>

app.insertAdjacentElement("beforebegin", div);
// Opción 1: <div>Ejemplo</div> <div id="app">App</div>

app.insertAdjacentElement("afterbegin", div);
// Opción 2: <div id="app"> <div>Ejemplo</div> App</div>

app.insertAdjacentElement("beforeend", div);
// Opción 3: <div id="app">App <div>Ejemplo</div> </div>

app.insertAdjacentElement("afterend", div);
// Opción 4: <div id="app">App</div> <div>Ejemplo</div>
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.insertAdjacentElement(), .insertAdjacentHTML(), .insertAdjacentText()

```
app.insertAdjacentElement("beforebegin", div);
// Opción 1: <div>Ejemplo</div> <div id="app">App</div>

app.insertAdjacentHTML("beforebegin", '<p>Hola</p>');
// Opción 2: <p>Hola</p> <div id="app">App</div>

app.insertAdjacentText("beforebegin", "Hola a todos");
// Opción 3: Hola a todos <div id="app">App</div>
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

ELIMINAR ELEMENTOS:

- No se borra, se desconecta del documento HTML.

Métodos	Descripción
UNDEFINED `.remove()`	Elimina el propio nodo de su elemento padre.
NODE `.removeChild(node)`	Elimina y devuelve el nodo hijo `node`.
NODE `.replaceChild(new, old)`	Reemplaza el nodo hijo `old` por `new`. Devuelve `old`.

.remove()

Se encarga de desconectarse del DOM a si mismo

```
const div = document.querySelector(".borrar");

div.isConnected;    // true
div.remove();
div.isConnected;    // false
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

.removeChild()

Se encarga de desconectar el nodo o elemento HTML proporcionado

Permite eliminar un nodo hijo de un elemento. En este caso, el parámetro node, es el nodo hijo a eliminar.

```
const div = document.querySelector(".item:nth-child(2)");   // <div class="item">2</div>

document.body.removeChild(div); // Desconecta el segundo .item
```

.replaceChild(new, old)

Cambia un nodo hijo old por uno nuevo nodo hijo new.

Devuelve el nodo reemplazado.

```
const div = document.querySelector(".item:nth-child(2)");

const newnode = document.createElement("div");
newnode.textContent = "DOS";

document.body.replaceChild(newnode, div);
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

Crea un documento HTML base que contenga las etiquetas html, head y body.

Añade todos los elementos necesarios desde Javascript, para poder visualizar un web como la de la siguiente imagen.

```
<body>
    <header>
        <h1>Cabecera</h1>
    </header>
    <section>
        <article>
            <h2>Título de sección</h2>
            <p>Lorem ipsum</p>
        </article>
    </section>
    <aside>
        <h3>
            Título de aside
        </h3>
        <ul>
            <li>
                <a href="#">item uno</a>
            </li>
            <li>
                <a href="#">item dos</a>
            </li>
        </ul>
    </aside>
    <footer>
        <h3>Pie de página</h3>
    </footer>
</body>
```

1.6. JS BOM - Objeto document

Insertar elementos en el DOM

Añade con javascript los siguientes elementos atendiendo a los números de la imagen:

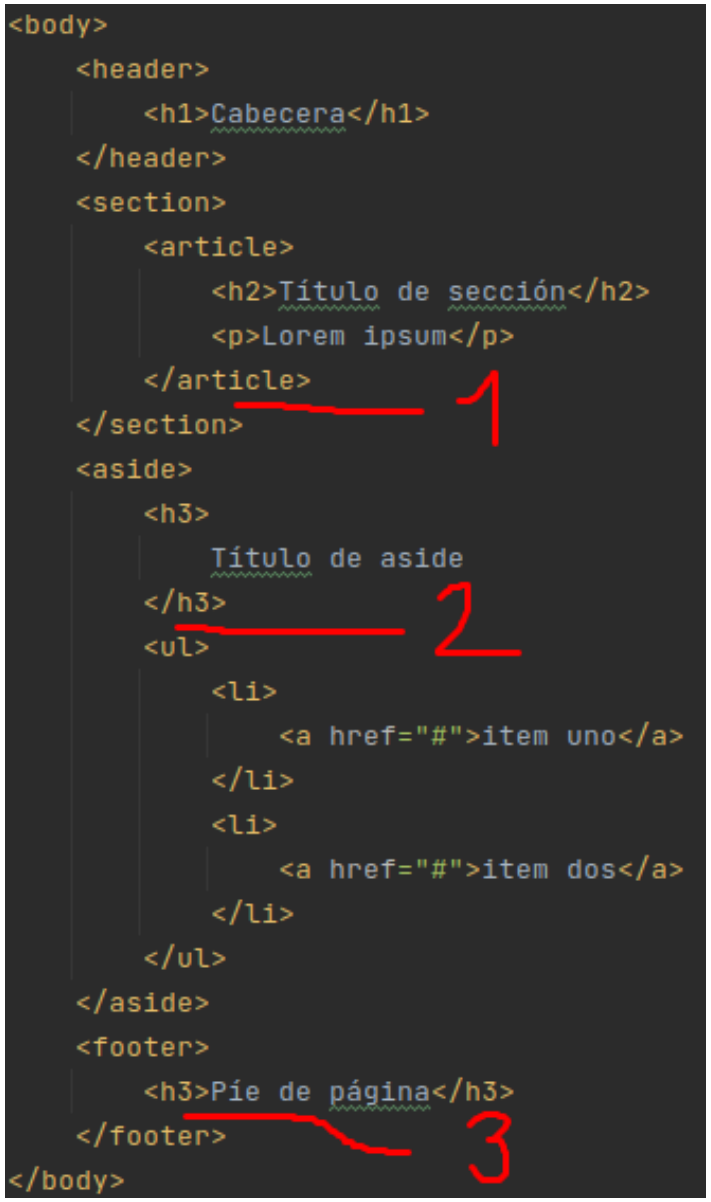
1. Inserta un elemento <article> que contenga dentro un <h2> y dos <p>

2. Añade una imagen

3. Añade un elemento, el que quieras.

Elimina el título de la cabecera y añade uno nuevo diferente al inicial

Reemplaza el título creado en el apartado anterior por un nuevo nodo <h2>Definitivo</h2>




```
<body>
    <header>
        <h1>Cabecera</h1>
    </header>
    <section>
        <article>
            <h2>Título de sección</h2>
            <p>Lorem ipsum</p>
        </article>
    </section>
    <aside>
        <h3>
            Título de aside
        </h3>
        <ul>
            <li>
                <a href="#">item uno</a>
            </li>
            <li>
                <a href="#">item dos</a>
            </li>
        </ul>
    </aside>
    <footer>
        <h3>Pie de página</h3>
    </footer>
</body>
```

1.6. JS BOM - Objeto document

Manipular clases CSS (classList)

className

- Propiedad disponible en todos los elementos HTML
- Contiene el valor del atributo HTML class. Se puede leer y reemplazar.

Propiedad	Descripción
`.className`	Acceso directo al valor del atributo HTML `class`. También se puede asignar.
`.classList`	Objeto especial para manejar clases CSS. Contiene métodos y propiedades de ayuda.

1.6. JS BOM - Objeto document

Manipular clases CSS (classList)

.className es como un getter [.getAttribute("class")] y setter [.setAttribute("class", valor)]

```
//div class="container dark"></div>
const div = document.querySelector(".container");
```

```
// Obtener clases CSS
div.className;            // "container dark"
div.getAttribute("class");  // "container dark"
```

```
// Modificar clases CSS
div.className = "contenedor oscuro";
div.setAttribute("class", "contenedor oscuro");
```

Inconveniente: cuando tenemos múltiples clases CSS en un elemento, y queremos manipular solo una de ellas sin modificar las demás.

1.6. JS BOM - Objeto document

Manipular clases CSS (classList)

classList

- Objeto especial que contiene una serie de ayudantes para trabajar con las clases de forma sencilla.
- Nos devuelve un array (lista) de clases CSS del elemento HTML. (no es un array como tal, puede carecer de algunos métodos o propiedades. Se puede convertir a array con Array.from())

Método	Descripción
ARRAY `.classList`	Devuelve la lista de clases del elemento HTML.
STRING `.classList.item(n)`	Devuelve la clase número `n` del elemento HTML.
UNDEFINED `.classList.add(c1, c2, ...)`	Añade las clases `c1`, `c2`... al elemento HTML.
UNDEFINED `.classList.remove(c1, c2, ...)`	Elimina las clases `c1`, `c2`... del elemento HTML.
BOOLEAN `.classList.contains(clase)`	Indica si la `clase` existe en el elemento HTML.
BOOLEAN `.classList.toggle(clase)`	Si la `clase` no existe, la añade. Si no, la elimina.
BOOLEAN `.classList.toggle(clase, expr)`	Si `expr` es `true`, añade `clase`. Si no, la elimina.
BOOLEAN `.classList.replace(old, new)`	Reemplaza la clase `old` por la clase `new`.

1.6. JS BOM - Objeto document

Manipular clases CSS (classList)

```
<div id="page" class="info data dark" data-number="5"></div>
```

Añadir y eliminar clases CSS

- Métodos classList.add() y classList.remove(): añadir o eliminar una o múltiples clases CSS

```
const div = document.querySelector("#page");

div.classList;       // ["info", "data", "dark"]

div.classList.add("uno", "dos");        // No devuelve nada.
div.classList;      // ["info", "data", "dark", "uno", "dos"]

div.classList.remove("uno", "dos");      // No devuelve nada.
div.classList;     // ["info", "data", "dark"]
```

1.6. JS BOM - Objeto document

Manipular clases CSS (classList)

```
<div id="page" class="info data dark" data-number="5"></div>
```

Conmutar o alternan clases CSS

- Métodos classList.toggle(): añade la clase CSS si no existía antes, la elimina si ya existía.
- Solo permite indicar una única clase CSS
- Devuelve un boolean: true si la añade y false si la elimina

```
const div = document.querySelector("#page");

div.classList;        // ["info", "data", "dark"]

div.classList.toggle("info");        // Como "info" existe, lo elimina. Devuelve "false"
div.classList;        // ["data", "dark"]

div.classList.toggle("info");        // Como "info" no existe, lo añade. Devuelve "true"
div.classList;      // ["info", "data", "dark"]
```

1.6. JS BOM - Objeto document

Manipular clases CSS (classList)

```
<div id="page" class="info data dark" data-number="5"></div>
```

Otros métodos

- **classList.item(n)**: devuelve la clase CSS ubicada en la posición n (comienza en 0)
- **classList.contains(name)**: devuelve si la clase CSS name existe o no (true o false)
- **classList.replace(old,new)**: sustituye la clase CSS old por la clase new (true o false)

```
const div = document.querySelector("#page");

div.classList;       // ["info", "data", "dark"]

div.classList.item(1);      // 'data'

div.classList.contains("info");       // Devuelve `true` (existe la clase)

div.classList.replace("dark", "light");       // Devuelve `true` (se hizo el cambio)
```

PRÁCTICA 03 – Modificar estilos DOM

1.6. JS BOM - Objeto document

Navegar a través de elementos

- Podemos navegar por la jerarquía de elementos HTML relacionados

Propiedades de elementos HTML	Descripción
ARRAY children	Devuelve una lista de elementos HTML hijos.
ELEMENT parentElement	Devuelve el padre del elemento o NULL si no tiene.
ELEMENT firstElementChild	Devuelve el primer elemento hijo.
ELEMENT lastElementChild	Devuelve el último elemento hijo.
ELEMENT previousElementSibling	Devuelve el elemento hermano anterior o NULL si no tiene.
ELEMENT nextElementSibling	Devuelve el elemento hermano siguiente o NULL si no tiene.

1.6. JS BOM - Objeto document

Navegar a través de elementos

```
<html>
 <body>
  <div id="app">
   <div class="header">
    <h1>Titular</h1>
   </div>
   <p>Párrafo de descripción</p>
   <a href="/">Enlace</a>
  </div>
 </body>
</html>
```

```
document.body.children.length;     // 1
document.body.children;        // <div id="app">
document.body.parentElement;    // <html>

const app = document.querySelector("#app");

app.children;                  // [div.header, p, a]
app.firstElementChild;         // <div class="header">
app.lastElementChild;          // <a href="/">

const a = app.querySelector("a");

a.previousElementSibling;      // <p>
a.nextElementSibling;          // null
```

1.6. JS BOM - Objeto document

Navegar a través de nodos

- Trabajar más al detalle a nivel de nodo.
- Interesante para trabajar con nodos de texto, ya que también influyen los espacios en blanco entre los elementos HTML

Propiedades de nodos HTML	Descripción
ARRAY childNodes	Devuelve una lista de nodos hijos. Incluye nodos de texto y comentarios.
NODE parentNode	Devuelve el nodo padre del nodo o NULL si no tiene.
NODE firstChild	Devuelve el primer nodo hijo.
NODE lastChild	Devuelve el último nodo hijo.
NODE previousSibling	Devuelve el nodo hermano anterior o NULL si no tiene.
NODE nextSibling	Devuelve el nodo hermano siguiente o NULL si no tiene.

1.6. JS BOM - Objeto document

Navegar a través de nodos

```
<html>
 <body>
  <div id="app">
   <div class="header">
    <h1>Titular</h1>
   </div>
   <p>Párrafo de descripción</p>
   <a href="/">Enlace</a>
  </div>
 </body>
</html>
```

```
document.body.childNodes.length;     // 3
document.body.childNodes;        // [text, div#app, text]
document.body.parentNode;        // <html>

const app = document.querySelector("#app");

app.childNodes;         // [text, div.header, text, p, text, a, text]
app.firstChild.textContent;      // "                    "
app.lastChild.textContent;       // "                    "

const a = app.querySelector("a");

a.previousSibling;              // #text
a.nextSibling;                 // #text
```

Trabajo en clase: Utilizando HTML de las prácticas anteriores, navega por los elementos y nodos. Añade lo que necesites a dichos HTML para poder probar todo lo visto en clase.