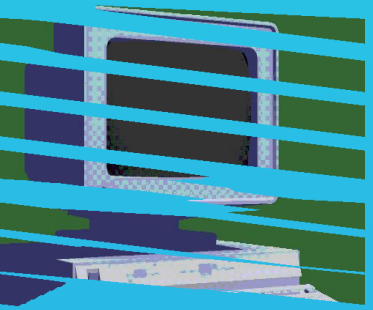
Desarrollo Web en Entorno Cliente

Unidad 4: DOM y BOM



2º DAW

Unidad 4: DOM y BOM

Objetivos

- Validar formularios en una interfaz de HTML.
- Conocer las propiedades del DOM.
- Conocer los objetos de BOM.

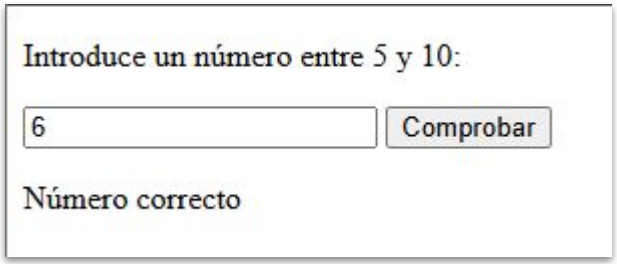
Validar formularios en una interfaz de HTML

Hay dos formas de validar formularios:

- Utilizando el método **document.getElementById** .
- Haciendo uso de **try** y recoger el error en **catch**.

Se va a implementar el siguiente ejercicio:

Validar un campo de tipo texto sabiendo que sólo puede contener un número entre 5 y 10 de ambas formas.



Unidad 4: DOM y BOM

Validar formularios en una interfaz de HTML

Validación utilizando el método **document.getElementById**:

```
<!DOCTYPE html>
<html>
<body>
<p>Introduce un número entre 5 y 10:</p>
<input type="text" id="aqui">
<button onclick="validar()">Comprobar</button>
<p id="msg"></p>
<script>
<script>
function validar() {
    const valor = document.getElementById("aqui").value;
    const msg = document.getElementById("msg");
    if (valor === "") {
        msg.textContent = "Está vacío";
    } else if (isNaN(valor)) {
        msg.textContent = "No es un número";
    } else if (Number(valor) < 5) {
        msg.textContent = "Es menor que 5";
    } else if (Number(valor) > 10) {
        msg.textContent = "Es mayor que 10";
    } else {
        msg.textContent = "Número correcto";
    }
}
</script>
</body>
</html>
```

Validar formularios en una interfaz de HTML

Validación haciendo uso de **try** y recoger el error en **catch**:

```
<!DOCTYPE html>
<html>
<body>
<p>Introduzca un número entre  5 y 10:</p>
<input type="text" id="aqui">
<button type="button"
onclick="validar()">comprueba</button>
<p id="msg"></p>
<script>
function validar(){
    var msg, x;
    msg = document.getElementById("msg");
    msg.innerHTML = "";
    x = document.getElementById("aqui").value;
    try{
        if(x == "") throw "Está vacío";
        if(isNaN(x)) throw "No es un número";
        x = Number(x);
        if(x < 5) throw "Es menor que 5";
        if(x > 10) throw "Se pasa, es mayor que
10";
    } catch(err){
        msg.innerHTML = "Resultado: " + err;
    }
}
</script>
</body>
</html>
```

Unidad 4: DOM y BOM

Validar formularios en una interfaz de HTML

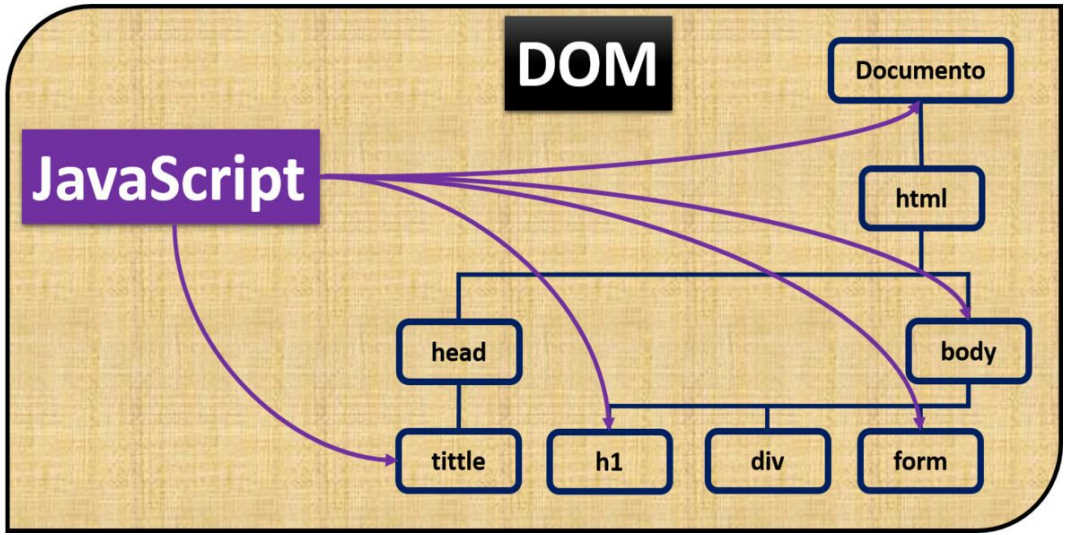
¿Cuál es la mejor forma de validar formularios?

- Usar **condicionales** (if / else).
- Acceder a los campos con **document.getElementById**.
- Comprobar valores vacíos, tipo y rango.
- Mostrar mensajes claros al usuario.
- **No usar try...catch** para validación, **excepto** si hay que capturar alguna excepción por ejemplo cuando se llama a una API externa, cuando se procesan archivos pesados, cuando se usan funciones que podrían fallar por razones ajenas a tu lógica (como un JSON mal formado), etc.

Unidad 4: DOM y BOM

DOM

El DOM (Document Object Model) es una plataforma e interfaz de un documento que permite a los scripts acceder y modificar su contenido, estructura y estilo.



Unidad 4: DOM y BOM

DOM

JavaScript puede acceder al DOM y hacer que una web sea dinámica cambiando el contenido HTML de cualquier elemento, por ejemplo: cambiar el estilo CSS, reaccionar a los eventos, añadir o eliminar elementos HTML, etc.

El objeto **document** se crea cuando se carga un documento en un navegador y además de ser la raíz de todo el documento HTML, tiene propiedades y métodos. Algunos son:

- **Propiedades:** activeElement, cookie, domain, images, links y referrer.
- **Métodos:** open(), querySelector(), getElementById() y createEvent().

Unidad 4: DOM y BOM

DOM: getElementById()

getElementById() permite obtener el identificador (ID) de un elemento HTML, en este caso para obtener "miTitulo" y "miBoton" y así poder añadirle un evento al botón y modificar el título:

```
<body>
    <h1 id="miTitulo">¡Hola, Mundo!</h1>
    <button id="miBoton">Resaltar Título</button>
    <script>
        //Seleccionar el elemento con el id "miTitulo"
        const titulo = document.getElementById("miTitulo");
        //Seleccionar el botón con el id "miBoton"
        const boton = document.getElementById("miBoton");
        // Añadir un evento de clic al botón
        boton.addEventListener('click', function () {
            //Cambiar el estilo
            titulo.classList.toggle('resaltado');
        });
    </script>
</body>
```

Unidad 4: DOM y BOM

DOM: Acceso al DOM document.querySelector()

El método **querySelector()** de document **devuelve** el **primer elemento** que encuentre y que concuerde con **el selector CSS** que se le introduzca cómo **parámetro**. Puede seleccionar: **“#id”, “.clase”** y **“etiqueta”.** Un ejemplo:

```
<body>
    <p>Primer párrafo</p>
    <p class="destacado">Segundo párrafo</p>
    <p>Tercer párrafo</p>
    <button id="boton">Cambiar texto</button>
    <script>
        // Selecciona el primer elemento con la clase "destacado"
        const parrafo = document.querySelector(".destacado");
        // Selecciona el botón por su id
        const boton = document.querySelector("#boton");
        // Al hacer clic se añade el texto en el párrafo
        boton.addEventListener("click", function () {
            parrafo.textContent = "Texto cambiado con querySelector" ;
        });
    </script>
</body>
```

Unidad 4: DOM y BOM

Nodos del DOM

Cuando el navegador recibe una página, la interpreta y va creando una estructura arborescente con los elementos recibidos llamada DOM.

En el DOM existen muchos tipos de nodos como Attr, Comment, Document, DocumentType, Element, Entity, Notation o Text. Los programadores web utilizan sobre todo cuatro nodos cuando desean manipular el DOM:

- **Attr.** Representa los atributos de las etiquetas, que tienen el formato atributo=valor.
- **Document.** El nodo más importante que constituye la raíz de la que derivan los demás nodos del DOM.
- **Element.** Cada etiqueta tiene un nodo tipo element. Estos nodos pueden tener atributos y otros nodos pueden derivar de ellos.
- **Text.** Almacena el texto de una etiqueta.

Unidad 4: DOM y BOM

¿Cómo se accede a los nodos?

En JavaScript se puede acceder a los nodos del DOM utilizando las funciones que este proporciona. Las operaciones más comunes son:

- Acceder al valor de un atributo.
- Establecer el valor de un atributo.
- Crear o añadir nuevos nodos.
- Borrar nodos.
- Mover nodos de su lugar en el árbol.

Unidad 4: DOM y BOM

¿Cómo se accede a los nodos?

Lo normal es acceder a los nodos de forma directa mediante alguna de las siguientes funciones:

- **getElementById().** Es el más utilizado. Se accede por el ID que le ha dado el programador al elemento y este ID debería ser el único en la página.
- **getElementsByTagName().** Accede a los nodos que tengan el tipo de etiqueta que se pasa por parámetro y devuelve un array de elementos.
- **getElementsByClassName().** Se accede al elemento por el nombre de la clase (CSS) o selector CSS al que pertenece y devuelve un array de elementos.
- **querySelector().** Permite acceder al primer elemento que concuerde con el selector que se pasa como parámetro.

Unidad 4: DOM y BOM

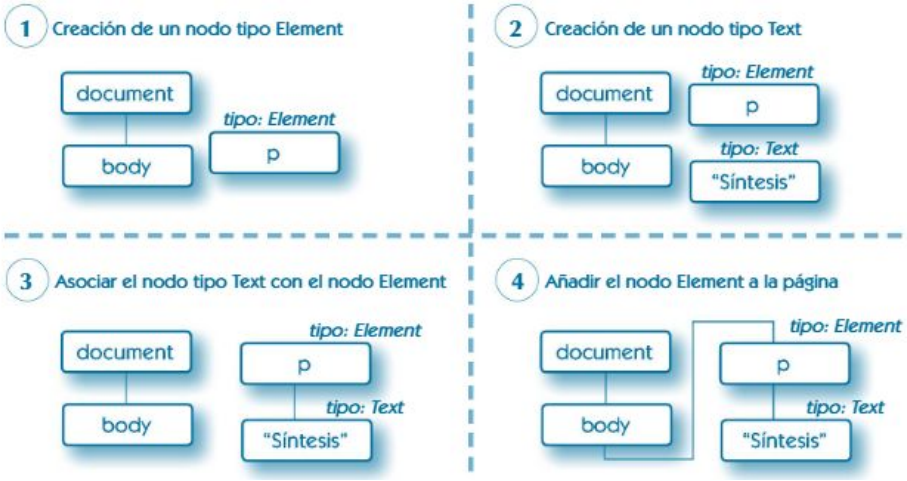
¿Cómo se accede a los nodos?

Ejemplo del método getElementsByClassName():

```
<body>
    <p class="parrafo"> Primer párrafo</p>
    <p class="parrafo"> Segundo párrafo</p>
    <button onclick="pulsar()">Cambiar texto</button>
    <script>
        function pulsar(){
            var p = document.getElementsByClassName("parrafo");
            //Se accede a ambos párrafos y se modifica el texto
            p[0].innerHTML = "Párrafo primero";
            p[1].innerHTML = "Párrafo segundo";
        }
    </script>
</body>
```

Unidad 4: DOM y BOM

¿Cómo crear un nuevo nodo?




```
// Hecho con código sería:
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Crear nodo de tipo Text
var contenido =
document.createTextNode("Síntesis");
// Vincular el nodo Text como hijo del nodo
Element
parrafo.appendChild(contenido);
//Vincular el nodo Element como hijo de la página
document.body.appendChild(parrafo);
```

¿Cómo se elimina un nodo?

Para eliminar un nodo hay que invocar desde el nodo padre a la función **removeChild()**, la cual requerirá como parámetro el nodo que se va a borrar.

Para acceder al nodo padre de un elemento HTML, se puede usar la propiedad **parentNode.**

Ejemplo:

```
//Dado el siguiente párrafo
<p id="parrafo"> Primer párrafo</p>
<script>
    //Se obtiene el ID del párrafo
    var parrafo = document.getElementById("parrafo");
    //se elimina el nodo desde el nodo padre
    parrafo.parentNode.removeChild("parrafo");
</script>
```

Cuando se elimina un nodo, se eliminan todos los nodos hijos que pueda poseer.

DOM: Atributos y Propiedades

Diferencias entre Atributos y propiedades:

- **Atributos:** son parte de la etiqueta HTML y se definen directamente en el código HTML. Ejemplos incluyen src, alt, class, id, etc.
- **Propiedades:** son representaciones en el DOM de los atributos, pero pueden tener valores diferentes a los de los atributos en el código HTML. Las propiedades también incluyen características no representadas en el HTML, como innerText o value.

Manipulación de atributos: usamos los métodos getAttribute(), setAttribute() y removeAttribute() para trabajar con atributos.

DOM: Atributos y Propiedades

- **getAttribute():** devuelve el valor de un atributo especificado en el elemento. Si el atributo no existe, devuelve null.
- **setAttribute():** establece el valor de un atributo en el elemento indicado. Si el atributo ya existe, el valor es actualizado, en caso contrario, el nuevo atributo es añadido con el nombre y valor indicado.
- **removeAttribute():** elimina un atributo HTML. Si el atributo no existe no ocurre nada y no se produce error.
- **hasAttribute():** devuelve un valor booleano que indica si un elemento tiene un atributo concreto.

Unidad 4: DOM y BOM

DOM: Atributos y propiedades

Ejemplo del uso de los métodos getAttribute(), setAttribute(), removeAttribute() y hasAttribute():

```
<body>
    <img id="foto" src="img1.jpg" alt="Imagen inicial">
    <button onclick="ver()">getAttribute()</button>
    <button onclick="cambiar()">setAttribute()</button>
    <button onclick="quitar()">removeAttribute()</button>
    <button onclick="comprobar()">hasAttribute()</button>
    <script>
        const img = document.getElementById("foto");
        function ver() { alert(img.getAttribute("src"));}
        function cambiar() { img.setAttribute("src", "img2.jpg");}
        function quitar() { img.removeAttribute("alt");}
        function comprobar() {
            if (img.hasAttribute("alt")) { alert("La imagen TIENE el atributo alt");
            } else { alert("La imagen NO tiene el atributo alt");}
        }
    </script>
</body>
```

DOM: Clases y Estilos

Las clases y estilos pueden considerarse tanto atributos como propiedades, pero se suelen manipular de formas específicas.

- **Manipulación de clases:** podemos añadir, quitar o comprobar si un elemento tiene una clase usando las propiedades **classList**: **add**, **remove** y **contains**.
- **Manipulación de estilos: l**os estilos en línea se pueden manipular utilizando la propiedad **style**.

DOM: Clases y Estilos

La propiedad **classList** permite trabajar con las clases CSS de un elemento de forma sencilla.

Los métodos más comunes son:

- **add()**: añade una o varias clases al elemento.
- **remove()**: elimina una o varias clases.
- **toggle()**: añade la clases si no existe, la elimina si existe.
- **contains()**: devuelve true si el elemento tiene la clase y false si no.

classList evita tener que manipular element.className con cadenas de texto largas. Y es mucho más seguro y limpio para añadir, eliminar o alternar clases dinámicamente.

Unidad 4: DOM y BOM

DOM: Clases y Estilos

Ejemplo:

```
<style>
    .resaltado {
      background-color:
yellow;
    }
    .grande {
      font-size: 2em;
    }
</style>
```

```
<body>
  <p id="miParrafo">¡Hola, mundo!</p>
  <button id="btnAdd">Añadir clase</button>
  <button id="btnRemove">Quitar clase</button>
  <button id="btnToggle">Alternar clase</button>
  <script>
    const p = document.getElementById("miParrafo");
    document.getElementById("btnAdd").addEventListener("click", () => {
      p.classList.add("resaltado", "grande");
    });
    document.getElementById("btnRemove").addEventListener("click", () => {
      p.classList.remove("resaltado", "grande");
    });
    document.getElementById("btnToggle").addEventListener("click", () => {
      p.classList.toggle("resaltado");
    });
  </script>
</body>
```

DOM: Manipulación del contenido y la estructura

Es muy común modificar el contenido y la estructura del DOM cuando se crean aplicaciones web interactivas. Ya sea que se necesite cambiar el texto de un elemento, añadir nuevos elementos o reorganizar los existentes, JS provee de una variedad de métodos y técnicas para hacerlo:

- **Modificación de contenido:** para cambiar el contenido de un elemento, se pueden usar las propiedades **innerText**, **innerHTML** y **textContent**.
- **Inserción de elementos:** se pueden insertar nuevos elementos usando métodos como **appendChild()**, **insertBefore()** e **innerHTML()**.

DOM: Manipulación del contenido y la estructura

- **Eliminación de elementos:** se pueden eliminar elementos del DOM utilizando métodos cómo **removeChild()** y **remove()**.
- **Reemplazo de elementos:** se puede reemplazar un elemento existente con uno nuevo usando **replaceChild()**.
- **Clonación de elementos:** se puede clonar un nodo existente y sus descendientes utilizando **cloneNode**.

Ejemplo: **EjemploManipulacionContenidoYEstructura.html**

DOM: Creación y Eliminación de elementos

La creación y eliminación de elementos del DOM es una operación esencial para cualquier aplicación web dinámica. Esta capacidad permite a los desarrolladores agregar nuevos contenidos y componentes a sus páginas web de manera dinámica, así como también eliminar los elementos que ya no son necesarios.

- **Creación de elementos:** para crear un nuevo elemento en el DOM, se utiliza el método **document.createElement()**.
- **Inserción de elementos:** se pueden usar varios métodos para esto: **appendChild()**, **insertBefore()**, **insertAdjacentElement()**, **insertAdjancentHTML()** e **insertAdjancentText()**.

DOM: Creación y Eliminación de elementos

- **appendChild()**: añade un nuevo nodo como el último hijo de un elemento
- **insertBefore()**: añade un nuevo nodo antes de un nodo existente.
- **insertAdjacentElement()**, **insertAdjancentHTML()** e **insertAdjancentText()**: permiten insertar nuevos nodos en posiciones específicas en relación con un elemento de referencia.
- **Eliminación de elementos: s**e pueden eliminar elementos del DOM utilizando removeChild() o remove().

Ejemplo: **EjemploCreacionYEliminacionElementos.html**

DOM: Eventos

JavaScript puede reaccionar a cualquier evento que ocurra en una página web. Algunos de los elementos que puede controlar JavaScript son:

- Pulsar una tecla.
- Enviar un formulario.
- Modificar un campo de texto.
- Cuando el ratón pasa sobre un elemento.
- Cuando se carga una imagen.
- Cuando el usuario hace clic sobre un elemento.
- Cuando se carga una página web.

Unidad 4: DOM y BOM

DOM: Eventos

Todos los eventos que ocurren a un objeto están basados en el objeto **event**. Este objeto contiene las propiedades y métodos que son comunes a todos los eventos.

Actualmente se usa el método **addEventListener**, en vez de añadir el evento en todos los elementos HTML afectados (esto independiza el código HTML del código JS). W3C defiende el uso de addEventListener.

DOM: Manejadores de Eventos

addEventListener permite asociar evento y función JavaScript desde el propio script. El manejador de eventos sirve para indicarle al navegador qué tiene que hacer cuando ocurre una acción, en este caso, cuando el usuario hace clic en un botón.

```
<body>
   <button id="boton">Pulsar<p id="texto"></p></button>
   <script>
      function muestraFecha() {
      document.getElementById("texto").innerHTML = Date();
      }
      document.getElementById("boton").addEventListener("click",muestraFecha);
   </script>
</body>
```

DOM: Manejadores de Eventos

El evento **DOMContentLoaded** se dispara cuando el contenido del archivo HTML se ha cargado en el navegador, sin necesitar esperar imágenes, hojas de estilo, etc. Es muy conveniente en la mayoría de las veces inicializar script de JS utilizando este evento en lugar de load, ya que permite ejecutar el código antes y mejora el rendimiento.

En este ejemplo, se ejecuta la función cuando ya se ha cargado el contenido del archivo HTML, si el script estuviese en el head y no se usase DOMContentLoaded, el elemento h1 no existiría y el código fallaría.

```
<body>
  <h1 id="titulo">Título original</h1>
  <script>
    // El código se ejecuta cuando el HTML ya está cargado
    document.addEventListener("DOMContentLoaded", function () {
      const titulo = document.getElementById("titulo");
      titulo.textContent = "El DOM ya está cargado";
    });
  </script>
```

DOM: Manejadores de Eventos

Asignación Directa

Otra forma de añadir un manejador de eventos es asignando una función directamente a la propiedad de evento del elemento.

En el siguiente ejemplo, con onclick sólo puede haber un manejador de evento, si se asigna otro, se sobreescribe.

```
var button = document.getElementById("myButton");
button.onclick = function() {
    alert("¡Botón clickeado!");
};
```

DOM: Manejadores de Eventos

Borrado de manejadores de eventos

Para eliminar un manejador de eventos, se usa **removeEventListener()**. Para eliminar un evento es muy importante indicar la misma función que añadimos con addEventListener() y no una función diferente que haga lo mismo que la primera.

```
function showAlert() { alert("¡Botón clickeado!");}
button.addEventListener("click", showAlert);
button.removeEventListener("click", showAlert);
```

DOM: Manejadores de Eventos

event.target y event.currentTarget

Cuando ocurre un evento, en el objeto evento dos de las propiedades más importantes de este son:

- **event.target:**
  - Es una propiedad de **solo lectura**.
  - Hace **referencia** al elemento exacto donde el usuario ha realizado la acción, es decir, el **elemento** que **ha provocado el evento**.
- **event.currentTarget:**
  - Hace **referencia** al **elemento** al que se le ha asignado el manejador de eventos.
  - Es el elemento que **está gestionando el evento**, no necesariamente donde se ha hecho clic.

DOM: Manejadores de Eventos

event.target y event.currentTarget:

En el siguiente ejemplo, el botón que pulsas se vuelve amarillo -> **event.target** y el contenedor div recibe un borde negro -> **event.currentTarget**.

```
<div id="contenedor">
        <button>Botón 1</button>
        <button>Botón 2</button>
        <button>Botón 3</button>
</div>
<script>
        const contenedor = document.getElementById("contenedor");
        contenedor.addEventListener("click", function (evt) {
            // Cambia el color del botón pulsado (target)
            evt.target.style.backgroundColor = "yellow";
            // Cambia el borde del contenedor (currentTarget)
            evt.currentTarget.style.border = "2px solid black";});
    </script>
```

DOM: Delegación de Eventos

En lugar de adjuntar un escuchador de eventos a cada elemento que pueda generar un evento, se adjunta un único escuchador en un ancestro común. Este escuchador único puede entonces manejar los eventos desencadenados por los elementos hijos.

¿Por qué usar la Delegación de Eventos?

Sobre todo cuando se trabaja con elementos generados dinámicamente o se necesita reducir el número de escuchadores de eventos en una página.

Algunas de las ventajas incluyen:

- **Mejora del rendimiento:** reducir el número de escuchadores de eventos que se deben adjuntar a los elementos.
- **Simplicidad de la gestión:** facilitar la gestión de eventos, especialmente cuando se añaden o eliminan elementos dinámicamente.

DOM: Delegación de Eventos

¿Cómo funciona la Delegación de Eventos?

La delegación de eventos aprovecha la técnica de propagación de eventos en el DOM, especialmente la fase de propagación. Durante esta fase, un evento se propaga desde el elemento hijo hasta los padres en el árbol DOM.

Uso de matches() y closest()

Para trabajar con la delegación de eventos, estos dos métodos son muy útiles.

- **matches(selector):** comprueba si el elemento invocante coincide con el selector CSS especificado. Devuelve true o false.
- **closest(selector):** devuelve el ancestro más cercano (subiendo por el árbol) que coincide con el selector CSS, o el propio elemento si ello coincide. Devuelve el elemento si coincide y si no hay coincidencias, devuelve null.

DOM: Delegación de Eventos

Ejemplo:

1. Se asigna un único manejador al <ul> listaPadre.
2. Cuando se hace clic en un <li>, el evento se propaga hasta el <ul>.
3. El manejador comprueba que el target sea un <li>.
4. Se ejecuta la acción sólo sobre el elemento cliqueado.

```
<ul id="listapadre">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
<script>
  var listaPadre = document.getElementById("listapadre");
  listaPadre.addEventListener("click", function(event) {
    // Solo reaccionar si se hace click sobre un <li>
    if (event.target && event.target.matches("li")) {
      console.log("Ítem clickeado: " + event.target.textContent);
    }
  });
</script>
```

DOM: Delegación de Eventos

Ejemplo de matches() y closest() :

1. El usuario pulsa el botón A o B.
2. El evento “burbujea” hasta el contenedor #c.
3. El if comprueba que el target sea un botón.
4. Se encuentra la tarjeta .tar que contiene el botón y se resalta en amarillo

```
<div id="c">
  <div class="tar"><button>A</button></div>
  <div class="tar"><button>B</button></div>
</div>
<script>
document.getElementById("c").addEventListener("click", e => {
  if(e.target.matches("button")){
     e.target.closest(".tar").style.background="yellow";
  }
});
</script>
```

Navegación en el DOM

El DOM es una estructura jerárquica de nodos, y JavaScript proporciona diversas propiedades y métodos para recorrer esta estructura y localizar elementos.

Un **nodo** es cualquier unidad del DOM. Representa cualquier parte del documento: elementos HTML, texto, comentarios, atributos, etc.

Un **elemento** es un tipo concreto de nodo que representa una etiqueta HTML.

Navegación en el DOM

Permiten recorrer la estructura del DOM y localizar elementos:

- **parentNode**: accede al nodo padre del elemento actual.
- **childNodes**: devuelve una colección de todos los nodos hijos (incluye nodos de texto, comentarios y elementos.
- **children**: devuelve únicamente los nodos de tipo elemento.
- **firstChild**: devuelve el primer nodo hijo sea del tipo que sea.
- **firstElementChild**: devuelve el primer hijo que sea de un elemento.
- **nextSibling**: devuelve el siguiente nodo al mismo nivel.
- **nextElementSibling**: devuelve el siguiente nodo hermano que sea un elemento.
- **closest** (selector): permite encontrar el ancestro más cercano que coincida con un selector CSS.

Unidad 4: DOM y BOM

Navegación en el DOM

Ejemplo:

```html
<div class="contenedor">
  <p id="texto">Hola
<span>Mundo</span></p>
  <p>Otro párrafo</p>
</div>
```

```
<script>
      const texto = document.getElementById("texto");
      // Nodo padre
      console.log(texto.parentNode);// <div class="contenedor">
      // Hijos
      console.log(texto.childNodes);// texto + <span>
      console.log(texto.children); // solo <span>
      // Primer hijo
      console.log(texto.firstChild);// nodo de texto "Hola "
      console.log(texto.firstElementChild);// <span>
      // Hermanos
      console.log(texto.nextSibling); // nodo de texto (salto de línea)
      console.log(texto.nextElementSibling);// <p>Otro párrafo</p>
      // Ancestro más cercano
      console.log(texto.closest(".contenedor"));//<div class="contenedor">
</script>
```

Unidad 4: DOM y BOM

BOM

El BOM (Browser Object Model) es el modelo de objetos del navegador que están presentes en JavaScript para controlar cualquier cosa dentro de la ventana del cliente web.

Diferencias entre DOM y BOM:

- Con el DOM, JS puede acceder a los elementos de un documento o página web mediante su estructura interna.
- Con el DOM, no se puede acceder a ciertos aspectos del navegador cómo la URL, las dimensiones de la ventana, cerrar o redimensionar la ventana del navegador, gestionar cookies, etc. Para esto se utiliza el BOM, que es otra estructura arborescente similar al DOM con otra serie de objetos.
- A diferencia del DOM, el elemento raíz del BOM es window.

Unidad 4: DOM y BOM

BOM

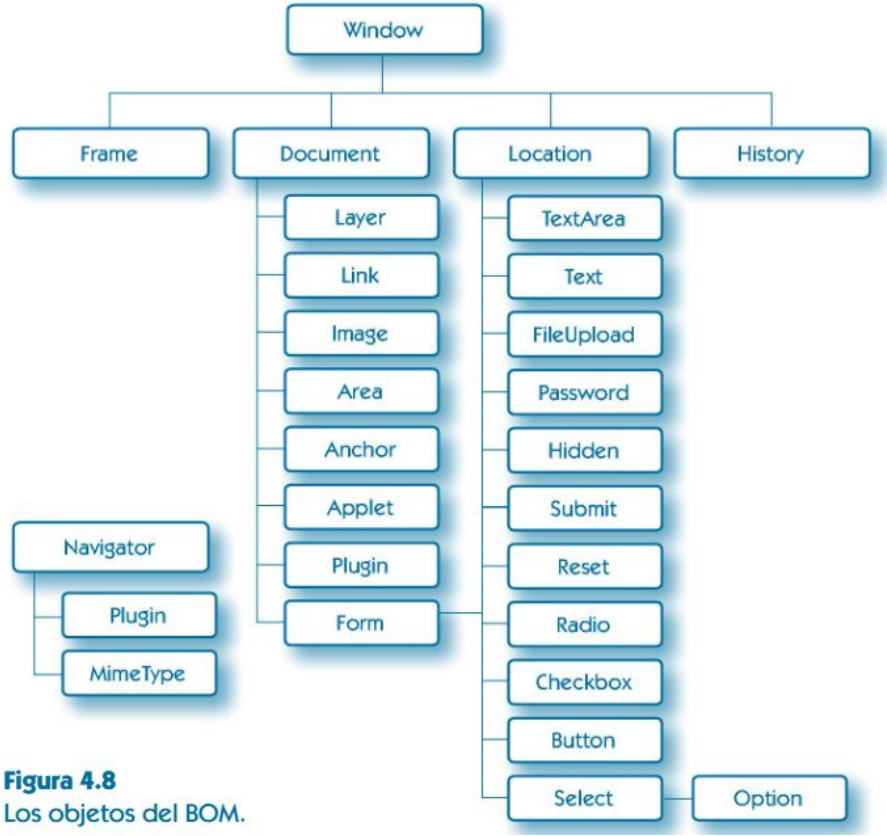


Figura 4.8
Los objetos del BOM.

BOM: Objeto window

El objeto document forma parte del objeto window, por lo que necesitamos referirnos a él cómo window.document. Sin embargo, JavaScript nos permite usarlo simplemente escribiendo document.

```
<body>
    <script>
        // Acceso al DOM a través del objeto window
        document.body.innerText = "¡Hola!";
        window.document.body.innerText = "¡Hola!";
    </script>
    <form>
        <input type="button" value="Púlsame" onclick="window.alert('¡Hola mundo!')">
    </form>
</body>
```

BOM: Objeto location

Representa la URL actual del documento que se está visualizando en la ventana navegador. Por medio del objeto location podemos hacer redirecciones hacia otras páginas.

Propiedades:

- **location.href:** obtiene la URL completa.
- **location.protocol:** obtiene el protocolo.
- **location.host:** obtiene el nombre del host incluyendo el puerto.
- **location.hostname:** obtiene el nombre del host sin incluir el puerto.
- **location.port:** devuelve el puerto de la URL actual o una cadena vacía si es el puerto por defecto.
- **location.origin:** obtiene la URL sin el path y otros segmentos posteriores.
- **location.pathname:** obtiene la ruta.
- **location.search:** obtiene el querystring.
- **location.hash:** obtiene la parte de la almohadilla (hash).

BOM: Objeto history

Este objeto nos permite interactuar con el historial del navegador, que tiene una lista de las URL que ha visitado el usuario recientemente.

window.history.back(); esto nos llevaría a la página anterior que el usuario tenga en el historial, si es que hay alguna.

Por ejemplo, si quisiéramos ver en consola el nº de páginas del historial escribiríamos:
console.log(**window.history.length**)

Métodos:

- **history.back()**: enviaría hacia la página anterior si hay.
- **history.forward()**: enviaría a la página siguiente, si hay.
- **history.go(-2)**: navega dos páginas hacia atrás.
- **history.go(1)**: navega una página hacia delante.

Para cambiar la URL sin recargar la página:

window.history.pushState({}, 'Otro título para la página', '/otra/url');

BOM: Objeto screen y Objeto navigator

El **objeto screen** proporciona información sobre la pantalla del dispositivo del usuario.

alert(screen.width);

El **objeto navigator** proporciona información sobre el navegador:

```
<body>
 <script>
      const idioma = navigator.language;
      const agente = navigator.userAgent;

      alert(
        "Idioma del navegador: " + idioma +
        "\nUser Agent: " + agente
      );
  </script>
```