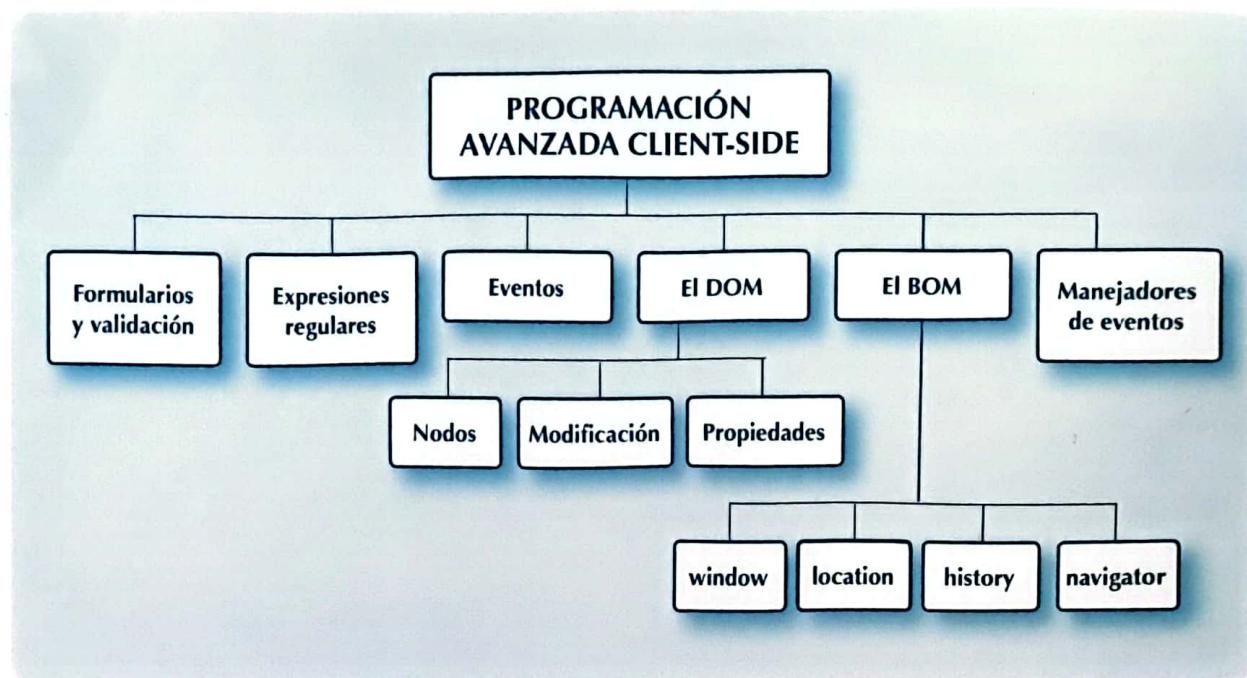


Mapa conceptual



Glosario

BOM. Objeto en el que se representa el navegador, no solo el documento. Mediante el BOM se puede acceder a datos como el historial de navegación, dimensiones de la ventana, etcétera.

DOM. Plataforma e interfaz de un documento que permite a los scripts y programas acceder y modificar dinámicamente su contenido, estructura y estilo.

EventListener. Código que se ejecuta cuando en la interfaz se genera un evento concreto (clic del ratón, pulsar una tecla en un campo de un formulario, etcétera).

Expresión regular. Secuencia de caracteres para realizar búsquedas, filtros o sustituciones en cadenas de caracteres.

Tag. Término en inglés que significa “etiqueta” y hace referencia a una palabra clave que sirve para describir un documento.

4.1. Introducción

En este capítulo, se trabajará en profundidad el concepto de *DOM* y de *BOM*. La ventaja que tiene JavaScript es poder acceder a ambos objetos, lo cual consigue que las aplicaciones sean interactivas.

Accediendo al DOM y al BOM, el programador va a poder conocer la potencia de la programación del lado del cliente. Este hecho permitirá que mucha de la lógica de las aplicaciones web corra en este lado dedicándose el servidor a temas más específicos de back-end.

4.2. Formularios

El trabajo con campos, ya sea en formularios o de forma aislada, es la base de las aplicaciones web. Con HTML5, es posible comprobar el contenido de muchos de los campos de una web, pero, en ocasiones, se necesitan ciertas validaciones y acciones extra que pueden ser realizadas en JavaScript.

4.2.1. La validación de campos

A continuación, se mostrará un ejemplo en el que se intenta validar un campo de texto, de tal manera que solamente se permita introducir un número entre 5 y 10:

```
<!DOCTYPE html>
<html>
<body>
<p>Introduzca un número entre 5 y 10:</p>
<input id="aqui" type="text">
<button type="button" onclick="dale()">Comprueba</button>
<p id="msg"></p>
<script>
function dale() {
    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 "No llega";
        if(x > 10) throw "Se pasa";
    }
    catch(err) {
        msg.innerHTML = "Resultado: " + err;
    }
}

</script>
</body>
</html>
```

En primer lugar, se puede observar que, en el código, existe un campo input de texto que es el que va a validarse con identificador (ID) "aqui".

Aparte del campo por validar, se añade un botón que, al presionarlo, ejecute la función `dale()`, que es la encargada de validar el campo anterior.

Para la validación, se utilizan las sentencias `try` y `catch`, de tal manera que, si el campo de `texto` contiene algún valor incorrecto, el código lanzará una excepción recogida con la sentencia `catch`. El bloque del `catch` se encargará de mostrar el texto lanzado en la excepción.



SABÍAS QUE...

La validación anterior es posible realizarla solamente con HTML.

4.3. Expresiones regulares

Las expresiones regulares son objetos que se pueden utilizar en los métodos `exec` y `test` del objeto `RegExp` y también en los métodos `search`, `search`, `replace` y `match` del objeto `string`.

Una expresión regular es una secuencia de caracteres que forman un patrón determinado. Los patrones se utilizan para realizar operaciones de búsqueda y reemplazamiento en `strings`. Las expresiones regulares son un concepto heredado del Unix donde se utilizan mucho en comandos del sistema como `grep`, `find`, `awk`.

Las expresiones regulares pueden ser sencillas o complejas dependiendo del objetivo que se necesite conseguir.

RECUERDA

- ✓ Las expresiones regulares son ampliamente utilizadas en la validación de campos por su potencia y efectividad.

Véase la sintaxis de una expresión regular:

`/patrón/modificadores`

Un ejemplo sencillo de una expresión regular sería el siguiente:

```
var patron = /myfpschool/i;
```

Dónde `/myfpschool/i` es la expresión regular. El objetivo es buscar la cadena “`myfpschool`” de forma case-insensitive (se le ha añadido el modificador `i`).

Obsérvese que no se utilizan comillas al definir el patrón. No es un string.

Aplicado a un código concreto, sería algo parecido a esto:

```
var cadena = "Aprende en MYfpschool javascript";
var patron = /myfpschool/i;
var pos = cadena.search(patron);
```

Al ejecutar el código anterior, el valor de la variable pos sería 11 porque la cadena myfpschool aparece en dicha posición.

Véase un ejemplo de utilización con otras expresiones regulares:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Expresiones regulares</title>
  </head>
  <body>

    <script>

var cadena = "Sintesis1@sintesis.com";
var patron = /\w/g; // \w selecciona los caracteres de una cadena de
caracteres
var resultado = cadena.match(patron);
//devuelve un array de palabras y números sin otros símbolos
//[ 'S', 'i', 'n', 't', 'e', 's', 'i', 's', 'l', 's', 'i', 'n', 't', 'e', 's', 'i',
's', 'c', 'o', 'm' ]
console.log(resultado);

var patron = /\d/g; // \d selecciona los dígitos de una cadena de ca-
racteres
var resultado = cadena.match(patron);
//devuelve un array con los dígitos
//[ '1' ]
console.log(resultado);

var patron = /\W/g; // \W selecciona los NO caracteres de una cadena
de caracteres
var resultado = cadena.match(patron);
//devuelve un array con los no caracteres
//[ '@', '.' ]
console.log(resultado);

var cadena = "Aprende JavaScript en Sintesis";
var patron = /\S/g; // \S elimina los espacios de una cadena de carac-
teres
var resultado = cadena.match(patron);
//devuelve un array de los elementos del string sin espacios en blanco
//[ 'A', 'p', 'r', 'e', 'n', 'd', 'e', 'J', 'a', 'v', 'a', 'S', 'c', 'r', 'i', 'p',
't', 'e', 'n', 'S', 'i', 'n', 't', 'e', 's', 'i', 's' ]
console.log(resultado);

var patron = /r+/g; // r+ busca si en la cadena tiene una o más erres.
var resultado = cadena.match(patron);
```

```

//devuelve un array de los elementos encontrados
//[‘r’,’r’]
console.log(resultado);

var cadena = “Ponte a 100 o a 1000 con Sintesis”;
var patron = /\d{2}/g; // busca si hay una secuencia de dos dígitos
var resultado = cadena.match(patron);
//devuelve un array de las secuencias encontradas
//[‘10’,’10’,’00’]
console.log(resultado);

var patron = /\d{2,3}/g; // busca si hay una secuencia de dos o tres
dígitos
var resultado = cadena.match(patron);
//devuelve un array de las secuencias encontradas
//[‘100’,’100’]
console.log(resultado);
</script>

</body>
</html>

```



PARA SABER MÁS

El objeto RegExp, como se ha explicado, tiene, entre otros, los métodos exec y test para ejecutar una expresión regular y testearla respectivamente. Las siguientes líneas de código tendrían la misma funcionalidad:

```

var patron = /myfpschool/i;
var re = new RegExp('/myfpschool/i');

```

Para más información sobre las expresiones regulares, se puede acceder a un recurso de MDN a través del siguiente código QR:



4.4. DOM

Según el W3C, el DOM es una plataforma e interfaz de un documento que permite a los scripts y los programas acceder y modificar dinámicamente su contenido, estructura y estilo.

En la figura 4.1, puede observarse un ejemplo visual del DOM.

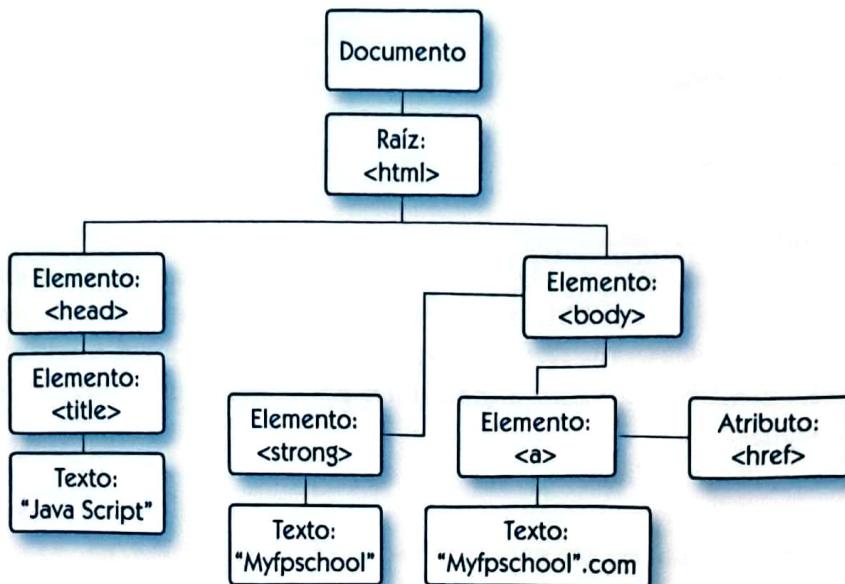


Figura 4.1
Detalle del DOM de una página concreta
(en la raíz, el objeto document).

Como ya se ha comentado, JavaScript puede acceder al DOM (Document Object Model) de cualquier página web. Al contrario que las páginas HTML estáticas, con JavaScript, se puede hacer que una web sea dinámica cambiando el contenido HTML de cualquier elemento, cambiando el estilo CSS de esta, reaccionando a los eventos del DOM o incluso añadiendo o eliminando elementos HTML de tu web.

FUNDAMENTAL

El objeto document

Cuando se carga un documento en el navegador, se crea el objeto `document`, el cual, además de ser la raíz de todo el documento HTML, tiene una serie muy amplia de propiedades y métodos. Algunas de ellas se citarán a continuación:

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

Desde el capítulo 1, se ha visto cómo se podía acceder al DOM y modificar su contenido. Recuérdese uno de los primeros ejemplos:

```

<!DOCTYPE html>
<html>
<body>
  <h1>Modificando el código HTML</h1>
  <p id="prueba">Modificando el contenido.</p>

```

```

<button type="button" onclick="document.getElementById('prueba').
    innerHTML = 'CAMBIANDO el contenido!'">¡Dale!</button>
</body>
</html>

```

Para ello, se utilizaba el método `getElementById()`, que permitía obtener el identificador (ID) de un elemento HTML, en este caso, el elemento prueba. Una vez seleccionado el elemento concreto, mediante la propiedad `innerHTML`, será posible cambiar el contenido de un elemento HTML.

Además de buscar elementos por el ID, se pueden buscar elementos por el tag [utilizando el método `getElementsByName()`] o también por la clase [utilizando el método `getElementsByClassName()`].

Observe el siguiente código:

```

<p id="texto">Curso de <strong>JavaScript</strong></p>
<p id="texto2">Curso de <strong>React</strong></p>
<script>
    var a = document.getElementById("texto");
    var b = a.getElementsByTagName("strong");
    alert (b[0].innerHTML);
</script>

```

En el ejemplo anterior, se muestra un mensaje de alerta con el texto “JavaScript”. En la primera línea del script, se selecciona el párrafo con ID igual a “texto” y, en la segunda, se extraen de ese texto todas las etiquetas en negrita, con lo cual, en la posición 0, se almacenará el objeto JavaScript.

Véase un ejemplo más completo de utilización del DOM con JavaScript. A continuación, se muestra un ejemplo de test sobre nomofobia:

```

<!DOCTYPE html>
<html>
<body>

<form id="frm1">
<ol>
    <input type="checkbox" name="nomo" >
    <strong>1.</strong> Cuando mandas un mensaje por whatsapp esperas
    siempre al doble check. Si no aparece vuelves a abrir el terminal
    para revisarlo al rato.
</ol>
<ol>
    <input type="checkbox" name="nomo" >
    <strong>2.</strong> Antes de acostarte siempre miras el móvil a ver
    si tienes mensajes o notificaciones.
</ol>
<ol>
    <input type="checkbox" name="nomo" >
    <strong>3.</strong> Te despiertas antes de tiempo para jugar, mandar
    mensajes, actualizar perfiles,... con el teléfono móvil.

```

```
</ol>
<ol>
    <input type="checkbox" name="nom0" >
    <strong>4.</strong> Si sales de casa sin el móvil volverías a co-
gerlo aunque llegues tarde a tu cita.
</ol>
<ol>
    <input type="checkbox" name="nom0" >
    <strong>5.</strong> Tienes miedo a quedarte sin batería.
</ol>
<ol>
    <input type="checkbox" name="nom0" >
    <strong>6.</strong> Cuando tienes la batería baja desactivas cier-
tas aplicaciones u opciones del teléfono como la WiFi, bluetooth
para no quedarte sin batería.
</ol>
<ol>
    <input type="checkbox" name="nom0" >
    <strong>7.</strong> Tienes ansiedad cuando tienes llamadas perdi-
das. Llamas a los números y te preocupas si no responden.
</ol>
<ol>
    <input type="checkbox" name="nom0" >
    <strong>8.</strong> Miras la cobertura cuando estás en algún sitio
con los amigos, esperando, etc.
</ol>
<ol>
    <input type="checkbox" name="nom0" >
    <strong>9.</strong> Sueles hacer alguna otra cosa mientras que mi-
ras al móvil como comer, lavarte los dientes, etc.
</ol>
<ol>
    <input type="checkbox" name="nom0" >
    <strong>10.</strong> Vas al baño siempre con el móvil.
</ol>
    <br><br>
</form>
```

<p>Recuerda marcar todas las cuestiones arriba expuestas que creas que siempre realizas.

</p>

```
<button onclick="myFunction()">Comprueba si sufres nomofobia</but-
ton>
```

```
<h3><p id="resultado"></p></h3>
<h4><p id="consejo"></p></h4>
```

```
<script>
function myFunction() {
    var cont=0
    var x = document.forms["frm1"];
    var text = "";
    var i;
    for (i = 0; i < x.length ;i++) {
        if (x.elements[i].checked){ cont++; }
    }

    document.getElementById("resultado").innerHTML = "Resultado: El
principio no tienes nada de que preocuparte.";
    document.getElementById("consejo").innerHTML = "Resultado: El
principio no tienes nada de que preocuparte.";

    if (cont>5){
        document.getElementById("resultado").innerHTML = "Resultado:
Empiezas a tener signos de dependencia del móvil. ";
        document.getElementById("consejo").innerHTML = "Consejo: Puedes utilizar técnicas como apagar el móvil cuando no lo necesitas, cuando duermes, etc.";
    }

    if (cont>6){
        document.getElementById("resultado").innerHTML = "Resultado:
Tienes una gran dependencia del móvil. ";
        document.getElementById("consejo").innerHTML = "Consejo: Deberías seguir un plan de desintoxicación del móvil como por ejemplo dejar el móvil en casa cuando vas a comprar, apagarlo durante la noche, apagarlo durante horas de clase o trabajo, etc.";
    }

    if (cont>8){
        document.getElementById("resultado").innerHTML = "Resultado:
Tus síntomas de dependencia son muy preocupantes. ";
        document.getElementById("consejo").innerHTML = "Consejo: Además de todas las técnicas de los apartados anteriores deberías plantearte un plan de desintoxicación del móvil que consista en estar una o dos semanas sin utilizarlo. Si ves que no puedes hacerlo por ti mismo, pide ayuda a un profesional.";
    }
}

</script>

</body>
</html>
```

En el script del ejemplo anterior, se puede ver cómo se hace un recorrido por todos los campos del formulario [función myFunction()] y se van registrando en la variable cont de la función anterior todos aquellos que estén seleccionados cambiando el contenido de dos párrafos con ID “resultado” y “consejo”.

Recurso web

www

Si necesitas saber más sobre el DOM en la página oficial del W3C, puedes acceder desde el siguiente código QR:



4.4.1. El acceso al DOM. El método document.querySelector()

El método querySelector() del objeto document devuelve el primer elemento que encuentre y que concuerde con el selector que se le introduzca como parámetro. Es importante recalcar que solamente devolverá el primer elemento y no todos. Para seleccionar todos los elementos que concuerden con un selector, habrá que utilizar el método querySelectorAll().

Véase un ejemplo de uso en una página web:

```
<!DOCTYPE html>
<html>
<body>

<h3 class="ejemplo">introduzca una expresi&oacute;n num&eacute;rica</h3>

<label for="texto">Introducir datos:</label>
<input type="text" id="texto">

<button>Ejecutar expresi&oacute;n</button>

<p class="ejemplo">Un p&aacute;rrafo de la clase ejemplo.</p>
<p>otro p&aacute;rrafo.</p>

<button onclick="laFuncion()">Cambiar color</button>

<script>
function laFuncion() {
    document.querySelector(".ejemplo").style.backgroundColor = "red";
    document.querySelector("p").innerHTML = "rojo";
}

```

```

var input = document.querySelector('input');
var boton = document.querySelector('button');
var parrafo = document.querySelector('p');
boton.onclick = function() {
    var dato = input.value;
    parrafo.innerHTML = eval(dato);
}

</script>

</body>
</html>

```

El aspecto de la página web creada con el código anterior será el que se muestra en la figura 4.2.

Figura 4.2
Resultado de la página creada con el código anterior.

Como se puede observar en el código, cuando se pulsa el botón “Cambiar color”, se ejecuta la función de nombre *laFuncion()*.

```

function laFuncion() {
    document.querySelector(".ejemplo").style.backgroundColor = "red";
    document.querySelector("p").innerHTML = "rojo";
}

```

En esta función, se cambiará el color de fondo (primera línea de la función) de aquel elemento de la clase ejemplo, que, en este caso, será el título h3. Además, se seleccionará el primer párrafo (segunda línea de la función) y cambiará el texto que tenga por “rojo”.

El resultado de pulsar este botón será el que se indica en la figura 4.3.

Figura 4.3
Pulsando el botón cambiar color.

En la segunda parte del script, se selecciona el campo input, el primer botón y el primer párrafo (véanse las tres primeras líneas siguientes) y se le asigna al botón una función.

```
var input = document.querySelector('input');
var boton = document.querySelector('button');
var parrafo = document.querySelector('p');
boton.onclick = function() {
    var dato = input.value;
    parrafo.innerHTML = eval(dato);
}
```

En esta función, se utiliza la función eval para ejecutar la expresión que se introduzca en el campo input.

El resultado de introducir la expresión $5+4$ y ejecutar el botón “Ejecutar expresión” será el que se muestra en la figura 4.4.

Figura 4.4

Ejecutando la expresión
 $5+4$.

4.5. Eventos del DOM

Ya en el segundo capítulo se hizo una rápida introducción a los eventos que se podían capturar en una página web, así como a los manejadores de eventos. En esta sección, se profundizará más sobre estos conceptos.

JavaScript puede reaccionar a cualquier evento que ocurra en una página web. Hasta ahora, el evento utilizado era el clic sobre un botón de la página web, pero ¿qué tipos de eventos puede controlar JavaScript? A continuación, se muestran algunos de los eventos que pueden ser capturados:

- 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.

Véase un ejemplo de control del ratón sobre un texto:

```
<!DOCTYPE html>
<html>
<body>
```

```

<div onmouseover="dentro(this)" onmouseout="fuera(this)">
Editorial Síntesis</div>

<script>
function dentro(obj) {
    obj.innerHTML = "Desarrollo web en entorno cliente"
}
function fuera(obj) {
    obj.innerHTML = "Editorial Síntesis"
}
</script>

</body>
</html>

```

En el ejemplo anterior, se controla si el ratón está sobre la etiqueta <div> cambiando el texto que la contiene y, en caso de que el ratón deje de estar sobre ella, se restablecerá el valor original de la etiqueta.

RECUERDA

- ✓ 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, los programadores prefieren el método addEventListener en vez de añadir el evento en todos y cada uno de los elementos HTML afectados. Esta forma de programar es más limpia, puesto que independiza el código HTML del código JavaScript. El W3C pone énfasis en cambiar el modelo tradicional de registro de eventos por este nuevo método.

Véase otro ejemplo que comprueba cuando se produce la carga de la página si las cookies están habilitadas o no:

```

<!DOCTYPE html>
<html>
<body onload="miraCookies()">
<p id="muestracookies"></p>
<script>
function miraCookies() {
    var text = "";
    if (navigator.cookieEnabled == true) {
        text = "Las cookies están habilitadas.";
    } else {
        text = "Las cookies están deshabilitadas.";
    }
    document.getElementById("muestracookies").innerHTML = text;
}

```

```

}
</script>
</body>
</html>

```

Como se puede observar, en este caso, se ejecutará el script en la carga de la página (evento onload).

4.6. Manejadores de eventos

Cuando ocurre un evento de un objeto del documento o página web, es posible con JavaScript asociarlo a una función. Ya en el primer capítulo se pudieron ver scripts que asociaban al evento click del ratón sobre un botón con una función JavaScript.

Con el método addEventListener, es posible asociar evento y función JavaScript desde el propio script limpiando de código el HTML. Véase un ejemplo de utilización:

```

<!DOCTYPE html>
<html>
<body>
<button id="unboton">Dale</button><p id="texto"></p>

<script>
function MuestraFecha() {
    document.getElementById("texto").innerHTML = Date();
}
document.getElementById("unboton").addEventListener("click", MuestraFecha);
</script>

</body>
</html>

```

Como puede observarse, por un lado, está la función MuestraFecha(), que asigna al contenido HTML del elemento con id=texto la fecha actual.

Por otro lado, está el método addEventListener asociado al botón que se ha creado con identificador=unboton. Lo que hace ese método es asociar el evento click del botón con la función MuestraFecha.



Actividad propuesta 4.1

Del ejemplo expuesto en este apartado, desliga el código JavaScript a otro fichero y haz que funcione la página web. Coloca el código JavaScript en una subcarpeta scripts para tener una estructura de proyecto más ordenada.

4.7. Nodos del DOM

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



TOMA NOTA

Hasta que el navegador no haya construido totalmente el árbol (DOM) cargándose la página por completo, no será posible acceder hasta él.

En el DOM, existen muchos tipos de nodo como Attr, CDataSection, Comment, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, ProcessingInstruction o Text. Generalmente, los programadores web utilizan sobre todo cuatro nodos cuando desean manipular el DOM:

1. *Attr*. Este nodo representa los atributos de las etiquetas, que tienen el formato `atributo=valor`.
2. *Document*. Es el nodo más importante que constituye la raíz del que derivan los demás nodos del DOM.
3. *Element*. Cada etiqueta tendrá un nodo de tipo element. Estos nodos pueden tener atributos y, además, otros nodos podrán derivar de ellos.
4. *Text*. En este nodo, se almacena el texto de una etiqueta.

4.7.1. ¿Cómo se accede a los nodos (elementos HTML)?

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

- 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.

Los nodos se pueden acceder de forma directa, a través de un nodo específico y realizando las operaciones que se deseen sobre él o a través de todos sus antecesores, lo cual es una operación menos usada y más tediosa. 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 identificador que le ha dado el programador al elemento y, generalmente, este identificador es único en la página (debería serlo).

- `getElementsByTagName()`. Acceder a los nodos mediante su tipo de etiqueta. Se accederá a todos los nodos que tengan un tipo de etiqueta concreta y esta llamada devolverá un array de elementos.
- `getElementsByName()`. Acceder a los nodos por su nombre. En HTML5, ya no se utiliza dado que el nombre (name) ha sido reemplazado por su identificador (ID).
- `getElementsByClassName()`. Se accede al elemento por el nombre de la clase (CSS) o selector CSS a la que pertenece. La llamada a este método devuelve una colección (array) de elementos, el cual se podrá ir recorriendo para acceder a todos ellos.
- `querySelector()`. Es otro método del DOM que permite acceder al primer elemento que concuerde con el selector que se le pasa como parámetro. Es un método parecido al anterior.

A continuación, se muestra un ejemplo del método `getElementsByClassName()`:

```
<!DOCTYPE html>
<html>
<body>

<p class="parrafo">Primer p&aacute;rrafo.</p>

<p class="parrafo">Segundo p&aacute;rrafo.</p>

<button onclick="dale()">Cambia el texto</button>

<script>
function dale() {
    var p = document.getElementsByClassName("parrafo");
    p[0].innerHTML = "P&aacute;rrafo primero.";
    p[1].innerHTML = "P&aacute;rrafo segundo.";
}
</script>

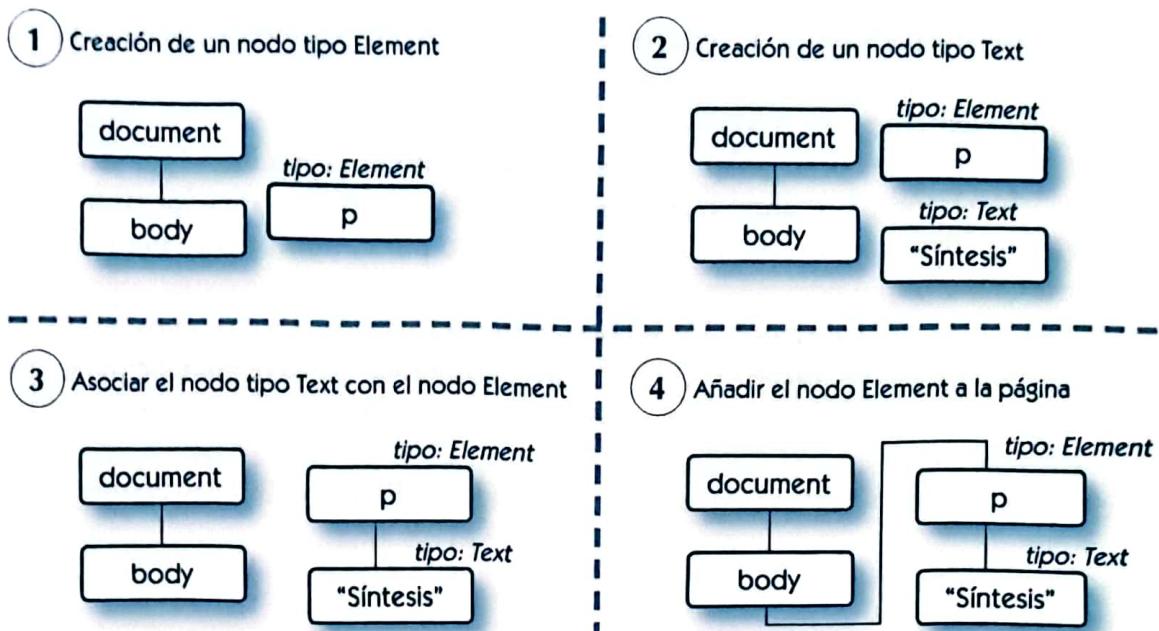
</body>
</html>
```

Como se puede ver en el ejemplo anterior, se accede a todos los elementos del tipo `<p>` y se modifica su contenido.

4.7.2. ¿Cómo se crea un nuevo nodo?

Como se puede ver, los pasos para crear un nuevo nodo son los siguientes:

- *Paso 1.* Crear un nuevo nodo de tipo Element.
- *Paso 2.* Crear un nuevo nodo de tipo Text (para albergar el contenido del elemento anterior).
- *Paso 3.* Vincular el nodo Text al nodo Element.
- *Paso 4.* Vincular el nodo Element a la página de tal manera que el nodo esté en el lugar correspondiente donde se quiera insertar el elemento.

**Figura 4.5**

Proceso gráfico de creación de un nodo tipo párrafo.

El código de los pasos anteriores sería el siguiente:

```
// 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);
```

Como se puede ver en el ejemplo anterior, se han utilizado las siguientes funciones del DOM:

- *createElement()*. Para crear un nodo de tipo Element.
- *createTextNode()*. Para crear un nodo de tipo Text.
- *appendChild()*. Para vincular un nodo hijo a un nodo padre.

4.7.3. ¿Cómo se elimina un nodo?

Imagínese que se tiene un elemento HTML en una web como el siguiente:

```
<p id="editorial">Editorial S&iacute;ntesis</p>
```

Para eliminar dicho elemento, habrá 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 utilizar la propiedad *parentNode*. Véase un ejemplo de código JavaScript para eliminar el párrafo anterior de una página web:

```
var pEditorial = document.getElementById("editorial");
pEditorial.parentNode.removeChild(pEditorial);
```

Como se puede apreciar, eliminar un nodo o elemento de una página web es más sencillo que su creación.

RECUERDA

- ✓ Cuando se elimina un nodo, se eliminarán todos los nodos hijos que pueda poseer. De esa manera, eliminar un nodo puede equivaler a eliminar más de un elemento.

4.7.4. Práctica guiada: creación y eliminación de elementos

En el siguiente ejercicio, se va a mostrar una lista de elementos y dos botones: añadir y eliminar. Pulsando el botón añadir, se podrá dar de alta un nuevo elemento a la lista y, pulsando eliminar, se eliminará el último elemento par.

El aspecto de la página será el que se observa en la figura 4.6.

Pulsa el botón para **añadir** o **eliminar** un elemento a la lista.
1. primer dato
2. segundo dato
3. tercer dato
4. cuarto dato

Figura 4.6
Aspecto de la página.

El código de la página será el siguiente:

```
<!DOCTYPE html>
<html>
<body>

<p>Pulsa el botón para
  <input type="button" value="añadir" onclick="aniade();"/>
  o
  <input type="button" value="eliminar" onclick="elimina();"/>
  un elemento a la lista.
</p>
<ol id="lista">
```

```
<li>primer dato</li>
<li>segundo dato</li>
<li>tercer dato</li>
<li>cuarto dato</li>
</ol>

<script>

function aniade(){

    do{
        var palabras = prompt('Introduce el texto que quieras.');
    }while (palabras == "");

    // Si ha pulsado cancelar no ejecutará el siguiente código
    if (palabras != null){
        var elemento = document.createElement('li');
        var texto = document.createTextNode(palabras);
        var lista = document.getElementById('lista');

        // con esto creamos el nodo li y el texto.
        elemento.appendChild(texto);
        lista.appendChild(elemento);
    }
}

function elimina(){
    var lista = document.getElementById('lista');

    if(lista.children.length > 1){
        var elemento = lista.children[lista.children.length -1];
        //Si es par borro el último.
        if(lista.children.length % 2 == 0){
            elemento.parentNode.removeChild(elemento);
        }else{
            // si no es par tengo que borrar el anterior
            elemento = lista.children[lista.children.length -2];
            elemento.parentNode.removeChild(elemento);
        }
    }else{
        alert("No hay elementos a borrar");
    }
}

</script>
</body>
</html>
```

Como se puede observar en el código, la función que añade elementos a la lista se ha visto en el apartado de la creación de un nodo, pero la de borrado añade una dificultad y es que el nodo por borrar tiene que ser par. Para ello, se utiliza la expresión (`lista.children.length % 2 == 0`). Si se cumple esa expresión, el último nodo de la lista será par y, en caso contrario, habrá que borrar el anterior a dicho nodo.

4.8. Propiedades del DOM

A continuación, se muestran algunas propiedades avanzadas de uso del DOM.

4.8.1. El acceso a los nodos

En apartados anteriores, se ha estudiado cómo se puede acceder a los elementos del DOM. El manejo de los métodos de acceso es importante, puesto que el programador puede, por ejemplo:

1. *Conocer el número de enlaces que tiene una página web.* Para ello, se puede utilizar el método `getElementsByName()` del DOM preguntando por los elementos `<a>`.

```
var numEnlaces = document.getElementsByTagName('a');
```

La línea anterior almacenará todos los enlaces de la página (en un array de objetos), pero, para saber cuántos enlaces hay en dicho array, habrá que utilizar la función `length` ya estudiada anteriormente.

```
var mensaje = 'Se han encontrado '+numEnlaces.length+' enlaces';
```

2. *Conocer la dirección a la que apunta el último enlace.* Bastaría con acceder a la última posición del array y preguntar por la propiedad `href` del objeto contenido en él.

```
var mensaje = 'El penúltimo enlace apunta a '+numEnlaces[numEnlaces.length-1].href;
```

3. *Conocer el número de enlaces que están en el tercer párrafo del documento.* Bastaría con extraer todos los párrafos del documento para, posteriormente, extraer los enlaces.

```
var parrafos = document.getElementsByTagName('p');
enlaces = parrafos[2].getElementsByTagName('a');
numEnlaces = enlaces.length;
```

Ahora, en `numEnlaces`, se almacenará el número de enlaces contenidos en el tercer párrafo.

4.8.2. Capturar ciertos eventos

Para capturar el evento `click` en una página, se le puede asignar al evento `document.onclick` una función creada por el programador. Si se desea saber dónde ha hecho clic el usuario en la página web, habrá que colocar la siguiente línea:

```
document.onclick = hahechoclickaqui;
```

El script hahechoclickaqui será una función que se disparará cuando el usuario haga clic. El código de esta función puede ser el siguiente:

```
function hahechoclickaqui(elEvento){
    var evento = elEvento || window.event;
    .....
}
```



PARA SABER MÁS

¿Qué significa el elEvento || window.event? En los navegadores tipo Internet Explorer, el objeto event se obtiene directamente mediante:

```
var evento = window.event;
```

Por otra parte, en el resto de navegadores, el objeto event se obtiene mágicamente a partir del argumento que el navegador crea automáticamente:

```
function manejadorEventos(elEvento) {
    var evento = elEvento;
}
```

Si se quiere programar una aplicación que funcione correctamente en todos los navegadores, es necesario obtener el objeto event de forma correcta según cada navegador. El siguiente código muestra la forma correcta de obtener el objeto event en cualquier navegador:

```
function manejadorEventos(elEvento) {
    var evento = elEvento || window.event;
}
```

Una vez obtenido el objeto event, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

Por lo tanto, en evento.clientX y evento.clientY, se almacenarán las coordenadas donde el usuario haya hecho clic.

4.9. Modificando el DOM

Ya se ha visto en apartados anteriores cómo modificar la estructura del DOM añadiendo o eliminando nodos, accediendo a ellos y capturando ciertos eventos. En este apartado, se va a trabajar con mayor profundidad la modificación del DOM cambiando propiedades de los elementos como textos, mostrando y ocultando elementos, deshabilitando objetos o comprobando la información introducida en campos de texto.

4.9.1. Cambiar el tamaño del texto

Véase la siguiente página web:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Cambiando el tamaño del texto</title>
</head>
<body>
    <p id="parrafol">Texto a agrandar, reducir o dejar de forma
    original</p>

    <button type="submit" onclick="modificarTexto('aumentar', 0.05);>Agrandar</button>
    <button type="submit" onclick="modificarTexto('reducir', 0.05);>Reducir</button>
    <button type="submit" onclick="modificarTexto('original', 0.05);>Original</button>

<script>
    var tamano = 1;
    var tamanoOriginal = 1;
    function modificarTexto(elEvento, pixel){
        var elemento = document.getElementById('parrafol');
        switch(elEvento){
            case 'aumentar':
                if(tamano > 2){
                    alert('superado el tamaño máximo');
                    break;
                }else{
                    tamano = tamano + pixel;
                    break;
                }
            case 'reducir':
                if(tamano < .9){
                    alert('superado el tamaño mínimo');
                    break;
                }else{
                    tamano = tamano - pixel;
                    break;
                }
            case 'original':
                tamano = tamanoOriginal;
                break;
        }
    }
</script>
```

```

        elemento.style.fontSize = tamaño+'em';
    }

</script>
</body>
</html>

```

El aspecto de esta página web será el que se observa en la figura 4.7.

Texto a agrandar, reducir o dejar de forma original

Agrandar Reducir Original

Figura 4.7

Aspecto de la página web anterior.

Como se puede ver, existen tres botones en ella que pueden agrandar, disminuir o restablecer el tamaño original del párrafo con ID “parrafo1”. En el evento onclick de los botones, se hace una llamada para modificar el párrafo de la página web.

Actividad propuesta 4.2



Partiendo del código expuesto en este apartado, comprueba que funciona en tu navegador e intenta realizar una página web con alguna variante.

4.9.2. Mostrar y ocultar elementos de una página web

Cambiar el estilo es muy útil en muchas aplicaciones JavaScript. A continuación, se presenta un ejemplo más sofisticado en el que se muestra o se oculta un elemento de una página web. Para ello, se han utilizado dos clases en CSS y, mediante JavaScript, se irá cambiando la clase al elemento del DOM:

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ocultando y mostrando texto</title>
    <style>
        .oculto {
            visibility: hidden;
        }

```

```
.visible {
    visibility: visible;
}
</style>
</head>
<body>
    <p id="parrafo1"> Este es el primer párrafo</p>
    <p id="parrafo2"> Este es el segundo párrafo</p>

    <button type="submit" onclick="ocultar('parrafo1');">Ocultar/mostrar</button>
    <button type="submit" onclick="ocultar('parrafo2');">Ocultar/mostrar</button>

<script>
    function ocultar(parrafo){
        var elemento = document.getElementById(parrafo);
        if (elemento.className != 'oculto'){
            elemento.className = 'oculto';
            // también funcionaría la siguiente línea pero se ha preferido modificar el estilo CSS
            //elemento.style.visibility = 'hidden';

        }else{
            elemento.className = 'visible';
            // también funcionaría la siguiente línea pero se ha preferido modificar el estilo CSS
            //elemento.style.visibility = 'visible';
        }
    }
</script>
</body>
</html>
```

De la manera anterior, se puede ocultar y mostrar un elemento de una página web de forma alternativa. Cambiar la clase CSS a cualquier elemento del DOM puede dar al programador muchos recursos para hacer formularios o páginas web más interactivas e intuitivas.



Actividad propuesta 4.3

Basándote en el código expuesto en este apartado, logra que tu programa pueda mostrar u ocultar cuatro párrafos.

4.9.3. Deshabilitar objetos

En ocasiones, los programadores necesitan deshabilitar ciertos elementos de la interfaz dependiendo de la interacción con el usuario. A continuación, se muestra el código de un botón que hace que se desactiven o activen ciertos elementos de la página web (en este caso, otros botones):

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Habilitar y desabilitar elementos del DOM</title>
</head>
<body>
    <input type="submit" value="Boton1">
    <input type="submit" value="Boton2">
    <input type="submit" value="Boton3">
    <input type="submit" value="Boton4">
    <input type="submit" value="Boton5">
    <br/><br/>
    <button type="submit" onclick="bloquear(); id='boton'>Desactivar todos</button>

<script>
    function bloquear(){
        var elementos = document.getElementsByTagName('input');
        if (elementos[0].className != 'desactivo'){
            for (var i = 0; i < elementos.length; i++) {
                elementos[i].disabled = true;
                elementos[i].className = 'desactivo';
            }
            document.getElementById('boton').innerHTML = "Activar todos";
        } else{
            for (var i = 0; i < elementos.length; i++) {
                elementos[i].disabled = false;
                elementos[i].className = 'activo';
            }
            document.getElementById('boton').innerHTML = "Desactivar todos";
        }
    }
</script>
</body>
</html>
```

4.9.4. Comprobar que se introduce información en un campo de texto

Muchas veces, la validación de un campo de texto implica que contenga información. A veces, los usuarios introducen espacios en blanco que hacen que un programa no funcione de forma correcta. Se muestra un código que obliga a que el usuario introduzca algo en un campo de texto con id="campo".

```
valor = document.getElementById("campo").value;
if( valor == null || valor.length == 0 || /^$/.test(valor) ) {
    return false;
}
```

Para que se dé por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que 0 y que no se hayan introducido solo espacios en blanco.

- La palabra reservada *null* es un valor especial que se utiliza para indicar “ningún valor”. Si el valor de una variable es *null*, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.
- La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a 0 caracteres, esto es, que no sea un texto vacío.
- Por último, la tercera parte de la condición que es una expresión regular (*/^\$/.test(valor)*) obliga a que el valor introducido por el usuario no solo esté formado por espacios en blanco. Esta comprobación se basa en el uso de *expresiones regulares*. Por lo tanto, solo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

A continuación, se mostrará cómo se validaría un email. Obsérvese que la expresión regular en este caso es más compleja, puesto que hay que realizar más comprobaciones:

```
valor = document.getElementById("campo").value;
if( !(/[\w+([-.\'])\w+]*@[ \w+([-.\'])\w+]*\.\w+([-.\'])\w+/.test(valor)) ) {
    return false;
}
```

A veces, también el programador tiene que tratar con teléfonos. A continuación, se muestra la expresión regular para validar teléfonos:

```
valor = document.getElementById("campo").value;
if( !(/^\d{9}$/.test(valor)) ) {
    return false;
}
```



Actividad propuesta 4.4

Realiza un formulario en el que se valide una cantidad de euros, el nombre de un usuario –que tendrá que ser mayor a 4 caracteres–, su email y su teléfono.

4.10. BOM

El BOM (Browser Object Model) en su momento fue implementado por los navegadores para que JavaScript pudiese hacer uso de sus métodos y propiedades de forma uniforme.

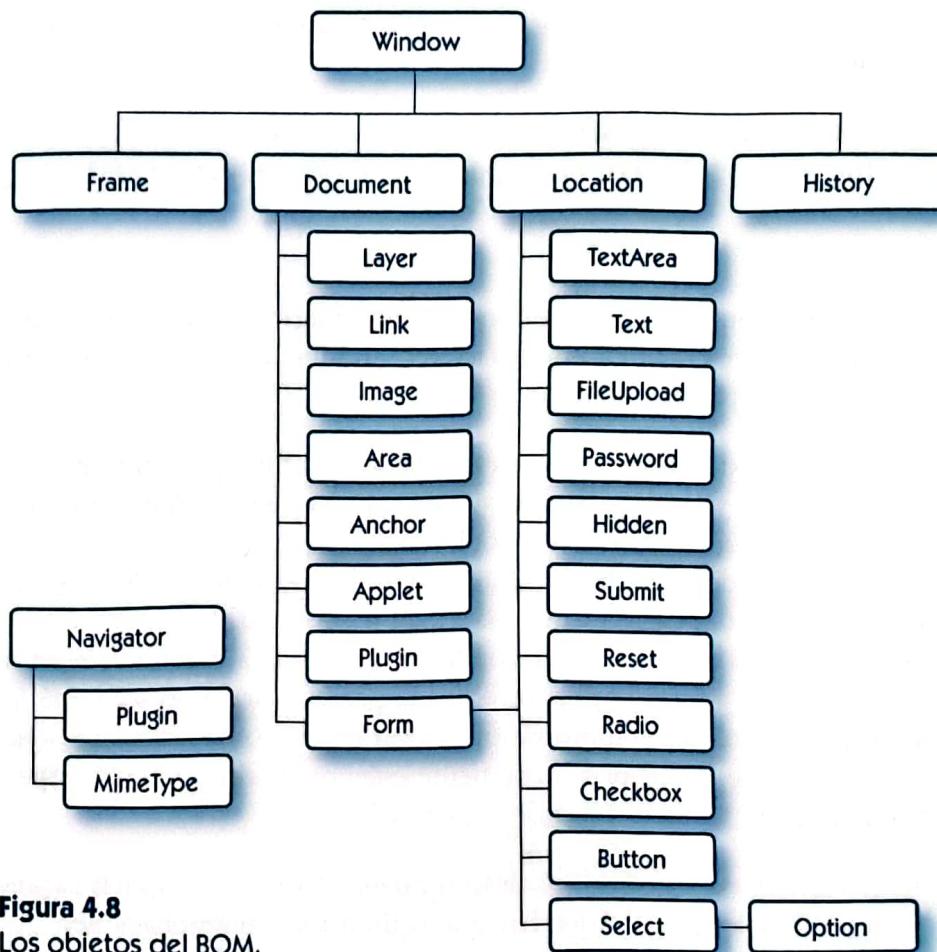


Figura 4.8
Los objetos del BOM.

Como se puede observar en la figura 4.8, el objeto raíz del BOM es el objeto window, que está soportado por cualquier navegador y que, a su vez, contiene otros objetos de menor rango jerárquico, como puede ser el objeto document (muy importante también) u otros.

Las diferencias entre BOM y DOM serían las siguientes:

- Con el DOM, JavaScript 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 como puede ser la URL o las dimensiones de la ventana (navegador) ni tampoco se puede cerrar o redimensionar la ventana del navegador, gestionar cookies, etc. Para ello, se utiliza el BOM, que es otra estructura arborescente similar al DOM y algo más completa al añadirsele otra serie de objetos.
- A diferencia del DOM, el elemento raíz es el objeto window.

Por lo tanto, podrían ejecutarse las siguientes líneas de código:

```
h = window.document.getElementById("header");
```

O bien:

```
h = document.getElementById("header");
```

Y el resultado de ambas líneas sería el mismo.

FUNDAMENTAL

El objeto document en el BOM y en el DOM

Como se puede observar, en el BOM existe un objeto document al igual que en el DOM. En el DOM, el objeto document es la raíz del árbol de objetos de la página web, mientras que en el BOM se puede acceder a propiedades como:

- `document.URL`. Contiene la URL de la página web actual.
- `document.referrer`. Contiene la URL desde que se accedió a la página actual.
- `document.title`. Se accede al texto de la etiqueta `<title>` de la página web.
- `document.lastModified`. Contiene la fecha de última modificación de la página web.

Véase un ejemplo de script que muestra el ancho y alto del navegador. Pruebe a recargar la página y se dará cuenta que las dimensiones van cambiando.

```
<!DOCTYPE html>
<html>
<body>
<p id="dimension"></p>
<script>
var ancho = window.innerWidth;
var alto = window.innerHeight;
document.getElementById("dimension").innerHTML = "Ancho: " + ancho + "
--- Alto: " + alto ;
</script>
</body>
</html>
```

TOMA NOTA



El navegador Internet Explorer suele ser un poco diferente a otros navegadores como Firefox, Safari u Opera. Siempre que utilice objetos del BOM, compruebe que su página web funciona en Explorer y en los demás navegadores.

4.10.1. El objeto window

Como se ha visto, el objeto window representa la ventana del navegador en la que se muestra un documento del DOM. El objeto window controla, entre otras cuestiones, las dimensiones de la ventana, las barras laterales de desplazamiento, la barra de estado, etc.

Actualmente, como los navegadores tienen varias pestañas, cada una de ellas tendrá un objeto window asociado. No obstante, métodos como window.resizeTo o window.resizeBy, como es de entender, se aplicarán a la ventana de la aplicación y no a la pestaña específica.



Recurso web

El objeto window tiene una infinidad de propiedades y métodos. Si se desea conocer el listado completo de propiedades y métodos, se puede acceder al siguiente recurso de MDN a través del código QR adjunto:



4.10.2. El objeto location

Un objeto muy útil del DOM es el objeto window.location. Véanse algunas de sus propiedades:

- *window.location.href*. Devuelve la URL de la página actual.
- *window.location.hostname*. Devuelve el dominio de la web.
- *window.location.pathname*. Devuelve el path y el fichero de la página actual.
- *window.location.protocol*. Devuelve el protocolo utilizado que debería ser o http: o https: (o file: si se está ejecutando una página desde el sistema de archivos local).
- *Método window.location.assign() o location.assign()*. Carga una nueva página web. Véase un ejemplo:

```
window.location.assign("http://www.sintesis.com");
```

El código anterior cargará en la misma página la web que se pasa como parámetro. En este caso, <http://www.sintesis.com>.

4.10.3. El objeto history

Es sabido que los navegadores tienen dos botones: uno que carga la web anterior (muy útil, por cierto) y otro que carga la siguiente web que se ha tecleado a la actual (la mayoría de las veces, el usuario está en la más reciente, salvo que vaya navegando hacia atrás).

Desde JavaScript, puede manejarse la historia del navegador con los métodos del objeto history:

- *history.back()*. Cargará la página web visitada anterior a la actual.
- *history.forward()*. Cargará la página web visitada siguiente a la actual.

4.10.4. El objeto navigator

El objeto navigator tiene las siguientes propiedades:

- *navigator.appName*.

- navigator.appCodeName.
- navigator.platform.

Las dos primeras contienen la aplicación que se está utilizando (por ejemplo, appName=Netscape y appCodeName=Mozilla) y en el tercero, la plataforma o sistema operativo utilizado (por ejemplo, Linux x86_64).



SABÍAS QUE...

¿Es posible pedir información al usuario de forma interactiva? No es normal, pero sí sería posible. Observe el siguiente ejemplo:

```
<!DOCTYPE html>
<html>
<body>
<p id="cp"></p>
<script>
var codigopostal = prompt("Por favor introduzca su código postal", "29680");
if (codigopostal != null) {
    document.getElementById("cp").innerHTML = "Su código postal es: " + codigopostal ;
}
</script>
</body>
</html>
```

En el ejemplo anterior, se pide de forma interactiva un código postal y se muestra en un párrafo en la página web.



Actividad propuesta 4.5

Basándote en el código expuesto en este apartado, crea un programa que pida un código postal y una población actualizando los campos correspondientes de la página web.

Resumen

- Para validar formularios en JavaScript, se pueden utilizar las sentencias try, throw y catch como en Java.
- Las expresiones regulares se utilizan ampliamente en la validación de campos por su potencia y efectividad.

- Las expresiones regulares son objetos que se pueden utilizar en los métodos `exec` y `test` del objeto `RegExp` y también en los métodos `search`, `replace` y `match` del objeto `string`.
- La sintaxis de una expresión regular es la siguiente:

/patrón/modificadores

- El DOM es una plataforma e interfaz de un documento que permite a los scripts y programas acceder y modificar dinámicamente su contenido, estructura y estilo.
- Cuando se carga un documento en el navegador, se crea el objeto `document`, el cual, además de ser la raíz de todo el documento HTML, tiene una serie muy amplia de propiedades y métodos.
- En el DOM, además de buscar elementos por el ID, se pueden buscar elementos por el tag [utilizando el método `getElementsByName()`] o también por la clase [utilizando el método `getElementsByClassName()`].
- El método `querySelector()` del objeto `document` devuelve el primer elemento que encuentre y que concuerde con el selector que se le introduzca como parámetro.
- Para seleccionar todos los elementos que concuerden con un selector, habrá que utilizar el método `querySelectorAll()`.
- 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.
- Los cuatro nodos más utilizados para manipular el DOM son `attr`, `document`, `element` y `text`.
- Las funciones para acceder a los nodos son:
 - `getElementById()`.
 - `getElementsByTagName()`.
 - `.getElementsByName()`.
 - `getElementsByClassName()`.
 - `querySelector()`.
- Los pasos que hay que seguir al crear un nodo son:
 - Paso 1. Crear un nuevo nodo de tipo `Element`.
 - Paso 2. Crear un nuevo nodo de tipo `Text` (para albergar el contenido del elemento anterior).
 - Paso 3. Vincular el nodo `Text` al nodo `Element`.
 - Paso 4. Vincular el nodo `Element` a la página de tal manera que el nodo esté en el lugar correspondiente donde se quiera insertar el elemento.
- El BOM es una estructura arborescente similar al DOM y algo más completa al añadirse otra serie de objetos.
- A diferencia del DOM, en el BOM, el elemento raíz es el objeto `window`.
- Algunos objetos importantes en el BOM son `window`, `location`, `history` y `navigator`.

Ejercicios propuestos



1. En el apartado 4.9.1, se muestra cómo modificar el tamaño de un párrafo. El problema es que solamente funciona para ese párrafo concreto. Añade un tercer parámetro a la función para que pueda modificarse el párrafo que se quiera de la web introduciendo su ID. Para comprobar si el script funciona correctamente, deberás crear más de un párrafo.
2. Crea una página web con dos campos de texto (nombre y apellidos) y un botón que valide que ambos contienen información.
3. Mejora el formulario con un campo de email que también sea validado por el botón anterior.
4. Desarrolla el formulario con un campo de teléfono que también sea validado por el botón anterior.
5. Realiza una página web con un script que verifique qué hace el código siguiente y explica su funcionalidad.

```
var str = "El sistema operativo más seguro es Android";
var res = str.replace(/android/i, "Linux");
```

6. Investiga qué hace el código siguiente y crea una página web para realizar la comprobación.

```
var str = "Aprende javaScript de 10 en Síntesis";
var m = str.search(/[S|t]/i);
console.log(m);
var n = str.search(/[0-9]/i);
console.log(n);
var a = str.search(/\d+/i);
console.log(a);
```

7. Elabora un test interactivo para verificar si un alumno está sufriendo acosos escolar.
8. Lleva a cabo un script que, cuando pase el ratón sobre un campo de texto, lo cambie a mayúsculas.
9. Diseña una página web que capture el evento de redimensionar la ventana por parte del usuario.
10. Concibe una página web con un botón que capture los eventos click, mouseover y mouseout.

El método removeEventListener es capaz de eliminar un evento asociado a un elemento del documento. Añade un segundo botón que vaya eliminando uno a uno los eventos asociados.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué función tiene el patrón /\d/g?:

- a) Selecciona los dígitos de una cadena de caracteres.
- b) Selecciona los datos de una cadena de caracteres.
- c) Selecciona los no caracteres de una cadena de caracteres.

2. ¿Qué hace el patrón /S/g?:

- a) Selecciona los no caracteres de una cadena de caracteres.
- b) Elimina los espacios de una cadena de caracteres.
- c) Busca si hay una secuencia de dos dígitos.

3. ¿Qué busca el patrón /\d{2,3}/g?:

- a) Busca si existe el 2 o el 3 en la cadena.
- b) Busca si existe el número 2 tres veces en la cadena.
- c) Busca si hay una secuencia de dos o tres dígitos.

4. Entre otros, el objeto RegExp tiene los métodos:

- a) pattern() y evaluateRegularExpression().
- b) exec() y test().
- c) pattern() y testPattern().

5. Entre otros, el objeto document tiene los métodos:

- a) openDocument(), querySelector(), getElementById(), etc.
- b) open(), querySelector(), getElementById(), etc.
- c) openDocument(), queryDOM(), getElementById(), etc.

6. ¿Qué propiedad se puede usar para saber si las cookies están habilitadas?:

- a) window.cookieEnabled.
- b) document.cookieEnabled.
- c) navigator.cookieEnabled.

7. ¿Cuál de las siguientes líneas de código es correcta?:

- a) document.getElementById("unboton").addEventListener("MuestraFecha","click");
- b) document.getElementById("unboton").addEventListener("click","MuestraFecha");
- c) document.getElementById("unboton").addEventListener("click", MuestraFecha);

8. ¿Cuáles son los nodos del DOM?:

- a) Attr, Document, Element y Text.
- b) Attr, window, Element y Text.
- c) Attr, location, history y Text.

9. Para acceder al selector de la clase CSS a la que pertenece un elemento, ¿cuál de las siguientes opciones se utiliza?:

- a) getElementsByTagName().
- b) getElementsByClassName().
- c) getElementsByClassName().

10. De las sentencias siguientes, ¿cuál se ejecuta para eliminar un nodo?:

- a) `elemento.parentNode.removeChild(elemento);`
- b) `elemento.removeNode(elemento);`
- c) `elemento.currentNode.removeDOMNode(elemento);`

SOLUCIONES:

1. a b c

2. a b c

3. a b c

4. a b c

5. a b c

6. a b c

7. a b c

8. a b c

9. a b c

10. a b c