1. SEPARAR ESTRUCTURA Y COMPORTAMIENTO

Hemos visto cómo podíamos acceder a las propiedades de los objetos, cómo utilizar sus métodos y cómo escribir código para responder a los eventos.

Bien, ahora es el momento de decirte que la forma de escribir el código que has visto hasta ahora no es la más adecuada de hacerlo.

Se podría decir que hemos visto el método *"de la vieja escuela"*, es decir, la forma en que se escribía código JavaScript hace unos años.

Aunque esta forma sigue siendo perfectamente válida y todos los navegadores entenderán ese tipo de código, la verdad es que presenta algunas deficiencias que ahora que ya tienes un poco más de experiencia vamos a corregir.

Y empezamos con ver cómo separar la estructura de la página web, es decir, su código HTML, del comportamiento o código JavaScript.

Así pues, aunque hemos visto que podemos incluir el código JavaScript junto al código HTML, no es la mejor forma de hacerlo.

Tal como ocurre con las hojas de estilo en cascada CSS, lo mejor es escribir el código JavaScript en **archivos independientes** y enlazar dicho código con los objetos del documento HTML.

Así pues, en el código HTML incluiremos una etiqueta **`<script>`** con el atributo **src**, que indicará la ubicación del archivo de JavaScript:

```
<script type="text/javascript" src="dom.js"></script>
```

Aunque tradicionalmente los script se incluían en la cabecera del documento HTML, ahora se recomienda hacerlo al final de la página, justo antes de la etiqueta de cierre **</body>**.

Esto permite que la carga de la página sea más rápida, ya que el navegador no tiene que esperar a cargar e interpretar el código JavaScript, para continuar con el resto de la página.

Por su parte, el código JavaScript se incluirá en un archivo cuyo nombre debe incluir la extensión **.js**. Habitualmente, estos archivos se guardan en una carpeta específica del sitio web. Por ejemplo, la carpeta **scripts**.

En este archivo no debe incluirse la etiqueta `<script>`, únicamente el código JavaScript:

```
// Archivo dom.js
function ContarElementos()
{
   var result = 0;
   var nodos = document.body.childNodes;
   for (var i = 0; i < nodos.length; i++)
   {
      if (nodos[i].nodeType == 1)
         result++;
   }
   return result;
}
function CambiarTexto()
{
   var elemento_em =
     document.getElementsByTagName("em")[0];
   elemento_em.firstChild.nodeValue ="en comunicación";
}
function NuevoTexto()
{
   var nuevo_elemento = document.createElement("p");
   var  nuevo_texto  =  document.createTextNode("Si  no
recibe respuesta tras 2 días, por favor, vuelva a enviar
la consulta.");
   nuevo_elemento.appendChild(nuevo_texto);
   document.body.appendChild(nuevo_elemento);
}
function NuevaTabla()
{
   var codigo = "<table>";
   codigo += "<tr><th>Título de la primera columna</th>"
   codigo += "<th>Título de la segunda columna</th>"
   codigo    +=    "<th>Título    de    la    tercera
columna</th></tr>"
   codigo    +=    "<tr><td>Segunda    fila,    primera
columna</td>";
   codigo += "<td>Segunda fila, segunda columna</td>";
   codigo     +=     "<td>Segunda     fila,     tercera
columna</td></tr>";
   codigo    +=    "<tr><td>Tercera    fila,    primera
columna</td>";
   codigo += "<td>Tercera fila, segunda columna</td>";
   codigo     +=     "<td>Tercera     fila,     tercera
columna</td></tr>";
   codigo += "</table>";

   var nuevo_elemento = document.createElement("div");

   nuevo_elemento.style.fontFamily = "Verdana, sans-serif";
   nuevo_elemento.style.color = "red";
   nuevo_elemento.innerHTML = codigo;
   document.body.appendChild(nuevo_elemento);
}
```

Además, en este mismo archivo **.js** incluiremos el código para registrar los manejadores de evento, eliminándolos de las etiquetas HTML de la página web.

Así pues, eliminaremos cualquier trozo de código `onclick="…"`, etc. de las etiqueta HTML y escribiremos dicho código en el archivo de JavaScript aunque de otra forma.

Con estos cambios separamos el código HTML y el código JavaScript. Esto es más importante de lo que parece ya que la página web debería ser completamente funcional sin el código JavaScript.

Piensa en lo que ocurre con las hojas de estilo CSS. El contenido de la página sigue siendo válido aunque no se puedan cargar. Eso sí, con un aspecto muy distinto al que conseguimos con una buena hoja de estilo. Pues lo mismo debe ocurrir con el código JavaScript.

Debemos pensar que si el navegador del usuario no puede ejecutar el código JavaScript, la página debe seguir siendo básicamente útil aunque no se consiga toda la funcionalidad que proporciona JavaScript.

2. AÑADIR LOS MANEJADORES DE EVENTO

Aunque hemos incluido la referencia al archivo **dom.js** en el documento HTML, realmente el código JavaScript incluido en él no es más que la definición de una serie de funciones.

Sabemos que la definición de una función no implica ninguna ejecución de su código y, además, hemos eliminado los manejadores de evento que se incluían en las etiquetas HTML **<body>** y **** de la página original.

Bien, la forma de hacerlo correctamente es incluir los manejadores de evento en el propio archivo **.js:**

```
window.onload = NuevaTabla;
```

En este caso estamos utilizando el manejador de evento **onload** del objeto **window**, lo que es equivalente a escribir el manejador **onload** de la etiqueta **<body>** del documento HTML.

Es decir, que indicamos que se tiene que ejecutar la función **NuevaTabla** cuando la página está completamente cargada.

Fíjate que no se incluyen los paréntesis ni se puede especificar ningún parámetro, solo el nombre de la función. En caso contrario, estarías realizando la llamada a la función y esta se ejecutaría.

Otra forma de incluir un manejador de evento en JavaScript es utilizar el método **addEventListener** sobre el objeto en el que queremos registrar el manejador de evento.

Este método requiere de tres parámetros:

addEventListener(evento, función, capture/bubble)

En nuestro ejemplo podría ser:

```
window.addEventListener("load", NuevaTabla, false);
```

En el primer parámetro se indica el nombre del evento (no el nombre del manejador del evento, por lo que no se incluye el prefijo **on**); en el segundo, el nombre de la función; y en el tercero, si el evento especifica qué objeto es el responsable de manejarlo.

Así, si el tercer parámetro tiene valor **true**, entonces el evento es capturado primero por el objeto **window** y después se propaga hacia el objeto destinatario del mismo.

Esto puede tener sentido. Por ejemplo, si se trata del evento correspondiente al click de un botón, ¿quién debe manejar el evento?

¿El botón sobre el que se ha hecho clic o la propia ventana del navegador? Al fin y al cabo, también se está pulsando en la superficie de la ventana.

Con el valor **true** en el tercer parámetro de **addEventListener**, el evento sería manejado primero por la ventana y después se propagaría hasta llegar al botón.

Si el tercer parámetro tiene el valor **false** (valor predeterminado), entonces ocurre justo al contrario. El evento primero es manejado por el botón y después sube hasta que llega al objeto **window**.

Esta forma de añadir manejadores de evento es mejor, ya que podemos incluir más de una función para el mismo evento utilizando varias veces el método **addEventListener**.

Lo mismo podemos hacer para el segundo evento que teníamos en la página:

```
document.getElementsByTagName("em")[0].onclick = CambiarTexto;
```

En este segundo caso, seguramente habría sido mejor incluir un identificador para el elemento "**em**" y utilizar el método **getElementById** para seleccionarlo.

También podríamos haber optado por esta otra notación:

```
document.getElementsByTagName("em")[0].addEventListener(
"click", CambiarTexto, false);
```

Por último, comentarte algo muy importante: debemos comprobar que la página se ha cargado completamente antes de registrar los manejadores de evento.

Esto significa que el código asociado con el evento **click** que acabamos de escribir, realmente debería estar asociado con el evento **load** del documento, ya que es la única forma de asegurarse de que la página se ha cargado completamente y que, por lo tanto, existe el elemento "**em**".

Por ejemplo, podríamos ejecutar más de una instrucción asociándola con el evento **load**. Fíjate cómo se hace:

```
window.onload = function () {
 NuevaTabla();
 document.getElementsByTagName("em")[0].onclick = CambiarTexto;
}
```

En este caso se utiliza lo que se conoce como una "**función anónima**", ya que no se incluye el nombre de la misma.

Esto es lo correcto porque el registro del evento **click** del elemento "**em**" se realiza cuando seguro que ya se ha cargado la página completamente.

3. DETECTAR CARACTERÍSTICAS DE LOS NAVEGADORES

Otro aspecto importante a la hora de trabajar con la funcionalidad del DOM es comprobar que realmente el navegador (o agente de usuario) que utiliza el usuario sabe interpretar el código correctamente.

Esto es importante, ya que incluso con los navegadores modernos, el usuario podría haber desactivado la ejecución del código JavaScript.

En ese caso, la página debe seguir funcionando sin JavaScript, aunque lo haga con menor funcionalidad.

Por otra parte, es posible que distintas versiones de un mismo navegador acepten unas u otras características del DOM.

Por todo ello, podemos detectar las características que admite un navegador y actuar en consecuencia.

Tradicionalmente, los desarrolladores comprobaban el nombre del navegador utilizando principalmente la propiedad **userAgent** del objeto **navigator**.

Sin embargo, esto no es una buena idea actualmente, ya que no es una característica fiable del navegador. Algunos navegadores enmascaran o cambian esta propiedad para que sus usuarios puedan acceder a páginas diseñadas exclusivamente para otro navegador e incluso también lo puede hacer el usuario.

Lo correcto actualmente es detectar las características u objetos del DOM que acepta el navegador del usuario. Fíjate cómo lo hacemos (el código de detección aparece en negrita):

```
window.onload = function () {
 if (!document.getElementById ||
     !document.getElementsByTagName) return false;
 NuevaTabla();
 document.getElementsByTagName("em")[0].onclick = CambiarTexto;
}
```

¿Qué estamos haciendo aquí? Pues simplemente detectando si el navegador entiende los métodos **getElementById** y **getElements–ByTagName**.

Si no es el caso, entonces salimos de la función devolviendo el valor **false**, por lo que no se ejecuta el código siguiente.

De esta forma, si el navegador no es compatible con alguna característica del DOM que estamos utilizando, simplemente no la ejecutará.

Podremos comprobar esto con cada propiedad o método del DOM que queremos verificar. Aquí lo importante es darse cuenta de que en el caso de un método no debemos incluir los paréntesis (ni parámetros), solo su nombre.

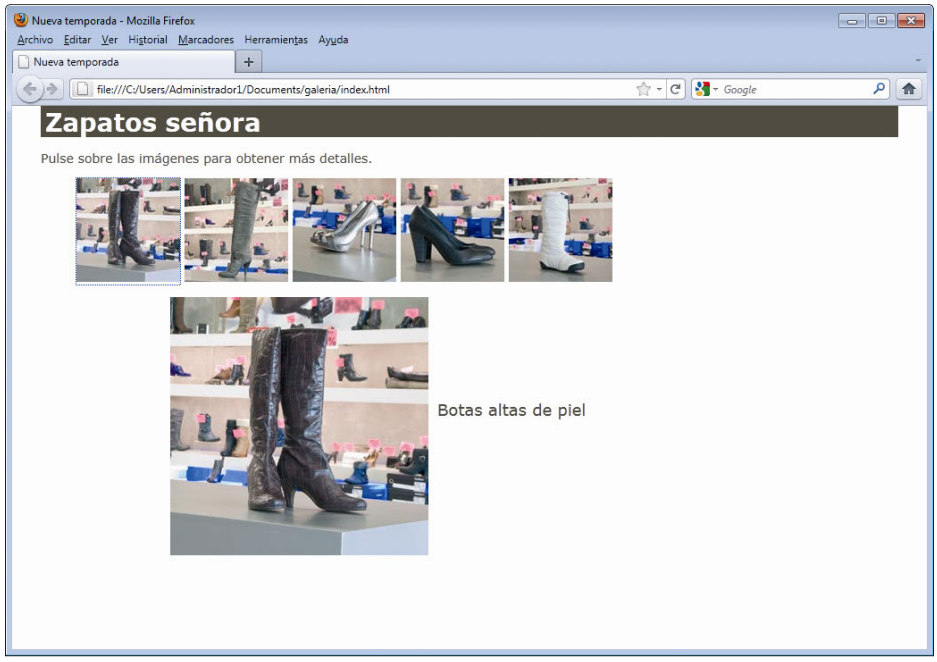
Ocurre lo mismo que antes. Si incluimos los paréntesis y/o parámetros, estaremos ejecutando realmente el método y esto no es lo que queremos ahora.

En los siguientes capítulos estudiaremos una página web un poco más compleja donde se han aplicado muchos de los conceptos que hemos ido viendo al estudiar el DOM.

4. EJEMPLO: UNA GALERÍA FOTOGRÁFICA

Para afianzar los conceptos y prácticas que hemos ido viendo durante las últimas lecciones, vamos a presentar un ejemplo sencillo pero útil.

Se trata de una *"galería fotográfica"*. El usuario accederá a la página web que puedes ver en la figura adjunta para visualizar algunas fotografías. Comprobaremos cómo podemos mejorar la experiencia del usuario con la tecnología DOM y JavaScript.



En la página encontramos una serie de fotografías "en miniatura", representando productos (en este caso zapatos de señora) de una tienda virtual.

La idea es que si un usuario desea obtener detalles adicionales sobre cualquiera de esos productos, simplemente tiene que pulsar en su imagen. Al hacerlo, se muestra una imagen más grande del producto y un pequeño texto descriptivo.

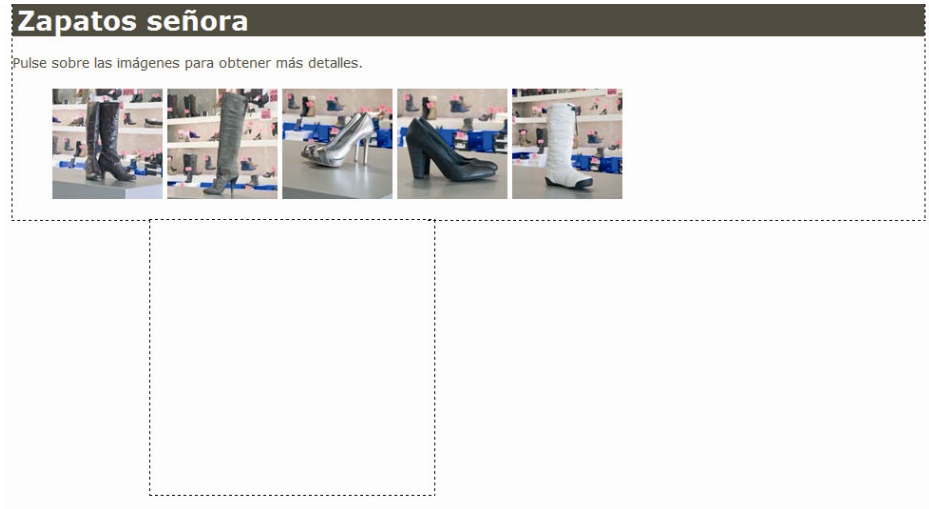
Fíjate que aunque no parece que la página sea demasiado compleja, realmente intervienen varios conceptos que se han estudiado: manejo de eventos, selección de elementos de la página y de sus atributos, actualización del contenido, creación de nuevos elementos, etc.

Además, veremos cómo se aplican las buenas prácticas que hemos visto durante esta lección.

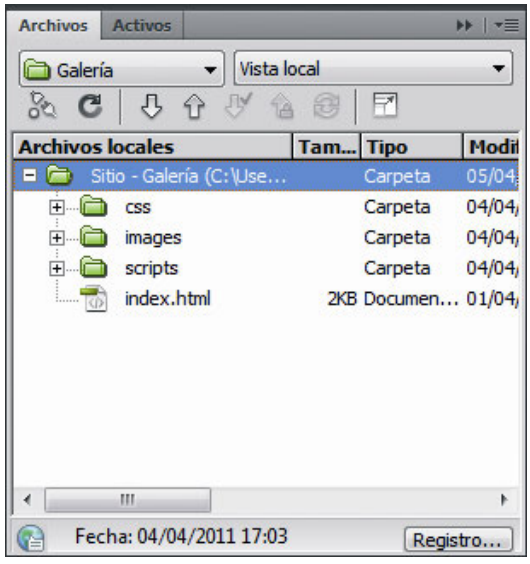
¡Vamos allá!

5. EJEMPLO: LA ESTRUCTURA DE LA PÁGINA

En cuanto a la estructura de la página web, podemos ver que en la página aparece una serie de fotografías en su parte superior y un recuadro en la parte inferior.



Este recuadro inferior es realmente otra imagen, que simplemente muestra un fondo blanco para que se confunda con el fondo de la página web. Sirve como marcador o lugar donde se muestran las imágenes grandes cuando se pulsa en alguna de las imágenes superiores.



Aquí vemos la distribución en carpetas del sitio web al que pertenece esta página. Fíjate que hay una carpeta para las imágenes, para las hojas de estilo y para los scripts.

Existen dos versiones de cada imagen: una más pequeña y la de tamaño normal.

También aparece la imagen **blanco.gif**, que es la imagen que se muestra inicialmente en el marcador de la parte inferior de la página web.

La imágenes más pequeñas aparecen en la subcarpeta **images/thumbnails**. En los dos casos tienen el mismo nombre, por lo que se diferencian por la ubicación en la que están almacenadas.

Ahora se adjunta el código fuente del documento HTML.

Archivo **index.html**:

```
<!DOCTYPE html>
<html>
<head>
<title>Nueva temporada</title>
<link rel="stylesheet" type="text/css"
href="css/estilo.css">
</head>
<body>
<div id="container">
<h1>Zapatos señora</h1>
<p>Pulse sobre las imágenes para obtener más
detalles.</p>
<ul>
<li>
<a href="images/zapatos01.jpg"><img id="modelo1"
src="images/thumbnails/zapatos01.jpg" alt="Botas altas
de piel" width="116" height="116"></a>
</li>
<li>
<a href="images/zapatos02.jpg"><img id="modelo2"
src="images/thumbnails/zapatos02.jpg" alt="Botas de
tacón alto" width="116" height="116"></a>
</li>
<li>
<a href="images/zapatos03.jpg"><img id="modelo3"
src="images/thumbnails/zapatos03.jpg" alt="Zapatos de
fiesta color plata" width="116" height="116"></a>
</li>
<li>
<a href="images/zapatos04.jpg"><img id="modelo4"
src="images/thumbnails/zapatos04.jpg" alt="Zapatos
negros para diario" width="116" height="116"></a>
</li>
<li>
<a href="images/zapatos05.jpg"><img id="modelo5"
src="images/thumbnails/zapatos05.jpg" alt="Botas blancas
sin tacón" width="116" height="116"></a>
</li>
</ul>
<img class="imagenGrande" id="actual"
src="images/blanco.gif" alt="Modelo visualizado"
width="289" height="289">
</div>
<script type="text/javascript"
src="scripts/funciones.js"></script>
</body>
</html>
```

Respecto de este código nos interesa comentar algunas cosas:

- El aspecto o formato de la página se consigue a través de una hoja de estilo en cascada externa. Fíjate que en la línea siguiente aparece el vínculo con el archivo **estilo.css** situado en la subcarpeta **css**:

```
<link rel="stylesheet" type="text/css"
href="css/estilo.css">
```

- Cada imagen tiene su correspondiente identificador y texto alternativo. Estos dos atributos son muy importantes ya que el atributo "**id**" permitirá seleccionar la imagen y el atributo "**alt**" proporcionará el texto descriptivo que se utilizará cuando se ha pulsado en ella.

- Además, vemos que rodeando a cada imagen aparece un hipervínculo, cuyo atributo **href** tiene el valor de la ubicación de cada una de las imágenes "grandes".

- En la siguiente línea encontramos el marcador de imagen o recuadro de la parte inferior donde se muestran las imágenes grandes. Fíjate que tiene el identificador "**actual**".

```
<img class="imagenGrande" id="actual"
src="images/blanco.gif" alt="Modelo visualizado"
width="289" height="289">
```

- Finalmente, en la línea siguiente encontramos la etiqueta **script**, que enlaza la página web con el archivo externo de JavaScript **funciones.js**, que está almacenado en la subcarpeta **scripts**.

```
<script type="text/javascript"
src="scripts/funciones.js"></script>
```

  Fíjate que hemos incluido el script justo al final del cuerpo del documento y no en la sección de cabecera.

  Además, no vemos ningún código JavaScript para manejar los eventos, ya que se encuentra en el archivo **.js**.

Como puedes ver, la estructura de la página web es sencilla, pero se sigue la buena práctica de separar la estructura, el formato y el comportamiento utilizando archivos distintos para cada una de esas tareas.

6. EJEMPLO: EL CÓDIGO JAVASCRIPT

Y ahora veamos el código JavaScript, donde trabajaremos con el DOM del documento.

Archivo **funciones.js**:

```
function MostrarFoto(elemento)
{
  var source = elemento.firstChild.getAttribute("src");
  source = source.replace("thumbnails/", "");
  document.getElementById("actual").setAttribute("src",
                                                  source);
  Describir(elemento.firstChild.getAttribute("id"));
}

function Describir(id)
{
   var texto =
     document.getElementById(id).getAttribute("alt");
   var objeto = document.getElementById("descripcion");
   if (!objeto)
   {
      var newText = document.createTextNode(texto);
      var newElement = document.createElement("p");
      newElement.setAttribute("id", "descripcion");
      newElement.setAttribute("class", "textoDescripcion");
      newElement.appendChild(newText);
      document.body.appendChild(newElement);
   }
   else
   {
      objeto.firstChild.nodeValue = texto;
   }
}


function PrepararHipervinculos()
{
   if (!document.getElementById ||
       !document.getElementsByTagName) return false;
   var hipervinculos =
           document.getElementsByTagName("a");
   for (var i = 0; i < hipervinculos.length; i++)
   {
      hipervinculos[i].onclick = function (){
         MostrarFoto(this);
         return false;
       }
   }
}

window.onload = PrepararHipervinculos;
```

Aquí encontramos algunas funciones que son las que hacen que la página funcione como hemos indicado.

Todo este código se ejecuta al cargar la página. De esta forma, nos aseguramos de que los elementos existen en el DOM del documento.

Así pues, en la línea siguiente vemos que se escribe el manejador del evento **load** del objeto **window**. Para este manejador se ha asociado la función **PrepararHipervinculos**:

```
window.onload = PrepararHipervinculos;
```

En la función **PrepararHipervinculos** lo primero que se hace es comprobar las características del DOM que soporta el navegador. Fíjate cómo se comprueba esto:

```
if (!document.getElementById ||
    !document.getElementsByTagName) return false;
```

Si el navegador no es capaz de ejecutar los métodos **getElementById** o **getElementsByTagName**, entonces simplemente no se sigue con el resto del código JavaScript.

Seguidamente, se utiliza el método **getElementsByTagName** para seleccionar los hipervínculos de la página. Así pues, la variable **hipervinculos** es un array con dicho conjunto.

Para cada uno de estos hipervínculos se crea su correspondiente manejador del evento **click**. Para ello, se recorre el array hipervinculos y se asocia una función anónima con el manejador **onclick**.

La función anónima consta de una llamada a la función **MostrarFoto**, que envía como parámetro el propio hipervínculo (fíjate que se utiliza la palabra clave **this**) y se devuelve el valor **false**.

Esto último es necesario, ya que debemos evitar que se produzca la acción predeterminada del hipervínculo.

Por defecto, cuando se pulsa en un hipervínculo se accede a la URL indicada en su atributo **href**. En este caso esto no nos interesa, ya que lo que realmente queremos que ocurra es que se ejecute el código JavaScript asociado con el evento **click**.

Para evitar la acción predeterminada del hipervínculo, simplemente se devuelve el valor **false**.

Veamos pues, qué ocurre al pulsar en cada hipervínculo. Es decir, estudiemos el código de la función **MostrarFoto**.

Primero se guarda el valor del atributo **src** de la imagen sobre la que se ha pulsado. Fíjate cómo conseguimos este valor.

Como la función recibe como parámetro el hipervínculo, utilizamos el método **firstChild** para acceder a la imagen relacionada. Seguidamente, con **getAttribute** podemos conseguir el valor de su atributo **src**:

```
var source = elemento.firstChild.getAttribute("src");
```

A continuación establecemos el valor de la imagen grande. Para ello, se está utilizando el método **replace** de una cadena de texto o String. Con este método simplemente eliminamos la subcadena "**thumbnails/**". De esta forma obtendremos la ubicación de la imagen grande correspondiente.

Por ejemplo, si se ha pulsado en la primera imagen, tendremos que su atributo **src** tiene este valor:

.../galería/images/thumbnails/zapatos01.jpg

Por lo que tras esta línea tendrá este otro valor:

.../galería/images/zapatos01.jpg

De esta forma obtenemos la ubicación de la imagen grande correspondiente.

Después se utiliza dicha ubicación para establecer el valor del atributo **src** de la imagen de identificador "**actual**", es decir, del marcador de la imagen grande.

Fíjate cómo se utilizan para ello los métodos **getElementById** y **setAttribute**:

```
document.getElementById("actual").setAttribute("src",
                                                source);
```

Con esto habremos conseguido que, al pulsar en una de las imágenes, se muestre la correspondiente imagen grande en la parte inferior de la página.

Nos faltaría ver cómo añadir el texto descriptivo. Para ello, no solo tenemos que modificar el DOM como hemos hecho hasta este momento, sino que tenemos que crear nuevo contenido

Este contenido se crea a través de la función **Describir**.

En la función **Describir** utilizamos los métodos del DOM para crear nodos y añadirlos al árbol del documento.

El texto en sí se recoge a partir del atributo **alt** de la imagen. Fíjate:

```
var texto =
  document.getElementById(id).getAttribute("alt");
```

Seguidamente seleccionamos el elemento cuyo identificador es "**descripcion**".

Este elemento es justamente el que estamos creando con la función **Describir**. Por ello, las siguientes líneas comprueban si ya existe o si hay que crearlo:

```
var objeto = document.getElementById("descripcion");
if (!objeto)
```

Si no existe el elemento "**descripcion**", entonces se crea. Para ello, se utilizan los métodos **createTextNode**, **createElement** y **appendChild**:

```
var newText = document.createTextNode(texto);
var newElement = document.createElement("p");
newElement.setAttribute("id", "descripcion");
newElement.setAttribute("class", "textoDescripcion");
newElement.appendChild(newText);
document.body.appendChild(newElement);
```

Si ya existe el elemento (se ha pulsado en alguna imagen con anterioridad), no es necesario volver a crearlo, sino que simplemente actualizamos el valor del nodo de texto. Para ello, se utiliza la propiedad **nodeValue**.

```
objeto.firstChild.nodeValue = texto;
```

Con todo este código JavaScript estamos mejorando significativamente la experiencia del usuario.

Por ejemplo, la página se carga mucho más rápidamente que si se mostraran las imágenes grandes desde el principio, ya que las que se muestran son más pequeñas.

Además, la página sigue siendo perfectamente funcional aunque no se pueda ejecutar el código JavaScript. En este último caso, cuando se pulse en una de las imágenes, se accederá a la imagen grande.

Es decir, que aunque el usuario no obtiene la funcionalidad que hemos visto, realmente sigue pudiendo ver el contenido de la página web.