Modelo de Objetos del Documento (DOM)

1. EL ÁRBOL DEL DOCUMENTO

Las últimas lecciones nos han servido de base para entender mejor el concepto de "*objeto*" y cómo se utiliza en JavaScript. Además, hemos conocido algunos objetos, como **window**, **location**, **document**, etc.

Bien, aunque trabajar con esos objetos tal como lo hemos visto hasta ahora puede ser de utilidad, realmente estamos limitados a utilizar los objetos que aparecen en la página web.

Sin embargo, donde realmente JavaScript demuestra todo su potencial es cuando se utiliza para actualizar el contenido de una página sin necesidad de refrescarla, es decir, sin necesidad de acudir de nuevo al servidor.

Para ello, el navegador proporciona lo que se conoce como el "**Modelo de Objetos del Documento**" o más conocido con las siglas en inglés **DOM**.

Según este modelo, el navegador representa en memoria los elementos de una página web, esto es, los que aparecen en el código HTML, como una colección de objetos programables cuyo objeto superior es **document**.

Cualquier título o párrafo de texto; los hipervínculos; las imágenes que aparecen en la página web; los controles de formulario, etc. se representan como objetos del **DOM** cuando la página se carga en el navegador.

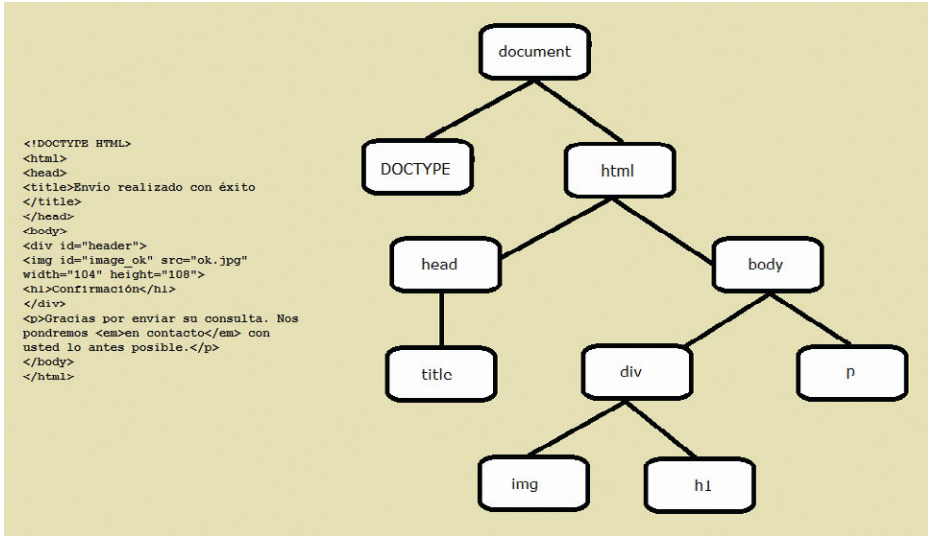
El estándar **W3C DOM** no especifica qué lenguaje de scripting utilizar para acceder a los objetos del documento. Sin embargo, en la práctica en la mayoría de las ocasiones se utiliza **JavaScript**.

Pero aquí viene lo interesante: podemos utilizar JavaScript para acceder a dichos objetos y modificar el contenido de la página web sin necesidad de acceder de nuevo al servidor web, ya que se mantienen en la memoria del navegador.

En definitiva, es la misma idea que hemos visto durante las lecciones anteriores, pero de una forma más potente y flexible.

Y para ello, es necesario entender cómo se representa este modelo en la memoria del navegador. Lo hace en forma de "*árbol de nodos*".

Fíjate en la figura siguiente.



En la parte izquierda tenemos el código HTML de una página muy sencilla, mientras que en la figura de la derecha se muestra el DOM de dicho documento.

Como puedes ver, el DOM queda representado como una estructura "*de árbol invertido*" en la que la raíz es el objeto **document**.

Utilizando DOM no nos tenemos que contentar únicamente con acceder a estos nodos, sino que incluso podremos crear nuevos nodos, eliminarlos y moverlos a lo largo del árbol del documento.

Fíjate lo que esto significa, ya que estarás creando contenido de forma dinámica sin necesidad de volver al servidor web.

Veremos que esto es mucho más potente que simplemente utilizar el método **write** del objeto **document**.

El problema que se encontraban los diseñadores web en el pasado era que cada navegador implementaba su modelo de objetos de forma distinta. Esto implicaba el esfuerzo de tener que comprobar el navegador que utilizaba el usuario y actuar en consecuencia.

Afortunadamente, esto empieza a ser menos serio con los navegadores modernos que se ajustan al estándar **W3C DOM**. Incluso Internet Explorer a partir de la versión 9 va por ese camino.

Durante esta lección conocerás los componentes fundamentales del modelo **DOM**. Veremos qué tipos de nodos podemos encontrar en el árbol del documento y cómo acceder a ellos.

Aunque son conceptos más difíciles de entender, recuerda que estaremos utilizando JavaScript para trabajar con la representación que hace el navegador en su memoria del documento HTML que tiene cargado.

Esta es la definición (traducida) del DOM que aparece en su especificación:

DOM: es una interfaz de programación o API neutra tanto desde el punto de vista de la plataforma como del lenguaje utilizado, que permite programar scripts para acceder y actualizar dinámicamente el contenido, estructura y formato de los documentos.

2. TIPOS DE NODOS Y DE RELACIONES

Pero volvamos a estudiar la estructura en árbol con la que el navegador representa el documento HTML.

Esta estructura proporciona mucha más información de la que parece a priori. Así, en el árbol podemos encontrar tres tipos de nodos:

- Nodos de **elemento**
- Nodos de **atributo**
- Nodos de **texto**

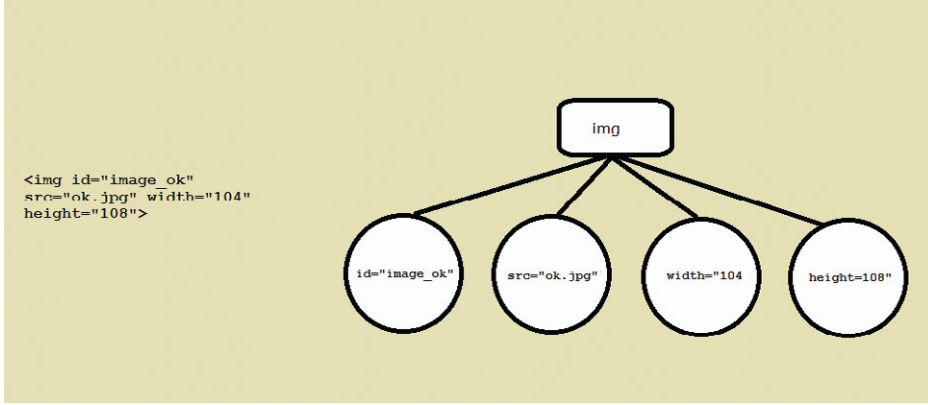
Los **nodos de elemento** se corresponden realmente con los de la figura anterior. Es decir, representan las etiquetas HTML utilizadas en el código fuente del documento.

Así pues, puedes ver que tenemos un nodo para la etiqueta **html**, otro para las etiquetas **body**, **div**, **img**, **h1**, **p**, etc.

Cada vez que el navegador encuentra una etiqueta HTML en el código, genera un nodo de elemento en el árbol, situándolo en la posición correcta.

Los **nodos de atributo** representan los atributos de esos elementos o etiquetas HTML.

En la figura de la página siguiente, los nodos de atributo aparecen representados por círculos. Por ejemplo, se muestra la representación de la etiqueta **img**.

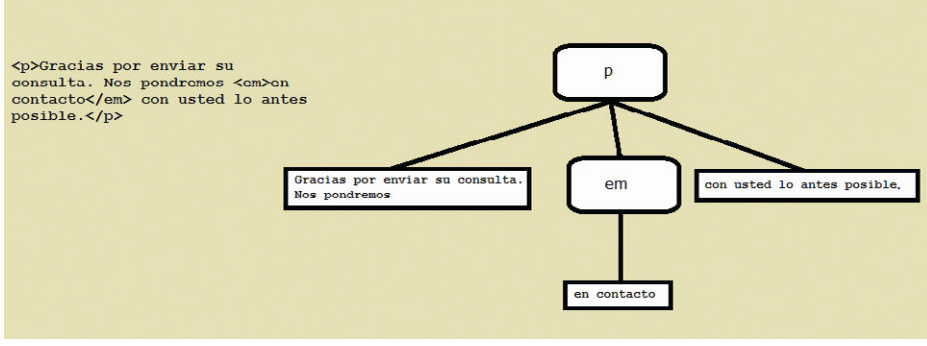


Finalmente, los **nodos de texto** representan fragmentos de texto del documento HTML.

Fíjate que se diferencia entre el nodo del elemento correspondiente al párrafo (p) o al título (h1) del nodo de texto.

En la figura siguiente, puedes comprobarlo. Es interesante ver que, como en el interior del texto aparece una etiqueta **em**, entonces realmente existe un nodo de elemento para ella que tiene por debajo otro nodo de texto.

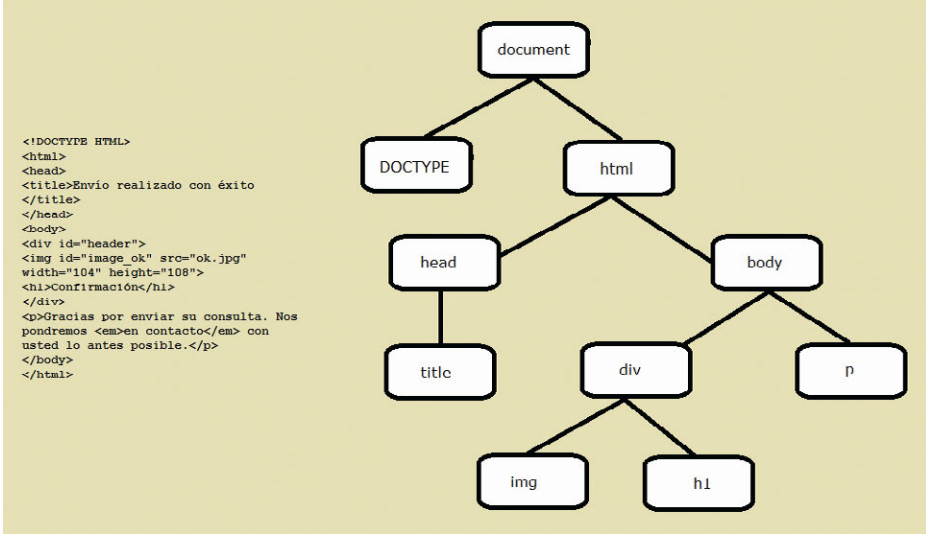
Y es que los nodos de elemento pueden tener otros nodos por debajo de ellos. Sin embargo, esto no ocurre con los nodos de texto.



A parte de conocer los tipos de nodos que podemos encontrar en la representación en árbol de un documento, el DOM también introduce el concepto de "relación" entre los nodos.

Y para entender esto, lo mejor es pensar en el árbol del documento como en un árbol genealógico.

Así, la posición que ocupan los distintos nodos puede representar dos tipos de relaciones:



- Relación **padre - hijo**.

  La encontramos cuando un nodo (el padre) tiene por debajo de él a otros nodos (los hijos).

  Por ejemplo, en el árbol de este sencillo documento HTML, el objeto **document** tiene como hijos a **DOCTYPE** y a **html**; mientras que el objeto **html** tiene como hijos (es decir, que a su vez es padre de) los objetos **head** y **body**.

- Relación de **hermanos**.

  La encontramos con los nodos del árbol situados en un mismo nivel. Por ejemplo, **head** y **body**; **div** y **p**; **img** y **h1** representan esta relación de hermanos.

Te estarás preguntando para qué sirve todo esto y qué relación tiene con la programación en JavaScript.

Pues es más sencillo de lo que parece. Si entiendes la representación que realiza el navegador en su memoria del documento HTML que tiene cargado, entonces entenderás mucho mejor cómo acceder a estos objetos y cómo modificarlos.

Para ello, el DOM proporciona una serie de propiedades, métodos y eventos que podremos utilizar si pensamos en el documento como en una serie de objetos dispuestos en un árbol de nodos como este.

Por ejemplo, seremos capaces de obtener los nodos "hijo" de un determinado nodo, el "padre" de un nodo o los nodos "hermano".

3. OBTENER ELEMENTOS

Una vez un documento HTML se ha cargado en el navegador, este construye el árbol del documento y nos permite acceder a él a través de la interfaz de programación del DOM.

Por lo tanto, lo primero que debe proporcionar el DOM es algún método para hacer referencia u obtener un determinado elemento del documento.

La verdad es que ya hemos visto cómo hacerlo en algún caso, pero utilizando la notación "*de la vieja escuela*". Por ejemplo, hemos visto que podíamos acceder a una determinada imagen si conocíamos cuál era su posición en el array **images**; o lo mismo pero referente a un control determinado de un **formulario**.

Bien, aunque esta forma de obtener elementos siempre estará disponible, el DOM proporciona tres métodos del objeto **document** que son mucho más potentes.

El primero se basa en el identificador del elemento al que queremos hacer referencia. Como ya hemos comentado en más de una ocasión, el atributo **id** de una etiqueta HTML es algo fundamental para poder utilizar JavaScript.

El atributo **id** de una etiqueta HTML:

- No puede contener espacios en blanco.
- No puede contener caracteres de puntuación excepto _ - . :
- Tiene que aparecer entre comillas.
- No puede empezar por un número.
- No puede repetirse en el mismo documento HTML.

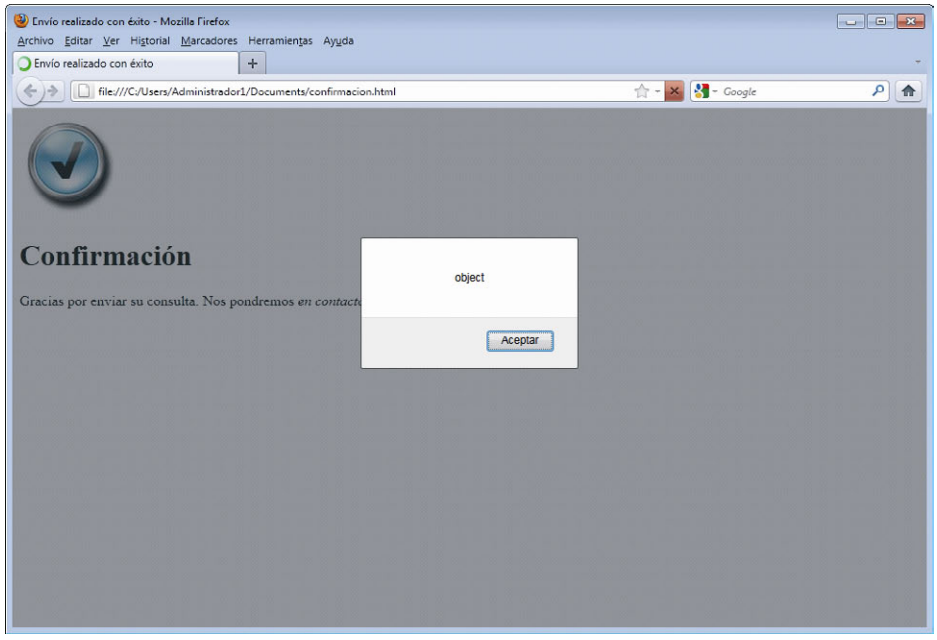
Para obtener un elemento a partir de su identificador, se emplea el método **getElementById** del DOM. Observa:

```
<script type="text/javascript">
<!--
var elemento = document.getElementById("image_ok");
alert (typeof elemento);
//-->
</script>
```

Con este sencillo *script* lo único que estamos haciendo es obtener el elemento cuyo identificador es **image_ok** y mostramos en un cuadro de diálogo **alert** el tipo de ese elemento.

Fíjate cómo se utiliza el método **getElementById**, al que le pasamos como parámetro el identificador del elemento que queremos obtener.

Después, con el operador **typeof** obtenemos el tipo de elemento de que se trata. Fíjate en el resultado.



El resultado que obtenemos es que el tipo del elemento cuyo identificador es **image_ok** es un objeto.

La verdad es que ya lo sabíamos, ya que todos los nodos de elemento son objetos.

Se trata de un objeto de tipo imagen que tendrá sus propiedades, métodos y manejadores de evento.

Pero aquí lo importante es comprobar que hemos podido hacer referencia a dicho objeto utilizando el método **getElementById** del objeto **document**.

Como es natural, no asignaremos identificadores a todos los elementos de un documento HTML.

Por ello, el DOM proporciona otros dos métodos que permiten obtener elementos en función bien del tipo de etiqueta HTML o de la clase que tienen asignados.

El primer método es **getElementsByTagName**.

Fíjate que el nombre de este método incluye la palabra **Elements** y no Element. Esto nos da una idea de que realmente lo que vamos a obtener es un conjunto o array de elementos.

En este caso el parámetro que se incluye en la llamada del método **getElementsByTagName** es el nombre de la etiqueta HTML.

Por ejemplo, si queremos conocer el número de párrafos de texto del contenido de un documento HTML podríamos escribir este código:

```
<script type="text/javascript">
<!--
alert(document.getElementsByTagName("p").length);
//-->
</script>
```

Lo que estamos haciendo en este caso es obtener el conjunto de elementos **p** del documento HTML y, como realmente se trata de un array, calcular su longitud o número de elementos mediante la propiedad **length**.

Podríamos utilizar el método **getElementsByTagName** como forma alternativa para trabajar con los arrays **images**, **links**, **forms**, etc.

Finalmente, el tercer método que nos proporciona el DOM para obtener elementos es **getElementsByClassName**. Se trata de una adición del HTML5.

¡Cuidado a la hora de escribir el nombre de estos tres métodos, ya que se tiene en cuenta la combinación de mayúsculas y minúsculas!

Es similar al que acabamos de ver, pero en este caso se utiliza un nombre de clase para obtener los elementos. Como ocurre con **getElementsByTagName**, el resultado es un array.

```
<script type="text/javascript">
<!--
alert(document.getElementsByClassName("texto").length);
//-->
</script>
```

Recuerda que el atributo **class** sirve para incluir la clase **CSS** con la que aplicaremos un determinado formato. Por ejemplo:

```
<h1 class="texto">Confirmación</h1>
```

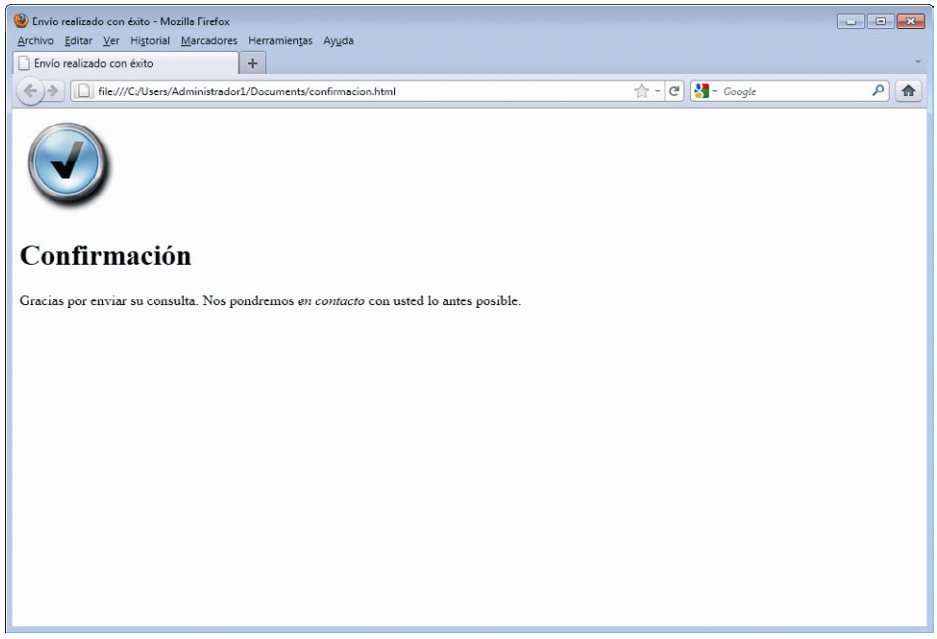
Poder seleccionar u obtener elementos de un documento mediante los métodos **getElementById**, **getElementsByTagName** y **getElementsByClassName** es el primer paso a la hora de trabajar con los objetos del DOM de un documento HTML.

4. OBTENER Y ESTABLECER ATRIBUTOS

Una vez hemos obtenido un elemento del documento, podemos acceder a sus atributos mediante el método **getAttribute** o también podemos establecerlos mediante el método **setAttribute**.

Estos métodos pertenecen al propio elemento, por lo que no se pueden aplicar directamente al objeto **document**.

Veamos un ejemplo y quedará más claro.



En la página anterior tenemos una imagen cuyo elemento DOM se corresponde con la etiqueta **img**. Fíjate en el código HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Envío realizado con éxito</title>
</head>
<body>
<div id="header">
<img id="image_ok" src="ok.jpg" width="104"
height="108">
<h1>Confirmación</h1>
</div>
<p>Gracias por enviar su consulta. Nos pondremos <em>en
contacto</em> con usted
lo antes posible.</p>
</body>
</html>
```

Aquí lo puedes ver mejor. Fíjate que la etiqueta **img** incluye el atributo **src** para indicar la ubicación de la imagen y los atributos **width** y **height** para establecer sus dimensiones.

Sin embargo, vemos que no se ha incluido el atributo **alt** para el texto alternativo a la imagen, algo que es muy recomendable.

Bien, lo que podríamos hacer es comprobar si el elemento imagen incluye dicho atributo y, en caso negativo, establecerlo.

Para lo primero será necesario obtener o comprobar el valor de **alt**; y para lo segundo, establecer dicho valor si no se ha incluido en el código HTML:

```
<script type="text/javascript">
<!--
var imagen = document.getElementById("image_ok");
if (!imagen.getAttribute("alt"))
  imagen.setAttribute("alt", "Imagen de envío correcto.");
//-->
</script>
```

Primero utilizamos **getElementById** para obtener la referencia al objeto correspondiente a la imagen y después utilizamos su método **getAttribute** para obtener el valor del atributo "**alt**".

Este método solo incluye como parámetro el atributo que queremos consultar.

En el caso de que dicho atributo no se haya establecido (fíjate en la condición del **if**), simplemente lo estableceremos con **setAttribute**.

En este caso, el método **setAttribute** requiere como primer parámetro, el nombre del atributo; y como segundo, su valor.

Como sabemos, el atributo **alt** se utiliza como texto alternativo cuando no se ha podido cargar la imagen. Obsérvalo en la figura de la página siguiente, donde se muestra el texto alternativo porque no se ha podido cargar la imagen.

Este texto alternativo se ha generado mediante el uso del método **setAttribute**, por lo que no se incluye en el código HTML original de la página.

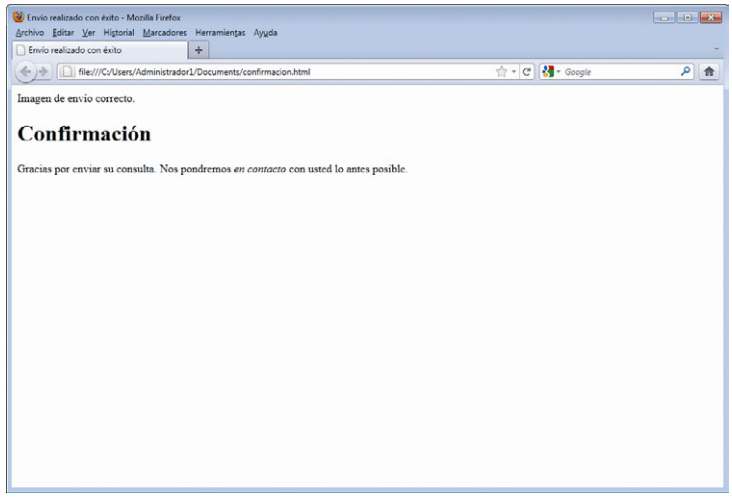
Aunque algunos objetos permiten obtener y establecer los atributos sin necesidad de utilizar estos dos métodos, siempre los aceptarán.

Por ejemplo, los objetos imagen aceptan esta otra sintaxis alternativa:

```
imagen.alt = "Imagen de envío correcto";
```

Sin embargo, otros tipos de objetos no la aceptan.

Por eso es recomendable utilizar los métodos **getAttribute** y **setAttribute** para no tener que recordar qué objetos permiten acceder o establecer qué atributos.



A continuación se proporciona el código completo de la página:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Envío realizado con éxito</title>
</head>
<body>
<div id="header">
<img id="image_ok" src="ok.jpg" width="104"
height="108">
<h1 class="texto">Confirmación</h1>
</div>
<p class="texto">Gracias por enviar su consulta. Nos
pondremos <em>en contacto</em> con usted
lo antes posible.</p>
<script type="text/javascript">
<!--
var imagen = document.getElementById("image_ok");
if (!imagen.getAttribute("alt"))
   imagen.setAttribute("alt", "Imagen de envío
                        correcto.");
//-->
</script>
</body>
</html>
```