Manipulación del DOM

1. RECORRER EL ÁRBOL DEL DOCUMENTO

En la lección anterior hemos visto que utilizando una serie de métodos del **DOM** es posible acceder al contenido de una página web.

Así, hemos conocido los métodos que permiten obtener elementos:

- **getElementById**
- **getElementsByTagName**
- **getElementsByClassName**

Además de los métodos que nos permiten trabajar con los atributos de estos elementos:

- **getAttribute**
- **setAtrribute**

Sin embargo, el DOM proporciona mucha más funcionalidad y realmente donde se ve toda su potencia es cuando lo utilizamos para la manipulación del documento, esto es, para añadir, modificar o eliminar contenido de la página web una vez ha sido cargada.

Y para ello, en esta lección conoceremos una serie de métodos y propiedades. Empezaremos con las propiedades que nos permiten recorrer el árbol del documento y seleccionar los elementos de otra forma:

- **parentNode**: propiedad que proporciona el nodo padre.
- **childNodes**: propiedad que proporciona el conjunto de nodos hijo, almacenándolos en un array.
- **firstChild**: propiedad que permite acceder al primer nodo hijo. Se trata de un atajo para el primer elemento del array childNodes.
- **lastChild**: propiedad que permite acceder al último nodo hijo. Se trata de un atajo para el último elemento del array childNodes.
- **previousSibling**: propiedad que devuelve el nodo "hermano" previo al actual.
- **nextSibling**: propiedad que devuelve el nodo "hermano" siguiente al actual.

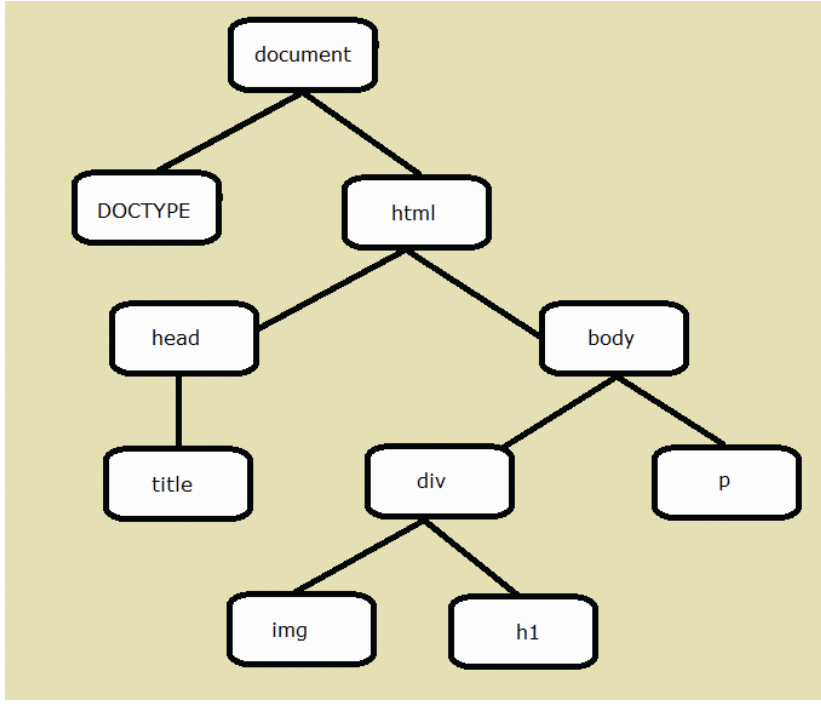
Para entender mejor todos estos métodos y propie–dades, recuerda que la estructura del DOM queda representada como un árbol de nodos.

Fíjate que para utilizar estas propiedades se necesita la referencia a un nodo en particular, por lo que siempre se tendrá que obtener ese nodo antes de aplicarlas.

```
<!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>
```

Este código HTML se corresponde a la página web con la que estuvimos trabajando en la lección anterior.

El árbol de este documento es sencillo y queda reflejado en esta figura.



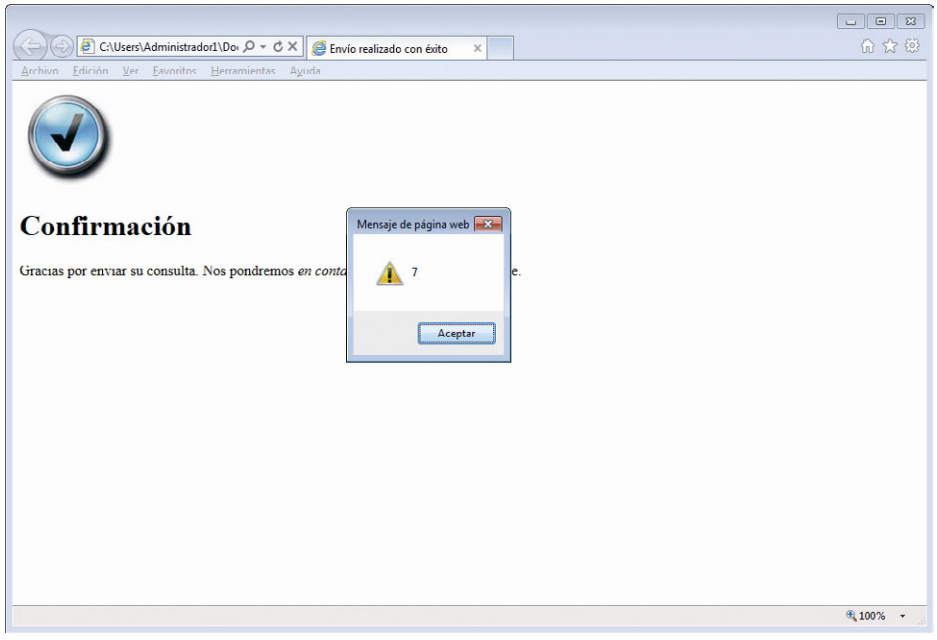
Con las propiedades mencionadas anteriormente podremos recorrer el árbol del documento, accediendo a los nodos con los que más tarde podríamos realizar algún proceso.

```
function ContarElementos()
{
   return document.body.childNodes.length;
}
```

Esta función utiliza la propiedad **childNodes** para obtener el array de nodos de elemento del documento. Para ello, se utiliza como nodo el objeto **document.body**. Una vez obtenido dicho array, conocemos el número de elementos a través de su propiedad **length**.

```
<body onload="alert(ContarElementos());">
```

De esta forma, la función se ejecutará tras cargarse completamente la página, mostrando el resultado a través de un cuadro de diálogo **alert**.



Se nos indica que hay **7** nodos en el documento.

La propiedad **childNodes** devuelve el número de nodos hijo directos, por lo que no parece que tenga mucho sentido este número. En todo caso deberían ser 3 nodos hijo: el elemento **div**, el párrafo **p** y el **script**.

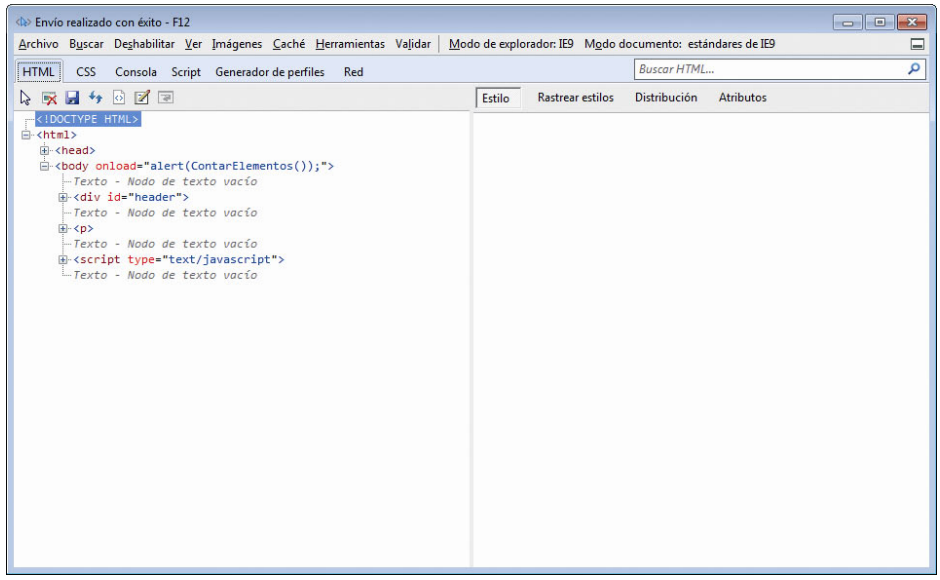
Bien, es importante entender que **childNodes** devuelve el número de nodos, tanto de elemento como de texto.

Y aunque no parece que haya ningún nodo de texto que sea hijo directo de **document.body**, realmente sí que lo hay.

Para el navegador, cualquier espacio en blanco o retorno de carro se representa también como un nodo de texto, por lo que aquí sí que tiene importancia cómo escribamos el código HTML.

Para el navegador Firefox hay un complemento equivalente a las Herramientas de desarrollo de Internet Explorer. Se trata del complemento Firebug.

Para verlo claramente, utilizaremos las **Herramientas de desarrollo** de Internet Explorer. Pulsa la tecla **F12** o accede a través del menú **Herramientas**.



Mediante esta herramienta, Internet Explorer nos proporciona el árbol del documento. Si te fijas, colgando de **body** hay un total de 7 nodos, que es lo que nos ha devuelto el script.

Fíjate que hay "*nodos de texto vacío*", que realmente se corresponden con los retornos de carro incluidos en el código HTML.

Esto no tiene mayor importancia excepto si recorremos el árbol del documento utilizando las propiedades que hemos introducido en este capítulo, ya que podríamos pensar que estamos accediendo a un nodo de elemento cuando realmente lo estamos haciendo a un nodo de texto vacío.

Por ejemplo, el primer nodo hijo de **body** es un nodo de texto vacío y no el elemento **div** que parecía inicialmente según el código HTML.

Si utilizamos la propiedad **document.body.firstChild** para acceder a **div**, lo estaremos haciendo incorrectamente.

Simplemente recuerda esta circunstancia cuando recorras el árbol del documento. Si escribieras todo el código HTML en un única línea, no tendrías este problema pero normalmente no será viable hacerlo si el código no es muy corto.

Como medida para resolver el problema que puede representar la presencia de estos nodos de texto vacío, podemos utilizar las propiedades:

- **nodeName**: proporciona el nombre del nodo (se corresponde con la etiqueta HTML en el caso de un nodo de elemento) o **#text** si se trata de un nodo de texto.

- **nodeType**: devuelve un valor numérico representando el tipo de nodo.

- **nodeValue**: permite acceder al contenido o valor del nodo. Por ejemplo, si es un nodo de texto, esta propiedad nos permite acceder al texto en sí.

En especial podemos utilizar la propiedad **nodeType**, que puede tener hasta 12 valores distintos, siendo estos los más importantes:

- Los nodos de elemento tienen el valor **nodeType = 1**.

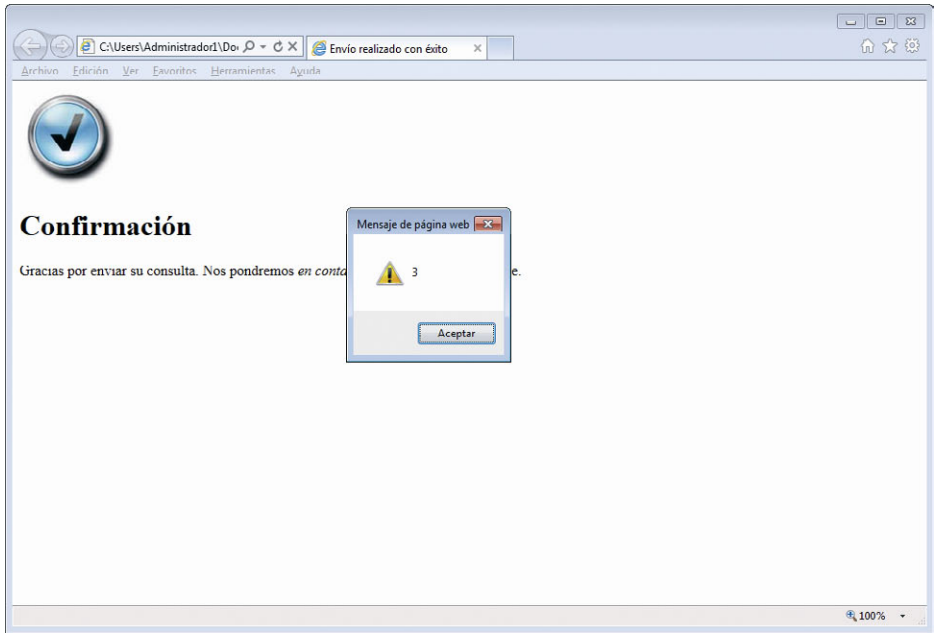
- Los nodos de atributo tienen el valor **nodeType = 2**.

- Los nodos de texto tienen el valor de **nodeType = 3**.

Por lo tanto, podemos utilizar esta propiedad para limitar el resultado a los nodos de elemento:

```
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;
}
```

Ahora el resultado indica únicamente el número de nodos de elemento del documento, como puedes ver en la figura de la página siguiente.



2. MODIFICAR EL VALOR DE LOS NODOS

Mediante la propiedad **nodeValue** podemos acceder y modificar el valor de un nodo.

Pero aquí es importante entender que **nodeValue** solo tiene sentido cuando se aplica sobre determinados tipos de nodo.

Por ejemplo, si lo aplicas sobre un nodo de elemento "**p**" no obtendrás ningún resultado significativo; sin embargo, si lo aplicas sobre el nodo de texto hijo de dicho elemento, sí que obtendrás el texto que se incluye en su interior.

Vamos a acceder al nodo que aparece como hijo del nodo "**em**" y modificaremos su texto. Así, en lugar de "*en contacto*", pondremos "*en comunicación*".

Fíjate que para ello, deberemos acceder al nodo de elemento "**em**" y después al nodo de texto.

```
function CambiarTexto()
{
var elemento_em = document.getElementsByTagName("em")[0];
elemento_em.firstChild.nodeValue = "en comunicación";
}
```

Fíjate en lo que estamos haciendo.

Primero obtenemos el nodo correspondiente al elemento "**em**" mediante el método **getElementsByTagName**. En este caso, dicho elemento es el primero (y único) de este tipo, por lo que accedemos al índice **0** del array devuelto por dicho método.

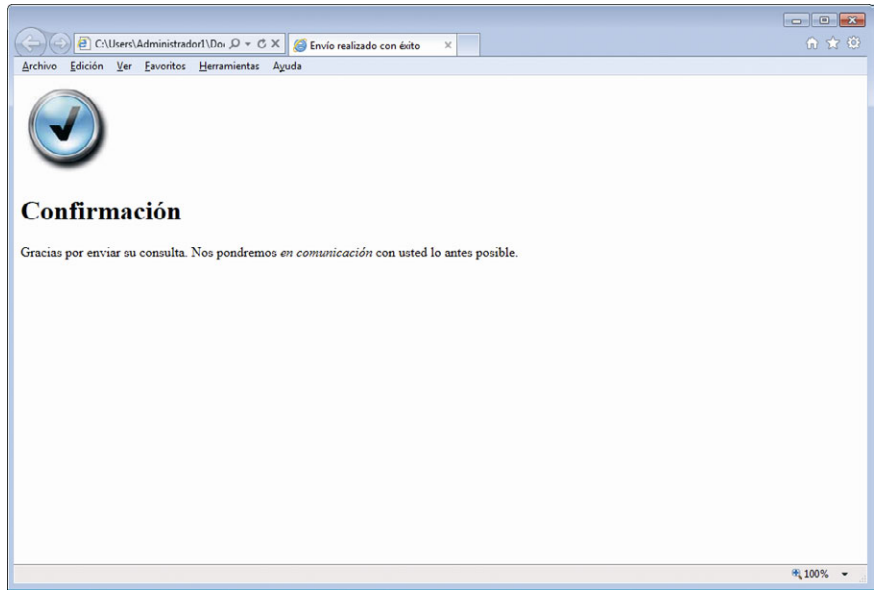
Una vez tenemos el nodo padre del texto, utilizamos la propiedad **firstChild** para acceder al primer nodo, que coincide con el nodo de texto.

Después simplemente actualizamos el valor de su propiedad **nodeValue**.

Utilizaremos esta función ante un evento del usuario:

```
<em onclick="CambiarTexto();">en contacto</em>
```

Por lo tanto, cuando pulsemos en este elemento del árbol del documento, se cambiará el texto.



Fíjate en la potencia que representa manipular el DOM una vez la página se ha cargado. No tenemos que volver al servidor web para realizar estas actualizaciones, ya que las lleva a cabo el propio navegador al ejecutar el código JavaScript.

3. CREAR, ELIMINAR Y REEMPLAZAR NODOS

También es posible utilizar métodos del DOM para crear nuevos nodos. Es algo parecido a crear contenido mediante el método **write** del objeto **document**, pero mucho más potente.

Para ello se pueden utilizar los métodos:

- **createElement**: crea un nodo de elemento.
- **createTextNode**: crea un nodo de texto.
- **cloneNode**: crea un nodo idéntico a otro existente.
- **appendChild**: añade un nodo como hijo de otro. Sirve para incorporar el nuevo nodo al árbol del documento.
- **insertBefore**: inserta un nodo antes de otro en el árbol del documento.

Y para eliminar o reemplazar nodos:

- **removeChild**: para eliminar un nodo hijo.
- **replaceChild**: para reemplazar un nodo hijo.

Estos dos métodos se aplican sobre el nodo padre del nodo que queremos eliminar o reemplazar, por lo que es posible utilizar la propiedad parentNode para obtenerlo.

Los dos primeros métodos, **createElement** y **createTextNode**, pertenecen al objeto **document**; mientras que los otros cinco, se pueden aplicar sobre cualquier nodo de elemento.

Por ejemplo, vamos a añadir un nuevo párrafo en este documento. El párrafo incluirá el texto "*Si no recibe respuesta tras 2 días, por favor, vuelva a enviar la consulta.*".

Para añadir un párrafo que incluya dicho texto, debemos crear dos tipos de nodos: el del elemento **p** y el del propio **texto**:

```
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);
}
```

Primero creamos el nuevo nodo de elemento, para lo que se utiliza el método **createElement**. Fíjate que se trata de un método del objeto **document** y que se utiliza como parámetro el nombre o tipo de elemento (en este caso un elemento **p**).

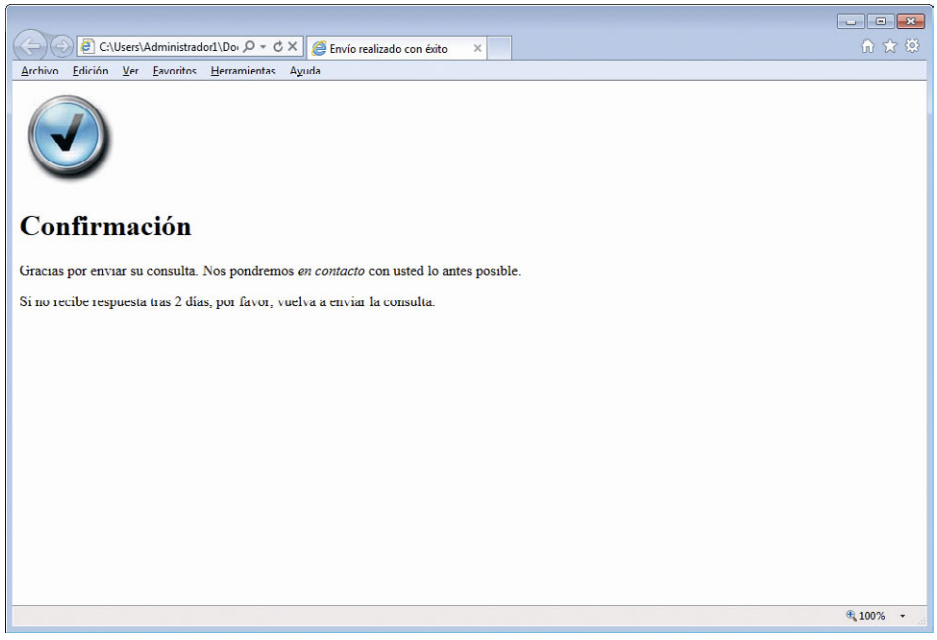
Seguidamente creamos el nodo de texto mediante el método **createTextNode**. Este método del objeto **document** requiere del texto en sí que se incluirá.

A continuación enlazamos los nuevos nodos con sus nodos padre. Para ello, se utiliza el método **appendChild**. Así pues, el nodo de texto será hijo del nuevo nodo de elemento y el nuevo nodo de elemento será hijo de **document.body**.

Falta asociar este código con algún evento. Utilizaremos de nuevo el evento **onload** de la etiqueta **body**.

```
<body onload="NuevoTexto();">
```

Este es el resultado:



En este caso se ha insertado al final del documento, ya que se ha creado una vez el documento original se había cargado.

Es posible utilizar el método **insertBefore** para insertarlo en otro lugar. Para ello, se requiere de tres referencias de nodo:

```
nodo_padre.insertBefore(nuevo_nodo, viejo_nodo);
```

Es decir, que **insertBefore** inserta el **nuevo_nodo** como un nodo hijo de **nodo_padre** y antes de **viejo_nodo**.

4. EL MÉTODO INNERHTML

En el ejemplo anterior hemos creado un nodo de texto y lo hemos incluido como hijo de un nodo de elemento. Más tarde incluimos ambos nodos en el árbol del documento.

Aunque el contenido añadido ha sido pequeño y sencillo (al fin y al cabo únicamente un párrafo de texto), ya hemos podido comprobar que se ha requerido de bastante código.

Imagínate lo que esto puede suponer si necesitas incorporar contenido más complejo, como una tabla HTML entera.

Tener que crear los nuevos nodos de elemento, incluir en ellos otros nodos de texto, asignar atributos, etc. no son tareas complejas pero sí que requieren de bastante código si utilizamos el API del DOM.

Por ello, durante mucho tiempo los programadores han utilizado una propiedad que, aunque no pertenece al estándar **W3C DOM**, prácticamente es un estándar *"de facto"*. Se trata de la propiedad **innerHTML**.

La propiedad **innerHTML** no pertenece al estándar **W3C DOM**, pero sí que aparece en la especificación de **HTML5**, por lo que seguramente se seguirá utilizando.

A través de esta propiedad podemos añadir o modificar texto y código HTML como valor de un nodo de 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.innerHTML = codigo;
   document.body.appendChild(nuevo_elemento);
}
```

La función **NuevaTabla** emplea una cadena de texto o String para englobar el código HTML necesario para crear una sencilla tabla de tres filas y dos columnas.

Después se crea un nuevo nodo de elemento con el método **createElement**. En este caso, se crea un contenedor genérico **div**.

Seguidamente se incluye el código HTML que aparecerá en el interior del **div** mediante la propiedad **innerHTML**.

Finalmente se añade el nuevo elemento en el árbol del documento.

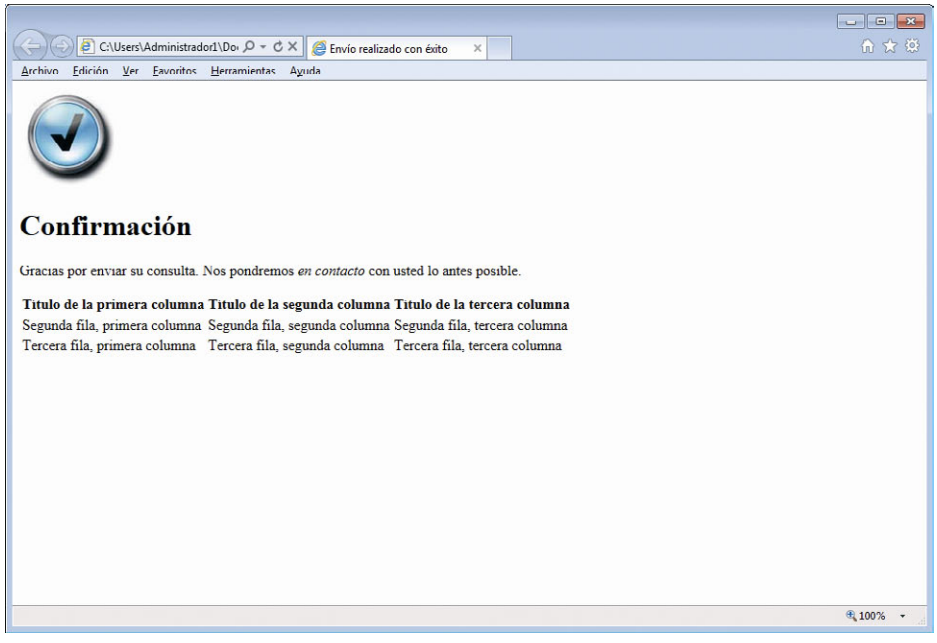
Fíjate lo rápido que esto puede ser en contraposición de tener que generar los correspondientes nodos correspondientes a este fragmento de código HTML.

Sin embargo, el código HTML se trata como una cadena de texto, por lo que no podemos hacer nada con él, a diferencia de lo que ocurre si generamos el conjunto de nodos DOM equivalente.

Ejecutaremos este código de nuevo al cargar la página:

```
<body onload="NuevaTabla();">
```

Este es el resultado:



Ahora se ha generado la tabla, añadiéndola al final del contenido de la página web.

En cualquier caso, los métodos del DOM pueden ser utilizados en lugar de la propiedad **innerHTML**. Aunque esto pueda ocasionar un código más largo y laborioso, normalmente es la mejor opción ya que nos proporciona un mayor control sobre el contenido que se genera sobre la marcha.

5. MODIFICAR EL FORMATO DINÁMICAMENTE

Finalizaremos la lección comprobando cómo modificar el **formato** de los elementos mediante el uso de algunas propiedades del DOM.

Aunque ya sabemos que el formato de una página web debe realizarse mediante la aplicación de las hojas de estilo en cascada **CSS**, el lenguaje JavaScript junto al DOM nos permiten modificar o actualizar dinámicamente dicho aspecto.

Esta es una de esas situaciones en las que la división de tareas del diseño de una página web (estructura, formato y comportamiento) no se cumple *"a rajatabla"*.

Para modificar el aspecto o estilo de un elemento, podemos emplear dos notaciones:

- La notación pura del DOM, que se basa en establecer los atributos **style** o **class** de un elemento.

Por ejemplo:

```
nodo.setAttribute("style", "font-family: Verdana, sans-
serif; font-size: 20px; color: red");
```

o

```
nodo.setAttribute("class", "encabezado");
```

En este caso se utiliza el método **setAttribute** para establecer las propiedades CSS adecuadas, ya sea mediante un estilo o mediante la aplicación de una clase.

- O la notación en la que se emplean las propiedades **style** y **className** directamente en el código:

Por ejemplo:

```
nodo.style.fontFamily = "Verdana, sans-serif";
nodo.style.fontSize = "20px";
nodo.style.color = "red";
```

o

```
nodo.className = "encabezado";
```

En este caso, las propiedades tienen el mismo nombre que las propiedades CSS equivalentes pero sin la presencia de guiones (-). En su lugar se utiliza una combinación de mayúsculas y minúsculas.

Así pues, la propiedad CSS **font-family** pasa a ser la propiedad del DOM **fontFamily**; **margin-left** pasa a ser **marginLeft**, etc.

Además, como la palabra "**class**" es una palabra reservada de JavaScript, la propiedad HTML **class** pasa a ser la propiedad del DOM **className**.

Podemos emplear cualquiera de las dos notaciones. Si estás acostumbrado a utilizar **getAttribute** y **setAttribute** con cualquier elemento, puedes seguir utilizándola; si prefieres emplear las propiedades concretas de estilo, simplemente recuerda la sintaxis necesaria.

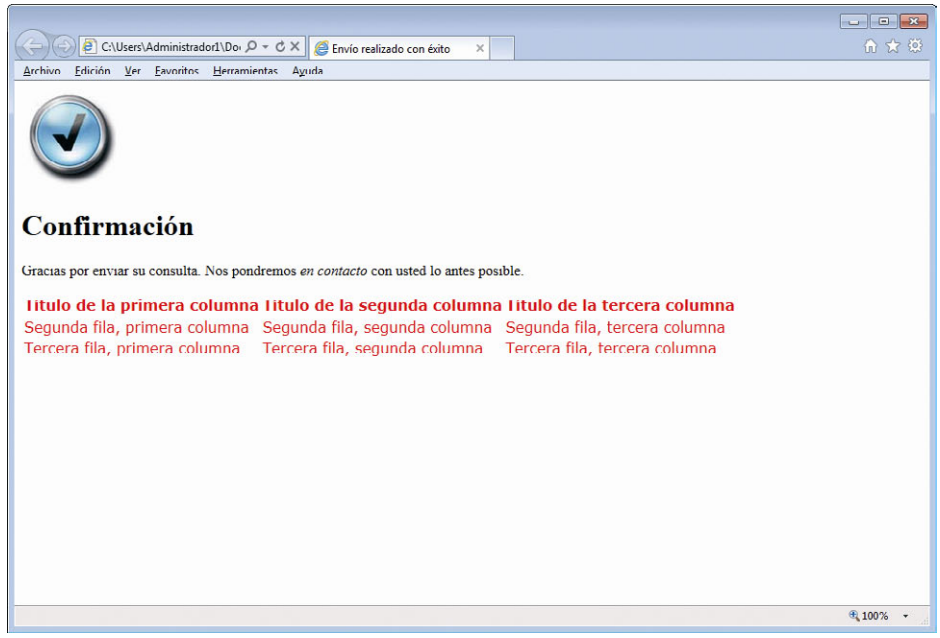
```
function NuevaTabla()
{
   var codigo = "<table>";
   codigo += "<tr><th>Título de la primera columna</th>"
   codigo += "<th>Título de la segunda columna</th>"
   codigo += "<th>Título de la tercera
              columna</th></tr>"
   codigo += "<tr><td>Segunda fila, primera
              columna</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.setAttribute("style", "font-family:
                  Verdana, sans-serif; color: red;");
   nuevo_elemento.innerHTML = codigo;
   document.body.appendChild(nuevo_elemento);
}
```

Con esto estaremos estableciendo unas características de formato para el elemento contenedor de la tabla.

Si en lugar de utilizar el atributo "**style**" utilizamos "**class**", entonces deberemos escribir una regla CSS utilizando la correspondiente clase.

En la figura siguiente puedes comprobar que el texto de la tabla se muestra con las nuevas características de formato.



Ahora emplearemos la notación equivalente que utiliza propiedades del DOM:

```
nuevo_elemento.style.fontFamily = "Verdana, sans-serif";
nuevo_elemento.style.color = "red";
```

Estas dos líneas son equivalentes a la primera que hemos escrito para cambiar el aspecto del texto.

Utilizando JavaScript y el DOM disponemos de una herramienta muy potente, aunque de cierta complejidad, para diseñar páginas web dinámicas, que se actualizan sin necesidad de acceder al servidor web.

Todo esto se muestra mucho más claramente cuando trabajamos con **HTML5**, ya que es un lenguaje donde el código de marcado (HTML) y de comportamiento (JavaScript y DOM) forman parte de una misma tecnología o de un conjunto de tecnologías.