Introducción a la PROGRAMACIÓN WEB

Antonio Javier Gallego Sánchez

Table of Contents

Contenidos

El objetivo de este libro es realizar una introducción a los conceptos más básicos de la programación web, como son HTML y CSS, hasta otros más avanzados y novedosos de HTML 5, CSS 3 o JavaScript.

En las primeras secciones se realiza una introducción a los conceptos fundamentales, desde la sintaxis del lenguaje, etiquetas, editores de HTML, estructura básica de una página web, etc. Se incluyen pequeños ejemplos para cada uno de los apartados además de algún ejemplo más elaborado y ejercicios finales.

También se tratan aspectos más avanzados como eventos HTML, procesamiento de formularios, pseudo-clases CSS, disposición o estructuración de una web en capas, y se realiza una introducción a los últimos elementos del lenguaje introducidos con HTML 5 y CSS: transiciones, transformaciones, audio, vídeo, geolocalización, etc.

Por último se incluye una sección de introducción a JavaScript en la que también se parte desde los conceptos más básicos (inclusión de scripts en HTML, variables, operadores, etc.) hasta algunos más avanzados como funciones, eventos o como trabajar con el DOM de una página Web.

Los contenidos principales de este libro son:

- Introducción a HTML
    - Editores HTML
    - Sintaxis del lenguaje
    - Estructura básica de una Web
    - Elementos de cabecera
    - Etiquetas básicas
    - Listas
    - Tablas
    - Cajas, etiqueta *DIV*
    - Enlaces
    - Imágenes
    - Formularios
    - Eventos
    - Símbolos HTML
- Introducción a CSS
    - Adjuntar una hoja de estilo
    - Definición de estilos para etiquetas HTML

Introducción a HTML

HTML, siglas de *HyperText Markup Language* (Lenguaje de marcado de hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

El código HTML se escribe en forma de "etiquetas", mediante las cuales podemos describir la estructura lógica y apariencia del contenido. La apariencia que podemos describir con HTML está bastante limitada, pero el código se puede complementar y mejorar mediante el uso de otros lenguajes como JavaScript o CSS.

En las siguientes secciones se realiza una introducción a las principales características de HTML, partiendo desde los posibles editores más adecuados para escribir código o la estructura básica que una página HTML debe tener, hasta conceptos más avanzados como la inclusión de formularios o el manejo de los eventos producidos en una página.

Editores HTML

El lenguaje HTML puede ser creado y editado con cualquier editor de textos básico, como puede ser Gedit en Linux o el Bloc de notas de Windows. Existen además otros editores para la realización de sitios web con características WYSIWYG (*What You See Is What You Get*, o en español: "lo que ves es lo que obtienes"). Estos editores permiten ver el resultado de lo que se está editando en tiempo real, a medida que se va desarrollando el documento. Ahora bien, esto no significa una manera distinta de realizar sitios web, sino que una forma un tanto más simple ya que estos programas, además de tener la opción de trabajar con la vista preliminar, tiene su propia sección HTML, la cual va generando todo el código a medida que se va trabajando. Algunos ejemplos de editores son Adobe Dreamweaver, KompoZer o Microsoft FrontPage.

Estos editores aceleran o facilitan la creación de código HTML, pero en algunas ocasiones también generan mucho más código del necesario (como es el caso de Microsoft FrontPage). Lo ideal es tener un control total sobre el código que se escribe y utilizar estos editores sólo como una pequeña ayuda. También podemos utilizar otro tipo de editores que simplemente comprueben que el código HTML escrito es correcto (que las etiquetas y atributos son correctos, las etiquetas se cierran correctamente, etc.).

Sintaxis del lenguaje

Las etiquetas HTML deben de ir encerradas entre corchetes angulares `<>`, y pueden ser de dos tipos:

- Se abren y se cierran, como por ejemplo: `<b>negrita</b>` o `<p>texto</p>`.
- Se abren y cierran en la misma etiqueta, como: `<br/>` o `<hr/>`.

En caso de que no cerremos una etiqueta que deba ser cerrada se producirá un error en la estructura del documento y probablemente también genere errores en la visualización.

Hay etiquetas que además pueden contener atributos, en este caso los atributos se deben de colocar en la etiqueta de inicio, de la forma:

```
<etiqueta atributo1="valor1" atributo2="valor2">...</etiqueta>
```

O para las etiquetas de solo apertura:

```
<etiqueta atributo1="valor1" atributo2="valor2"/>
```

Por ejemplo:

```
<img src="imagen.jpg" alt="Imagen de cabecera"/>
```

Estructura básica de una Web

Un documento HTML comienza con la etiqueta `<html>` y termina con `</html>`. Dentro del documento (entre las etiquetas de principio y fin de html) hay dos zonas bien diferenciadas: el encabezamiento, delimitado por `<head>` y `</head>`, que sirve para incluir definiciones iniciales válidas para todo el documento; y el cuerpo, delimitado por `<body>` y `</body>`, donde reside la información del documento.

Las etiquetas básicas o mínimas son:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.d
td">
<html>
<head>
   <title>Ejemplo</title>
</head>
<body>
   ¡Hola mundo!
</body>
</html>
```

La primera línea es el DOCTYPE, o el tipo de documento que viene a continuación. En este caso se usa el estándar de HTML 4.01 (el último estándar adoptado en 1999, ya que HTML5 a fecha de 2011 sigue siendo un borrador). La siguiente etiqueta, `<html>`, define el inicio del documento HTML, e indica que lo que viene a continuación debe ser interpretado como código HTML. Como podemos ver en la última línea, se cierra la etiqueta `</html>`.

Dentro de estas etiquetas `<html> ... </html>` encontramos las dos secciones mencionadas:

- Encabezamiento (`head`): Esta sección inicial contiene todos los elementos *no visuales* de nuestra web, como por ejemplo los metadatos descriptivos (autor, palabras clave, descripción del contenido, etc.), los estilos a utilizar en los elementos visuales del cuerpo, el título que aparecerá en la barra superior del navegador (como en el ejemplo superior), y otra serie de elementos que se estudiarán más en detalle en la siguiente sección.
- Cuerpo (`body`): Aquí hemos de incluir todos los contenidos *visuales* de nuestra web, todos los textos, imágenes, enlaces, etc. En el ejemplo de arriba lo único que se incluye es el texto "¡Hola mundo!" por lo que al abrir esta web nos aparecerá una página web que incluirá únicamente ese texto.

En los siguientes apartados se describirán más en detalle los elementos que podemos utilizar dentro del encabezamiento o del cuerpo de una web. En primer lugar se estudiarán los elementos de la cabecera y a continuación todas las etiquetas HTML que se suelen utilizar para la construcción de una web.

Elementos de la cabecera

La sección de cabecera `<head> ... </head>` se utiliza para describir el tipo de contenido y aspecto visual que tendrá la web. Es importante destacar que todo el contenido de la sección de cabecera **no** se muestra directamente al usuario, sino que es únicamente información descriptiva y metadatos. Por ejemplo, nos permitirá indicar metadatos que son muy útiles para la indexación de la web en buscadores, como el tipo de contenido, palabras clave o el autor, o indicar los estilos con los cuales se mostrarán los elementos visuales, entre otra información.

Algunas de las principales etiquetas que podemos utilizar dentro de la cabecera son:

- `<title></title>` : define el título de la página. Por lo general el título aparece en la barra de título encima de la ventana.
- `<link/>` : para vincular el sitio con hojas de estilo externas (ver la sección de CSS para más información):

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

  El atributo `rel` es requerido y describe el tipo de documento enlazado (en este caso una hoja de estilo). El atributo `type` es simplemente indicativo del tipo de hoja de estilo enlazada (en este caso CSS).

- `<style></style>` : se utiliza para añadir definición de estilo en línea. No es necesario colocarlo si se va a utilizar una hoja de estilo externa usando la etiqueta `<link/>` (que es lo más habitual y recomendable). El uso correcto sería de la forma:

  ```
  <html>
  <head>
   ...
   <style type="text/css">
      Estilos CSS
   </style>
  </head>
  <body></body>
  </html>
  ```

  Para más información ver la sección CSS del manual.

- `<meta/>` : para indicar metadatos como la descripción de la web, los keywords, o el autor:

```
<meta name="description" content="Descripción de la web" />
<meta name="keywords" content="key1,key2,key3" />
<meta name="author" content="Nombre del autor" />
```

Una etiqueta "meta" **muy útil** es la de la codificación, que nos permitirá escribir texto con acentos y se se vea bien (sin símbolos extraños) en todos los navegadores:

```
<meta charset="utf-8"/>
```

- `<script></script>` : permite incluir un script en la Web. El código se puede escribir directamente entre las etiquetas de `<script>` o cargar desde un fichero externo utilizando el atributo `src="url del script"` para indicar la dirección del fichero. Se recomienda incluir el tipo MIME en el atributo `type` , que en el caso de código JavaScript sería `text/javascript` . A continuación se incluyen algunos ejemplos de uso:

```
<script src="fichero.js" type="text/javascript"></script>
<script type="text/javascript">
  Código de un script integrado en la página
</script>
```

Cuando usamos el atributo `src` el contenido de estas etiquetas está vacío (no encierra nada), esto es porque lo carga directamente desde el fichero indicado. En el capítulo sobre JavaScript podréis encontrar mucha más información sobre como utilizar estas etiquetas.

A continuación se incluye un código de ejemplo en el que se ha ampliado la estructura HTML básica de una web que vimos en la sección anterior para añadir algunas de estas etiquetas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Ejemplo</title>

    <meta charset="utf-8"/>
    <meta name="description" content="Descripción de la web" />
    <meta name="keywords" content="key1,key2,key3" />
    <meta name="author" content="Nombre del autor" />

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

    <script src="javascript.js" type="text/javascript"></script>
</head>
<body>
   ¡Hola mundo!
</body>
</html>
```

Etiquetas básicas HTML

Dentro de la sección del cuerpo (`<body> ... </body>`) utilizaremos etiquetas HTML para crear el contenido *visual* de la web. Estas etiquetas nos permitirán ir añadiendo textos, imágenes, encabezados, tablas, etc. para componer el diseño de la web. En primer lugar veremos las etiquetas más básicas que se suelen utilizar, estas son:

- `<h1></h1>` a `<h6></h6>` : encabezados o títulos del documento con diferente relevancia, siendo `<h1>` la cabecera de mayor nivel.
- `<p></p>` : definición de un párrafo.
- `<br/>` : salto de línea.
- `<b></b>` : texto en negrita (etiqueta desaprobada. Se recomienda usar la etiqueta `<strong></strong>`).
- `<i></i>` : texto en cursiva (etiqueta desaprobada. Se recomienda usar la etiqueta `<em></em>`).
- `<s></s>` : texto tachado (etiqueta desaprobada. Se recomienda usar la etiqueta `<del></del>`).
- `<u></u>` : texto subrayado.
- `<center></center>` : texto centrado.
- `<pre></pre>` : texto preformateado, respeta los espacios y saltos de línea.
- `<sup></sup>` : Superíndice.
- `<sub></sub>` : Subíndice.
- `<blockquote></blockquote>` : Indica una cita textual, se representa como un párrafo indexado con respecto al margen.
- `<hr/>` : Línea horizontal, usada, por ejemplo, para separar diferentes secciones.
- `<!-- comentario -->` : Comentarios en HTML. El texto del comentario no será visible en el navegador.
- `<span></span>` : Esta etiqueta no aplica ningún formato por si misma, sino que provee una forma de definir un estilo o formato a un trozo de texto. Se utiliza junto con una hoja de estilo. Por ejemplo, lo podemos utilizar para marcar palabras en algún color o con algún formato especial.

A continuación se incluye un código de ejemplo en el que se ha ampliado el ejemplo del "Hola Mundo" y se han añadido algunas de las etiquetas HTML que hemos visto:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Ejemplo</title>
    <meta charset="utf-8">
</head>
<body>
   <h1>¡Mi primera Web!</h1>

   <p>
        Ejemplo de web con textos en <strong>negrita</strong>, <em>cursiva</em>
        y <u>subrayado</u>.
   </p>

   <blockquote>
        Ejemplo de nota, como se puede ver se le aplica un margen por la izquierda.
   </blockquote>

   <hr/>

   <!-- Este texto es un comentario que no se verá en el navegador!! -->

   <h2>Otro encabezado</h2>

   <p>
           Texto de ejemplo con superíndices (m<sup>2</sup>) y subíndices (H<sub>2</sub>O).
   </p>

</body>
</html>
```

Si guardamos este código en un fichero con extensión "html" (por ejemplo "index.html") y lo abrimos se nos mostraría un resultado similar al siguiente:

¡Mi primera Web!

Ejemplo de web con textos en **negrita**, *cursiva* y subrayado.

> Ejemplo de nota, como se puede ver se le aplica un margen por la izquierda.

Otro encabezado

Texto de ejemplo con superíndices (m^2) y subíndices (H_2O).

Listas

Para definir una lista utilizamos las siguientes etiquetas:

- `<ol></ol>` : Lista ordenada (con numeración).
- `<ul></ul>` : Lista con puntos (o viñetas).

Las etiquetas `<ol></ol>` y `<ul></ul>` se utilizan como etiquetas contenedoras de los elementos de la lista, dentro de las cuales tendremos que utilizar la etiqueta `<li></li>` para ir añadiendo cada uno de los elementos de la misma.

Por ejemplo, para crear una lista ordenada con dos elementos:

```
<ol>
   <li>Elemento 1</li>
   <li>Elemento 2</li>
</ol>
```

Con lo que obtendríamos un resultado similar al siguiente:

1. Elemento 1
2. Elemento 2

O por ejemplo, para crear una lista con puntos o viñetas de tres elementos usaríamos el siguiente código:

```
<ul>
   <li>Elemento 1</li>
   <li>Elemento 2</li>
   <li>Elemento 3</li>
</ul>
```

Con lo que obtendríamos un resultado similar al siguiente:

- Elemento 1
- Elemento 2
- Elemento 3

Dentro de las etiquetas `<li>` a su vez podemos usar otras etiquetas, como por ejemplo poner negritas, cursivas, encabezados, o cualquier otra cosa que se nos ocurra, por ejemplo:

```
<ul>
   <li><em>Elemento 1</em>: El elemento 1 es el <u>principal</u> de la lista.</li>
   <li><em>Elemento 2</em>: El elemento 2 es muy <strong>peligroso!</strong></li>
</ul>
```

Con este código HTML obtendríamos un resultado similar al siguiente:

- *Elemento 1*: El elemento 1 es el principal de la lista.
- *Elemento 2*: El elemento 2 es muy **peligroso!**

Dado que dentro de un elemento de la lista podemos poner cualquier otra etiqueta, también podremos poner otras listas. Esto nos permitirá es hacer listas anidadas de la forma:

```
<h1>Menú</h1>

<ol>
    <li>
        Primer plato
        <ul>
            <li>Ensalada</li>
            <li>Gazpacho andaluz</li>
       </ul>
    </li>
    <li>
        Segundo plato
        <ul>
            <li>Macarrones</li>
            <li>Pollo al horno</li>
            <li>Pescado</li>
        </ul>
    </li>
    <li>
        Postre
        <ul>
            <li>Tarta</li>
            <li>Yogur</li>
            <li>Café</li>
        </ul>
    </li>
</ol>
```

Con lo que obtendríamos una lista como la siguiente:

Menú

1. Primer plato
    - Ensalada
    - Gazpacho andaluz
2. Segundo plato
    - Macarrones
    - Pollo al horno
    - Pescado
3. Postre
    - Tarta
    - Yogur
    - Café

Tablas

Las tablas se definen básicamente mediante tres etiquetas:

- `<table></table>` : define una tabla.
- `<tr></tr>` : fila de una tabla, debe de estar dentro de las etiquetas de una tabla.
- `<td></td>` : celda de una tabla, debe estar dentro de una fila.

Ejemplo de una tabla:

```
<table>
   <tr>
      <td>Fila 1 izquierda</td>
      <td>Fila 1 derecha</td>
   </tr>
   <tr>
      <td>Fila 2 izquierda</td>
      <td>Fila 2 derecha</td>
   </tr>
</table>
```

Con lo que obtendríamos un resultado similar al siguiente:

Fila 1 izquierda	Fila 1 derecha
Fila 2 izquierda	Fila 2 derecha

Además también podemos utilizar la etiqueta `<th>` en lugar de `<td>` para indicar una celda de "cabecera", de esta forma el contenido será resaltado en negrita y en un tamaño ligeramente superior al normal. Por ejemplo, para crear una tabla con dos elementos de cabecera y dos celdas normales:

```
<table>
   <tr>
      <th>Cabecera 1</th>
      <th>Cabecera 2</th>
   </tr>
   <tr>
      <td>Celda 1</td>
      <td>Celda 2</td>
   </tr>
</table>
```

En la etiqueta de apertura `<table>` podemos utilizar los siguientes atributos:

- `border="num"` : Ancho del borde de la tabla en puntos. Si indicamos border="0" tendremos una tabla cuyas divisiones no serán visibles, esta propiedad se suele utilizar para distribuir los elementos en una página Web.
- `cellspacing="num"` : Espacio en puntos que separa las celdas que están dentro de la tabla.
- `cellpadding="num"` : Espacio en puntos que separa el borde de cada celda y el contenido de esta.
- `width="num"` : Indica la anchura de la tabla en puntos o en porcentaje en función del ancho de la ventana. Si no se indica este parámetro, el ancho dependerá de los contenidos de las celdas.
- `height="num"` : Indica la altura de la tabla en puntos o en porcentaje en función del alto de la ventana. Si no se indica este parámetro, la altura dependerá de los contenidos de las celdas. Este atributo también se puede utilizar en las etiquetas `<tr>` para indicar la altura de cada fila de forma individual.

En las etiquetas de apertura de celda (`<td>` o `<th>`) podemos utilizar los siguientes atributos:

- `align="pos"` : Indica como se debe alinear el contenido de la celda, a la izquierda (left), a la derecha (right), centrado (center) o justificado (justify).
- `valign="pos"` : Indica la alineación vertical del contenido de la celda, en la parte superior (top), en la inferior (bottom), o en el centro (middle).
- `rowspan="num"` : Indica el número de filas que ocupará la celda. Por defecto ocupa una sola fila. Este atributo se utiliza para crear celdas "multifila", es decir, una celda que por ejemplo ocupe 3 filas. Tendremos que tener en cuenta que esa celda no se deberá de definir en las siguientes 2 filas (para esas filas se definirá una celda menos).
- `colspan="num"` : Indica el número de columnas que ocupará la celda. Por defecto ocupa una sola columna. Este atributo se utiliza para crear celdas "multicolumna", es decir, una celda que por ejemplo ocupe 3 columnas. Tendremos que tener en cuenta que en esa fila tendremos que definir 2 celdas menos.
- `width="num"` : Indica la anchura de la columna en puntos o en porcentaje en función del ancho de la ventana. Si no se indica este parámetro, el ancho dependerá del tamaño de los contenidos.

Cajas (etiqueta `<div>`)

La etiqueta `<div></div>` permite crear cajas contenedoras de otros elementos. Esta etiqueta no muestra (por defecto) ningún estilo ni formato visual, sino que es utilizada únicamente para organizar la disposición de los elementos en la página. Es muy sencillo indicar su posición de forma absoluta o relativa en la página y crear divisiones del espacio para distribuir los elementos.

Estas cajas pueden contener cualquier tipo de elemento (texto, imágenes, etc.) u otras etiquetas `<div>` para crear subdivisiones.

A continuación se incluye un pequeño ejemplo de su uso:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Ejemplo de uso de la etiqueta DIV</title>
    <meta charset="utf-8">
</head>
<body>
   <div>Menu superior</div>

   <div>
        <h1>Contenido central</h1>

        <div>
            Artículo 1
        </div>

        <div>
            Artículo 2
        </div>
   </div>

   <div>Pie de página</div>
</body>
</html>
```

Como se puede ver se ha utilizado una etiqueta `<div>` para crear una sección que contendrá el menú superior, otra para el contenido central y otra para el pie de página. La sección con el contenido central se divide a su vez en otras dos secciones o cajas que contendrán los artículos.

Si guardamos este código de ejemplo en un archivo y lo abrimos veremos que esta etiqueta no muestra ningún formato ni estilo, solamente nos aparecerán los textos que hemos puesto. Para poder completar este código e indicar la posición y estilos nos hará falta utilizar CSS. En la sección de "Introducción a CSS > Capas" se explica más a fondo el uso de esta etiqueta y como la podemos utilizar para alinear los contenidos o crear secciones con estilos definidos.

> Para alinear el contenido de una página se recomienda el uso de la etiqueta `<div>` junto con CSS. No es recomentable el uso de la etiqueta `<table>` para crear alineaciones.

Enlaces

Los enlaces permiten vincular partes del documento con otros documentos o con otras partes del mismo documento. Por ejemplo, que al pulsar con el ratón sobre un texto o sobre una imagen se nos redirija a una nueva Web con un contenido diferente.

Para crear un enlace se utiliza la etiqueta `<a href=""></a>` cuyo atributo href establece la dirección URL a la que apunta el enlace. Por ejemplo, un enlace a la Wikipedia sería de la forma:

```
<a href="http://es.wikipedia.org">Wikipedia</a>
```

Para crear un enlace a una sección de nuestra propia web únicamente nos hará falta escribir el nombre del fichero HTML, por ejemplo:

```
<a href="pagina2.html">Pulsa aquí</a>
```

Ejemplo

En este ejemplo vamos a crear un enlace desde la página principal de nuestro sitio web (almacenada en el fichero `index.html`) a una página secundaria con un artículo (que estaría en el fichero `articulo1.html`). Además, en la página secundaria añadiremos también un enlace para volver a la página principal.

El contenido de la página principal (fichero `index.html`) sería el siguiente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.d
td">
<html>
<head>
    <title>Ejemplo</title>
    <meta charset="utf-8">
</head>
<body>
   <h1>Web de ejemplo</h1>
   <p>Pulsa en <a href="articulo1.html">este enlace</a>
      para consultar nuestro primer artículo.</p>
</body>
</html>
```

El contenido de la página secundaria (fichero `articulo1.html`) sería el siguiente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Ejemplo</title>
    <meta charset="utf-8">
</head>
<body>
   <h1>Artículo 1</h1>
   <p>Texto de ejemplo del artículo 1.</p>
   <p>Pulsa en <a href="index.html">este enlace</a> para volver a la página principal.
</p>
</body>
</html>
```

Enlaces sobre otros elementos

También se pueden crear enlaces sobre otros elementos, como por ejemplo imágenes. Para esto simplemente tenemos que escribir la/s etiqueta/s de dicho elemento dentro del enlace:

```
<a href="dirección_URL"><img src="imagen.jpg"/></a>
```

Abrir una nueva ventana (o pestaña del navegador)

La etiqueta de enlace `<a>` también permite el atributo `target="_blank"`, mediante el cual indicamos que el enlace se tiene que abrir en una nueva ventana o en una pestaña nueva del navegador. Esta opción se suele utilizar a menudo para los enlaces externos para que no se cierre la web actual.

Por ejemplo, para añadir un enlace a una web externa que se abra en otra ventana para que el usuario consulte más información escribiríamos el siguiente código:

```
<a href="https://es.wikipedia.org/wiki/Polinomio" target="_blank">
    Para más información pulsa aquí
</a>
```

Dirección base

La dirección principal de la web no es necesario escribirla ya que se añadirá automáticamente. Por ejemplo, si nuestra web es "http://www.webejemplo.es" y en el enlace escribímos únicamente `<a href="pagina2.html">Pulsa aquí</a>`, al pulsar nos redirigirá a la web "http://www.webejemplo.es/pagina2.html".

Hay que tener **cuidado** cuando colocamos las páginas dentro de carpetas. Por ejemplo, si creamos una carpeta para meter todas las páginas con artículos y escribimos un enlace de la forma: `<a href="articulos/articulo1.html">Ir al artículo 1</a>`, esto nos redirigirá a la dirección "http://www.webejemplo.es/articulos/articulo1.html". El **problema** está en los enlaces que coloquemos dentro de una página que esté en una subcarpeta. Por ejemplo, si en la página "articulo1.html" (que está en la carpeta "articulos") añadimos un enlace para volver al índice de la forma: `<a href="index.html">Volver</a>`, este enlace en realidad nos llevaría a la dirección "http://www.webejemplo.es/**articulos/index.html**". Es decir, buscaría la página "index.html" dentro de la carpeta actual.

Para solucionar este problema y hacer referencia a la raíz de nuestro sitio web, se suele anteponer **siempre** la barra "/" a todas las direcciones. Por ejemplo, en el caso del enlace erróneo anterior tendríamos que escribir: `<a href="/index.html">Volver</a>`. Pero esta barra es recomendable escribirla siempre, en todas las direcciones, para evitar errores.

Imágenes

Para incluir una imagen se utiliza la etiqueta `<img src="" alt=""/>`, la cual requiere el atributo `src` con la ruta en la que se encuentra la imagen. Es conveniente poner siempre el atributo `alt="texto alternativo"`, el cual indica el texto a mostrar en caso de no poder cargar la imagen y también se utiliza para opciones de accesibilidad.

Por ejemplo, para cargar una imagen llamada "cabecera.jpg" utilizaremos la etiqueta de la forma:

```
<img src="cabecera.jpg" alt="Imagen de cabecera"/>
```

Además existen otros atributos interesantes como `width` y `height` para redefinir el ancho y la altura de la imagen. Sin embargo se recomienda indicar estas propiedades (ancho y alto) modificando los estilos de la etiqueta (ver capítulo "Introducción a CSS").

Formularios

Los formularios permiten solicitar información al visitante de una página Web. Están compuestos por campos de diferente tipo, cuya información se enviará a una dirección URL (indicada en el código) al pulsar el botón de envío.

La declaración de formulario queda recogida por las etiquetas `<form></form>`, las cuales deben encerrar la definición de todos los campos del formulario. En la etiqueta de apertura `<form>` tenemos que indicar los atributos básicos:

- `action=""`: Entre comillas se indica la acción a realizar al enviar el formulario. En general se indicará el nombre de un fichero alojado en el servidor, el cual se encargará de procesar la información. Aunque también se le puede indicar una dirección de correo para que envíe directamente todo el contenido, de la forma: `mailto:direccion_de_correo`.
- `method=""` (`post` o `get`): Indica el método de transferencia de las variables. El método " `post` " envía los datos de forma no visible, mientras que el método " `get` " los adjunta a la URL a la que se redirige.
- `enctype= ""` : Especifica el tipo de codificación de la información enviada. Con `method="get"` no se realiza codificación, solo se cambian caracteres especiales como el espacio, por lo que no es necesario indicar `enctype`. Cuando el valor del atributo " `method` " es " `post` ", podemos utilizar los siguientes valores:
  - `application/x-www-form-urlencoded` : Es el valor predeterminado. Codifica todos los caracteres antes de enviarlos.
  - `multipart/form-data` : Es requerido al enviar archivos mediante un formulario. No codifica la información.
  - `text/plain` : No codifica la información, solo cambia los espacios por el símbolo "+".

Por ejemplo, una posible cabecera de un formlario sería:

```
<form action="http://www.miweb.com/procesarformulario" method="POST">

    <!-- Como no indicamos el enctype se utilizará la condificación por defecto -->

    <!-- Campos del formulario -->
    <!-- Campos del formulario -->
    <!-- Campos del formulario -->

</form>
```

Tipos de campos básicos

Para añadir campos al formulario se utiliza la etiqueta `<input/>`, esta etiqueta debe de tener siempre dos atributos:

- `name=""` : Indica el nombre que se asigna a un determinado campo. Este nombre no aparece visible en la Web, pues se utiliza para poder distinguir cada campo al enviar la información al servidor o por correo. Es como si fuera el nombre de la variable a la que se asigna el valor del campo.
- `type=""` : Indica el tipo de campo a utilizar. Puede ser de muchos tipos: text, password, checkbox, radio, file, hidden, submit, reset.

A continuación se describen más detalladamente los diferentes tipos de campos `<input/>` según su valor `type` :

- `type="text"` : campo de tipo texto de una línea. Sus atributos son:
  - `maxlenght=""` : Seguido de un valor que limitará el número máximo de caracteres.
  - `size=""` : Seguido de un valor que limitará el número de caracteres a mostrar en pantalla. A diferencia de maxlenght este atributo no limita la longitud del texto que se puede introducir, sino que modifica el tamaño visible del campo.
  - `value=""` : Indica el valor inicial del campo. A continuación se incluye un ejemplo de uso:

    ```
    <input name="usuario" type="text" maxlenght="24"/>
    ```

- `type="password"` : Este campo funciona exactamente igual que el de tipo "text", pero ocultará el texto introducido cambiando las letras por asteriscos o puntos. Sus atributos son los mismos que para "text".
- `type="checkbox"` : Este campo mostrará una casilla cuadrada que nos permitirá marcar opciones de una lista (podremos marcar varias opciones a la vez). Para indicar que varias casillas pertenecen al mismo grupo se les debe de dar el mismo nombre para el

atributo "name". El texto que queramos que aparezca a continuación de la casilla del "checkbox" se tendrá que escribir después de cerrar la etiqueta `<input/>`. Sus atributos son:

  - `value=""` : Define el valor que será enviado si la casilla está marcada.
  - `checked` : Este atributo es opcional, y hace que la casilla aparezca marcada por defecto. No necesita indicarle ningún valor. Ejemplo:

```
<input type="checkbox" name="option1" value="leche"/> Leche<br/>
<input type="checkbox" name="option1" value="pan" checked/> Pan<br/>
<input type="checkbox" name="option1" value="queso"/> Queso<br/>
```

- `type="radio"` : El campo se elegirá marcando de entre varias opciones una casilla circular. Al marcar una casilla el resto de casillas de ese grupo de desmarcarán automáticamente. Para indicar que varias casillas pertenecen al mismo grupo se les debe de dar el mismo nombre para el atributo "name" (ver ejemplo). Además debemos de indicar:
  - `value=""` : Define el valor que será enviado si la casilla está marcada.
  - `checked` : Este atributo es opcional, y hace que la casilla aparezca marcada por defecto. Solo se podrá usar para una casilla. No necesita indicarle ningún valor. Ejemplo:

```
<input type="radio" name="group1" value="leche"/> Leche<br/>
<input type="radio" name="group1" value="pan" checked/> Pan<br/>
<input type="radio" name="group1" value="queso"/> Queso<br/>
```

- `type="file"` : El atributo file permite al usuario subir archivos. Necesitaremos un programa que gestione estos archivos en el servidor mediante un lenguaje diferente al HTML. El único atributo opcional que podemos utilizar es `size=""` mediante el cual podremos indicar la anchura visual de este campo. Ejemplo:

```
<input type="file" name="datafile" size="40"/>
```

- `type="hidden"` : Este valor no puede ser modificado, pues permanece oculto. Se suele utilizar para enviar al método encargado de procesar el formulario algún dato adicional necesario para su procesamiento. Para indicar el valor de este campo utilizamos el atributo: `value = "valor"` .
- `type="submit"` : Representa el botón de "Enviar". Al pulsar este botón la información de todos los campos se enviará realizando la acción indicada en `<form>` . Mediante el atributo:
  - `value="texto"` : podemos indicar el texto que aparecerá en el botón.

- `type="reset"` : Al pulsar este botón se borra el contenido de todos los campos del formulario. Mediante el atributo:
  - `value="texto"` : podemos indicar el texto que aparecerá en el botón.

Etiquetas

Las etiquetas se utilizan para poner un texto o descripción de los campos de un formulario. Se escriben usando la etiqueta HTML `<label>` y tienen un único atributo " `for` " que se utiliza para indicar el nombre (atributo `name`) del campo asociado. Por ejemplo:

```
<label for="campo1">Etiqueta</label>
<input name="campo1" type="text"/>
```

Campos de Selección

Mediante la etiqueta `<select></select>` podemos crear listas de opciones, que nos permitirán seleccionar entre una o varias de ellas. Sus atributos son:

- `name=""` : Nombre del campo.
- `size=""` : Número de opciones visibles a la vez. Si no se indica nada o se le asigna un valor de uno se presentará como un menú desplegable. En el caso de valores mayores que uno aparecerá como una lista con una barra de desplazamiento.
- `multiple` : Permite seleccionar mas de un valor a la vez para el campo.

Las diferentes opciones de la lista se indicarán mediante la etiqueta `<option></option>` . El nombre que se visualizará debe de indicarse dentro de estas etiquetas. Mediante el atributo `value=""` podemos indicar el valor que se enviará con el formulario. También podemos utilizar el atributo `selected` para indicar la opción seleccionada por defecto. Si no lo especificamos, siempre aparecerá como seleccionado el primer elemento de la lista.

```
<select name="Colores" multiple>
   <option value="r">Rojo</option>
   <option value="g">Verde</option>
   <option value="b">Azul</option>
</select>
<select name="Colores" SIZE="1">
   <option value="r">Rojo</option>
   <option value="g" selected>Verde</option>
   <option value="b">Azul</option>
</select>
```

Áreas de texto

Mediante las etiquetas `<textarea></textarea>` podemos crear un campo de texto de múltiples líneas. Los atributos que podemos utilizar son:

- `name=""` : Nombre del campo.
- `cols="num"` : Número de columnas de texto visibles. Este atributo es opcional.
- `rows="num"` : Número de filas de texto visibles. Este atributo es opcional.

Ejemplo

A continuación se incluye un ejemplo de uso de los formularios en el que se han incluido la mayoría de los campos que hemos visto:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.d
td">
<html>
<head>
   <title>Ejemplo</title>
   <meta charset="utf-8"/>
</head>
<body>
    <h2>Formulario de registro</h2>
    <p>Escribe tus datos de usuario en el siguiente formulario y
        por último aprieta el botón enviar.</p>

    <form action="http://www.miweb.com/procesarformulario" method="POST">
        <p>
            <label for="usuario">Usuario: </label>
            <input name="usuario" type="text" maxlenght="32"/>
        </p>
        <p>
            <label for="password">Contraseña: </label>
            <input name="password" type="password" maxlenght="16"/>
        </p>
        <p>
            <label for="nombre">Nombre completo: </label>
            <input name="nombre" type="text" maxlenght="64"/>
        </p>
        <p>
            <label for="direccion">Dirección: </label>
            <input name="direccion" type="text" maxlenght="128"/>
        </p>
        <p>
            <label for="ciudad">Ciudad: </label>
            <select name="ciudad">
                <option value="A Coruña">A Coruña</option>
                <option value="Álava">Álava</option>
```

```
                <option value="Albacete">Albacete</option>
                <option value="Alicante">Alicante</option>
                <option value="Almería">Almería</option>
                <option value="Asturias">Asturias</option>
                <!-- resto de ciudades... -->
            </select>
        </p>
        <p>
            <label for="tipo">Tipo de cliente: </label><br/>
            <input type="radio" name="tipo" value="particular" checked/> Particular<br
/>
            <input type="radio" name="tipo" value="profesional"/> Profesional<br/>
        </p>
        <p>
            <label for="comentarios">Comentarios: </label><br/>
            <textarea name="comentarios" rows="4" cols="50"></textarea>
        </p>
        <p>
            <input type="checkbox" name="terminos" value="terminos"/>
            <label for="terminos">Acepto los términos y condiciones de uso</label>
        </p>
        <p>
            <input type="submit" value="Enviar"/>
            <input type="reset" value="Borrar formulario"/>
        </p>
    </form>

</body>
</html>
```

Como se puede ver cada par de etiqueta y campo de formulario se ha encerrado dentro de un párrafo `<p>` para que ocupe una única línea y que el siguiente campo baje a la línea siguiente. Este mismo efecto se podría conseguir usando la etiqueta `<div>`. Para mejorar el aspecto visual del formulario se tendrían que aplicar estilos CSS, los cuales se tratarán en el siguiente capítulo.

Si guardamos este código en un fichero ".html" y lo abrimos con el navegador obtendríamos un resultado similar al siguiente:

Formulario de registro

Escribe tus datos de usuario en el siguiente formulario
y por último aprieta el botón enviar.

Usuario:

Contraseña:

Nombre completo:

Dirección:

Ciudad: A Coruña

Tipo de cliente:
- (•) Particular
- ( ) Profesional

Comentarios:

[ ] Acepto los términos y condiciones de uso

Enviar | Borrar formulario

Eventos

En esta sección de describe un concepto un poco más avanzado: los eventos. Un evento, como su nombre indica, es cuando sucede una determinada acción sobre un elemento. HTML permite escuchar estos eventos y asociarles un comportamiento o acción que se realizará cuando suceda dicho evento. La forma de definirlos es similar a los atributos (`evento="ACCION"`), la acción hará referencia a una función o método en lenguaje JavaScript. Algunos de los eventos que podemos utilizar son:

- **onload**: se activa cuando el navegador termina de cargar todos los elementos de la página.
- **onclick**: cuando se presiona el botón del ratón sobre un elemento.
- **onmouseover**: se dispara cuando el cursor del ratón pasa sobre un elemento.
- **onmousemove**: cuando se mueve el cursor del ratón mientras está sobre un elemento.
- **onmouseout**: se activa cuando el cursor del ratón sale fuera de un elemento (sobre el que estaba).
- **onfocus**: ocurre cuando un elemento recibe el enfoque (el cursor de escritura), ya sea con el puntero o con mediante la tecla tabulador.
- **onkeypress**: ocurre cuando se presiona una tecla (dentro de un elemento, por ejemplo un campo de escritura).
- **onkeydown**: se dispara cuando una tecla es presionada (dentro de un elemento)
- **onkeyup**: cuando una tecla es soltada.
- **onsubmit**: se activa cuando un formulario es enviado.
- **onreset**: ocurre cuando un formulario es reseteado.
- **onchange**: ocurre cuando un control pierde el enfoque y su valor ha sido modificado desde que recibió el enfoque.
- etc. (Ver sección eventos en el capítulo de JavaScript)

Por ejemplo, podemos enlazar el evento " `onkeyup` " de un `textarea` con una función de JavaScript de la forma:

```
<script type="text/javascript">
   function saveText() {
      // acciones JavaScript
   }
</script>

<textarea id="myarea" cols="80" rows="15" onkeyup="saveText()"></textarea>
```

Los eventos se verán más en detalle en el capítulo correspondiente de la sección dedicada a Javascript (Ver sección eventos). De momento, solo es importante que aprendráis que se puede asociar código JavaScript a determinados eventos o acciones que se producen en los campos HTML de una página Web.

Símbolos HTML

Los caracteres especiales como signo de puntuación, letras con tilde o diéresis, o símbolos del lenguaje; se deben convertir en entidades HTML para que se muestren correctamente en un navegador. La siguiente es una lista de caracteres españoles junto con algunos símbolos especiales y su correspondiente entidad HTML:

Caracter	Código		Caracter	Código
á	`&aacute;`		Á	`&Aacute;`
é	`&eacute;`		É	`&Eacute;`
í	`&iacute;`		Í	`&Iacute;`
ó	`&oacute;`		Ó	`&Oacute;`
ú	`&uacute;`		Ú	`&Uacute;`
ü	`&uuml;`		Ü	`&Uuml;`
ñ	`&ntilde;`		Ñ	`&Ntilde;`
espacio en blanco	` `		€	`&euro;`
`<` (Menor que)	`<`		`>` (Mayor que)	`>`
&	`&`		º (grados)	`&deg;`

Recordad que para escribir letras acentuadas u otros símbolos y que el navegador los muestre correctamente símplemente tenemos que añadir la cabecera `meta` con la codificación:

```
<meta charset="utf-8"/>
```

Sin embargo hay determinados caracteres que si queremos escribirlos nos veremos obligados a escribir el código del símbolo. Por ejemplo, HTML solamente muestra o renderiza un espacio en el navegador aunque nosotros escribamos muchos espacios. Si por alguna razón queremos dar varios espacios tendremos que escibir el símbolo "` `". O si por ejemplo queremos poner los símbolos de mayor (`>`) o menor (`<`) y no correr el riesgo de que se confunda con el inicio o cierre de una etiqueta HTML, también tendremos que escribir el código correspondiente.

Para obtener una lista mucho más completa de símbolos podemos buscar en Google: "*HTML symbols*" o visitar la siguiente dirección http://www.ascii.cl/htmlcodes.htm.

Introducción a CSS

El nombre hojas de estilo en cascada viene del inglés *Cascading Style Sheets*, del que toma sus siglas. CSS es un lenguaje usado para definir la presentación o la apariencia de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). CSS se creó para separar el contenido de la forma, a la vez que permite a los diseñadores mantener un control mucho más preciso sobre la apariencia de las páginas. El W3C (*World Wide Web Consortium*) es el encargado de formular la especificación de las hojas de estilo que sirven de estándar para los navegadores.

En versiones antiguas de HTML se debía de añadir el formato dentro de las propias etiquetas, para indicar por ejemplo su color o tamaño. Esto obligaba a tener que especificar el mismo formato en todas las etiquetas para tener un diseño consistente, además, al cambiar un formato también había que cambiarlo para todas las etiquetas.

Cuando se utiliza CSS, las etiquetas HTML no deberían proporcionar información sobre cómo serán visualizadas. La información de la hoja de estilo será la que especifique cómo se han de mostrar: color, fuente, alineación del texto, tamaño, etc.

Las ventajas de utilizar CSS (u otro lenguaje de estilo) son:

- Control centralizado de la apariencia de un sitio web completo, con lo que se agiliza de forma considerable la actualización del mismo.
- Los navegadores permiten a los usuarios especificar su propia hoja de estilo local, que será aplicada a un sitio web, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, personas con deficiencias visuales pueden configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.
- Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o, incluso, a elección del usuario. Por ejemplo, para ser impresa o mostrada en un dispositivo móvil.
- El documento HTML en si mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

Adjuntar una hoja de estilo

La información de estilo puede ser adjuntada de tres formas diferentes:

- **Hoja de estilo externa**: es una hoja de estilo que está almacenada en un archivo diferente al archivo HTML (por ejemplo llamado "estilo.css"). Esta es la manera de programar más potente y la que deberemos de utilizar por defecto, ya que separa completamente las reglas de estilo de la página HTML y además nos permite usar dicha hoja de estilo en todas las páginas HTML que queramos. La hoja de estilo debe de ser enlazada con el código HTML de la forma:

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

- **Hoja de estilo interna**: es una hoja de estilo que está incrustada dentro del documento HTML. En general, el motivo para usar una hoja de estilo interna, es cuando se quiere diferenciar con algún estilo uno de los ficheros HTML de nuestra Web. Este código debe de estar incluido en la sección de cabecera `<head>` y entre las etiquetas `<style>` . La forma de incluir el código sería de la forma:

```
<html>
<head>
   <style type="text/css">
      H1 { color:blue; text-align:center }
   </style>
</head>
...
```

- **Estilo en línea (inline):** es un método para insertar los estilos CSS directamente dentro de una etiqueta HTML. Esta opción la usaremos **únicamente** cuando queramos aplicar un estilo sobre una única etiqueta o un único elemento. En cualquier otro caso usaremos alguna de las opciones anteriores, ya que si por ejemplo aplicamos un mismo estilo a muchas etiquetas usando esta opción y después queremos cambiar algo de ese estilo tendríamos que cambiarlo en todas las etiquetas, mientras que si tenemos una hoja de estilo centralizada con un único cambio será suficiente. Para incluir un estilo en línea o *inline* se usa el atributo `style` de la forma:

```
<h1 style="color:blue; text-align:center">...</h1>
```

Definición de estilos para etiquetas HTML

Si lo que queremos es dar formato o redefinir una etiqueta HTML existente, usaríamos la sintaxis:

```
etiqueta {
   estilo CSS 1;
   estilo CSS 2;
   ...
}
```

En "etiqueta" pondríamos el nombre de la etiqueta (por ejemplo "`h1`", "`p`", etc. pero sin los signos `< >`) y los estilos que definirían esa etiqueta irían encerrados entre las llaves "`{...}`". Por ejemplo:

```
h1 {
   estilo CSS 1;
   estilo CSS 2;
   ...
}
```

Comentarios

En las hojas de estilo también se pueden escribir comentarios usando los símbolos: `/* texto del comentario */`. Pero es importante usar esta notación y no ninguna otra, ya que es la única soportada. A continuación se incluye un ejemplo con comentarios:

```
/* Definimos el estilo de la cabecera principal */
h1 {
   estilo CSS 1; /* Cambiamos el estilo de su... */

   /* También cambiamos este otro estilo porque... */
   estilo CSS 2;
}
```

Definición de varios estilos a la vez

También podemos redefinir varias etiquetas a la vez que compartirán los mismos estilos, separándolas por comas, de la forma:

```
etiqueta1, etiqueta2, etiqueta3 {
   /* estilos CSS */
}
```

> En esta sección y la siguiente nos centraremos las cabeceras de los estilos y dejaremos los estilos CSS que podemos utilizar para la sección "estilos CSS básicos" y siguientes. Por este motivo escribiremos `/* estilos CSS */` en el lugar donde irán los estilos que definirá la etiqueta.

Imaginad por ejemplo que queréis cambiar el color de todas las cabeceras, lo podéis hacer a la vez escribiendo:

```
h1, h2, h3, h4, h5, h6 {
   /* estilos CSS */
}
```

Estilos anidados

Otra opción interesante es definir el estilo de etiquetas "dentro" de otras etiquetas. Para esto tenemos que escribir primero la etiqueta contendora, seguida de un espacio y por último la etiqueta a definir. En este caso el estilo CSS solo se aplicará cuando la *etiqueta definida* se encuentre dentro de la *etiqueta contenedora*:

```
contenedor etiqueta {
   /* estilos CSS */
}
```

Por ejemplo, una etiqueta `<span>` dentro de una sección `<p>`:

```
p span {
   /* estilos CSS */
}
```

Este estilo solo se aplicaría cuando se encuentre la etiqueta `<span>` dentro de una sección `<p>` de la forma:

```
<p>
    Párrafo de ejemplo donde
    <span>el estilo solo se aplicará sobre este texto</span>
    y no sobre el resto del texto.
</p>
```

Esta opción es muy útil pues nos permitirá definir diferentes estilos para la misma etiqueta dependiendo de donde se encuentre.

Identificadores y Clases

A veces tenemos varias etiquetas del mismo tipo pero queremos aplicar diferentes estilos según donde estén. Para esto usamos los identificadores y las clases.

La principal diferencia entre ellos es que los identificadores tienen que ser **únicos en todo el documento HTML** mientras que las clases pueden repetirse todas las veces que queramos. La otra diferencia es la forma de definirlos y de utilizarlos: En HTML para indicar el identificador de una etiqueta usaremos el atributo " `id` ", mientras que para indicar la clase usaremos " `class` ":

```
<div id="capitulo2">
   <p>...</p>
   <p class="parrafogris">....</p>
   <p>...</p>
   <p class="parrafogris">....</p>
   <p>...</p>
   <p class="parrafogris">....</p>
</div>
```

En este ejemplo se asigna el identificador "capitulo2" a la etiqueta `<div>` inicial. Esta etiqueta sería una sección **única** en todo el documento sobre la cual podemos aplicar un estilo concreto. El estilo de la clase "parrafogris" se aplicaría sobre las etiquetas " `p` " indicadas, y como se puede ver si ha aplicado varias veces.

Otra diferencia entre identificadores y clases es la forma de definir sus estilos CSS en la hoja de estilos. Para indicar un identificador escribiremos su nombre precedido por una almohadilla " `#` ", y para referenciar una clase usaremos como prefijo el punto ".", por ejemplo:

```
#identificador {
    /* estilos CSS */
}
.clase {
    /* estilos CSS */
}
```

Por ejemplo, para indicar los estilos del ejemplo anterior, escribiríamos el siguiente código:

```
#capitulo2 {
    /* estilos CSS */
}
.parrafogris {
    /* estilos CSS */
}
```

> Es importante diferenciar cuando tenemos que usar la almohadilla " `#` " y el punto ".", los cuales solo los pondremos en la hoja de estilos y **no** en el código HTML. Esto es un error común y haría que los estilos no funcionasen. Es decir, si escribimos `<div id="#capitulo2">` (con " `#` ") o escribimos `<p class=".parrafogris">` (con ".") sería un **error** y no funcionaría.
>
> Los identificadores se suelen usar menos que las clases y solo para elementos específicos que se quieren diferenciar. Normalmente se aplican sobre etiquetas "neutras" como `<div>` o `<span>` para marcar partes de un documento y después indicar sus estilos (como por ejemplo identificar la cabecera, un logotipo, el menú principal, etc.).

Definición de varios estilos a la vez

Igual que hemos visto antes, podemos definir estilos a la vez para varios identificadores y clases:

```
#capitulo1, #capitulo2, #capitulo3 {
    /* estilos CSS */
}
.parrafogris, .parraforojo, .parrafoverde {
    /* estilos CSS */
}
```

Podemos mezclar identificadores, con clases y con etiquetas sin problema:

```
#capitulo1, .parrafogris, h1 {
    /* estilos CSS */
}
```

Anidación de estilos

Podemos aplicar estilos a identificadores y clases solo cuando cuando estén dentro de otros:

```
#capitulo1 #cabecera {
    /* estilo a aplicar a #cabecera solo cuando esté dentro de #capitulo1 */
}
.parrafogris .resaltado {
    /* estilo a aplicar a .resaltado solo cuando esté dentro de .parrafogris */
}
```

Igual que antes también podemos combinar identificadores, con clases y con etiquetas sin problema:

```
#cabecera h1 {
    /* estilos a aplicar a h1 solo cuando esté dentro de la sección #cabecera */
}
.parrafogris span {
    /* estilos a aplicar a la etiqueta span solo cuando esté dentro de .parrafogris */
}
```

Si queremos podemos crear más niveles de profundidad, por ejemplo:

```
#cabecera p .resaltado {
    /* estilos a aplicar a ".resaltado" solo cuando esté dentro
    de una etiqueta "p" que a su vez esté dentro de la sección #cabecera */
}
```

Filtrar etiquetas con estilos

También podemos aplicar estilos filtrando por etiquetas que tenga una determinada clase, por ejemplo:

```
etiqueta1.clase1 {
   /* estilos CSS */
}
```

En este caso sólo se aplicaría el estilo a las etiquetas "etiqueta1" que se marque que son de la clase "clase1", por ejemplo: `<etiqueta1 class="clase1">...</etiqueta1>`. Si intentáramos aplicar esta clase a una etiqueta diferente no funcionaría.

Por ejemplo, el estilo:

```
h1.resaltado {
   /* estilos CSS */
}
```

Solo se aplicaría a las cabeceras `h1` que tengan aplicada la clase `.resaltado` de la forma: `<h1 class="resaltado">...</h1>`

Estilos CSS básicos

La sintaxis básica para definir un estilo es:

```
atributo: valor;
```

Los diferentes estilos siempre se separan con punto y coma (;), y después del nombre se pone dos puntos (y no un igual "=", que es un error típico al confundirse con el HTML). Por ejemplo, si queremos definir una clase que aplique tres estilos usaríamos la notación:

```
.estilo_de_ejemplo {
    atributo: valor;
    atributo: valor;
    atributo: valor;
}
```

> Es importante usar esta notación correctamente ya que si se nos olvida el punto y coma (;) o no ponemos los dos puntos (:) los estilos no funcionarán.

Muchos de los valores que podemos aplicar a un atributo de estilo tendrán **unidades de medida**, por ejemplo, el valor del tamaño de un margen o el tamaño de la fuente. Las unidades de medida que podemos utilizar son:

- pixels (px)
- puntos (pt)
- centímetros (cm)
- pulgadas (in)

A continuación se incluye un resumen de los principales estilos CSS y los valores que se les pueden aplicar a los siguientes elementos:

- Estilos para textos
- Estilos para párrafos
- Estilos para fondos
- Estilos para cajas

Estilos para textos

En esta sección se describen los principales estilos CSS que podemos utilizar para cambiar la apariencia de los textos de una Web. Para cada uno de ellos se indica el nombre del atributo, los posibles valores que le podemos asignar, algunos ejemplos y una explicación de uso.

- **color:** valor RGB o nombre de color
  Ejemplos: color: #009900; color: red;
  Sirve para indicar el color del texto. Lo admiten casi todas las etiquetas de HTML. No todos los nombres de colores son admitidos en el estándar, es aconsejable entonces utilizar el valor RGB. Algunos de los principales nombres de colores son: white, black, gray, blue, red, green o yellow, para más nombres podemos consultar la dirección "http://www.w3schools.com/colors/colors_names.asp".

- **font-size:** xx-small|x-small|small|medium|large|x-large|xx-large|Unidades CSS
  Ejemplos: font-size: 12pt; font-size: x-large;
  Sirve para indicar el tamaño de las fuentes de manera más rígida y con mayor exactitud.

- **font-family:** serif | sans-serif | cursive | fantasy | monospace | Etc.
  Ejemplos: font-family: arial,helvetica; font-family: fantasy;
  Con este atributo indicamos la familia de tipografía del texto. Los primeros valores son genéricos (serif, sans-serif, etc.), es decir, los navegadores las comprenden y utilizan las fuentes que el usuario tenga en su sistema.
  También se pueden definir con tipografías normales. Si el nombre de una fuente tiene espacios se utilizan comillas para que se entienda bien.

- **font-weight:** normal | bold | bolder | lighter | 100 | 200 | 300 | ... | 900
  Ejemplos: font-weight: bold; font-weight: 200;
  Sirve para definir la anchura de los caracteres, o dicho de otra manera, para poner negrita con total libertad.
  Normal y 400 son el mismo valor, así como bold y 700.

- **font-style:** normal | italic | oblique
  Ejemplos: font-style: normal; font-style: italic;
  Es el estilo de la fuente, que puede ser normal, itálica u oblicua. El estilo "oblique" es similar al "italic".

- **text-decoration:** none | underline | overline | line-through
  Ejemplos: text-decoration: none; text-decoration: underline;
  Establece la decoración de un texto, si está subrayado, sobre-rayado o tachado.
- **text-transform:** capitalize | uppercase | lowercase | none
  Ejemplos: text-transform: none; text-transform: capitalize;
  Nos permite transformar el texto, para que tenga la primera letra en mayúsculas de todas las palabras, o todo en mayúsculas o minúsculas.

Ejemplos

Por ejemplo, para definir un párrafo en negrita, cursiva y además cambiar el color podemos definir estos estilos *inline* en el atributo `style` del propio párrafo:

```
<p style="font-weight:bold; font-style:italic; color:red">
    Texto en negrita, cursiva y en color rojo.
</p>
```

Si quisieramos cambiar solamente el estilo de una o varias palabras dentro de un párrafo podemos utilizar la etiqueta `<span>` y asignarle dichos estilos en su atributo `style` :

```
<p>
    Este párrafo contiene una
    <span style="font-weight:bold">sección en negrita</span>.
</p>
```

La definición de estilos dentro de la etiqueta `style` se suele hacer cuando dicho estilo solo se va a aplicar de forma puntual. Pero si por el contrario queremos definir un estilo que lo vamos a aplicar varias veces en un sitio Web lo más aconsejable es declarar una clase en una hoja de estilos externa o interna. Por ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Ejemplo</title>
    <meta charset="utf-8"/>
    <style type="text/css">
        h1 {
            color: blue;
        }
        p {
            color: black;
            font-size: 12pt;
        }
        .parrafo-aviso {
            font-weight:bold;
            font-style:italic;
            color:red;
        }
        .nota {
            text-align: right;
            font-size: 10pt;
            color: gray;
        }
    </style>
</head>
<body>
    <h1>Artículo 1</h1>
    <p>
        Párrafo normal al que se le aplica el color negro y un tamaño de 12pt.
    </p>
    <p class="parrafo-aviso">
        Texto de aviso destacado en color rojo, cursiva y negrita.
    </p>
    <p class="nota">
        Nota alineada a la derecha, en color gris y con tamaño de letra 10pt.
    </p>
</body>
</html>
```

Estilos para párrafos

Los estilos para párrafos nos permiten cambiar propiedades de todo un párrafo o bloque de texto, como por ejemplo el espaciado entre las líneas, la alineación, etc.

- **line-height:** normal | unidades CSS
  Ejemplos: line-height: 12px; line-height: normal;
  El alto de una línea, y por tanto, el espaciado entre líneas. Es una de esas características que no se podían modificar utilizando HTML.
- **text-align:** left | right | center | justify
  Ejemplos: text-align: right; text-align: center;
  Sirve para indicar la alineación del texto. Es interesante destacar que las hojas de estilo permiten el justificado de texto, aunque recuerda que no tiene por que funcionar en todos los sistemas.
- **text-indent:** Unidades CSS
  Ejemplos: text-indent: 10px; text-indent: 2in;
  Un atributo que sirve para hacer sangrado o márgenes en las páginas.

Ejemplos

Para alinear el texto de un párrafo a la derecha podemos utilizar:

```
<p style="text-align:right">
    Texto alineado a la derecha
</p>
```

En este otro ejemplo vamos a colocar un texto centrado, con un interlineado mayor, con el texto en cursiva y gris. Además definiremos la hoja de estilo en la cabecera de la página:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.d
td">
<html>
<head>
    <title>Ejemplo</title>
    <meta charset="utf-8"/>
    <style type="text/css">
        h1 {
            color: #222;
            text-align: center;
        }
        .poema {
            text-align: center;
            font-style:italic;
            line-height: 30pt;
            font-size: 12pt;
            color: #222;
        }
        .autor {
            font-weight:bold;
            color:gray;
            font-size: 10pt;
            text-align: center;
        }
    </style>
</head>
<body>
    <h1>Me gusta cuando callas</h1>
    <p class="poema">
        Me gustas cuando callas porque estás como ausente, <br/>
        y me oyes desde lejos, y mi voz no te toca. <br/>
        Parece que los ojos se te hubieran volado <br/>
        y parece que un beso te cerrara la boca.
    </p>
    <br/>
    <p class="autor">
        Pablo Neruda
    </p>
</body>
</html>
```

Con lo que obtendríamos un resultado similar al siguiente:

Me gusta cuando callas

Me gustas cuando callas porque estás como ausente,

y me oyes desde lejos, y mi voz no te toca.

Parece que los ojos se te hubieran volado

y parece que un beso te cerrara la boca.

Pablo Neruda

Estilos para fondos

- **background-color:** Un color, con su nombre o su valor RGB
  Ejemplos: background-color: green; background-color: #000055;
  Sirve para indicar el color de fondo de un elemento de la página.
- **background-image:** El nombre de la imagen con su camino relativo o absoluto
  Ejemplos: background-image: url(mármol.gif); background-image: url(http://www.url.com/fondo.gif)
  Permite colocar una imagen de fondo en cualquier elemento de la página.

Estilos para cajas

(`<div>` o `<table>`)

- **width:** Unidades CSS | Porcentaje
  height: Unidades CSS | Porcentaje
  Ejemplos: width: 50px; width: 100%; height: 15px;
  Permiten indicar el ancho y altura de un elemento. Se pueden aplicar sobre muchos elementos, como tablas, etiquetas div, imágenes, párrafos "p", etc. Con algunas etiquetas no funciona, tampoco sirve para indicar espaciado (padding), bordes o márgenes.

- **margin-left:** Unidades CSS
  Ejemplos: margin-left: 1cm; margin-left: 0,5in;
  Indica el tamaño del margen izquierdo.

- **margin-right:** Unidades CSS
  Ejemplos: margin-right: 5%; margin-right: 1in;
  Define el tamaño del margen derecho.

- **margin-top:** Unidades CSS
  Ejemplos: margin-top: 0px; margin-top: 10px;
  Indica el tamaño del margen superior.

- **margin-bottom:** Unidades CSS
  Ejemplos: margin-bottom: 0pt; margin-top: 1px;
  Indica el tamaño del margen inferior.

- **margin:** `<arriba> <derecha> <abajo> <izquierda>` | `<arriba> <derecha> <abajo>` | `<arriba-abajo> <izquierda-derecha>` | `<los 4 márgenes>`
  Ejemplos: margin: 4px 2px 1px 2px; margin: 4px;
  También podemos utilizar el estilo "margin" para indicar todos los márgenes a la vez, esta etiqueta nos permite indicarle desde 4 valores (para cada uno de los márgenes), hasta 1 valor (para aplicarlo sobre todos los márgenes).

- **padding-left:** Unidades CSS
  Ejemplos: padding-left: 0.5in; padding-left: 1px;
  Indica el espacio insertado, por la izquierda, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas. El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.

- **padding-right:** Unidades CSS
  Ejemplos: padding-right: 0.5cm; padding-right: 1pt;
  Indica el espacio insertado, en este caso por la derecha, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas. El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.

- **padding-top:** Unidades CSS
  Ejemplos: padding-top: 10pt; padding-top: 5px;
  Indica el espacio insertado, por arriba, entre el borde del elemento-continente y el contenido de este.

- **padding-bottom:** Unidades CSS
  Ejemplos: padding-bottom: 0.5cm; padding-bottom: 1pt;
  Indica el espacio insertado, en este caso por abajo, entre el borde del elemento-continente y el contenido de este.

- **padding:** `<arriba> <derecha> <abajo> <izquierda>` | `<arriba> <derecha> <abajo>` | `<arriba-abajo> <izquierda-derecha>` | `<los 4 márgenes>`
  Ejemplos: padding: 4px 2px 1px 2px; padding: 4px;
  Al igual que para "margin", esta etiqueta nos permite indicarle desde 4 valores (espaciado hasta cada uno de los bordes por separado), hasta 1 valor (para indicar el mismo espaciado hasta todos los bordes).

- **border-color:** color RGB o nombre de color
  Ejemplos: border-color: red; border-color: #ffccff;
  Para indicar el color del borde del elemento de la página al que se lo aplicamos. Se puede poner colores por separado con los atributos border-top-color, border-right-color, border-bottom-color, border-left-color.

- **border-style:** none | dotted | solid | double | groove | ridge | inset | outset
  Ejemplos: border-style: solid; border-style: double;
  El estilo del borde, los valores significan: none=ningún borde, dotted=punteado, solid=solido, double=doble borde, desde groove hasta outset son bordes con varios efectos 3D.

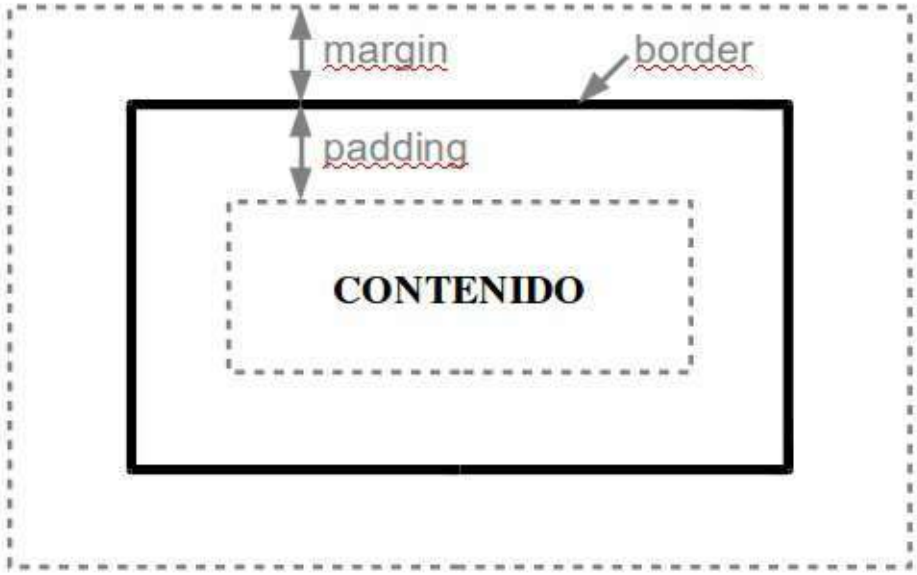
- **border-width:** Unidades CSS
  Ejemplos: border-width: 10px; border-width: 0.5in;
  El tamaño del borde del elemento al que lo aplicamos.

- **border:** `<grosor> <tipo> <color>`
  Ejemplo: border: 2px solid red;
  De esta forma podemos indicar las tres propiedades del borde a la vez. También podemos utilizar border-top, border-right, border-bottom y border-left para indicar estas tres propiedades para un borde en concreto.

- **float:** none | left | right
  Ejemplo: float: right;
  Sirve para alinear un elemento a la izquierda o la derecha haciendo que el texto se agrupe alrededor de dicho elemento.

- **clear:** none | both | right | left
  Ejemplo: clear: right;
  Indica que no se permiten elementos por ese lado del objeto. Por ejemplo, si tenemos varias cajas una a continuación de otra, al poner "clear:left" en la última caja, esta pasaría a la siguiente línea.

En la siguiente imagen se puede ver un esquema de un contenedor *DIV* en el que se representa la diferencia entre *margin*, *padding* y *border*:



Pseudo-clases

Una *pseudo-clase* permite tener en cuenta diferentes condiciones o eventos al definir una propiedad para una etiqueta HTML, por ejemplo si un enlace ha sido visitado o si el cursor del ratón está sobre un elemento. Algunas de las pseudo-clases que podemos utilizar son:

- *a:link* - enlace que no ha sido explorado por el usuario.
- *a:visited* - se refiere a los enlaces ya visitados.
- *a:active* - enlace seleccionado con el ratón.
- *a:hover* - enlace con el puntero del ratón encima, pero no seleccionado.
- *a:focus* - enlace con el foco del sistema. También puede ser usado para un input.
- *p:first-letter* - primera letra de un párrafo.
- *p:first-line* - primera línea de un párrafo.

Utilizando estos elementos podemos configurar por ejemplo:

```
a { color: black; }
a:hover { color: blue; }
a:visited { color: darkgreen; }
p:first-letter {color: green; font-size: x-large;}
```

En este ejemplo se aplica el color azul al texto de los enlaces solo cuando el ratón esté encima. Es decir, el texto del enlace tendrá por defecto el color negro, pero cuando el cursor del ratón pase por encima el color cambiará a azul. Además también se asigna el color del texto verde oscuro a los enlaces ya visitados, por lo que el usuario podrá ver marcados de este color los enlaces que ya ha pulsado anteriormente. Por último también se indica que la primera letra de los párrafos tenga el color verde y un tamaño más grade.

Las *pseudo-clases* no se pueden incluir en el estilo en línea de un elemento, por lo tanto las tendremos que definir bien en la hoja de estilos externa o en la sección *style* de la cabecera, por ejemplo:

```
<html>
<head>
    <style type="text/css">
        a {
            background-color: white;
        }
        a:hover {
            background-color: blue;
        }
    </style>
</head>
...
```

Capas

Normalmente la posición de los elementos de una página es **relativa**, es decir, depende de los demás elementos de la página. Por ejemplo, un párrafo estará más abajo si antes de él hay más párrafos o elementos. Debido a esto, normalmente cuando se quería colocar elementos en un sitio concreto, se recurría al uso de tablas (invisibles, solo para estructurar).

Con CSS podemos colocar los elementos en posición **absoluta**, es decir, indicando el tamaño y coordenadas exactas en las que queremos que se coloque. Para organizar la disposición en una Web con CSS se suele usar el elemento `<div>`. Además se le suele dar un identificador único a cada uno, mediante el cual, desde la hoja de estilo, podemos configurar su disposición. También podemos colocar estos elementos con posición relativa a otro elemento que lo contenga, por ejemplo, un `<div>` dentro de otro.

Es común en el diseño Web crear contenedores `<div>` generales en una posición absoluta o centrados en la página, con un tamaño definido, los cuales se utilizarán para contener y disponer el resto de elementos de nuestra Web. Estos otros elementos se pueden alinear de forma sencilla con una alineación "relativa" a sus contenedores. Por ejemplo un contenedor para la cabecera que contenga un par de contenedores para la disposición de logotipo y el texto de cabecera.

Distribución

Para indicar el tipo de distribución o disposición de un elemento lo hacemos mediante el atributo "**position: valor**". El cual puede tomar los valores:

- *absolute*: La posición del elemento no depende de ninguna otra etiqueta. Esta posición se calcula desde la esquina superior izquierda de la página.
- *fixed*: Al igual que el anterior la posición es absoluta, pero el elemento se queda fijo en el sitio al hacer "scroll".
- *relative*: Posición relativa a su elemento contenedor. Es la propiedad predeterminada.
- *static*: Al igual que el anterior la posición es relativa, pero no podemos redimensionar (por ejemplo) el objeto.

Posición

Para indicar la posición concreta de una capa utilizamos los atributos: *top*, *bottom*, *left* y *right*, de la forma:

```
top: <posición>;
left: <posición>;
```

Normalmente sólo se utilizan un par de ellos, como *top* y *left*, o *botton* y *right*. La posición se especifica mediante unidades de CSS, como por ejemplo en "px", aunque también admite porcentajes.

Un ejemplo de la definición de una capa sería:

```
#micapa {
   position: absolute;
   top: 200px;
   left: 150px;
   width: 175px;
   height: 175px;
   border: solid 1px blue;
   text-align: center;
   color: gray;
}
```

En nuestro documento HTML tendremos un elemento definido de la forma: `<div id="micapa"> ... </div>`, dentro del cual colocaremos texto u otros elementos.

> La posición absoluta la podemos definir respecto a la ventana del navegador o de forma relativa a un elemento contenedor. En este segundo caso tenemos que indicar que la posición del elemento contenedor es relativa (o de otra forma no funcionará).

Orden

A veces tenemos varias capas, unas por encima de otras, y queremos especificar un orden, para poder controlar las ocultaciones entre capas. Para esto usamos el z-index, de la forma:

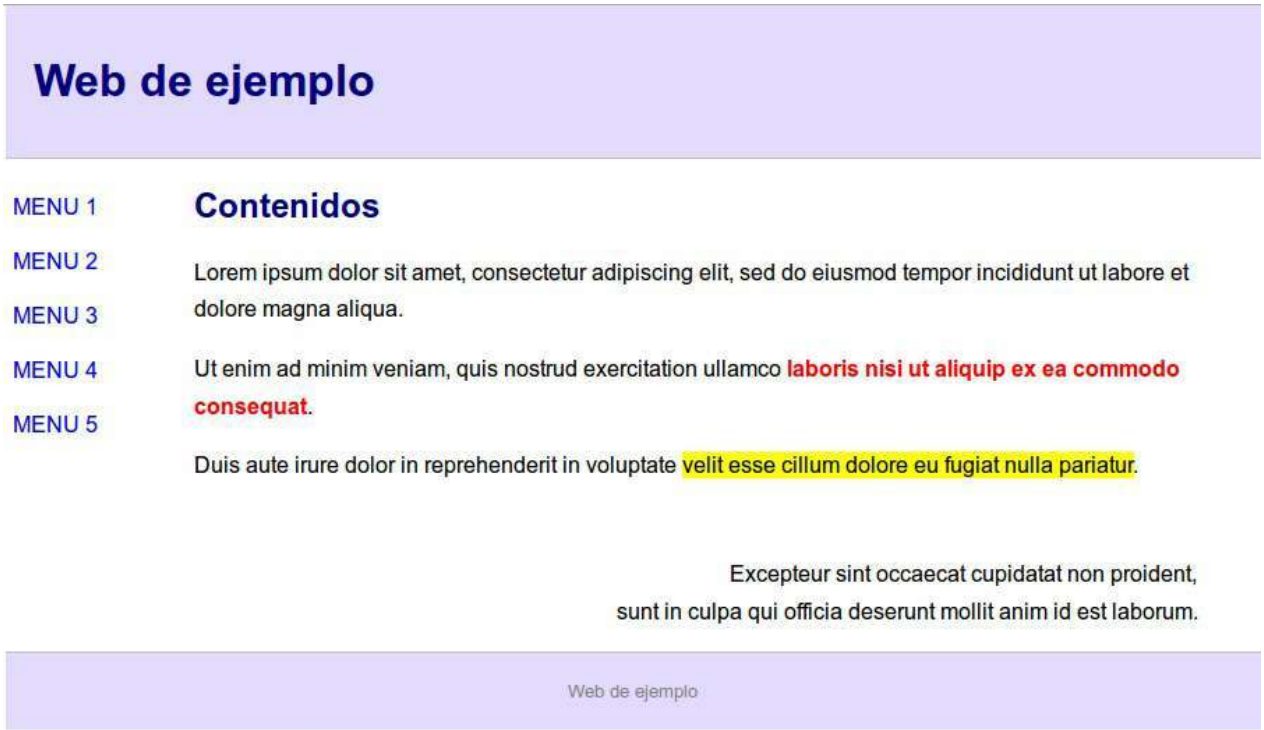
```
z-index: <índice>;
```

Las capas con un índice de Z-index mayor aparecerán por encima de las capas con un z-index menor.

Ejemplo

En esta sección se incluye un ejemplo más completo de una pequeña página web HTML a la que se le han aplicado estilos definidos en una sección *style* interna además de algunos estilos *inline*.

En la siguiente imagen se incluye una captura de la apariencia final del código de ejemplo. Como se puede ver la web consta de una cabecera, un menú lateral, una sección principal de contenido y un pie de página:



A continuación se incluye el código HTML y CSS utilizado para componer la web de ejemplo. En primer lugar, en la sección de cabecera (`head`), se ha incluido una sección de estilos entre las etiquetas `style` . En la sección `body` se incluye todo el código HTML que define la página con los estilos aplicados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.d
td">
<head>
    <title>Ejemplo</title>
    <style type="text/css">
        body { /* Redefinimos la etiqueta body */
            padding: 0px;
            margin: 0px;
            font-family: arial, helvetica;
            width: 900px;
            margin: 0 auto;
```

```
        }
        h1, h2 { /* Redefinimos las etiquetas de cabecera */
            color: navy;
        }
        #header { /* Estilo para la cabecera usando su identificador */
            padding: 15px 20px;
            background-color: #E3DAFF;
            border-bottom: 1px solid silver;
        }
        #sidebar {
            float: left;
            width: 15%;
        }
        #article {
            float: left;
            width: 80%;
        }
        #footer {
            clear: both;
            text-align: center;
            border-top: 1px solid silver;
            font-size: small;
            color: gray;
            background-color: #E3DAFF;
            padding: 20px;
        }
        #sidebar a { /* Estilo para los enlaces de la barra lateral */
            text-transform: uppercase;
            text-decoration: none;
            padding: 10px 5px;
            display: block;
        }
        #sidebar a:hover { /* Pseudo-clase para los enlaces */
            background-color: navy;
            color: white;
        }
        #article p {
            line-height: 20pt;
        }
        .nota { /* Definimos la clase .nota */
            background-color: yellow;
        }
        .alineado-derecha { /* Definimos la clase .alineado-derecha */
            text-align: right;
        }
    </style>
</head>
<body>
    <div id="header">
        <h1>Web de ejemplo</h1>
    </div>
    <div id="content">
```

```
        <div id="sidebar" style="padding-top: 15px">
            <a href="#">Menu 1</a>
            <a href="#">Menu 2</a>
            <a href="#">Menu 3</a>
            <a href="#">Menu 4</a>
            <a href="#">Menu 5</a>
        </div>
        <div id="article">

            <h2>Contenidos</h2>

            <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmo
d
                tempor incididunt ut labore et dolore magna aliqua.
            </p>
            <p>
                Ut enim ad minim veniam, quis nostrud exercitation ullamco
                <span style="font-weight:bold; color:red">laboris nisi ut aliquip ex e
a
                commodo consequat</span>.
            </p>
            <p>
                Duis aute irure dolor in reprehenderit in voluptate
                <span class="nota">velit esse cillum dolore eu fugiat nulla pariatur</
span>.
            </p>
            <br/>
            <p class="alineado-derecha">
                Excepteur sint occaecat cupidatat non proident,
                <br/>
                sunt in culpa qui officia deserunt mollit anim id est laborum.
            </p>

        </div>
    </div>
    <div id="footer">
        Web de ejemplo
    </div>
</body>
</html>
```

Introducción a HTML 5

La quinta revisión del lenguaje de programación HTML pretende remplazar al actual (X)HTML, corrigiendo problemas con los que los desarrolladores web se encuentran, así como rediseñar el código y actualizándolo a nuevas necesidades que demanda la web de hoy en día.

Actualmente se encuentra en modo experimental, lo cual indica la misma W3C; aunque ya es usado por múltiples desarrolladores web por sus avances, mejoras y ventajas.

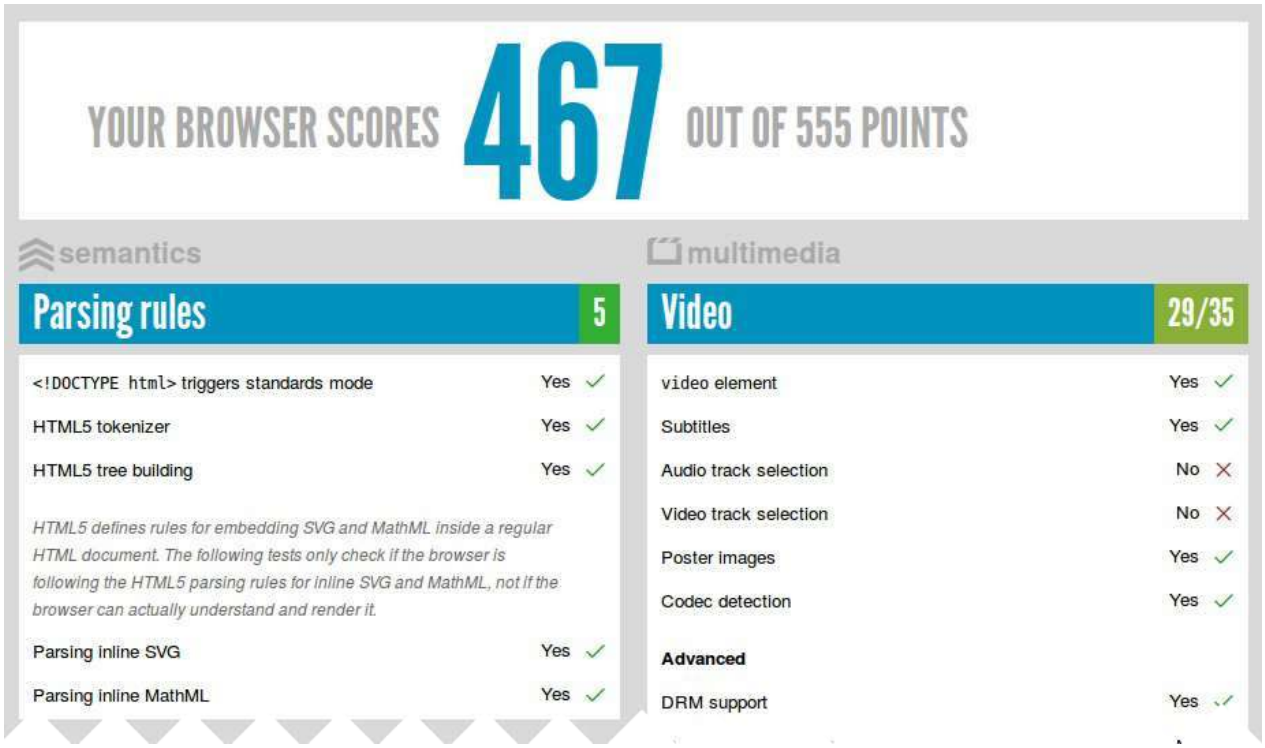
A diferencia de otras versiones de HTML, los cambios en HTML5 comienzan añadiendo semántica y accesibilidad implícitas. Establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas `<div>` y `<span>`, pero tienen un significado semántico, como por ejemplo `<nav>` (bloque de navegación del sitio web) o `<footer>`. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos `<audio>` y `<video>`.

Algunos elementos de HTML 4.01 han quedado obsoletos, incluyendo elementos puramente de presentación, como `<font>` y `<center>`, cuyos efectos se deben de realizar utilizando CSS. También hay un renovado énfasis en la importancia del scripting DOM para el comportamiento de la web.

Navegadores que lo soportan

Actualmente, de los navegadores de escritorio, el que mayor soporte da es Google Chrome, seguido muy de cerca por Mozilla Firefox y Apple Safari. El que menor compatibilidad ofrece es Internet Explorer.

Para comprobar la compatibilidad de un navegador podemos visitar la Web "http://www.html5test.com/" donde se realiza un test de todas las funcionalidades de HTML5.



Doctype

El doctype es el encargado de indicarle al navegador el tipo de documento que está abriendo, con el fin de renderizar la pagina de manera correcta. Por ejemplo, el doctype de HTML 4 es:

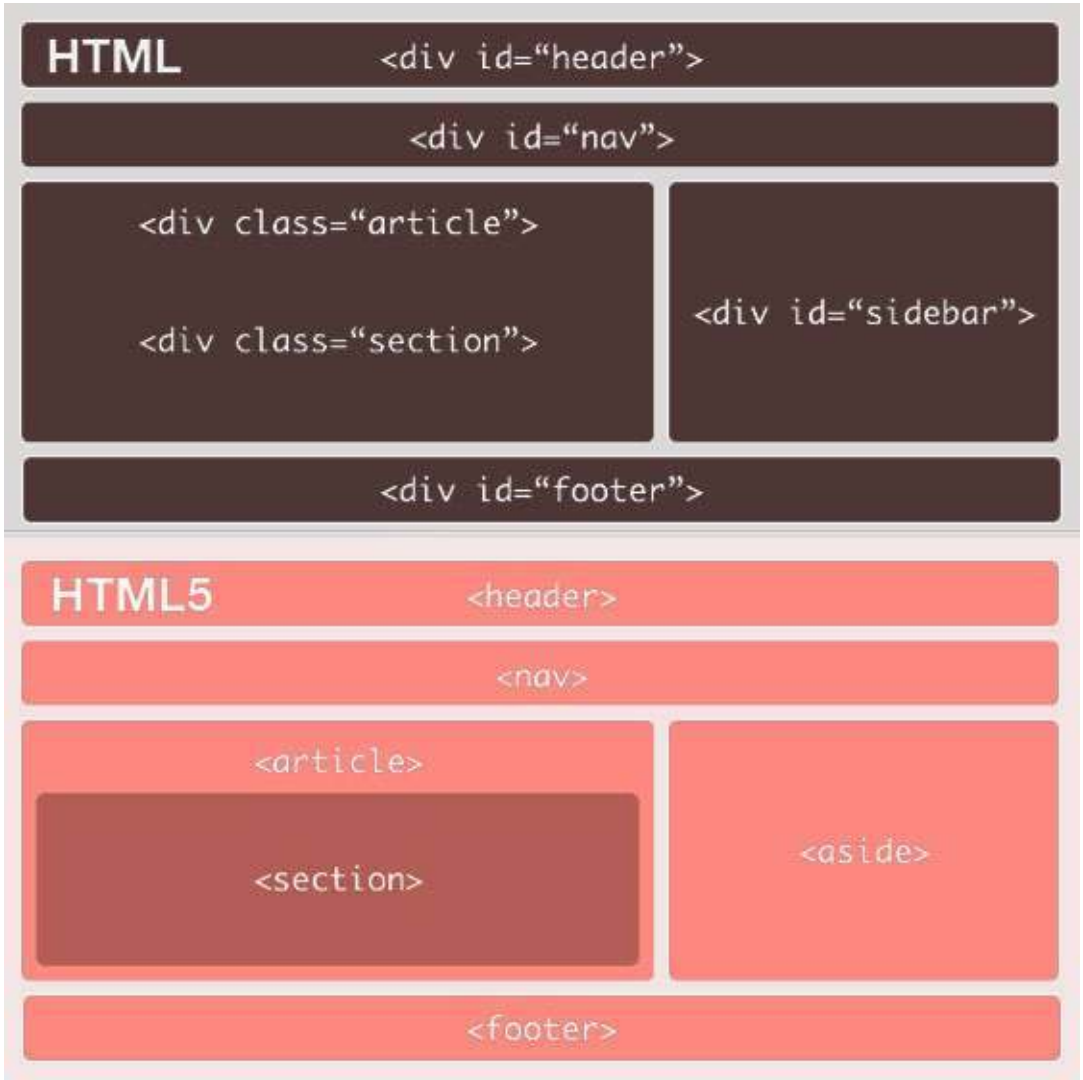
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

Para HTML 5 el doctype se ha simplificado mucho y además es compatible con las versiones anteriores de HTML:

```
<!DOCTYPE html>
```

Estructura semántica

Hasta ahora se utilizaba de forma abusiva la etiqueta `<div>` y las tablas para estructurar una web en bloques. El HTML5 nos brinda nuevas etiquetas que perfeccionan esta estructuración. Estas nuevas etiquetas introducen un nuevo nivel semántico que hace que la estructura de la web sea más coherente y fácil de entender. Además los navegadores podrán darle más importancia a determinadas secciones, facilitándole además la tarea a los buscadores, así como cualquier otra aplicación que interprete sitios Web. En la siguiente imagen se puede ver una comparación entre la estructuración realizada con HTML (hasta la versión 4) y HTML 5:



Las Webs se dividirán en los siguientes elementos:

- `<section></section>` : Se utiliza para representar una sección "general" dentro de un documento o aplicación, como un capítulo de un libro. Puede contener subsecciones y si lo acompañamos de las etiquetas `<h1>..<h6>` podemos estructurar mejor toda la

página creando jerarquías del contenido, algo muy favorable para el buen posicionamiento web. Por ejemplo:

```
<section>
   <h1>Introducción al elemento section</h1>
   <p>El elemento section se usa para agrupar
   contenido relacionado entre si.</p>
   <p>...</p>
</section>
```

- `<article></article>` : Se usa para definir contenido autónomo e independiente, con la intención de ser reutilizable de modo aislado. El elemento article puede contener uno o varios elementos section. Si por ejemplo nuestro contenido puede ser redistribuido como RSS y sigue manteniendo íntegro su significado, entonces, probablemente es un elemento article. De hecho, el elemento article está especialmente indicado para sindicación. El elemento article es especialmente útil para posts en blogs, noticias en prensa digital, comentarios y posts en foros.
  La especificación de HTML5 añade además que el elemento article debe ser usado por widgets autónomos como; calculadoras, relojes, marcos de clima y cosas por el estilo. Hay que analizar si el contenido de un widget es autónomo, independiente y puede ser reutilizable o incluso sindicado.

- `<aside></aside>` : Representa una sección de la página que abarca un contenido no directamente relacionado con el contenido que lo rodea, por lo que se le puede considerar un contenido independiente. Dentro de este elemento pueden incluirse: elementos publicitarios, barras laterales, grupos de elementos de la navegación, efectos tipográficos, u otro contenido que se considere separado del contenido principal de la página.

- `<header></header>` : Es la cabecera de la página o de una sección. Existe una diferencia clave entre el elemento header y el uso habitual del término header (o cabecera) utilizado comúnmente para situar los elementos del encabezado de un sitio web.
  Una página web debe definir un header principal donde normalmente irá el logo o el nombre del sitio y seguramente un menú de navegación, pero además puede —y debe — definir otros elementos `<header>` dentro de los elementos `<section>` :

```
<section>
   <header>
      <h1>Cabecera se sección</h1>
   </header>
   <p>...</p>
</section>
```

- `<nav></nav>` : Contiene información sobre la navegación por el sitio web, usualmente una lista de enlaces. Este elemento debe de ser utilizado solo para la navegación principal del sitio y no para enlaces externos por ejemplo. Normalmente el elemento nav aparece dentro de un elemento *header* o *footer*.
- `<footer></footer>` : Representa el pié de una sección o la parte inferior de una página Web, contiene información acerca de la página/sección que poco tiene que ver con el contenido de la página, como el autor, el copyright, la fecha de última modificación, etc. Igual que con la etiqueta `<header>`, este elemento también se puede utilizar dentro de una sección para indicar información como: quien lo ha escrito, información de propiedad intelectual, enlaces, etc.

Es muy importante tener en cuenta que estas etiquetas no indican su posición en la página Web, sino su valor semántico. Por ejemplo, las etiquetas *header, footer o aside* no indican que esos elementos tengan que ir en la parte superior, inferior o lateral del contenido principal, sino que indican su función en esa sección o en esa página.

Además debemos tener en cuenta que estas nuevas etiquetas se comportan igual que una etiqueta de caja `<div>` por lo que podemos aplicarles los mismos estilos CSS. Podemos redefinir la propia etiqueta o aplicarle una clase, por ejemplo:

```
header { width: 100%; padding: 10px; margin-bottom: 20px; }
.webheader { height: 30px; border: 1px solid gray; background-color: silver; }
.sectionheader { font-size: 20px; }
```

Formularios

La estructura de los formularios con HTML 5 no varía con respecto a las anteriores de HTML. Pero sí que se añaden muchos nuevos tipos de campos que podemos utilizar, cada uno específico para cada tipo de dato.

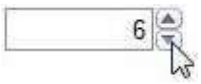
En el caso de que utilicemos estas características y el navegador no sea compatible, simplemente las ignorará sin causarnos mayores problemas. También podemos detectar si el navegador soporta una determinada característica y en caso negativo emularla mediante código JavaScript (para más información ver la sección "Detectar funcionalidades de HTML5").

Los nuevos tipos de campos son:

- **search**: se utiliza para crear cajas de búsqueda. Tiene un aspecto similar a un campo de tipo texto. Además podemos utilizar el atributo *results="num"* para añadir un histórico de búsquedas con "num" resultados. De momento no funciona ni en Firefox ni en Chrome.

```
<label for="busqueda">Búsqueda con histórico: </label>
<input type="search" name="busqueda" id="busqueda" results="5"/>
```

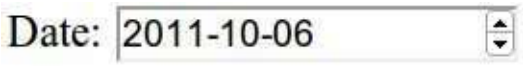
- **number**: campo numérico, incorpora dos botones para para incrementar o decrementar el valor del campo. Además podemos usar atributos para asignar restricciones, como *min=""*, *max=""* o *step=""*. El valor es almacenado en el atributo *value=""*.



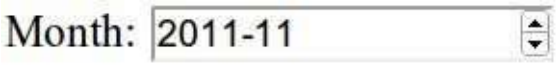
- **range**: campo numérico que permite seleccionar mediante una barra de desplazamiento un valor entre dos valores predeterminados, especificados mediante min="" y max="". El valor actual es almacenado en el atributo *value=""*. Además podemos indicar el incremento mínimo al desplazar la barra con *step=""*.

- **color**: permite seleccionar un color. De momento solo funciona en Opera 11.
- **tel**: es un campo de texto normal pero valida si el valor introducido es un número telefónico (todavía no funciona).
- **url**: valida direcciones web. De momento requiere "http://" o "http:" simplemente. En algunos navegadores cambia el aspecto del cambio.

- **email**: valida direcciones de email. Funciona en algunos navegadores, mostrando además un aspecto diferenciado. Para iPhone además adapta el teclado.
- **date**: seleccionar un día en un calendario. En algunos navegadores (para móvil) aparece un calendario desplegable (como en Opera).



- **month**: selector para meses. En algunos navegadores (para móvil) aparece un calendario desplegable.

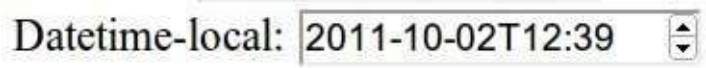


- **week**: selector para semanas. En algunos navegadores (para móvil) aparece un calendario desplegable.

- **time**: campo con formato para hora.

- **datetime**: permite seleccionar fecha y hora.

- **datetime-local**: permite seleccionar fechas y hora local.



- **output**: este campo se utiliza para visualizar valores, por ejemplo el valor de un campo "range". De momento solo funciona en Opera. Se suele utilizar junto con la propiedad "onformchange" para actualizar su valor:

```
<output onformchange="value = rango.value">0</output>
```

Además, junto con estos nuevos tipos de campos, también se han incorporado nuevos tipos de atributos. Estos nuevos atributos son aplicables a la mayoría de los campos:

- **Autocomplete**: La mayoría de los navegadores incorporan la funcionalidad de autocompletar algunos campos de los formularios con valores introducidos anteriormente. Esta funcionalidad no siempre resulta útil, sobre todo si alguien nos roba nuestro portátil o dispositivo móvil. La nueva especificación de HTML5 nos permite desactivar el autocompletado en un formulario completo o solo en campos específicos. El atributo *autocomplete* nos permite definir dos valores: "on" o "off".

```
<form action="formaction.php" autocomplete="off">
   ...
</form>
```

El código anterior desactivaría el autocompletado de todo el formulario. Si por el contrario solo queremos desactivar el autocompletado de un solo campo podemos especificarlo así:

```
<input type="text" name="cuentadelbancosupersecreta"  autocomplete="off" />
```

Esta funcionalidad no se puede emular mediante código JavaScript.

- **Placeholder**: El atributo *placeholder="texto"* se utiliza para colocar el valor de su texto dentro del campo a modo de ayuda. Si se focaliza dicho campo, se elimina el *placeholder*. Si abandonamos el campo sin añadir ningún valor, se vuelve a añadir el *placeholder*. Esta funcionalidad siempre ha requerido del uso de JavaScript para ser llevado a cabo, pero con la nueva especificación este comportamiento puede definirse de la forma:

```
<label for="referer">Nombre</label>
<input id="referer" name="referer" type="text"
            placeholder="Escribe tu nombre completo" />
```

Obteniendo como resultado:

- **Required**: Una de las tareas de validación más extendidas es la de los campos requeridos. La nueva especificación de HTML5 incluye el atributo *required* que nos sirve para definir si un campo es requerido o no. Si un campo requerido está en blanco el formulario no será enviado y además avisará con un mensaje:

```
<label for="username">Su nombre de usuario</label>
<input id="username" name="username" type="text" required/>
```

NOTA: Es un error grave de seguridad validar los formularios únicamente desde el lado del cliente, es imprescindible además realizar la validación en el servidor.

- **Autofoco**: El atributo de autofoco asigna el foco (cursor de escritura) al campo indicado en cuando la página se ha cargado. Sólo se puede asignar a un elemento de la página. De momento este atributo solo lo soportan Safari, Chrome y Opera. Firefox e IE, lo ignoran, pero se puede emular fácilmente mediante código JavaScript (ver la siguiente sección "Detectar funcionalidades de HTML5").

```
<input name="b" autofocus/>
```

- **List**: Usando el atributo list con un elemento `<input>` podemos especificar una lista de opciones. Esto permite al usuario seleccionar un valor de la lista o escribir uno que no esté en ella (este tipo de elemento se suele llamar *Combo Boxes*). Los elementos de la lista se deben de indicar utilizando otro nuevo elemento de HTML5, el `<datalist>`. El cual simplemente nos permite crear una lista de valores. En algunos navegadores estas funcionalidades todavía no funcionan, como en Chrome.

```
<label for="diasemana">Día de la semana:</label>
<input type="text" name="diasemana" id="diasemana" list="dias"/>
<datalist id="dias">
   <option value="Lunes" />
   <option value="Martes" />
   <option value="Miércoles" />
   <option value="Jueves" />
   <option value="Viernes" />
   <option value="Sábado" />
   <option value="Domingo" />
</datalist>
```

Con este código obtendríamos un resultado similar al de la siguiente imagen:

- **Pattern (formatting)**: Este atributo se utiliza para validar la entrada del usuario mediante expresiones regulares. En la dirección "http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular" podemos obtener más información sobre las expresiones regulares. Ejemplo de uso (en Firefox y Chrome funciona):

```
<label for="cp">Código Postal</label>
<input id="cp" name="cp" pattern="[\d]{5}(-[\d]{4})" />
```

Mark

HTML5 también introduce un conjunto nuevo de elementos *inline*, solo que ya no se llaman elementos inline sino *text-level semantics* o semántica a nivel de texto. Uno de ellos es la etiqueta mark. Cuando realizamos una búsqueda en ciertos sitios, los elementos encontrados en la página aparecen remarcados para facilitar su localización. Hasta ahora el estilo se aplicaba con etiquetas `<span>`, pero esta solución no es semántica. Es ahí donde entra en escena la nueva etiqueta `<mark>`:

```
<h1>Resultados de la búsqueda de la palabra 'anillo'</h1>
<ol>
   <li>El señor de los <mark>anillo</mark>s...</li>
   <li>el cliente compró este <mark>anillo</mark></li>
</ol>
```

Si queremos podemos redefinir el estilo de esta nueva etiqueta de la misma forma que lo hacíamos con las etiquetas de HTML, por ejemplo, para cambiar el color de fondo a rojo:

```
mark { background-color: red; }
```

Canvas

El elemento canvas puede definirse como un entorno para crear imágenes dinámicas. Utilizando su API en JavaScript podemos manipular el elemento canvas para dibujar en él y crear gráficos dinámicos de todo tipo (incluidas interfaces de aplicaciones web completas). La API, aunque de momento está en desarrollo, la podemos encontrar en: http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html

Para empezar a usarlo lo único que hay que especificar son sus dimensiones. El texto que escribamos entre la apertura y cierre de la etiqueta canvas solamente será interpretado por navegadores que no soporten esta etiqueta:

```
<canvas id="myCanvas" width="360" height="240">
   <p>Tu navegador no soporta canvas</p>
</canvas>
```

El resto de trabajo con canvas se ha de realizar con código JavaScript. Primero debemos referenciar este elemento y adquirir su contexto (que de momento solo está disponible para 2D):

```
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
```

Una vez adquirimos el contexto podemos empezar a dibujar. La API bidimensional ofrece muchas de las herramientas que podemos encontrar en cualquier aplicación de diseño gráfico: trazos, rellenos, gradientes, sombras, formas y curvas Bézier. Los principales métodos disponibles son:

- **fillRect(x, y, width, height)**: dibuja un rectángulo relleno de color según el estilo activado.
- **strokeRect(x, y, width, height)**: dibuja solo el borde de un rectángulo, el interior será transparente.
- **clearRect(x, y, width, height)**: borra el área indicada.
- **beginPath()**: inicializa el dibujado de un "trazo".
- **closePath()**: cierra la figura creando una línea desde el último punto hasta el primero.
- **moveTo(x, y)**: mueve el puntero del trazo hasta las coordenadas indicadas (para poder seguir dibujando).
- **lineTo(x, y)**: dibuja un trazo desde la posición actual hasta las coordenadas indicadas.
- **stroke()**: dibuja el trazo indicado desde el último "beginPath()".

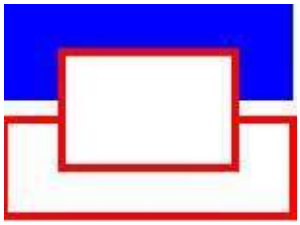
- **fill()**: cierra el trazo definido desde el último "beginPath()" y lo rellena.
- **arc(x, y, radius, startAngle, endAngle, anticlockwise)**: dibuja un arco con centro en "x, y" y el radio definido. Los ángulos se definen en radianes (radianes = (PI/180)*grados) y el último parámetro es un valor booleano.
- **quadraticCurveTo(controlx, controly, x, y)**: dibuja una curva de bezier cuadrática.
- **bezierCurveTo(control1x, control1y, control2x, control2y, x, y)**: dibuja una curva de bezier cúbica.
- **drawImage(x, y)**: dibuja una imagen (como objeto JavaScript) en el canvas.
- **createImageData(width, height)**: crea un objeto ImageData como un array de píxeles para ser manipulado como un array de enteros.
- **getImageData(x, y, w, h)**: carga un objeto ImageData a partir del dibujo actual para ser manipulado.
- **putImageData(imageData, x, y)**: mapea los valores de un objeto ImageData en el dibujo actual.
- **strokeText(string, x, y)**: dibuja una cadena de texto usando solo su borde.
- **fillText(string, x, y)**: dibuja una cadena de texto.

A continuación mostramos un ejemplo de dibujado en un objeto canvas una vez capturado su contexto:

```
// Primero definimos las propiedades con las que vamos a dibujar
context.fillStyle   = '#0000ff'; // color de relleno azul
context.strokeStyle = '#ff0000'; // color de borde rojo
context.lineWidth   = 4;         // grosor de línea

// Y a continuación dibujar algunas figuras
context.fillRect  (0,   0, 150, 50);   // rectángulo relleno
context.strokeRect(0,  60, 150, 50);   // rectángulo solo borde
context.clearRect (30, 25,  90, 60);   // borrar área del canvas
context.strokeRect(30, 25,  90, 60);   // Orden de coordenadas: izqda, arriba, ancho,
largo
```

Obteniendo finalmente un resultado similar a:



Webs muy importantes están cambiando sus contenidos a canvas y dejando de usar Flash, como Slideshare (ver http://www.slideshare.net/AmitRanjan/slideshare-is-html5-now).

Audio

El nuevo elemento audio permite insertar archivos sonoros en diferentes formatos, incluyendo mp3 y ogg. Además provee de una interfaz de control sobre la reproducción del mismo con una API en JavaScript sin necesidad de plugins de ningún tipo (como Flash). Añadir un reproductor de audio en HTML5 es muy simple:

```
<audio src="archivo.mp3" controls>
   <p>Tu navegador no soporta el elemento audio</p>
</audio>
```

En Firefox obtendríamos un resultado similar a:

El texto que se encuentra entre las etiquetas audio solo es tenido en cuenta por navegadores que no soporten la nueva etiqueta. El atributo "controls" indica al navegador que muestre los controles de reproducción. En caso de no activarlo no se visualizaría nada, pero podríamos controlar la reproducción mediante funciones JavaScript, de la forma:

```
<audio id="player" src="archivo.mp3"></audio>
<button onclick="document.getElementById('player').play();">Reproducir</button>
<button onclick="document.getElementById('player').pause();">Pausa</button>
<button onclick="document.getElementById('player').volume += 0.1;">Subir Volumen</butt
on>
<button onclick="document.getElementById('player').volume -= 0.1;">Bajar Volumen</butt
on>
```

También podemos usar los atributos "autoplay" y "loop" para que se auto-reproduzca y para que se cree un bucle de reproducción infinito:

```
<audio src="archivo.mp3" autoplay loop></audio>
```

El formato de audio a utilizar vendrá impuesto por el navegador usado y no por el estándar:

Códec	IE>=9	Firefox	Chrome	Safari	Opera
Ogg Vorbis	no	sí	sí	no	sí
WAV PCM	no	sí	sí	sí	sí
MP3	sí	no	sí	sí	sí
AAC	sí	no	sí	sí	sí
Speex	no	no	sí	no	no

Como puede verse, combinando Vorbis y MP3 podremos ofrecer audio a todos los navegadores mayoritarios. Existe una forma de definir más de un archivo de audio para la etiqueta audio, en lugar de usar el atributo "src", utilizaremos la etiqueta "source" para poder definir múltiples archivos. Estas etiquetas se irán leyendo de arriba a abajo hasta que el navegador encuentre un formato soportado. De esta manera podremos complacer las necesidades de todos los usuarios sin discriminar a ningún navegador.

```
<audio controls>
   <source src="archivo.ogg" type="audio/ogg" />
   <source src="archivo.mp3" type="audio/mpeg" />
   <object type="application/x-shockwave-flash" data="player.swf?soundFile=archivo.mp3"
>
      <param name="movie" value="player.swf?soundFile=archivo.mp3" />
      <a href="archivo.mp3">Descarga el archivo de audio</a>
   </object>
</audio>
```

En este ejemplo hemos añadido además una tercera línea con un reproductor Flash por si no fuesen soportados ninguno de los formatos anteriores, y un link directo de descarga para aquellos que tampoco soporten Flash. Así estaremos ofreciendo nuestro contenido a todos los navegadores y dispositivos manteniendo unas buenas prácticas en cuanto a accesibilidad del contenido se refiere.

Vídeo

La nueva especificación de HTML5 soporta la inclusión de vídeo empotrado en las páginas web de forma nativa. El elemento *video* no especifica el formato del mismo sino que el uso de uno u otro vendrá impuesto por el fabricante del navegador:

Códec	IE>=9	Firefox	Chrome	Safari	Opera
Ogg Theora	no	sí	sí	no	sí
H.264	sí	no	no	sí	no
VP8	no	sí	sí	no	sí

El elemento *video* dispone de los atributos "*autoplay*", "*loop*" y "*preload*", para activar la auto-reproducción, para indicar que se reproduzca en bucle y para activar/desactivar la precarga del vídeo. Asimismo puedes utilizar los controles que te ofrece el navegador de forma nativa utilizando el atributo *controls* o bien puedes ofrecer tus propios controles en JavaScript. Dado que el vídeo ocupa un espacio, también podremos definir sus dimensiones con los atributos "*width*" y "*height*". E incluso podemos indicar una imagen para que se muestre antes de la reproducción mediante el atributo "poster":

```
<video src="archivo.mp4" controls width="360" height="240" poster="poster.jpg"> </video
>
```

Con lo que obtendríamos un resultado similar a:

Para dar soporte a todos los navegadores, podemos especificar diferentes archivos en diferentes formatos. Además podemos usar el mismo truco que usábamos con el elemento audio para seguir dando soporte al *plugin* de Flash a través de la etiqueta *object*, e incluso incluir un link de descarga:

```
<video controls width="360" height="240" poster="poster.jpg">
   <source src="archivo.ogv" type="video/ogg" />
   <source src="archivo.mp4" type="video/mp4" />
   <object type="application/x-shockwave-flash" width="360" height="240"
                   data="player.swf?file=archivo.mp4">
      <param name="movie" value="player.swf?file=archivo.mp4" />
      <a href="archivo.mp4">Descarga la película</a>
   </object>
</video>
```

Geolocalización

La geolocalización es la forma de obtener tu posición en el mundo y si quieres, compartir esta información. Existen muchas maneras de descubrir donde te encuentras, por tu dirección IP, la conexión de red inalámbrica, la torre de telefonía móvil por la que se conecta tu móvil, o usando directamente el posicionador GPS.

HTML5 incorpora una nueva funcionalidad para facilitar esta tarea, que dependerá de que el navegador le de soporte. Está disponible a partir de las versiones de Opera 10.6, Firefox 3.5, Chrome 5, Safari 5 e Internet Explorer 9.

Para obtener la localización simplemente tienes que utilizar el objeto `navigator` de JavaScript. Inicialmente es recomendable comprobar si está disponible la localización y de ser así ya podemos utilizar el método `getCurrentPosition` de la forma:

```
if (navigator.geolocation)
{
   navigator.geolocation.getCurrentPosition(showPosition);
}

function showPosition( position )
{
      var lat = position.coords.latitude;
      var lng = position.coords.longitude;

      alert( "Latitud: " + lat + ", longitud: " + lng );
}
```

Almacenamiento Offline

El almacenamiento web está ampliamente soportado por los navegadores modernos, tanto en plataforma escritorio como en plataforma móvil, Android 2.1+, iPhone 3.1+, iPad 4.2+, Opera Mobile 11.00+, Palm WebOS 1.4+ y BlackBerry 6.0+, Crome 4.0+, Firefox 3.5+, IE 8.0+, Opera 10.5+ y Safari 4.0+.

Tipos de almacenamiento

El almacenamiento web ofrece dos áreas de almacenamiento diferentes, el almacenamiento local (*localStorage*) y el almacenamiento por sesión (*sessionStorage*), que difieren en alcance y tiempo de vida. Los datos alojados en un almacenamiento local es solo accesible por dominio y persiste aún cuando se cierre el navegador. El almacenamiento por sesión es por ventana y su tiempo de vida está limitado a lo que dure la ventana (o pestaña).

Los datos se almacenan de forma muy sencilla, por pares clave/valor, de la forma:

```
// Para almacenamiento persistente en local:
localStorage.setItem("miValor", valor);

// Para almacenamiento por sesión:
sessionStorage.setItem("miValor", valor);
```

Para recuperarlos posteriormente solo tenemos que hacer:

```
var miValor = localStorage.getItem("miValor");
var miValor = sessionStorage.getItem("miValor");
```

Las variables guardadas con *sessionStorage* sólo se mantendrían en caso de que cambiemos de página o que el navegador se refresque, mientras que localStorage guardaría los datos aunque el navegador sea cerrado.

También podemos borrar los valores almacenados, indicando un valor en concreto o todos ellos:

```
localStorage.remove("miValor");
localStorage.clear();
```

Offline Application Cache (appCache)

Esta nueva característica de HTML5 permite ejecutar aplicaciones Web aun cuando no estamos conectados a Internet. Al visitar por primera vez una página web (que use appCache) el navegador descarga y guarda todos los archivos necesarios para esa página. La siguiente vez que la visitemos el navegador usará directamente los archivos descargados (a no ser que estemos conectados y se compruebe que hay una versión más actual de la Web).

El principal componente del appCache es el archivo de manifiesto (manifest file), un archivo de texto con la lista de archivos que el navegador cliente debe almacenar. En primer lugar, para usar esta característica debemos de indicar el archivo de manifiesto en la etiqueta de apertura HTML:

```
<html manifest="app.manifest">
```

Este fichero debe de empezar con el texto `CACHE MANIFEST`. A continuación en cada nueva línea indicaremos un recurso a almacenar (usando URLs absolutas o relativas), además podemos poner comentarios anteponiendo el símbolo "#".

```
CACHE MANIFEST
# Esto es un comentario
index.html
js/scripts.js
css/estilos.css
imgs/logo.gif
imgs/image1.jpg
```

Una vez cargada la página, la única petición que realizará el navegador será por el fichero de *manifiest*. Aunque solo haya cambiado un letra del fichero, se descargarán todos los recursos de nuevo. Para asegurarnos que servimos la última versión de nuestra página cuando realizamos cambios, la forma más sencilla y segura es actualizar el fichero de manifiesto con un comentario indicando la fecha de la última actualización (o un número de versión, etc.), de la forma:

```
CACHE MANIFEST
# Actualizado el 2011-10-12
```

Para más información podéis consultar las fuentes:

- http://www.w3.org/TR/offline-webapps/
- http://www.w3.org/TR/html5/offline.html

Detectar funcionalidades de HTML5

Modernizr es una librería de JavaScript con licencia MIT de código abierto que detecta si son compatibles elementos de HTML5 y CSS3. Podemos descargar la librería desde "http://www.modernizr.com/". Para utilizarla solo hay que incluir en el `<head>` de tu página de la forma:

```
<head>
   <script src="modernizr.min.js"></script>
</head>
```

Modernizr se ejecuta automáticamente, no es necesario llamar a ninguna función. Cuando se ejecuta, se crea una objeto global llamado *Modernizr*, que contiene un *set* de propiedades *Boleanas* para cada elemento que detecta. Por ejemplo si su navegador soporta elementos *canvas*, la propiedad de la librería "*Modernizr.canvas*" será “*true*”. Si tu navegador no soporta los elementos *canvas*, la propiedad será “*false*”, de la forma:

```
if (Modernizr.canvas) {
   // sí que hay soporte
} else {
   // no hay soporte para canvas
}
```

Para comprobar elementos de un formulario también podemos crearnos dos simples funciones que validan el soporte para diferentes tipos de inputs y atributos:

Comprobar si un *input* es soportado

Con la siguiente función podemos comprobar si un navegador soporta o no los nuevos tipos de inputs:

```
function inputSupports(tipo) {
   var input = document.createElement('input');
   input.setAttribute('type', tipo);
   if (input.type == 'text') {
      return false;
   } else {
      return true;
   }
}
```

Por lo que podemos usarlo de la siguiente forma:

```
if (!inputSupports('range')) {
   // Input tipo range no soportado
}
```

Comprobar si un atributo es soportado

Para comprobar si hay soporte para un atributo

```
function attrSupports(el, attr) {
   var telement = document.createElement(el);
   if (attr in telement) {
      return true;
   } else {
      return false;
   }
}
```

Por lo que podemos usarlo para comprobar, por ejemplo, los atributos autofocus, placeholder o required:

```
if (!attrSupports('input', 'autofocus')) {
   document.getElementById('search_string').focus();
}
if (!attrSupports('input', 'placeholder')) {
   // Atributo placeholder no soportado
}
if (!attrSupports('input', 'required')) {
   // Atributo required no soportado
}
```

Introducción a CSS3

La especificación de CSS3 viene con interesantes novedades que permitirán hacer webs más elaboradas y más dinámicas, con mayor separación entre estilos y contenidos. Dará soporte a muchas necesidades de las webs actuales, sin tener que recurrir a trucos de diseñadores o lenguajes de programación.

Aunque CSS3 está todavía en fase de desarrollo, la mayoría de navegadores ya dan soporte a casi todos los nuevos estilos, como Firefox, Chrome o Safari. Por el contrario Internet Explorer no ha empezado a incorporar estos nuevos elementos hasta la versión 9.

Las principales propiedades nuevas en CSS3 son:

- Selectores de atributos y propiedades
- Nuevas pseudo-clases
- Formatos de color: colores HSL, colores HSLA, colores RGBA, Opacidad
- Bordes: border-color, border-image, border-radius, box-shadow
- Fondos: background-origin, background-clip, background-size, capas con múltiples imágenes de fondo
- Texto: text-shadow, text-overflow, rotura de palabras largas, Web Fonts, creación de múltiples columnas de texto
- Modelo de caja básico: overflow
- Transiciones y transformaciones

A continuación veremos con más detalle cada una de estas nuevas propiedades.

Nuevos selectores de atributos

En primer lugar encontramos 3 nuevos selectores de atributos:

- **elemento[atributo^="valor"]**: Selecciona los elementos con ese atributo y que su valor comienza por la cadena de texto indicada en "valor".
- **elemento[atributo$="valor"]**: Selecciona los elementos con ese atributo y que su valor termina por la cadena de texto indicada en "valor".
- **elemento[atributo*="valor"]**: Selecciona los elementos con ese atributo y que su valor contiene la cadena de texto indicada en "valor".

Por ejemplo:

```
// Selecciona todos los enlaces que apunten a una dirección de correo:
a[href^="mailto:"]{...}

// Selecciona todos los enlaces que apuntan a páginas .php
a[href$=".php"]{...}

// Selecciona todos los enlaces que lleven a una página que contenga la palabra ejempl
o:
a[href*="ejemplo"]{...}
```

También incorpora nuevas formas de seleccionar etiquetas adyacentes:

- **h1 + h2{...}**: Etiquetas inmediatamente adyacentes.
- **h1 ~ h2{...}**: Selector general de hermanos. Válido cuando `<h2>` se encuentre después de `<h1>`, pero puede haber otras etiquetas de por medio.

Ejemplo:

```
<h1>Título</h1>
<h2>Subtítulo adyacente</h2>

<h1>Título</h1>
<p> párrafo de separación</p>
<h2>Subtítulo con selector general de hermanos</h2>
```

También podemos indicar atributos específicos de una etiqueta, con:

- **etiqueta1[atributo1="valor1"]**: seleccionaría todas las etiquetas "etiqueta1" que contengan un atributo llamado "atributo1" cuyo valor sea igual a "valor1". Por ejemplo, si queremos indicar un estilo para todas las etiquetas input que sean de tipo texto:

```
input[type="text"] {
   background: #eee;
}
```

Nuevas pseudo-clases

Una pseudo-clase es un estado o uso predefinido de un elemento al que se le puede aplicar un estilo independientemente del estilo aplicado al de su estado por defecto. En CSS3 se han añadido muchas nuevas pseudo-clases para facilitar a los programadores el uso de algunos estilos avanzados en el diseño de páginas Web. Las nuevas pseudo-clases son:

- **:nth-child(n)** - Fija el aspecto de una ocurrencia específica del elemento nodo hijo especificado. Por ejemplo, el tercer elemento nodo hijo de una lista sería "li:nth-child(3)". Además se pueden usar pequeñas expresiones como parámetro para por ejemplo seleccionar todos los elementos impares: "nth-child(2n+1)" los pares "nth-child(2n)", etc. Los elementos impares y pares también se pueden seleccionar usando "nth-child(odd)" y "nth-child(even)"
- **:nth-last-child(n)** - igual que ":nth-child(n)" pero empezando a contar desde el final.
- **:nth-of-type(n)** - Fija la apariencia de una ocurrencia específica del elemento con el tipo de selector especificado en un elemento padre. Por ejemplo la segunda lista no ordenada sería ul:nth-of-type(2). También permite los mismos parámetros que ":nth-child(#)".
- **:nth-last-of-type(n)** - igual que ":nth-of-type(n)" pero empezando a contar desde el final.
- **:first-child** - Fija el aspecto del primer elemento de un tipo de selector solo si es el primer nodo hijo de su elemento padre, por ejemplo la primera etiqueta `<li>` de una lista `<ol>` .
- **:last-child** - Ultimo elemento de una lista de elementos de un tipo dado.
- **:first-of-type** - Selecciona el primer elemento de un tipo concreto dentro de la lista de hijos.
- **:last-of-type** - Selecciona el último elemento de un tipo.
- **:only-child** - Selecciona el elemento si es el único elemento hijo.
- **:only-of-type** - Selecciona el elemento si es el único elemento hijo de ese tipo.
- **:empty** - Selecciona los elementos que no tienen hijos (incluyendo nodos de texto).
- **:enabled** - Selecciona los elementos de la interfaz que tengan el estado "enable".
- **:disabled** - Selecciona los elementos de la interfaz que tengan un estado "disabled".

- **:not(s)** - Selecciona los elementos que no coincidan con el selector especificado.
- **:lang(language)** - nos permite especificar estilos que dependen del idioma especificado por la propiedad language (en, sp, etc.)

Ejemplos de uso:

```
tr:nth-child(even) {
   background: silver;
}
tr:nth-child(odd) {
   background: white;
}
p:lang(en) {
   color: gray;
   font-style: italic;
}
```

Pseudo-clases para formularios

Además también se han añadido nuevas pseudo-clases que podemos usar en los formularios para aplicar un formato según el estado de un campo. Estas propiedades van en concordancia con los nuevos campos introducidos en HTML5 (ver la sección de formularios de HTML5). estas son:

- **:valid** - campo válido (dependerá del tipo de campo).
- **:invalid** - campo inválido (dependerá del tipo de campo).
- **:required** - campo requerido (marcado con el atributo "required").
- **:optional** - campo opcional (campo no marcado con el atributo "required").
- **:checked** - elemento marcado (o checked, válido para radio button o checkbox).
- **:in-range** - valor dentro del rango indicado (para campos numéricos o de rango).
- **:out-of-range** - valor fuera de rango (para campos numéricos o de rango).
- **:read-only** - campo de solo lectura.
- **:read-write** - campo de lectura / escritura.

Algunos ejemplos de uso:

```
<head>
   <style>
      #form1 input:valid { background:lightgreen; }
      #form1 input:invalid  { border-color:red; }
      #form1 specialInput input:valid { background:green; }
   </style>
</head>
<body>
   <form id="form1" name="form1" method="post" action="formaction.php">
      <p>Nombre:
       <input type="text" name="nombre" id="nombre" required/>
      <p/>
      <p>Usuario:
         <specialInput>
         <input type="text" name="usuario" id="usuario" required/>
         </specialInput>
      <p/>
   </form>
</body>
```

En este ejemplo cabe destacar la etiqueta "specialInput", que no es ninguna etiqueta existente, sino una nueva etiqueta que hemos creado para aplicar un formato especial.

Además podemos aplicar estas pseudo-clases en cadena y hacer cosas como:

```
input:focus:required:invalid {
   background: pink url(ico_validation.png) 379px 3px no-repeat;
}
input:required:valid {
   background-color: #fff; background-position: 379px -61px;
}
```

Dado que Internet Explorer 6-8 no soporta la mayoría de pseudo-clases se han desarrollado algunas librerías de JavaScript que realizan las mismas funciones para estos navegadores, como "select[ivizr]" que podréis descargar de su página oficial "http://selectivizr.com/".

Color

En CSS3 se han incorporado nuevas formas para definir los colores:

- **rgba(red, green, blue, opacity);** - Color RGBA. El valor de opacidad debe de estar entre 0 y 1, siendo 0 totalmente transparente. Por ejemplo, podemos usarlo de la forma:

```
background-color: rgba(255, 115, 135, 0.5);
color: rgba(255, 115, 135, 0.5);
```

- **hsl(hue, saturation, lightness);** - Modelo de color HSL.
- **hsla(hue, saturation, lightness, alpha);** - Modelo de color HSLA.
- **cmyk(cyan, magenta, yellow, black);** - Modelo de color CMYK.
- **opacity: 0.5;** - También podemos indicar el valor de transparencia u opacidad por separado, debiendo de estar este valor entre 0 y 1, siendo 0 totalmente transparente y 1 totalmente opaco. Para dar también soporte a Internet Explorer usaremos: "*filter:alpha(opacity=50);*".

Bordes

En CSS3 se han incorporado cuatro nuevas propiedades para dar formato a los bordes de una caja. Estas propiedades no están todavía plenamente soportadas en todos los navegadores, por lo que para que funcione en la mayoría de ellos tendremos que usar también las propiedades nativas del navegador (simplemente añadiremos los prefijos -webkit- y -moz-). Las nuevas propiedades son:

- **border-radius**: permite crear cajas con esquinas redondeadas. Hasta ahora esto solo se podía hacer insertando imágenes que simularan esta característica. Ahora lo podemos hacer de una forma mucho más sencilla:

```
-webkit-border-radius: 30px;
-moz-border-radius: 30px;
border-radius: 30px;
```

Además también podemos indicar cada uno de los bordes por separado:

```
-moz-border-radius-topleft: 10px;
-moz-border-radius-topright: 20px;
-moz-border-radius-bottomright: 30px;
-moz-border-radius-bottomleft: 40px;
-webkit-border-radius: 10px 20px 30px 40px;
border-radius: 10px 20px 30px 40px;
```

- **border-image**: este nuevo estilo nos permite usar una imagen como borde de una caja. Tenemos que indicar tres atributos: la imagen a utilizar, el grosor y la forma de aplicar la imagen (stretch, repeat, round, none). Ejemplo de uso:

```
-webkit-border-image: url(imagen.png) 27 repeat;
-moz-border-image: url(imagen.png) 27 repeat;
border-image: url(imagen.png) 27 repeat;
```

El resultado dependerá de la imagen que utilicemos para el borde, pero por ejemplo podríamos obtener resultados como el siguiente:

- **border-color**: Permite crear degradados en los bordes de una caja indicando la

secuencia de colores del degradado (píxel a píxel y de dentro hacia fuera), de la forma:

```
-webkit-border-bottom-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-webkit-border-top-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-webkit-border-left-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-webkit-border-right-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-bottom-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-top-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-left-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-right-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
border: 8px solid #000;
```

Con lo que obtendríamos un resultado similar a:

- **box-shadow**: Permite dar sombra a elementos de bloque. Tiene 4 atributos: la distancia horizontal de la sombra, la distancia vertical de la sombra, el desenfoque (blur) y el color de la sombra. Además podemos usar valores negativos para las distancias horizontal y vertical para crear sombras en otros sentidos. Un ejemplo de sombra en color gris:

```
-moz-box-shadow:3px 3px 6px #888888;
-webkit-box-shadow:3px 3px 6px #888888;
box-shadow:3px 3px 6px #888888;
```

Con lo que obtendríamos un resultado similar a:

Fondos

CSS3 también ha introducido nuevas propiedades para definir el estilo de las imágenes de fondo:

- **background-origin: border-box | padding-box | content-box** - permite definir el origen de coordenadas sobre el que se va a colocar la imagen de fondo. Acepta tres posible valores: "border-box" para que el fondo empiece desde el mismo borde del elemento, "padding-box" para que la imagen de fondo se coloque a partir del espaciado de padding, y por último "content-box" para que la imagen de fondo se coloque donde empieza el contenido del elemento, sin tener en cuenta el borde ni el padding.
- **background-clip: border-box | padding-box | content-box** - define el área sobre la que se extiende la imagen de fondo, puede tomar tres valores: "border-box" se extiende por toda el área dentro de la zona definida a partir del borde, "padding-box" se extiende a partir del espaciado de padding y "content-box" el fondo se extiende solo dentro del área de contenido.
- **background-size**: Permite indicar el tamaño de la imagen de fondo. Acepta diferentes atributos:
    - background-size: 200px; // especifica ancho y alto a la vez
    - background-size: 200px 100px; // 200px de ancho y 100px de alto
    - background-size: auto 200px; // ajustar la anchura automáticamente
    - background-size: 50% 25%; // También podemos indicar el tamaño con porcentajes
    - background-size: contain; // Escalar la imagen al tamaño máximo posible (conservando las proporciones originales) para que quepa dentro del área asignada.
    - background-size: cover; // Escalar la imagen para que cubra completamente el área asignada (conservando las proporciones originales).
    - Capas con múltiples imágenes de fondo: Con la propiedad **background** ahora podemos indicar varias imágenes de fondo, simplemente separándolas con comas. Para cada propiedad background debemos definir cuatro valores: imagen de fondo, posición vertical, posición horizontal, modo de repetición (repeat, repeat-x, repeat-y, no-repeat). Ejemplo:

```
background: url(imagen1.png) 10px center no-repeat,
            url(imagen2.png) 0 center repeat-x;
```

Dado que estas propiedades no son soportadas todavía en todos los navegadores, deberemos de definirlas también añadiendo los prefijos "-webkit-" y "-moz-" para dar un mayor soporte.

Texto

Las nuevas propiedades de CSS3 para dar formato a textos son:

- **text-shadow**: Permite dar sombra a un texto. Sus propiedades son distancia horizontal, distancia vertical, desenfoque (*blur*) y color de la sombra. Por ejemplo:

```
text-shadow: 2px 2px 2px #9e9e9e;
filter: dropshadow(color=#9e9e9e, offx=2, offy=2);
```

Con lo que obtendríamos un resultado similar a:

- **word-wrap: break-word;** - Permite separar palabras muy largas dentro de un elemento de bloque. Por defecto toma el valor "normal", por lo que las palabras largas se saldrían del borde del elemento. Con el valor "break-word" indicamos que las palabras pueden ser partidas para que quepan en el ancho de la caja, de la forma:

- **text-overflow: clip | ellipsis;** - Indica la forma de partir texto cuando excede el tamaño de su contenedor. Con "clip" el texto sobrante será cortado directamente aunque se quede una palabra por la mitad, mientras que "ellipsis" quitará la última palabra que no quepa y pondrá en su lugar unos puntos suspensivos. Esta propiedad de momento no funciona en Firefox.
- **font-face**: Permite utilizar tipografías diferentes a las estándar, que serán importadas desde un fichero indicado. De momento soporta los formatos: .eot, .ttf y .otf. Para importar una fuente hay que seguir la siguiente sintaxis:

```
@font-face{
   font-family: <nombre-fuente>;
   src: <source>;
   [font-weight: <weigth>];
   [font-style: <style>];
}
```

Con "font-family" indicamos el nombre que le damos a la fuente, y "src" nos permite seleccionar el fichero a cargar. Los otros dos parámetros son opcionales y tendrán valor "normal" por defecto. Por ejemplo:

```
@font-face {
   font-family: ’LeagueGothic’;
   src: url(LeagueGothic.otf);
}

// Ahora ya podemos usar esta fuente:
p {
   font-family: ’LeagueGothic’;
}
```

Columnas

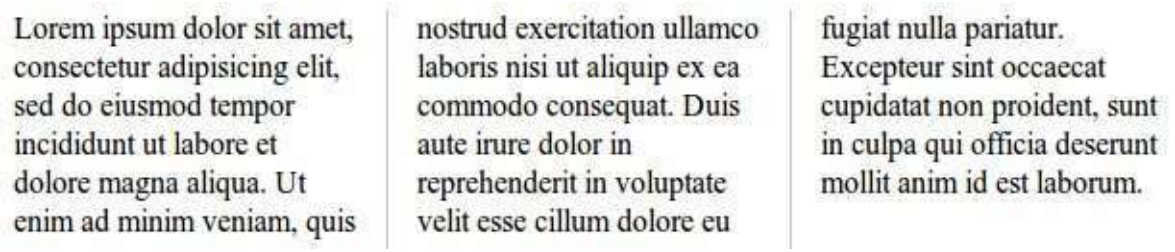
Se han añadido nuevas propiedades que nos permiten crear columnas directamente a partir de un texto, estas son:

- **column-count:** Define el número de columnas en el que se va a dividir el texto. El texto será dividido de la mejor forma posible para que ocupe todo el espacio.
- **column-width:** Define el ancho de la columna (en unidades CSS).
- **column-gap:** Define el espacio entre las columnas (en unidades CSS).
- **column-rule:** Mediante esta propiedad podemos añadir una línea separadora entre las columnas, si no especificamos esta propiedad no se añadirá ninguna línea. Debemos de indicarle tres valores: ancho de la línea (en unidades CSS), estilo de la línea (solid, dotted, double, etc.) y color de la línea.

Para dar un mayor soporte antepondremos los prefijos -webkit- y -moz-, de la forma:

```
-webkit-column-count: 3;
-webkit-column-rule: 1px solid silver;
-webkit-column-gap: 10px;
-moz-column-count: 3;
-moz-column-rule: 1px solid silver;
-moz-column-gap: 10px;
column-count: 3;
column-rule: 1px solid silver;
column-gap: 10px;
```

Con lo que obtendríamos un resultado similar a:


Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Modelo de caja básico

Se han añadido nuevas propiedades para la disposición de elementos dentro de una caja:

- **overflow: visible | hidden | scroll | auto;** - permite indicar que ocurrirá si el contenido excede el área de un elemento, acepta cuatro posibles valores:
  - *visible*: No se recorta el contenido, la parte que quede fuera será visible. Es el valor por defecto.
  - *hidden*: El contenido que sobresalga será ocultado y tampoco se mostrará la barra de scroll.
  - *scroll*: El contenido se recorta y el navegador muestra la barra de scroll para ver el resto del contenido.
  - *auto*: Si el contenido se recorta el navegador mostrará una barra para ver el resto del contenido.
- **overflow-x**: igual que overflow pero indicaremos solo la propiedad en horizontal.
- **overflow-y**: igual que overflow pero solo para vertical.
- **resize: none | horizontal | vertical | both;** - habilita la posibilidad de redimensionar "manualmente" una caja. Puede tomar los valores: none, horizontal (permitir redimensionar solo en horizontal), vertical (solo en vertical), o both (redimensionar ambas dimensiones). Se recomienda además añadir la propiedad "overflow: hidden" para ocultar los elementos al redimensionar. Por ejemplo:

```
resize:both;
overflow:auto;
```

Transiciones

Una de las propiedades más novedosas que incorpora CSS3 es la posibilidad de crear animaciones mediante transiciones y transformaciones. Se pueden aplicar transiciones a la mayoría de propiedades (posiciones, fondo, color, tamaño, etc.). Desafortunadamente, no todos los navegadores usan los nombres estándares, por lo que tendremos que añadir los prefijos "-webkit-", "-moz-" y "-o-" para dar un mayor soporte. La buena noticia es que la sintaxis para los valores en todos ellos es consistente:

- **transition-property: propertyName;** - Indica la propiedad sobre la que se aplicará la transición. Se puede aplicar sobre casi todas las propiedades: background, color, height, width, border, etc. Además también podemos usar el valor "all" para que se aplique sobre todas las propiedades disponibles, por ejemplo:

```
-webkit-transition-property: all;
-moz-transition-property: all;
-o-transition-property: all;
transition-property: all;
```

- **transition-duration: duration;** - Indica el tiempo que debe durar la transición en segundos (0.5s) o en milisegundos (500ms):

```
-webkit-transition-duration: 1s;
-moz-transition-duration: 1s;
-o-transition-duration: 1s;
transition-duration: 1s;
```

- **transition-timing-function: timingFunction;** - Es la función de tiempo que seguirá la transición, indica los cambios de velocidad a lo largo de la animación. Puede tomar cinco valores diferentes: ease (valor por defecto), linear, ease-in, ease-out, ease-in-out y cubic-bezier(cp1x, cp1y, cp2x, cp2y). Por ejemplo:

```
-webkit-transition-timing-function: linear;
-moz-transition-timing-function: linear;
-o-transition-timing-function: linear;
transition-timing-function: linear;
```

- **transition-delay: delay;** - Permite establecer un retraso inicial antes de ejecutar la transición. El tiempo de retraso se debe de indicar en segundos (0.5s) o en milisegundos (500ms):

```
-webkit-transition-delay: 0.2s;
-moz-transition-delay: 0.2s;
-o-transition-delay: 0.2s;
transition-delay: 0.2s;
```

- **transition: propertyName duration timingFunction delay;** - También podemos indicar las cuatro propiedades explicadas en una sola línea:

```
-webkit-transition: all 1s linear 0.2s;
-moz-transition: all 1s linear 0.2s;
-o-transition: all 1s linear 0.2s;
transition: all 1s linear 0.2s;
```

En general, lo mejor es declarar la transición en la propiedad base, sin pseudo-clases. De esta forma conseguiremos que se ejecute en ambas direcciones, por ejemplo:

```
.btn1 {
   background: #9c3;
   -webkit-transition: background 0.3s ease;
   -moz-transition: background 0.3s ease;
   -o-transition: background 0.3s ease;
   transition: background 0.3s ease;
}
.btn1:hover {
   background: #690;
}
```

Transformaciones

La propiedad "**transform**" nos permite aplicar transformaciones 2D o 3D a un elemento. Por ejemplo nos permite rotar, escalar, mover, etc. el elemento indicado. Esta propiedad todavía no es soportada por todos los navegadores, por lo que tendremos que añadir los prefijos "-ms-", "webkit-", "-moz-" y "-o-" para dar un mayor soporte. Algunas de las funciones de transformación que podemos utilizar son:

- **none:** Indica que no se tiene que aplicar ninguna transformación.
- **translate(x,y):** Define una traslación 2D.
- **translateX(x):** Traslación en la coordenada X.
- **translateY(y):** Traslación en la coordenada Y.
- **scale(x,y):** Define una transformación de escalado 2D, deberemos de indicar valores entre 0.1 y 2.
- **scaleX(x):** Escalado en la coordenada X, deberemos de indicar valores entre 0.1 y 2.
- **scaleY(y):** Escalado en la coordenada Y, deberemos de indicar valores entre 0.1 y 2.
- **rotate(angle):** Aplica una rotación, el ángulo debe ser indicado en grados (ejem: "30deg").
- **skew(x-angle,y-angle):** Define una transformación 2D de sesgo (o torsión), indicada en grados (deg).
- **skewX(angle):** Define una transformación de sesgo sobre la coordenada X (indicada en grados).
- **skewY(angle):** Define una transformación de sesgo sobre la coordenada Y (indicada en grados).

Además también podemos indicar varias transformaciones en una misma línea, de la forma:

```
#myDIV {
    -moz-transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg, 5deg);
    -webkit-transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg, 5deg);
    -o-transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg, 5deg);
    -ms-transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg, 5deg);
    transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg, 5deg);
}
```

Hay muchos más tipos de transformaciones, aunque algunas de ellos no son funcionales todavía (sobre todo las funciones 3D), para más información consulta: "http://www.w3schools.com/cssref/css3_pr_transform.asp".

Introducción a JavaScript

JavaScript (comunmente abreviado como "js") es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Se utiliza principalmente en su forma del lado del cliente (*client-side*), aunque existe una forma de JavaScript del lado del servidor (*Server-side JavaScript o SSJS*). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

JavaScript no es Java, aunque el nombre incluye la palabra y su sintaxis tenga algo de influencia de dicho lenguaje (también de C). Sin embargo son lenguajes totalmente diferentes, diseñados por personas diferentes, con objetivos diferentes. Conceptualmente Java es un lenguaje estático, el compilador comprueba los tipos de datos, y además es un lenguaje interpretado. JavaScript es dinámico, existen los tipos, pero solo se resuelven en tiempo de ejecución, y no es un lenguaje interpretado.

JavaScript es en general un lenguaje sencillo aunque expresivo, que ha triunfado donde Java ha fallado, en el cliente, y tiene la ventaja de que no necesita instalación, ya que está presente en todos los navegadores.

Como incluir JavaScript en nuestra página Web

La integración de JavaScript y XHTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

Incluir desde un archivo externo

Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos XHTML enlazan mediante la etiqueta `<script>`.

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

Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite. La principal ventaja de enlazar un archivo JavaScript externo es que se simplifica el código de la página, que se puede reutilizar el mismo código JavaScript en todas las páginas del sitio web y que cualquier modificación realizada en el archivo JavaScript se ve reflejada inmediatamente en todas las páginas que lo enlazan.

Incluir en el mismo documento HTML

El código JavaScript se encierra entre etiquetas `<script>` y se incluye en cualquier parte del documento:

```
<script type="text/javascript">
    alert("Hola mundo!");
</script>
```

Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (sección `<head>`) o al final de la página (antes de la etiqueta de cierre `</body>`. Con esta segunda opción se consigue mejorar el tiempo de carga de la página, ya que primero se mostrará todo el contenido de la web y por último se cargarán los javascript.

Incluir en los elementos HTML

Consiste en incluir trozos de JavaScript dentro del código HTML de la página, por ejemplo:

```
<p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
```

El principal inconveniente de este método es que ensucia innecesariamente el código HTML de la página y complica el mantenimiento del código JavaScript.

Etiqueta *noscript*

Algunos navegadores no disponen de soporte completo de JavaScript, otros permiten bloquearlo parcialmente e incluso algunos usuarios bloquean completamente el uso de JavaScript porque creen que así navegan de forma más segura.

En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento, se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página.

```
<noscript>
    <p>Bienvenido a Mi Sitio</p>
    <p>La página que estás viendo requiere para su funcionamiento el uso de
    JavaScript. Si lo has deshabilitado intencionadamente,
    por favor vuelve a activarlo.</p>
</noscript>
```

Consideraciones sobre el lenguaje JavaScript

Algunas consideraciones a tener en cuenta sobre el lenguaje JavaScript antes de empezar a ver su sintaxis son:

- No se tienen en cuenta los espacios en blanco y las nuevas líneas.
- Se distinguen las mayúsculas y minúsculas.
- No se define el tipo de las variables: una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
- No es necesario terminar cada sentencia con el carácter de punto y coma (;), pero sí que es muy recomendable.
- Se pueden incluir comentarios con " `//` " y con " `/* */` ".
- La lista de palabras reservadas que no se pueden utilizar libremente para definir variables o funciones son: *break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with*.

Variables

Las variables en JavaScript se crean mediante la palabra reservada `var`, que solamente es necesario utilizarla la primera vez (al declarar la variable), de la forma:

```
var numero1 = 2;
var numero2 = 3;
var resultado = numero1 + numero2;
```

En javascript no es necesario declarar las variables con `var`, y en este caso lo que hace es crear una variable global a la que asigna el valor correspondiente. Por esta razón se recomienda declarar siempre las variables y llevar cuidado con esta característica.

El nombre de la variable solo puede estar formado por letas, números, el símbolo de dólar "$" y el guión bajo "_", además la primera letra del nombre no puede ser un número. A continuación se incluyen algunos ejemplos de nombres válidos e incorrectos:

```
// Ejemplos correctos
var $numero1;
var _$letra;
var $$$otroNumero;
var $_a__$4;

// Ejemplos incorrectos
var 1numero;  // Empieza por un número
var numero;1_12.3;  // Contiene los carácteres ";" y "."
```

Tipos de variables

Dado que todas las variables se crean de la misma forma (mediante la palabra reservada `var`), el tipo de la variable vendrá definido según el valor que se le asigne o almacene en ella. A continuación se detallan los diferentes tipos de variables posibles:

Variables numéricas

Si a una variable se le asigna un entero o un valor decimal dicha variable se convertirá a tipo numérico, por ejemplo:

```
var num1 = 16;
var num2 = 3.1415;
```

Variables tipo cadenas de texto

Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final, por ejemplo:

```
var mensaje = "Bienvenido a nuestro sitio web";
var nombreProducto = 'Producto ABC';
var letraSeleccionada = 'c';
var texto1_1 = "Una frase con 'comillas simples' dentro";
var texto1_2 = 'Una frase con \'comillas simples\' dentro';
var texto2_1 = 'Una frase con "comillas dobles" dentro';
var texto2_2 = "Una frase con \"comillas dobles\" dentro";
```

Booleanos

Una variable de tipo boolean almacena un tipo especial de valor que solamente puede tomar dos valores: true (verdadero) o false (falso). Por ejemplo:

```
var clienteRegistrado = false;
var ivaIncluido = true;
```

Arrays

Para definir un array, se utilizan los caracteres `[` y `]` para delimitar su comienzo y su final y se utiliza el carácter `,` (coma) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Un nuevo array se puede declarar asignado valores iniciales (como en el ejemplo anterior), o vacío haciendo `var array1 = new Array();` o también `var array2 = [];` y después asignarle valores:

```
var array1 = new Array();
array1[0] = "hola";
array1[1] = "mundo";

var saludo = array1[0];  // Obtener el valor de un elemento del array
```

Las posiciones o índices del array empiezan en 0 y terminan en el tamaño del array menos uno.

Tipo de una variable

En JavaScript se puede comprobar el tipo de una variable mediante el operador `typeof`, por ejemplo:

```
typeof "John"                // string
typeof 3.14                  // number
typeof false                 // boolean
typeof [1,2,3,4]             // object
```

JavaScript define los siguientes tipos primitivos de variables: *undefined, null, boolean, number, string* y *object*. Los dos primeros (equivalentes) se utilizan para identificar cuando se accede a una variable que está sin definir. Los tipos *boolean, number* y *string* ya los hemos visto en las secciones anteriores, pero hay que destacar que el tipo *object* se utiliza para definir tanto a los arrays como a los objectos (los cuales no se tratarán en este manual de iniciación).

Operadores

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

Asignación

Este operador se utiliza para guardar un valor específico en una variable.

```
var numero1 = 3;
var numero2 = 4;
var numero3 = numero1 + numero2;
```

Incremento y decremento

Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar (++) o decrementar (--) en una unidad el valor de una variable.

```
var numero = 5;
++numero;   // Pre-incremento
numero++;   // Post-incremento
--numero;   // Pre-decremento
numero--;   // Post-decremento
```

Si el operador se utiliza como prefijo el decremento o incremento se realiza antes de la operación, si por el contrario se utiliza como sufijo se realizará después, por ejemplo:

```
var numero1 = 5;
var resultado = 2 + numero1++;
// resultado = 7

var numero1 = 5;
var resultado = 2 + ++numero1;
// resultado = 8
```

Operadores lógicos

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o booleano.

- **Negación**: Se utiliza para obtener el valor contrario al valor de la variable: `var negacion = !valor_booleano;`
- **And**: El operador se indica mediante el símbolo `&&` y su resultado solamente es *true* si los dos operandos son *true*: `var resultado = valor1 && valor2;`
- **Or**: El operador se indica mediante el símbolo || y su resultado es *true* si alguno de los dos operandos es *true*: `var resultado = valor1 || valor2;`

Operadores matemáticos

Los operadores definidos son: suma (+), resta (-), multiplicación (*), división (/) y módulo (%). A continuación se incluyen algunos ejemplos:

```
var numero1 = 10;
var numero2 = 5;
var resultado = numero1 / numero2;  // resultado 2
resultado = 3 + numero1;   // resultado 13
resultado = numero2 – 4;   // resultado 1
resultado = numero1 * numero2; // resultado 10
resultado = numero1 % numero2; // resultado 0
```

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

```
var numero1 = 5;
numero1 += 3; // numero1 = numero1 + 3 = 8
numero1 -= 1; // numero1 = numero1 - 1 = 4
numero1 *= 2; // numero1 = numero1 * 2 = 10
numero1 /= 5; // numero1 = numero1 / 5 = 1
numero1 %= 4; // numero1 = numero1 % 4 = 1
```

Operadores relacionales o de comparación

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=). El resultado de todos estos operadores siempre es un valor booleano:

```
var numero1 = 3;
var numero2 = 5;
resultado = numero1 > numero2; // resultado = false
resultado = numero1 < numero2; // resultado = true

numero1 = 5;
numero2 = 5;
resultado = numero1 >= numero2; // resultado = true
resultado = numero1 <= numero2; // resultado = true
resultado = numero1 == numero2; // resultado = true
resultado = numero1 != numero2; // resultado = false
```

Estructuras de control de flujo

Las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa.

Estructura if

Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {
    ...
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de `{...}`. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en `{...}` y el programa continúa ejecutando el resto de instrucciones del script.

La condición se evaluará de forma booleana y por lo tanto podemos utilizar cualquiera de los operadores que hemos visto para ello: !, &&, ||, >, <, ==, etc.

Esta estructura permite añadir una sección `else` que se ejecutará en caso de que no se cumpla la condición anterior e incluso concatenar varias sentencias `if` para realizar varias comprobaciones. A continuación se incluyen algunos ejemplos:

```
var nombre = "";
if(nombre == "") {
    alert("Aún no nos has dicho tu nombre");
}
else {
    alert("Hemos guardado tu nombre");
}

// If ... else if anidado
var valor = 3;
if( valor == 1 )
    alert( "La variable vale 1" );
else if( valor == 2 )
    alert( "La variable vale 2" );
else if( valor == 3 )
    alert( "La variable vale 3" );
else
    alert( "La variable tiene otro valor" );
```

Estructura switch

La estructura `switch` se utiliza para agilizar la toma de decisiones múltiples, trabaja de la misma manera que lo harían sucesivos `if`, `if else` anidados en el que según el valor de una variable entraría en uno de los casos definidos o en el caso por defecto:

```
switch (variable) {     // La variable puede ser cualquier tipo de dato
    case "ok":
        // si variable == "ok"
        break;
    case 1:
        // si variable == "1"
        break;
    default:
        // Si no es igual a ninguno de los casos anteriores
        break;
}
```

El ejemplo anterior sería igual a hacer:

```javascript
if( variable == "ok" ) {
    // si variable == "ok"
}
else if( variable == 1 ) {
    // si variable == "1"
}
else {
    // Si no es igual a ninguno de los casos anteriores
}
```

Estructura for

Esta estructura permite realizar una serie de repeticiones (también llamado bucle) mientras se cumpla una condición:

```
for(inicializacion; condicion; actualizacion) {
    ...
}
```

Donde:

- La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La "condición" es el único elemento que decide si continua o se detiene la repetición.
- La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

Ejemplo de uso con un array:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
for(var i=0; i<7; i++) {
    alert(dias[i]);
}
```

Estructura while

Es similar a `for`, repetirá el contenido del bucle mientras se cumpla la condición inicial:

```
while (condicion) {
    ...
}
```

Estructura do...while

Esta estructura es similar al bucle tipo `while` pero evalua la condición al final del bucle, con lo que se asegura que al menos se ejecutará una iteración:

```
do {
    ...
} while (condicion);
```

Estructura for...in

Este tipo de bucle, derivado de la estructura tipo `for`, permite iterar entre los elementos de un array o de un objeto (no tratados en esta introducción) de una forma muy sencilla:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

for(i in dias) {
    alert(dias[i]);
}
```

Funciones útiles de JavaScript

JavaScript incorpora una serie de herramientas y utilidades para el manejo de las variables.

Funciones útiles para mostrar resultados

- La función " `alert(texto);` " nos permite mostrar un valor o cadena en una ventana emergente.
- La función " `console.log(texto);` " nos permite mostrar un valor o cadena en la consola del sistema. Esta función se suele utilizar para depuración.

Funciones útiles para cadenas de texto

- Longitud de una cadena: se obtiene a partir de su propiedad " `.length` "
- Concatenación de cadenas: operador " `+` "
- Convertir en mayúsculas: `toUpperCase()`
- Convertir en minúsculas: `toLowerCase()`
- Obtener un carácter de la cadena: `charAt(posicion)`

Ejemplos:

```
var mensaje = "Hola Mundo";
var numeroLetras = mensaje.length; // numeroLetras = 10
var concatenacion = "¡" + mensaje + "!";
var mayusculas = mensaje.toUpperCase();
var letra = mensaje.charAt(0); // letra = H
```

Funciones útiles para arrays

- Longitud del array: se obtiene a partir de su propiedad " `.length` "
- Añadir un elemento al final del array: " `push(valor)` "

Ejemplos:

```
var array = [1, 2, 3];
var numeroElementos = array.length; // numeroElementos = 3
array.push(4);  // contenido del array = [1, 2, 3, 4]
```

Funciones útiles para números

- Comprobar posibles valores numéricos no definidos: " `isNaN(valor)` "
- Comprobar si el valor es finito: " `isFinite(valor)` "
- Formatear / redondear números decimales: " `.toFixed(digitos)` "

Ejemplos:

```
var valor1 = 3.14159265358979323846;
var valor2 = 0;

if( isNaN(valor1/valor2) || !isFinite(result) )
    alert("La división no está definida para los números indicados");

var valor3 = valor1.toFixed(2); // 3.14
```

Funciones

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente. Las funciones en JavaScript se definen mediante la palabra reservada `function`, seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion() {
    ...
}
```

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. A continuación se incluye un ejemplo de función y su llamada:

```
<script>
function calculaSuma() {
    var valor1 = 3;
    var valor2 = 7;
    var resultado = valor1 + valor2;
    alert( "El resultado es " + resultado );
}
calculaSuma();  // llamada a la función "calculaSuma"
</script>
```

Igual que en otros lenguajes, a las funciones se le pueden pasar valores o *argumentos* de entrada y pueden devolver un valor como resultado:

```
function calculaSuma(valor1, valor2) {
    var resultado = valor1 + valor2;
    return resultado;
}
var resultado = calculaSuma(10, 4);  // llamada a la función "calculaSuma"
alert( resultado );
```

Es importante saber que:

- Se puede utilizar un número ilimitado de argumentos.
- El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios. En caso de que un argumento no esté definido tendrá el valor "undefined".

Ámbito de las variables

Con respecto al ámbito de las variables tenemos que tener en cuenta las siguientes consideraciones:

- Si una variable se define con `var` dentro de una función el ámbito de dicha variable será local.
- Si una variable se define fuera de una función su ámbito será global, y por lo tanto serán accesibles desde dentro de las funciones.
- Si una variable se define dentro de una función pero sin usar la palabra reservada `var` el ámbito de dicha variable será global.

Ejemplos:

```
var global1 = "Variable Global 1";

function mostrarMensaje() {
    var local1 = "Variable local 1";
    global2 = "Variable Global 2";
    console.log( global1 );  // mostrará "Variable Global 1"
}
mostrarMensaje();
console.log(local1);  // error, no existe la variable "local1"
console.log(global2); // mostrará "Variable Global 2"
```

DOM

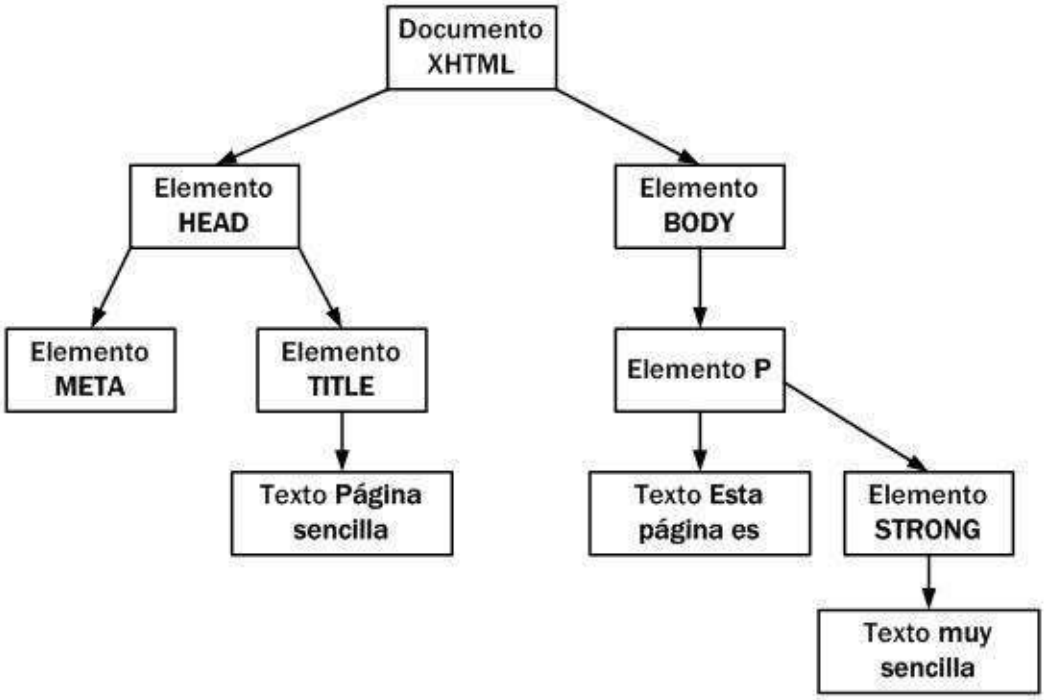
El *Document Object Model* o DOM ('Modelo de Objetos del Documento' o 'Modelo en Objetos para la Representación de Documentos') es esencialmente una interfaz que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, aportando una interfaz estándar para acceder a ellos y manipularlos. A través del DOM se pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, y es para lo que se diseñó principalmente.

DOM transforma todos los documentos XHTML en un conjunto de elementos llamados nodos, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama "árbol de nodos".

Por ejemplo, la siguiente página HTML:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Página sencilla</title>
</head>
<body>
    <p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se transformaría en el siguiente árbol de nodos:



Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Como acceder a un nodo del árbol es equivalente a acceder a "un trozo" de la página, por lo que es posible manipular de forma sencilla la página: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc.

DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado.

Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar a él descendiendo a través de todos sus nodos padre.

Acceso a nodos del árbol DOM

getElementsByTagName()

Obtiene todos los elementos de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función, por ejemplo:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que se indica delante del nombre de la función (en este caso *document*, que sería la raíz del árbol DOM) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor *document* como punto de partida de la búsqueda.

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos del DOM, no un array de cadenas de texto o un array de objetos normales.

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++) {
    var parrafo = parrafos[i];
}
```

La función `getElementsByTagName()` se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");
```

getElementsByName()

Esta función es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo `name` sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementsByName("especial");
<p name="prueba">...</p>
<p name="especial">...</p>
<p>...</p>
```

Internet Explorer 6.0 no implementa de forma correcta esta función, ya que sólo la tiene en cuenta para los elementos de tipo `<input>` y `<img>`. Además, también tiene en consideración los elementos cuyo atributo id sea igual al parámetro de la función.

getElementById()

Esta es la función más utilizada cuando se desarrollan aplicaciones web dinámicas ya que permite acceder directamente a un nodo y poder leer o modificar sus propiedades.

La función `getElementById()` devuelve el elemento HTML cuyo atributo `id` coincide con el parámetro indicado en la función. Como el atributo `id` debe ser único en toda la página, la función devuelve únicamente el nodo deseado.

```
var cabecera = document.getElementById("cabecera");
<div id="cabecera">
<a href="/" id="logo">...</a>
</div>
```

Internet Explorer 6.0 también interpreta incorrectamente esta función, ya que devuelve también aquellos elementos cuyo atributo name coincida con el parámetro proporcionado a la función.

Creación y eliminación de nodos

Con JavaScript también se pueden añadir y eliminar elementos dinámicamente del árbol DOM. Para esto se utilizan las funciones `createElement(etiqueta)`, `createTextNode(text)`, `appendChild(child)` y `removeChild(child)`.

Dado que su uso se queda fuera de los contenidos de este curso, si se desea obtener más información se puede consultar la web:

- http://www.w3schools.com/jsref/met_document_createelement.asp
- http://www.w3schools.com/jsref/met_document_createtextnode.asp
- http://www.w3schools.com/jsref/met_node_appendchild.asp
- http://www.w3schools.com/dom/met_element_removechild.asp

Acceso a atributos de un nodo

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos y todas las propiedades CSS de cualquier elemento de la página.

Los atributos de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo detrás del nombre del nodo.

El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
// HTML
<a id="enlace" href="http://www.ua.es">Universidad de Alicante</a>
<img id="imagen" style="margin:0; border:0;" src="logo.png" />
<p id="parrafo" style="font-weight: bold;" class="destacado">Texto</p>

// JavaScript
var enlace = document.getElementById("enlace");
var imagen = document.getElementById("imagen");
var parrafo = document.getElementById("parrafo");

console.log(enlace.id);       // muestra "enlace"
console.log(enlace.href);     // muestra "http://www.ua.es"
console.log(enlace.innerHTML);// muestra "Universidad de Alicante"

console.log(imagen.id);       // muestra "imagen"
console.log(imagen.src);      // muestra "logo.png"
console.log(imagen.style.margin); // muestra "0px"
console.log(imagen.style.border); // muestra "0px none"

console.log(parrafo.id);               // muestra "parrafo"
console.log(parrafo.innerHTML);        // muestra "Texto"
console.log(parrafo.style.fontWeight); // muestra "bold"
console.log(parrafo.className);        // muestra "destacado"
```

Es importante destacar que si el nombre de una propiedad o estilo es compuesto, como "font-weight", que tiene un guión "-" en el medio, para acceder dicho guión se elimina y además se escribe en mayúsculas la primera letra se la siguiente palabra, por ejemplo:

- *font-weight* se transforma en *fontWeight*
- *line-height* se transforma en *lineHeight*
- *border-top-style* se transforma en *borderTopStyle*
- *list-style-image* se transforma en *listStyleImage*

También es importante destacar que como la palabra `class` está reservada por JavaScript, no es posible utilizarla para acceder al atributo `class` del elemento XHTML. En su lugar, DOM utiliza el nombre `className` para acceder.

Eventos

Los eventos permiten ejecutar acciones cuando sucede un determinado evento o se realiza una determinada acción sobre un elemento HTML de nuestra página Web. La forma de definirlos es similar a los atributos HTML (`evento="acción"`), donde la acción hará referencia a una función o método en lenguaje JavaScript:

```
<div onclick="CODIGO-JAVASCRIPT"></div>
```

Algunos de los eventos que podemos utilizar son:

- **onload**: se activa cuando el navegador termina de cargar todos los elementos de la página.
- **onunload**: se activa al cerrar una página.
- **onclick**: cuando se presiona el botón del ratón sobre un elemento.
- **ondblclick**: se activa al hacer doble clic sobre un elemento.
- **onmousedown**: se activa al presionar el botón del ratón (mientras que está presionado).
- **onmouseup**: cuando el botón del ratón es liberado.
- **onmouseover**: se dispara cuando el cursor del ratón pasa sobre un elemento.
- **onmousemove**: cuando se mueve el cursor del ratón mientras está sobre un elemento.
- **onmouseout**: se activa cuando el cursor del ratón sale fuera de un elemento (sobre el que estaba).
- **onfocus**: ocurre cuando un elemento recibe el enfoque (el cursor de escritura), ya sea con el puntero o con mediante la tecla tabulador.
- **onblur**: se dispara cuando un elemento pierde el enfoque (ya sea por hacer clic fuera o por presionar la tecla tabulador).
- **onkeypress**: ocurre cuando se presiona una tecla (dentro de un elemento, por ejemplo un campo de escritura).
- **onkeydown**: se dispara cuando una tecla es presionada (dentro de un elemento)
- **onkeyup**: cuando una tecla es soltada.

- **onsubmit**: se activa cuando un formulario es enviado.
- **onreset**: ocurre cuando un formulario es reseteado.
- **onselect**: cuando el usuario selecciona un texto en un campo de texto.
- **onchange**: ocurre cuando un control pierde el enfoque y su valor ha sido modificado desde que recibió el enfoque.

Ejemplos de uso:

```
<!-- Dos eventos en una misma etiqueta... -->
<div onclick="alert('Has hecho click');" onmouseover="alert('Acabas de pasar por encim
a');">
    Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima
</div>

<!-- Comprobar que la página se ha cargado... -->
<body onload="alert('La página se ha cargado completamente');">
    ...
</body>

<!-- También se pueden llamar a funciones desde los eventos... -->
<script type="text/javascript">
   function saveText() {
      // acciones JavaScript
   }
</script>

<textarea id="myarea" cols="80" rows="15" onkeyup="saveText()"></textarea>
```

Eventos y la variable *this*

JavaScript define una variable especial llamada *this* que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable *this* para referirse al elemento que ha provocado el evento.

Esta variable es muy útil para ejemplos como el siguiente: queremos que al pasar por encima de un `<div>` el color del borde cambie, y al salir del contenedor restablezca el color inicial. Si no usamos la variable *this* el código sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="document.getElementById('contenidos').style.borderColor='black';"
onmouseout="document.getElementById('contenidos').style.borderColor='silver';">
    Sección de contenidos...
</div>
```

Sin embargo, usando la variable *this* quedaría mucho más claro:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="this.style.borderColor='black';"
onmouseout="this.style.borderColor='silver';">
    Sección de contenidos...
</div>
```

Si quisieramos llamar a una función externa, también es posible usar la variable *this* para pasarle como parámetro el elemento sobre el cual se quiere actuar, por ejemplo:

```
<script>
function setColor(element, color) {
    element.style.borderColor = color;
}
</script>

<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="setColor(this,'black');" onmouseout="setColor(this,'silver');">
    Sección de contenidos...
</div>
```

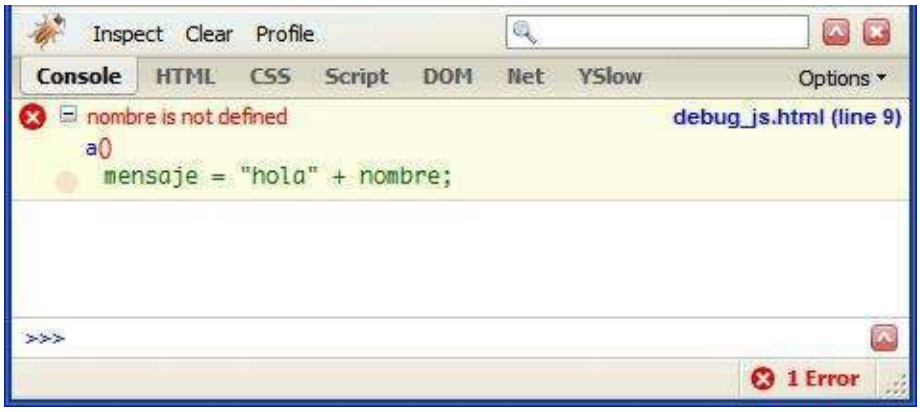
Detección de errores con Firefox y Chrome

Firefox y Chrome proporcionan herramientas para ayuda al desarrollador que están incluidas por defecto y que son de gran utilidad a la hora de depurar y detectar errores en una web en desarrollo.

Para abrir estas utilidades se puede pulsar sobre la opción correspondiente del menú (en la sección "Herramientas") o directamente pulsando la tecla de función F12.

La consola de errores permite diferenciar los mensajes de información, los mensajes de aviso y los mensajes de error. Además, permite visualizar todos los errores de la página simultáneamente. Por cada error detectado se indica la posible solución mediante un mensaje en inglés y se muestra el trozo de código del script donde se ha producido el error. Además, pulsando sobre el enlace incluido se accede a la línea concreta del archivo concreto donde se ha producido el error.

Firefox permite instalar pequeñas mejoras y ampliaciones en el navegador, que se conocen con el nombre de extensiones. Una de las extensiones más interesantes para los desarrolladores de aplicaciones web es Firebug, que se puede descargar gratuitamente desde http://www.getfirebug.com/



Más información

Para obtener una referencia mucho más completa sobre las hojas de estilo podemos consultar alguna de las siguientes Webs:

- Referencia detallada de todos los estilos: http://www.w3schools.com/cssref/default.asp
- Especificaciones: http://www.w3.org/Style/CSS/
- Tutoriales: http://www.desarrolloweb.com/manuales/manual-css-hojas-de-estilo.html

Sobre CSS3 podemos obtener más información en:

- http://www.w3schools.com/css3/default.asp
- http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/

Además, existen algunas páginas Web que proporcionan "generadores de estilos CSS" que nos pueden ayudar, como:

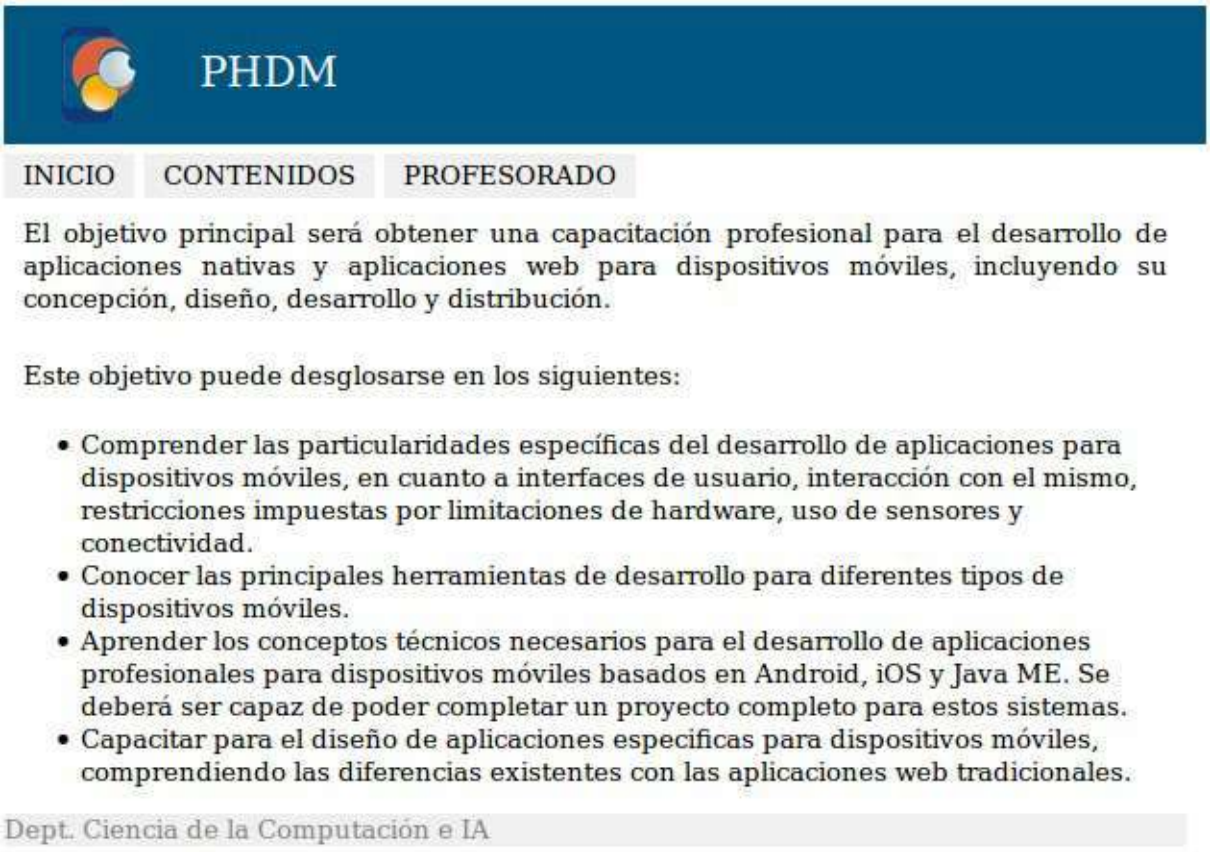
- http://css3generator.com/
- http://www.colorzilla.com/gradient-editor/

Ejercicios 1 - Ejercicios sobre HTML, HTML5, CSS y CSS3

En esta primera sesión vamos a realizar algunos ejercicios para practicar con las distintas versiones de HTML, HTML5, CSS y CSS3. Descarga la plantilla para los ejercicios de los materiales.

Ejercicio 1 - Estructura en HTML (0.6 puntos)

En este primer ejercicio vamos a hacer una pequeña Web de ejemplo utilizando las etiquetas DIV para estructurar el contenido. Nuestra Web va estar dividida en cuatro apartados, cada uno tendrá un identificador único: *header, nav, content, footer*. El resultado final tendrá que ser similar al de la siguiente imagen:



En la parte superior vamos a crear un campo DIV con identificador "header", en el que incluiremos el logo del curso (logo.jpg, con un alto de 50px) y el texto "PHDM" (color blanco con tamaño de 16 puntos). A continuación incluiremos la barra de navegación, otro campo DIV con identificador "nav". Dentro de este a su vez crearemos 3 cajas tio DIV con la **clase**

"navElement". Las cajas tendrán los textos "Inicio", "Contenidos" y "Profesorado". Como estilo indicaremos que la clase "navElement" tenga el color de fondo "#f0f0f0" y que cambie al color "#005682" cuando el ratón pase por encima.

La sección central la crearemos también mediante una etiqueta DIV con el identificador "content". Esta sección contendrá una lista no ordenada (del tipo `UL`), el contenido de esta lista lo podemos obtener del fichero con la plantilla para este ejercicio.

El pie de página lo crearemos también utilizando una etiqueta DIV con el identificador "footer" y el texto "Dept. Ciencia de la Computación e IA". Como estilo le aplicaremos el color de texto gris.

Por último tenemos que terminar de ajustar los estilos CSS para que la página se vea correctamente. Para los elementos "header", "nav", "content" y "footer" definiremos un ancho del 100% e indicaremos que no se permiten elementos por su lado izquierdo (es decir, que deben de estar en una nueva línea, esto lo haremos mediante `clear:left;`). Para el estilo "navElement" indicaremos que se tiene que situar a continuación del anterior (en la misma línea, esto lo haremos mediante `float: left;`).

Ejercicio 2 - Terminando la Web de ejemplo (0.6 puntos)

En este ejercicio vamos a terminar la web del ejercicio 1, añadiéndole el contenido de los enlaces que faltan. En primer lugar copiamos el fichero resultado del ejercicio anterior y lo renombramos a "ejercicio2.html". En este fichero vamos a modificar las opciones de menú para añadir enlaces a *Inicio* (un enlace a este mismo fichero), *Contenidos* (enlace a "ejercicio2_contenidos.html") y *Profesorado* (enlace a "ejercicio2_profesorado.html").

Al añadir estos enlaces se modificará su apariencia, por lo que tendremos que modificar la hoja de estilo. Añadimos dos nuevos estilos "`.navElement a`" para indicar que el color de los enlaces es negro y que no se dibuje la línea de subrayado del enlace (`text-decoration: none;`). Y otro estilo "`.navElement a:hover`" para que el enlace cambie a color blanco cuando el ratón pase por encima.

El siguiente paso es pasar todos estos estilos a un fichero independiente, llamado "ejercicio2_css.css". Simplemente tendremos que crear este fichero y cortar y pegar en él todos los estilos que ya tenemos creados. En el fichero principal HTML tendremos que cargar esta hoja de estilo, quedando solo una línea (`<link href="ejercicio2_css.css" rel="stylesheet" type="text/css" />`).

Ahora vamos a crear los dos ficheros HTML que faltan. Para esto realizamos dos copias del fichero HTML principal y las renombraremos a "ejercicio2_contenidos.html" y "ejercicio2_profesorado.html". En cada una de estas copias solo tendremos que cambiar el contenido de la zona central. Para el fichero de contenidos buscaremos el índice general de contenidos del curso y lo añadiremos en una lista no numerada (UL). Y para el fichero de profesorado añadiremos también en una lista no numerada los nombres de los profesores.

Por último vamos a hacer que al cambiar de sección se quede marcado el enlace correspondiente. Esto lo haremos añadiendo la clase "visited" solo al enlace de la sección actual, es decir, en cada página (inicio, contenidos o profesorado), añadiremos la clase "visited" solo a la opción del menú que está abierta en ese momento.

Nota: Para añadir más de un clase a un elemento HTML lo podemos hacer separando las clases con espacios, de la forma: `<div class="navElement visited">`. Finalmente definimos el estilo ".visited" en la hoja de estilo con el color de fondo "#005682" y el color de texto blanco.

Ejercicio 3 - Estructura de HTML 5 (0.6 puntos)

En este ejercicio vamos a modificar la web que hemos hecho en el ejercicio anterior para aplicarle las nuevas etiquetas semánticas de HTML5. Para esto seguiremos los siguientes pasos:

- Copiamos los ficheros del ejercicio anterior y los renombramos por "ejercicio3".
- Cambiamos la dirección de los enlaces y la inclusión del fichero CSS en la cabecera para que apunten correctamente a los nuevos ficheros.
- En cada uno de los ficheros HTML cambiamos las etiquetas DIV principales (con identificadores *header, nav, content, footer*) y las sustituimos por las etiquetas semánticas de HTML5 (*header, nav, article y footer*).
- Modificamosel fichero CSS para aplicar los estilos sobre las nuevas etiquetas semánticas de HTML5 (simplemente tendremos que cambiar los identificadores por los nombres de estas etiquetas, por ejemplo: "#header" por "header").

Ejercicio 4 - Canvas (0.2 puntos)

Para practicar con el elemento Canvas vamos a dibujar unas sencillas figuras geométricas. En primer lugar, en el cuerpo del documento, tenemos que crear el canvas con identificador "myCanvas" y dimensiones de 360x240.

En el código JavaScript btendremos la instancia del canvas a partir de su identificador "myCanvas" y adquiriremos su contexto 2D. Definiremos un estilo de relleno con color '#0000ff', un color de borde '#ff0000' y un grosor de línea de 4 píxeles.

Dibujamos un rectángulo relleno en las coordenadas (0, 0, 150, 50), y otro rectángulo usando solo el borde en las coordenadas (0, 60, 150, 50). Por último dibujaremos un triángulo usando la herramienta trazo (path). Iniciamos el trazo (beginPath), definimos el primer punto en (160, 0) y los siguientes puntos en (270, 0), (160, 110) y (160, 0). Por último indicamos que rellene la figura y que cierre el trazo.

Ejercicio 5 - Geolocalización (0.2 puntos)

En este ejercicio solo tenemos que añadir una línea, y es el comando necesario para obtener la posición actual, al cual le pasaremos como parámetro el nombre de la función "showPosition". Esta función será la encargada de mostrar nuestras coordenadas (utilizando la API de Google Maps).

Nota: si no funciona correctamente es posible que sea por problemas de permisos si abrimos el fichero html directamente. Para solucionarlo podemos colocar el ejercicio en un servidor web local y acceder a él a través del *localhost*.

Ejercicio 6 - Almacenamiento Offline (0.3 puntos)

Para practicar con la nueva funcionalidad de almacenamiento Offline vamos a hacer un pequeño ejemplo que guarde de forma automática una nota. Si abrimos la plantilla correspondiente solo tenemos que definir las funciones de cargar, guardar y borrar la nota. Para esto utilizaremos el almacenamiento en local (localStorage) y el identificador 'savedNote', y para el borrado eliminaremos todo el contenido directamente (clear). Además en la sección de estilo CSS indicaremos para el "contenedor-nota" que utilice la imagen de fondo "imgs/stickynote.jpg".

Nota: si no funciona correctamente es posible que sea por problemas de permisos si abrimos el fichero html directamente. Para solucionarlo podemos colocar el ejercicio en un servidor web local y acceder a él a través del *localhost*.

Ejercicio 7 - CSS3 (0.5 puntos)

En este ejercicio vamos a probar algunas de las funcionalidades nuevas de CSS3. Para todos los ejemplos recuerda indicar los nombres de las propiedades usando también los nombres nativos de los navegadores con los prefijos -webkit-, -moz- y -o-.

En el primer ejemplo "borderRadius" vamos a indicar que el cuadro tenga bordes redondeados con un radio de 30 píxeles.

En el segundo ejemplo (borderShadow) crearemos una sombra para el borde, con los siguientes atributos: distancia horizontal de la sombra de 5px, distancia vertical de la sombra 5px, desenfoque de 6px y color de la sombra grisáceo (#888888).

En el tercer ejemplo (textShadow) vamos a crear una sombra para el texto de 2px para sus distancias horizontal y vertical, de 2 píxeles para el desenfoque y "#9e9e9e" como color de sombra.

En el cuarto ejemplo (multiColumn) vamos a probar la funcionalidad de columnas. Aquí solo tendremos que indicar que el número de columnas es de 2 y que el espacio entre las columnas es de 15px.

En el último ejemplo (boxTransition) vamos a crear un efecto de transición. Usando la propiedad "transition" (para establecer todos los valores en una sola línea), indicaremos que vamos a realizar la transición sobre "margin-left", con una duración de 3s, y usando la función de tiempo "ease-in-out".

Ejercicios 2 - Ejercicios sobre JavaScript

Ejercicio 1 - Calculadora sencilla (1 punto)

Para practicar con javascript vamos a crear una calculadora sencilla como la que se muestra en el siguiente esquema:

Para ellos seguiremos los pasos:

- En primer lugar escribiremos el código HTML para diseñar una calculadora como la que se muestra en el esquema de la imagen. Es importante que asignemos un identificador único a los campos tipo " `input` " de entrada de datos y al campo en el que se mostrará el resultado (para este campo podemos asignar un identificador a una etiqueta `span` vacía).
- Crearemos una función para cada operación que se llamará en el evento " `onclick` " de cada botón.
- Las funciones deben comprobar que se haya escrito algún valor en los campos y en caso de error mostrar un aviso. Además, en la función de división se deberá de comprobar que el resultado sea correcto (finito) y en caso de error se mostará también un mensaje.

> Para obtener o asignar valor a los campos tipo `input` usaremos su propiedad `.value` , mientrar que para asignar un valor a otro tipo de elementos HTML (como párrafos (p), cajas (div), span, etc.) utilizaremos su propiedad `.innerHTML` .
>
> Al obtener el valor de un `input` se obtiene con tipo cadena, para realizar las operaciones correctamente tendréis que convertirlo a decimal mediante la función `parseFloat(valor)` .

Ejercicio 2 - Calculadora avanzada (1 punto)

En este ejercicio vamos a crear una calculadora un poco más avanzada que la del ejercicio anterior. En primer lugar escribiremos el código HTML para crear una calculadora con un diseño similar al de la siguiente figura:

Nos podemos ayudar de una tabla de HTML para la disposición de los elementos. La pantalla de la calculadora será un campo DIV al cual asignaremos valores mediante la función `.innerHTML`.

El código JavaScript constará de 3 funciones:

- Una función "`limpiar()`" que se llamará al pulsar la tecla "C" y que borrará el contenido de la pantalla.
- Una función "`setValue(valor)`" que añadirá el valor pasado por parámetro al contenido ya existente en la pantalla. Esta función se utilizará tanto para añadir números (`setValue(2)`) como para añadir las operaciones (`setValue('+')`) y el separador decimal (`setValue('.')`).
- Una función "`calcular()`" que calculará la operación introducida en la pantalla y mostrará el resultado de la misma. Para realizar los cálculos haremos uso de la función de javascript `eval`, la cual evalua la expresión que recibe por parámetro y devuelve el resultado. Además, dado que la expresión puede contener errores es necesario introducirla en un bloque `try...catch` como el siguiente:

```
try {
    pantalla.innerHTML = eval( expr );
} catch (e) {
    // error
}
```

Notas:

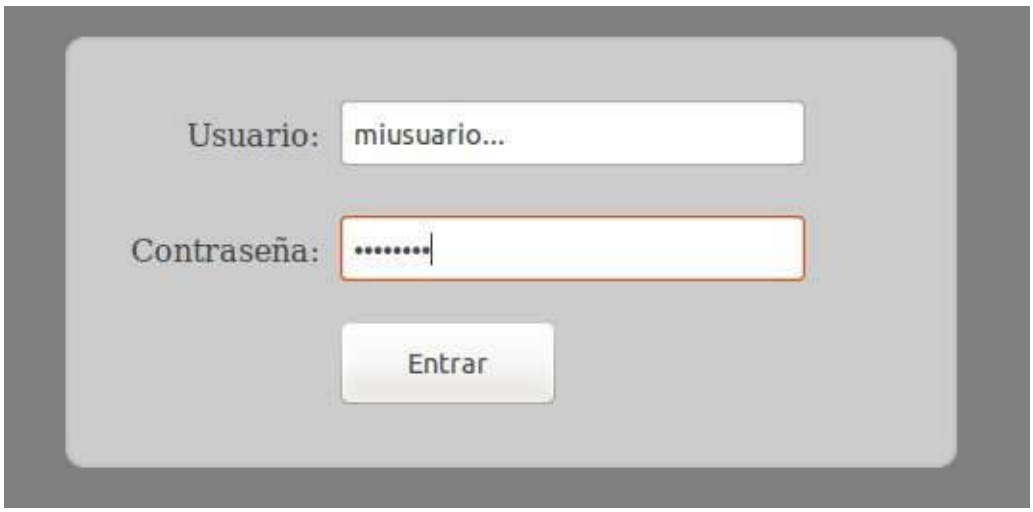
- Es necesario controlar los siguientes errores:
  - Cuando se llame a "`calcular()`" y no haya nada introducido en la pantalla.

    - Si se produce una excepción al evaluar la expresión.
    - Si el resultado de la operación no es un número o no es finito.
- En caso de error se mostrará el aviso "ERROR" por la pantalla.
- Después de un error, si se pulsa limpiar o se introduce un valor se borrará el aviso.

Ejercicio 3 - Validación de un formulario (1 punto)

En este último ejercicio vamos a crear un formulario para el acceso a la sección privada de una web mediante usuario y contraseña, esto nos valdrá para practicar con la funcionalidad de javascript validando sus campos.

En primer lugar crearemos el HTML del formulario de login, el cual deberá ser similar al de la siguiente imagen:



- El fondo de la web tendrá el color "*gray*".
- La caja contenedora tendrá un ancho de 400px y estará centrada en la pantalla. Su color de fondo será #ccc y tendrá un *border* de 2px de color *silver*. Además tendrá un espaciado interior de 30px.
- Se tendrá que adaptar también el resto de elementos para que se muestren con un aspecto y disposición similar al de la figura.

Se aconseja crear una sección de estilos CSS para agrupar los estilos del formulario.

A continuación crearemos una función de JavaScript para realizar la validación del formulario, la cual mostará los errores que hubiera o indicará que se ha validado correctamente. Para incluir los mensajes de error se pueden incluir directamente en el HTML inicial del formulario, pero se marcarán como ocultos usando la propiedad CSS: `display:none` y cuando se quierdan mostrar se cambiará a `display:block`.

> Nota: para cambiar el estilo de un campo en JavaScript se puede hacer accediendo a su propiedad `style` de la forma: `campo.style.display = 'block';`

Al pulsar sobre el botón "Entrar" se llamará a una función " `validar()` " que realizará las siguientes acciones:

- Si alguno de los campos está vacío mostrará un mensaje de error indicándolo.
- Si el campo no está vacío pero la longitud es inferior a 3 caracteres también se mostará un error indicándolo (ver la imagen inferior).
- Si no se produce ningún error aparecerá el aviso: "¡Validación correcta!

Notas:

- Si se pulsa varias veces el botón "Entrar" hay que actualizar los errores: si un error anterior ya no se produce el aviso tendrá que desaparecer.
- Solo se podrá mostrar un mensaje de error por cada campo: campo requerido o longitud mínima incorrecta.