Licencia

[1] http://creativecommons.org/licenses/by-nc-sa/2.5/es/

XHTML + CSS
¡de una maldita vez!

Belén Albeza (BenKo)

3 de diciembre de 2006

Índice general

Capítulo 1

Introducción

¡Yo te convoco, dragón invoco!

Dragon Fall GTI
Pilaf, villano de Dragon Fall

Este manual cubre cómo crear páginas web usando las **tecnologías estándares** recomendadas por el World Wide Web Consortium[1]. No son necesarios conocimientos previos, así que lo único que necesitas es:

- Un **editor** de texto plano. Si eres un desafortunado usuario de Windows, te sirve el Bloc de Notas. Ten cuidado si usas un procesador de textos, como el Word, ya que da formato y no queremos eso. Si usas GNU/Linux, tienes un montón para elegir: vim, emacs, Joe, Kate, etc.
- Un **navegador** (o varios) que funcione **bien** y respete los **estándares**. El Mozilla Firefox, por ejemplo. Es multiplataforma, libre, y lo puedes bajar desde la web del proyecto Mozilla[2].
- Un **navegador** que funcione **mal**, tenga todo el mundo, y pase de los estándares: *Ya-Sabes-Cuál*[3].

[1] Es el organismo que se encarga de regular los estándares de la Web. Algo así como la RAE con el castellano. Su dirección es www.w3c.org

[2] http://www.mozilla.org

[3] Su nombre está escrito en la Lengua Negra, que no pronunciaré aquí.

Además de eso, es muy recomendable que tengas a mano el editor de texto y un navegador para ir probando los ejemplos que vayan saliendo. Es la única forma de aprender.

1.1. ¿Qué es el XHTML?

XHTML significa *eXtensible HyperText Markup Language* y es la versión modernizada del tradicional HTML[4]. Si ya conoces HTML, al final del manual hay un apéndice para que sea más fácil la migración a XHTML.

XHTML es un lenguaje **semántico**, lo que quiere decir que no definimos el *aspecto* de las cosas, sino lo que *significan*. Por ejemplo, si tenemos el título de nuestra página, en lugar de decir *"Lo quiero grande en letras rojas"*, le indicamos al navegador que *"Este es el título principal de la página. Haz algo para que destaque"*. Y ese "algo" lo dejamos a decisión del navegador.

Obviamente, podemos controlar el aspecto que tienen nuestras páginas, pero eso es tarea de las hojas de estilo CSS, no del XHTML.

1.2. ¿Y eso de CSS?

CSS son las siglas de *Cascading Style Sheets* y son un regalo del cielo. Si el documento XHTML está bien estructurado, podemos cambiar totalmente su apariencia sin tocar una sola línea de código en el archivo `.html`. Esto nos permite **separar el contenido del aspecto**, y es de gran importancia.

Si quieres ver un ejemplo de cómo con el mismo código XHTML y distintos archivos CSS se pueden conseguir resultados totalmente diferentes, a la vez que espectaculares, échale un vistazo al *CSS Zen Garden*[5] y al *Proyecto Camaleón*[6].

1.3. ¿Cómo funciona el XHTML?

XHTML es un lenguaje basado en **etiquetas** (*tags*). Una etiqueta tiene la siguiente forma:

[4] Lenguaje utilizado para crear páginas web.
[5] http://www.csszengarden.com
[6] http://www.camaleoncss.com

```
<etiqueta>Algo aquí dentro</etiqueta>
```

Volviendo al ejemplo anterior de nuestro título: la etiqueta para poner el título principal en la página es `<h1>`. Entonces nos quedaría:

```
<h1>What is the Matrix?</h1>
```

Como ves, `<h1>` marca el **inicio** de la etiqueta, mientras que `</h1>` se encarga de **cerrarla**. Hay etiquetas que funcionan con una sola parte, y son así:

```
<etiqueta />
```

Observa el espacio en blanco que hay entre el nombre de la etiqueta y la barra `/`. Es muy importante para que los navegadores antiguos no se vuelvan locos.

Hay etiquetas que pueden modificarse con **atributos**. Ahora mismo no hace falta entender qué hacen, ya los veremos más adelante. Sólo quédate con cómo van escritos:

```
<etiqueta atributo="valor">
```

Por último, queda comentar un par de reglas que siguen las etiquetas: siempre en **minúsculas** y los atributos **entre comillas**[7].

1.4. ¿Pero para hacer webs no se usa el FrontPage?

Sí. También puedes freír huevos con aceite para el coche.

1.5. Mi amigo que es diseñador usa Dreamweaver

El Macromedia Dreamweaver es un editor WYSIWYG[8] muy extendido e idolatrado por infinidad de diseñadores. El problema es que nos permite hacer webs sin tocar nada de código.

Sí, eso es un problema. Genera código basura y no tenemos control sobre lo que hacemos. Así que antes de usar Dreamweaver

[7] Esto es un diferencia respecto al HTML, ya que podíamos poner atributos sin comillas y no importaba si escribíamos en mayúsculas o minúsculas

[8] *What You See Is What You Get* (en castellano, "lo que ves es lo que obtienes").

o algo similar, tenemos que aprender a hacer las webs nosotros solitos. Que quede claro que es mi opinión y no Ley Divina...

1.6. Una preguntilla...

¿Has mirado primero en Google[9]? Es muy listo y lo sabe (casi) todo. Si aún así no te aclaras, mi e-mail es `benko@ladybenko.net`.

[9] http://www.google.com

Parte I

XHTML

Capítulo 2

Estructura XHTML

> La ciencia es aquello que entendemos lo suficiente como para explicárselo a un ordenador. El arte es todo lo demás.
>
> Prólogo de A=B
> Donald Knuth, científico de la computación

Este capítulo es tremendamente aburrido, pero muy importante. No es muy largo, así que presta atención. Ahora aprenderemos a formar el esqueleto de nuestros archivos para poder usarlo más adelante a modo de plantilla.

2.1. La codificación

La primera línea que debemos tener en un documento XHTMl es la que marca la **codificación**. ¿Qué es esto? Simplemente el formato en el que se guardan los caracteres en el archivo. La codificación estándar es la **Unicode** (`utf-8`)[1] y soporta caracteres de todas las lenguas: occidentales, griegos, chinos, árabes, japoneses, coreanos...)

Asegúrate de que el editor de textos que uses te guarda el archivo en formato Unicode (normalmente es una opción a elegir en el cuadro de diálogo de *Guardar como*).

[1] Existen multitud de codificaciones, en su mayoría Criaturas del Mal. Deberías usar siempre la Unicode. ¿La razón? Échale un vistazo al apéndice C.

Al grano, hay que escribir esto[2]:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Esa etiqueta es de XML, por eso tiene esa forma tan "rarita". No volveremos a usar ese tipo de etiquetas en nuestro documento.

La línea anterior tiene que aparecer al principio de todo el documento, antes que ninguna otra. Lo que pasa es que **da problemas si trabajamos con PHP**[3]. Si usas PHP, la etiqueta que marca el inicio del script puede ser `<?`, así que los navegadores se hacen un lío. ¿La solución? Podemos **omitir** esa línea del principio y **declarar la codificación dentro de la sección** `head` (ahora veremos qué es eso). Si elegimos esto último, la línea a incluir dentro del `head` sería:

```
<meta http-equiv="Content-Type"
content="text/html; charset=utf-8" />
```

Escoge la opción que quieras, pero sólo una. Por cierto, por motivos de espacio, la línea aparece cortada. No importa, porque el navegador interpreta los saltos de línea como si fueran espacios en blanco[4]. En realidad podríamos escribir todo el archivo XHTML en una sola línea. O cada palabra en una línea diferente.

2.2. El DOCTYPE

Lo siguiente que toca es indicar el **DOCTYPE**. Se encarga de decirle al navegador qué demonios contiene el archivo que está abriendo. Nosotros usaremos la especificación **XHTML 1.0 Strict**, que es esta:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3c.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Después del DOCTYPE tenemos a la cabecera y al cuerpo encerrado entre las etiquetas `<html>` y `</html>`.

---

[2] No somos masocas, así que no nos lo tenemos que aprender de memoria. Simplemente copia y pega.

[3] ¿No sabes lo que es PHP? No pasa nada, hay otras formas de alcanzar la felicidad. Las galletas de dinosaurios, por ejemplo.

[4] Y por cierto, muchos espacios en blanco seguidos se interpretan como si sólo hubiera uno.

2.3. El elemento raíz (HTML)

El resto de nuestro documento tiene que ir dentro de la etiqueta `<html>`. Pero en esa etiqueta debemos indicar una serie de cosas, como que nuestro documento es XHTML y qué lengua estamos usando. Si escribimos en castellano, nos quedaría así:

```
<html xmlns="http://www.w3.org/1999/xhtml\"
xml:lang="es" lang="es">
```

Las letras `es` son el código de la lengua castellana. Podéis ver algunos otros códigos[5] en el cuadro 2.1.

Lengua	Código
castellano	es
catalán	ca
gallego	gl
euskera	eu
inglés	en
francés	fr
alemán	de
japonés	ja

Cuadro 2.1: Códigos de lenguas

2.4. La cabecera (HEAD)

La **cabecera** contiene información que no forma parte del contenido de la página en sí, como el título, vínculos a hojas de estilos CSS, información para robots de búsqueda, scripts, etc. Por ahora nos quedaremos sólo con el **título** de la página. La cabecera va encerrada entre `<head>` y `</head>`, mientras que para el título usamos la etiqueta `<title>`. El título de la web aparece en la barra de título de la ventana del navegador, es el nombre por defecto que aparece si añadimos a favoritos la página, etc.

Quedaría así:

[5]Los *trekkies* pueden escribir en su lengua con el código `x-klingon`. ¡Exigimos *quenya* ya!

```
<head>
  <title>Título de la web</title>
</head>
```

El sangrado no es obligatorio, pero sí que viene muy bien para aclararnos con el código

2.5. El cuerpo (BODY)

Por último, nos encontramos con el cuerpo, encerrado entre `<body>` y `</body>`, y que contiene toda la "chicha". Quedaría tal que así:

```
<body>
Aquí va el cuerpo de la web
</body>
```

2.6. La plantilla

Recopilando todo, nos quedaría algo como esto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3c.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="es" lang="es">

<head>
  <meta http-equiv="Content-Type" content="text/html;
  charset=utf-8" />

  <title>Título de la web</title>
</head>

<body>
  Aquí va el contenido
</body>

</html>
```

Y está pidiendo a gritos que guardes una copia en un archivito llamado `plantilla.html` para usos posteriores.

Capítulo 3

Etiquetas básicas

¡Disidente!

La Vida de Brian
Frente Popular de Judea,
revolucionarios

Ahora que conocemos la estructura de un documento XHTML, aprenderemos las **etiquetas básicas** que nos permitan empezar a crear el **contenido de nuestra página web**: párrafos, saltos de línea, títulos, etc.

3.1. Párrafos

Los **párrafos** sirven para **estructurar el contenido**. En la web funcionan igual que en el Mundo Real™: contienen una o más frases relacionadas entre sí. Para crear un párrafo, metemos texto entre las etiquetas `<p>` y `</p>`. Un ejemplo[1]:

```
<p>
Hola, me llamo Luke Skywalker y soy piloto
de una X-Wing en el Rogue Squadron. También
soy un Jedi del Lado Luminoso de la Fuerza.
Mis maestros han sido Yoda y Obi-Wan Kenobi.
</p>
```

[1] Da igual insertar un salto de línea entre la etiqueta y el contenido, ya que será interpretado como un espacio en blanco.

Si pruebas ese ejemplo, te darás cuenta de que todo forma un párrafo y que el navegador pasa de los saltos de línea. Esto es correcto. Podrías haber puesto todo el párrafo en la misma línea y obtendrías el mismo resultado.

3.2. Saltos de línea

Hay veces que necesitamos forzar un **salto de línea** dentro de un párrafo. Para ello, usamos la etiqueta `<br />`, así:

```
<p>
Dark Chest of Wonders<br />
Seen trought the eyes<br />
Of the one with pure heart<br />
Once, so long ago.
</p>
```

Chulísima canción de Nightwish[2], por cierto.

Aunque estéticamente podamos obtener el mismo resultado mediante párrafos (con `<p>`) que saltos de línea de forma indiscriminada, hay que tener en cuenta que usar `<br />` para separar párrafos es algo malvado, propio de una mente retorcida.

3.3. Los títulos (headings)

Los **títulos** nos sirven para agrupar información. Imaginemos la sección de enlaces de nuestra página. El título principal podría ser *Mis links favoritos*. Luego tendríamos los links clasificados por secciones, cada una de ellas bajo un subtítulo diferente: *Blogs*, *Descargas* y *Videojuegos*. Incluso podríamos tener subcategorías dentro de una misma sección, como por ejemplo *RPG's*, *Aventuras Gráficas* y *Arcades*.

Para conseguir esto, tenemos las etiquetas `<hx>` y `</hx>`, donde `x` es un número del 1 al 6. `<h1>` corresponde al título más importante y sólo debería haber uno por página. Veamos todo lo anterior en un ejemplo:

```
<h1>Mis links favoritos</h1>

<h2>Blogs</h2>
```

[2] *Dark Chest of Wonders*, de Nightwish, en el álbum *Once*.

```
<!-- bla bla bla -->

<h2>Videojuegos</h2>
<h3>RPG's</h3>
<!-- bla bla bla -->
<h3>Arcades</h3>
<!-- bla bla bla -->
```

Si ves el código anterior en un navegador, aparecerán los títulos más importantes de mayor tamaño que los menos importantes (pero el tamaño de cada título siempre puede cambiarse con CSS).

3.4. Citas

Existen tres etiquetas para poder escribir **citas**: `<blockquote>`, `<q>` y `<cite>`. Mientras que `<blockquote>` y `<q>` se usan para escribir la cita en sí (las frases), `<cite>` se emplea para marcar el origen (persona, libro, canción o lo que sea).

Entonces, ¿en qué se diferencian `<blockquote>` y `<q>`? Pues que `<blockquote>` está hecha para contener citas largas. Es decir, que dentro de un `<blockquote>` tendremos que poner **párrafos**. `<q>` funciona al revés, puesto que está pensada para escribir citas cortas. `<q>` va **dentro** de párrafos. La razón técnica es que `<blockquote>` es un elemento de bloque (**block**), mientras que `<q>` es **inline**, y un inline no puede estar "aislado".

Todo lo anterior puede resultar confuso, así que ahora toca un ejemplo clarificador:

```
<p>Aquí os dejo un fragmento de la canción
<cite>Die for Rock 'n' Roll</cite>, de Dover:</p>

<blockquote>
<p>
Everybody danced (while)<br />
I was lying on the floor<br />
I was ready to die<br />
for Rock 'n' Roll<br />
</p>
</blockquote>

<p>Mi parte preferida es cuando dice lo de
<q>I was ready to die[...]</q>.</p>
```

3.5. Separadores horizontales

Los **separadores horizontales** (horizontal rules) han caído en desuso, ya que podemos conseguir bordes delimitadores muy cucos mediante estilos CSS. Pero como el saber no ocupa lugar, nos quedaremos con la etiqueta `<hr />`:

```
<h2>Los videojuegos</h2>
<p>Bla bla bla...</p>

<hr />

<h2>Música</h2>
<p>Bla bla bla...</p>
```

3.6. Comentarios

Los **comentarios** son notitas que ponemos en el código fuente de una página, pero que no se muestran en pantalla. Para el navegador, son **invisibles**. Son útiles para indicar qué hacen ciertas partes del código. Para insertar un comentario, lo escribimos entre `<!--` y `-->`. Ten en cuenta que el comentario tiene que estar en una sola línea.

```
<!-- Esto es un comentario -->
```

Capítulo 4

Enlaces

Hablar es fácil. Enséñame el código.

Lista de correo del kernel de Linux
Linus Torvalds, ingeniero de software

Ya sabemos las etiquetas necesarias para escribir texto. Ahora toca aprender a usar una de las características más importantes de la web: los **enlaces** (o links). Se implementan con la etiqueta `<a>`.

4.1. Enlace a una página externa

Si queremos enlazar a una página o archivo que está en **otro servidor** (normalmente webs que no son nuestras), usamos `<a>` de esta forma:

```
<a href="http://www.algo.com" title="Descripción">
Texto del enlace
</a>
```

El atributo `href` contiene la **URL** a la que queremos enlazar. Es muy importante que no se nos olvide el protocolo[1] (en este caso *http://*) o no funcionará.

En `title` escribimos una **descripción** de la página que enlazamos. Al igual que con `<acronym>` y `<abbr>`, en la mayoría de navegadores este título aparecerá al pasar el ratón por encima del

[1] *http://* es el protocolo para la web, *ftp://* para los FTP, etc.

link. No hay que confundir el título con el texto del enlace. Son completamente independientes.

Imagina que quieres enlazar a Barrapunto[2]:

```
<a href="http://www.barrapunto.com" title="La
información que te interesa">Barrapunto</a>
```

4.2. Enlace a una página local

Para enlazar a una página que esté en **nuestro servidor**, necesitamos saber la **ruta** (*path*) desde donde estemos hasta la ubicación del archivo.

Si el origen (página donde está el link) está **en el mismo directorio** que el destino (página a la que apunta el link), entonces sólo tenemos que escribir su nombre:

```
<a href="profile.html" title="Información
sobre mí">Ficha personal</a>
```

Si el destino estuviera en **un subdirectorio**, utilizaríamos una barra `/` para indicar el camino[3]:

```
<a href="galeria/color.html" title="Galería
color">Ver dibujos a color</a>
```

Si el destino estuviera **un directorio por encima**, pondríamos dos puntitos y una barra `../` de esta manera:

```
<a href="../index.html" title="Página principal">
Volver al inicio</a>
```

4.3. Enlace a una dirección de e-mail

Podemos crear un enlace que, al pulsar sobre él, se abra automáticamente una ventana del cliente de correo que tenga el usuario[4] para que escriba un mensaje a esa dirección.

---

[2] Blog colectivo geek, similar a Slashdot. Ya no es lo que era, pero sigue usando Debian.

[3] ¡Atención usuarios del Maligno! En Windows se utilizan las barras invertidas \ para separar los directorios. En el resto del universo, se utilizan las barras normales.

[4] Normalmente el infame Outlook. Usa Thunderbird (http://www.mozilla.org), que es mejor y libre.

Para ello, sólo tenemos que usar *mailto:* en el atributo `href`, seguido de una dirección de correo electrónico:

```
<a href="mailto:leia@alianza.net"
title="E-mail de la Princesa Leia">Leia</a>
```

4.4. Anclas

Las **anclas** nos permiten enlazar a una posición concreta de la página, en plan teletransporte. Funcionan así:

Primero **creamos el ancla** en el sitio al que queremos enlazar después. Para esto, usamos el atributo `id` (podemos ponerlo prácticamente en cualquier etiqueta). Por ejemplo, así:

```
<h3 id="comentarios">Lista de comentarios</h3>
```

Ahora **creamos un link** que nos transporte a ese ancla. Ponemos en `href` la ID de antes precedida de una almohadilla `#`:

```
<a href="#comentarios" title="Lista de
comentarios"> Leer comentarios</a>
```

También podemos enlazar a un ancla que esté en **otra página**:

```
<a href="post005.html#comentarios" title="Lista de
comentarios">Comentarios del post nº 5</a>
```

¡Así de fácil! Hay otro modo de crear anclas, y es usando el atributo `name` de una etiqueta `a` sin atributo `href` y sin texto para el enlace; pero esta manera es obsoleta. De todos modos, revisando códigos de otras páginas te las puedes encontrar, así que no está de más conocer cómo se hace. Tienen este aspecto (y ojo, ¡son invisibles!):

```
<a name="comentarios" />
```

Y ya está todo lo que hay que saber sobre anclas. Son útiles para páginas muy largas, como las FAQ's[5]: tienen un índice de preguntas que enlazan al ancla de la respuesta correspondiente. Así, si hacemos clic en la pregunta número 3, nos enlazará a ese

[5]FAQ significa *Frequently Asked Questions* y es una recopilación de las preguntas más frecuentes sobre un tema, contestadas. Está muy mal no leerse las FAQ's y luego preguntar sobre cosas que ya están respondidas, así que ya sabes qué hacer si no quieres que Matrix te castigue.

punto concreto de la página. Normalmente las respuestas tienen otro enlace que nos devuelve a un ancla situada en el índice de preguntas, para facilitar la navegación.

Capítulo 5

Listas

> All your base are belong to us.
>
> ZERO WING, videojuego arcade

Ahora veremos cómo implementar **listas** en nuestras páginas. Las hay de tres tipos: **ordenadas**, **sin ordenar**, y de **definición**. Venga, que esto es muy facilito.

5.1. Listas ordenadas

Las **listas ordenadas** muestran sus elementos numerados. Se crean con la etiqueta `<ol>`:

```
<p>Mis escritores favoritos
(en orden de preferencia):</p>

<ol>
  <li>R. A. Salvatore</li>
  <li>George R. R. Martin</li>
  <li>Isabel Allende</li>
</ol>
```

Hay que tener en cuenta que con CSS podemos cambiar el número por el que queremos empezar a contar, así como el tipo de numeración (números arábigos, romanos, letras del abecedario, mayúsculas...).

5.2. Listas sin ordenar

Las **listas sin ordenar** marcan cada elemento con una viñeta, de modo que no se sigue una numeración. Se crean con la etiqueta `<ul>`.

```
<p>El helado perfecto (¡ñam!):</p>

<ul>
  <li>1 bola de helado de chocolate</li>
  <li>1 bola de helado de limón</li>
  <li>Trocitos de piña y melocotón en almíbar</li>
  <li>Sirope de chocolate</li>
</ul>
```

Al igual que con las listas ordenadas, podemos modificar su apariencia con CSS y elegir el tipo de viñetas que queremos[1].

Por cierto, prueba el helado, está riquísimo[2].

5.3. Listas de definición

Las **listas de definición** se diferencian de las anteriores en que cada ítem está compuesto por un par de elementos: un **término** y su **definición**. Se usan estas etiquetas: `<dl>` para crear la lista, `<dt>` para cáda término y `<dd>` para las definiciones.

```
<p>Significado de algunos smileys:</p>

<dl>
  <dt>:)</dt>
  <dd>Sonrisa</dd>

  <dt>xD</dt>
  <dd>Carcajada</dd>

  <dt>:P</dt>
  <dd>Sacar la lengua</dd>
</dl>
```

Y sí, también se puede cambiar el aspecto y bla, bla, bla. Comentar que las palabras `término` y `definición` no sólo se refie-

[1] Es más, podemos sustituir las viñetas por imágenes.

[2] También puedes añadir guindas y nata.

ren a una palabra y a su significado. También podemos usar una lista de definición para crear un perfil (por ejemplo), relacionando los pares *Nombre-Leia*, *Ciudad-Coruscant* y *Profesión-Senadora*.

5.4. Listas anidadas

No, **las listas anidadas** no son un nuevo tipo de lista, sólo una combinación de las anteriores. *Anidar* significa meter una lista dentro de otra. Por ejemplo:

```
<p>Algunos libros de Salvatore:</p>

<ul>
  <li>I Trilogía de El Elfo Oscuro
    <ol>
      <li>La Morada</li>
      <li>El Exilio</li>
      <li>El Refugio</li>
    </ol>
  </li>
  <li>Trilogía de El Valle del Viento Helado
    <ol>
      <li>La Piedra de Cristal</li>
      <li>Ríos de Plata</li>
      <li>La Gema del Halfling</li>
    </ol>
  </li>
</ul>
```

No es difícil, sólo debemos tener cuidado cerrando la etiqueta que toque. ¿Cómo lo sabemos? Fácil: **primero se cierran las interiores**, y después las de fuera. Es por esto, que cuando insertamos una lista dentro de un ítem de otra lista (esto es, primero `<li>` y después `<ol>`, después debemos cerrar primero la lista, y luego el ítem de la lista "exterior". Un buen sangrado como el del ejemplo ayuda mucho.

Capítulo 6

Imágenes

> Nada en la vida existe para que lo temamos, simplemente para que lo entendamos.
>
> Marie Curie, científica

Las imágenes son un elemento importante a la hora de hacer más atractiva una web, o de aportar más información. No obstante, hay que saber cuándo utilizarlas y no abusar de ellas.

Podemos usar tres **formatos** de imagen: `GIF`, `JPEG` y `PNG`. El `JPEG` es el más adecuado para imágenes con muchos colores, como **fotografías**, y que no tengan grandes áreas de colores planos. Las imágenes `GIF` sólo pueden almacenar hasta 256 colores diferentes, pero uno de estos colores puede ser **transparente**. El formato `PNG` es el estándar y podemos elegir varias profundidades de paleta (número de colores). Además, les podemos añadir un **canal alpha** para crear efectos con transparencias de diferente opacidad. Pero Ya-Sabéis-Cuál navegador en su versión 6 y anteriores no implementa correctamente los `PNG`, así que hay que llevar cuidado.

6.1. Insertar una imagen

Para poner una imagen, hacemos uso de la etiqueta `<img>` con unos cuantos atributos:

```
<img src="image.gif" width="ancho"
height="alto" alt="descripción" />
```

Con `src` establecemos qué imagen queremos mostrar. Al igual que con los links, hay que tener en cuenta la **ruta** hacia la imagen. Por motivos de organización, normalmente tendremos las imágenes en un subdirectorio (o varios) llamado `images`, así que escribiríamos `src="images/algo.gif"`.

Los atributos `width` y `height` nos permiten establecer el **ancho** y el **alto** de la imagen. Podemos indicar un valor absoluto en píxeles o uno relativo en tanto por ciento. Por ejemplo, `width="200"` mostraría la imagen con 200 píxeles de ancho, mientras que `width="100%"` hace que la imagen ocupe toda la pantalla de ancho. Estos dos atributos no son obligatorios, pero sí es conveniente que indiquemos las dimensiones en píxeles reales, ya que ahorramos tiempo al navegador a la hora de maquetar la página.

El atributo `alt` contiene una **descripción** de la imagen, que veremos cuando no se haya podido cargar por cualquier motivo. La mayoría de navegadores también muestran esta descripción al pasar el ratón por encima de la imagen. Este atributo es **obligatorio**, por cuestiones de acesibilidad: hay personas que deshabilitan las imágenes para ahorrar tiempo; otras, usan navegadores de texto como Lynx; y otras, sencillamente son ciegas[1].

Como ejemplo, así podríamos insertar la imagen de un *banner*:

```
<img src="images/banner.gif" width="200"
height="40" alt="BenKo's Art" />
```

6.2. Imágenes como links

También podemos hacer que una imagen sea a su vez un **enlace** a una página. Los navegadores suelen mostrarla con un reborde para indicarnos que se trata de un link, pero lo podemos cambiar con CSS.

Para poner una imagen como un link, la introducimos dentro de la etiqueta `<a>`:

```
<a href="http://art.ladybenko.net"
title="Mi portafolio">
<img src="images/banner.gif" width="200"
height="40" alt="BenKo's Art" />
```

[1] Una cita muy común entre desarrolladores web suele ser: "El visitante más importante de tu web es ciego y se llama Google".

```
</a>
```

Hay una técnica muy popular a la hora de implementar **galerías de imágenes** que consiste en usar *thumbnails*. ¿Qué es esto? Una *thumbnail* es una imagen más pequeña que la original, de modo que al hacer clic sobre ella, cargamos la imagen a tamaño completo. Entonces, algunos que reciben Iluminación Divina hacen:

```
<a href="matrix.jpg" title="Wallpaper">
<img src="matrix.jpg" width="100" height="50"
alt="Wallpaper de Trinity" />
</a>
```

Eso **está mal**. Si nuestro wallpaper de Trinity ocupa 100KB (o más), tendremos esos 100KB ¡como *thumbnail*! Es decir, justo lo que queremos evitar. El escalar una imaagen con `width` y `height` no hace que pese menos[2]. Tenemos que coger un programa de dibujo, escalar la imagen, y guardar esta copia más pequeña (de 5KB, por ejemplo):

```
<a href="matrix.jpg" title="Wallpaper">
<img src="matrix_thumb.jpg" width="100"
height="50" alt="Wallpaper de Trinity" />
</a>
```

6.3. Sobre el uso y abuso de imágenes

Dicen que una imagen vale más que mil palabras. Cierto, pero muchas imágenes, o pocas mal puestas, pueden llegar a desesperar.

¿Te resulta esto familiar? Entras a una web con un fondo de color chillón, letras verde fosforito, una cantidad obscena de GIF's animados, marquesinas, applets de Java, etc[3]. ¿Cuánto tardas en cerrar la página? Yo menos de un segundo, es ya un acto reflejo.

Es por esto que debemos limitar los GIF's animados, así como evitar el uso indiscriminado de imágenes. Recuerda: **sólo hay que poner las imágenes necesarias**. Además, te ahorrarás una pasta en ancho de banda de tu servidor.

[2]Firefox no es *tan* bueno.

[3]Aunque parezca mentira, esto estaba muy de moda en los tiempos de los módems de 56K baudios.

Capítulo 7

Tablas

¡¡¡Bienvenido a mi pesadilla!!!

Warcraft 2: Beyond the Dark Portal
Grom Hellscream, héroe orco

Las **tablas** son el mecanismo que nos proporciona XHTML para presentar **información tabulada**, como horarios o los resultados de la quiniela. Son un poco engorrosas de utilizar, pero a veces son necesarias, así que ¡allá vamos!

7.1. Una tabla básica

Las principales etiquetas que disponemos para crear una tabla son estas:

- `<table>`: crea la tabla
- `<caption>`: pone título a la tabla
- `<tr>`: crea una fila
- `<td>`: crea una celda
- `<th>`: crea una celda de encabezamiento

Se entiende mejor con un ejemplo. Es muy conveniente utilizar bien los sangrados, ya que hay que tener mucho cuidado al cerrar las etiquetas `<tr>`, `<td>` y `<th>`. Aquí tenemos una tabla de 2x2:

```
<table>
  <caption>Videojuegos</caption>

  <tr>
    <th>Título</th>
    <th>Género</th>
  </tr>
  <tr>
    <td>Sonic</td>
    <td>Plataformas</td>
  </tr>
</table>
```

Daría como resultado algo parecido a esto:

Título	Género
Sonic	Plataformas

El método es siempre el mismo. Por cada fila que queramos, abrimos una etiqueta `<tr>` e insertamos allí las celdas que correspondan. Dentro de cada celda podemos meter prácticamente **cualquier cosa**, pero debemos tener siempre en mente que el objetivo de las tablas es tabular información.

7.2. Atributos de table

La etiqueta `<table>` dispone de una serie de **atributos**[1] que nos permiten modificar su borde y los márgenes de las celdas.

Para cambiar el tamaño del **borde** de la tabla, usamos `border` con un valor en píxeles. Si no indicamos nada, los navegadores suelen tomar como valor por defecto 1 ó 0. Si no queremos ningún borde, debemos utilizar `border="0"`.

Si lo que queremos es cambiar la **distancia entre una celda** y otra, empleamos el atributo `cellspacing` con un valor en píxeles. Y para modificar la distancia del contenido de la celda a los bordes de esta, usamos `cellpadding`[2]. La diferencia entre `cellspacing` y `cellpadding` puede confundir al principio, así que lo mejor es verlo con un ejemplo[3], modificando la tabla anterior (pruébalo):

[1] Los viejos atributos que en HTML 4 nos permitían establecer dimensiones y colores han sido sustituidos por CSS.

[2] Aunque, en mi opinión, es mejor usar CSS para esto.

[3] Los valores están muy exagerados para que se note la diferencia.

```
<table border="1" cellpadding="10"
cellspacing="30">
  <caption>Videojuegos</caption>

  <tr>
    <th>Título</th>
    <th>Género</th>
  </tr>
  <tr>
    <td>Sonic</td>
    <td>Plataformas</td>
  </tr>
</table>
```

7.3. Expandir filas y columnas

Muchas veces necesitamos que una celda ocupe más de un espacio. Pongamos como ejemplo nuestra socorrida tabla de videojuegos. ¿Qué pasa si en vez de un género por cada juego, queremos meter dos? Podemos hacer que la celda de género ocupe dos columnas. Así:

```
<table>
  <caption>Videojuegos</caption>

  <tr>
    <th>Título</th>
    <th colspan="2">Género</th>
  </tr>
  <tr>
    <td>Sim City</td>
    <td>Simulación</td>
    <td>Estrategia</td>
  </tr>
</table>
```

Título	Género	
Sim City	Simulación	Estrategia

Como ves, para expandir una celda por varias columnas hemos utilizado el atributo `colspan`. Podemos hacer lo mismo con las filas, mediante `rowspan`. Vamos a hacer la misma tabla de antes, pero girada 90 grados:

```
<table>
  <caption>Videojuegos</caption>

  <tr>
    <th>Título</th>
    <td>Sim City</td>
  </tr>
  <tr>
    <th rowspan="2">Géneros</th>
    <td>Simulación</td>
  </tr>
  <tr>
    <td>Estrategia</td>
  </tr>
</table>
```

Título	Sim City
Géneros	Simulación
	Estrategia

¿Que es un rollo esto de las tablas? Pues sí, pero no le des mucha importancia. La mayoría de editores de código (X)HTML, como el Quanta o el Homesite+, traen asistentes para crear tablas de forma rápida y sencilla.

7.4. ¿Tablas para layouts? ¡Insensato!

A día de hoy, la mayoría de las páginas web están maquetadas usando tablas con `border="0"`. Antes de la llegada del CSS, era totalmente imposible crear texto a columnas y, en definitiva, maquetas un sitio web[4]. Afortunadamente, CSS implementa **capas**, con lo que se puede configurar totalmente la apariencia y la colocación de cada elemento de la página mediante la hoja de estilos, quedando el código XHTML muy simple y sencillo.

¿Entonces por qué la gente sigue usando tablas? Porque muchos piensan que eso de las capas es algo muy complicado. ¡Mentira y gorda! Lo que pasa es que nadie nace enseñado, y aprender algo nuevo siempre da un poco de pereza. En la parte de CSS de este

[4]Bueno, se podía apañar algo utilizando frames, pero era peor el remedio que la enfermedad.

tutorial aprenderemos a crear varios tipos de diseño populares para así tumbar el mito de que las capas son difíciles.

Y dándole la vuelta a la tortilla... **¿Por qué no usar tablas?** Pues porque las tablas no se han creado para maquetar y el WWW Consortium lo desaconseja. Además, en navegadores no visuales (de texto, para ciegos, o cualquier dispositivo que no sea un monitor) el resultado es totalmente **imprevisible**. ¿Quieres otra razón de peso? Las tablas dan más trabajo. El código queda más enrevesado, y si queremos renovar el diseño de nuestra página, **hemos de cambiar prácticamente todo el código**. Sin embargo, si hubiéramos usado un layout por capas, sólo tendríamos que modificar el archivo CSS. ¿Más razones? Al ser el código más complicado, las páginas son **más pesadas** y gastan más ancho de banda. Además, **tardan más en cargar**, ya que hasta que no se carga todo lo que hay en la tabla, no se muestran los resultados parciales.

El futuro es XHTML y sitios web importantes, como Blogger[5], ya han rediseñado su layout con capas. Así que te animo a que cumplas los estándares y no uses las tablas para maquetar[6] :)

[5] Comunidad de blogs de Google: http://www.blogger.com

[6] En la web de *Steal these buttons!* (http://gtmcknight.com/buttons/) hay botoncitos muy cucos para mostrar al mundo que formas parte del Clan de Paladines del CSS.

Capítulo 8

Formularios

> Mi fortaleza reside únicamente en mi tenacidad.
>
> Louis Pasteur, microbiólogo y químico

Los **formularios** nos permiten recoger **información introducida** por el visitante de nuestra web. Esta información podemos enviarla por correo electrónico o procesarla con un script.

Si se manda por **correo electrónico**, es muy importante tener en cuenta de que se trata de **información no cifrada** y que podría ser interceptada, así que no debe contener datos importantes. Un uso aceptable sería un formulario con comentarios sobre nuestra página o para pedir un intercambio de links.

Si nuestro servidor dispone de tecnología como **PHP** o **CGI** (por ejemplo), podemos hacer más cosas con esa información, como guardarla en una base de datos o generar una página dinámicamente.

8.1. La etiqueta FORM

Todos los formularios están encerrados por `<form>` y `</form>`. Dentro de esta etiqueta, van los campos del formulario, y podemos usar párrafos y saltos de línea para organizarlos. Vamos a ver un ejemplo de etiqueta `<form>`, suponiendo que va a ser un formulario que se enviará por correo electrónico:

```
<form action="mailto:leia@alianza.net"
method="post" enctype="text/plain">
```

El atributo `action` recoge qué se va a hacer una vez que se pulse el botón de enviar (`Submit`). En ese ejemplo, el formulario se envía a la dirección `leia@alianza.net`[1]. Si tuviésemos un script para procesar el formulario, hubiésemos puesto algo como `action=.enviar_info.php"`.

Con `method` especificamos cómo va a ser enviada la información. Si utilizamos correo electrónico, le damos el valor de `post`. Para scripts se suele utilizar `get`, que pone el valor de las variables en la *query string* (URL)[2].

Por último, con `enctype="text/plain"` conseguimos que llegue a nuestra bandeja de entrada el formulario en forma de texto plano sin caracteres extraños.

8.2. Campos de texto

La mayoría de los campos de un formulario se crea con una sola etiqueta, `<input>`, y mediante su parÃ¡metro `type` especificamos el tipo de campo que queremos. Un **campo de texto** básico quedaría así:

```
<input type="text" id="nombre" name="nombre"
size="20" />
```

Vamos a pegarle un vistazo a los atributos. Con `type="text"` indicamos que se trata de un campo de texto. El atributo `size` recoge el **ancho** del campo, medido en caracteres. Ahora bien, ¿qué hacen `id` y `name`?

Pues `id` es un **identificador**. Esto implica que **nada** en todo el documento puede llamarse igual que este elemento. Este parámetro sirve para CSS[3] y para más cosillas, como usarlo con la etiqueta `<label>` (que veremos ahora).

---

[1] Lo siento chicos, pero la princesa Leia no me ha autorizado a publicar aquí su verdadera dirección de e-mail :(

[2] Si esto te suena a chino, no te preocupes. Sólo es necesario saberlo en el caso de que vayas a usar un script... Y en ese caso, ya lo aprenderás cuando te enseñen a programarlos.

[3] Estableciendo un aspecto exclusivo para ese elemento, por ejemplo.

Con `name` damos **nombre a la variable** de ese campo. Por ejemplo, si el visitante escribe *"Morpheo"* en el formulario que hemos puesto de ejemplo, recibiríamos un e-mail que contendría algo así:

```
nombre = Morpheo
```

De todos modos, para ahorrarnos problemas, siempre que podamos es mejor escribir el mismo valor para `id` y `name`. Hay que complicarse la vida lo menos posible.

Hay otros atributos adicionales para nuestros campos de texto. Podemos indicar el `número máximo de caracteres` que puede introducir el usuario con `maxsize` Y si queremos que aparezca un `valor por defecto`, podemos usar `value=.ªlgo"`. Por ejemplo, para pedir la dirección de la página web del visitante:

```
<input type="text" name="url" id="url" size="30"
maxsize="255" value="http://" />
```

Y, por supuesto, no debemos olvidarnos de nuestro socorrido `title`, que funciona igual que con la etiqueta `<a>`.

8.3. Campos de contraseña

Los `campos de contraseña` son exactamente iguales que los de texto, sólo que el visitante en lugar de ver los caracteres que ha introducido, ve asteriscos. Lo de "exactamente iguales" quiere decir eso: es un campo de texto, la información **no va cifrada** de ninguna manera. La diferencia entre un campo de texto y uno de contraseña es meramente estética.

Los atributos son los mismos que los de los campos de texto, lo único que cambia es qe debemos introducir `type="password"`. Ejemplillo:

```
<input type="password" name="pass" id="pass" />
```

8.4. Etiquetar campos

Ya hemos aprendido a crear campos de texto para nuestro formulario, ¿pero cómo le decimos al visitante qué es lo que tiene que introducir en cada campo? Podríamos hacer algo así:

```
<p>Nombre:
<br />
<input type="text" name="nombre" id="nombre" />
</p>
```

Mal. Podemos tener problemas con navegadores no visuales. Entonces, ¿cómo sabemos que la palabra *"Nombre"* hace referencia al campo con el atributo `id="nombre"`? Para eso disponemos de una etiqueta nueva: `<label>`.

Esta etiqueta se encarga de **asociar texto con su campo** correspondiente. Sólo tiene un atributo, `for`, y en él tenemos que indicar la `id` del campo al que queremos hacer referencia. El ejemplo anterior sería correcto de esta manera:

```
<p>
<label for="nombre">Nombre:</label>
<br />
<input type="text" name="nombre" id="nombre" />
</p>
```

8.5. Áreas de texto

Con las **áreas de texto** damos a nuestros visitantes la posibilidad de introducir texto con varias líneas. La etiqueta a usar es `<textarea>`, y su funcionamiento es un poco diferente al de `<input>`.

La etiqueta `<textarea>` dispone de los atributos `id`, `name` y `title`, que funcionan como en el resto de campos de formulario. Además, disponemos de otros dos para indicar las dimensiones del área de texto: `cols` se encarga de establecer el **ancho** (medido en caracteres) y `rows` el **alto**, medido en líneas.

Como ejemplo, imaginemos que en una parte del formulario queremos poner un campo para que el visitante deje un comentario. Como probablemente sea largo, utilizamos un textarea:

```
<p>
<label for="comentario">¿Algún comentario?</label>
<br />
<textarea name="comentario" id="comentario"
cols="30" rows="5">
Bla bla bla
</textarea>
```

Fíjate en que `<textarea>` tiene etiqueta de **cierre**. El texto que hay entre la etiqueta de apertura y la de cierre es el **valor por defecto** que contendrá el campo; en este caso *"Bla bla bla"*.

8.6. Casillas de verificación

Una **casilla de verificación** (más conocida como *checkbox* es un cuadradito que el usuario puede activar y desactivar pulsando en él. Se crean con la etiqueta `<input>` y `type=çheckbox"`. Los atributos *id*, *name* y *title* funcionan con normalidad, pero *value* funciona de manera algo distinta. Lo que escribamos en `value` es lo que nos saldrá en el e-mail que recibamos como el valor de la variable (indicada en `name`) si la casilla está activada. Es decir, que si ponemos esto...

```
<p>Has jugado a...<br />
<input type="checkbox" name="monkey1" id="monkey1"
value="si" />
<label for="monkey1">Monkey Island I</label>
</p>
```

...y el usuario activa la casilla, recibiremos un mail como este:

```
monkey1=si
```

También podemos hacer que una casilla esté activada por defecto, si añadimos el atributo `checked=çhecked"`. Así:

```
<p>Has jugado a...<br />

<input type="checkbox" name="monkey1" id="monkey1"
value="si" checked="checked" />
<label for="monkey1">Monkey Island I</label>

<input type="checkbox" name="xwing" id="xwing"
value="si" />
<label for="xwing">X-Wing Alliance</label>
</p>
```

Nos quedaría entonces la casilla etiquetada como *"Monkey Island"*[4], mientras que la de *"X-Wing Alliance"* sin activar.

[4] Yo soy cola, tú pegamento.

8.7. Botones de selección

No sé si el término "botones de selección" es en realidad el más adecuado como traducción, así que mejor pondré una explicación de lo que hacen. El nombre en inglés es ***radio buttons***, y son casillas circulares agrupadas, en las que sólo una puede estar activada a la vez. Sirven para cuando queremos que el visitante sólo seleccione una opción de las múltiples que se le dan.

Aunque se crean con la etiqueta `<input>` indicando el parámetro `type="radio"`, los radio buttons son algo "especiales", así que los veremos un poco más despacio.

Pongamos el caso de que queremos que el visitante de nuestra web nos indique cuál película de la trilogía de Star Wars[5] es su favorita. Evidentemente, sólo puede coger una, así que nos toca emplear radio buttons en lugar de casillas de verificación. Necesitamos entonces un botón por cada película (tres en total). ¿Cómo los agrupamos? Pues dando el mismo nombre de variable a cada botón. Es decir, **el atributo `name` es el mismo para todo el grupo**. ¿Y qué hacemos con `id`? Bien, no puede haber dos valores de `id` repetidos, así que la *id* en cada botón ha de ser distinta. En otras palabras, los radio buttons son los únicos campos en los que la *id* y `name` deben ser diferentes. Disponemos además del atributo `checked` por si queremos marcar alguna opción por defecto. El código para este ejemplo sería así:

```
<p>Peli preferida:</br>

<input type="radio" name="peli" id="sw_hope"
value="hope" checked="checked" />
<label for="sw_hope">A New Hope</label>

<input type="radio" name="peli" id="sw_empire"
value="empire" />
<label for="sw_empire">The Empire Strikes
Back</label>

<input type="radio" name="peli" id="sw_jedi"
value="jedi" />
<label for="sw_jedi">
The Return of the Jedi</label>
```

[5] De la trilogía original, por supuesto. No queremos ofender a nadie.

```
</p>
```

Como ves, usamos como nombre de variable (`name`) la palabra *"peli"*. Según la película que sea, cada botón tiene asignada una *id* diferente. Por ejemplo, *A New Hope* tiene asignada la *id "peli_hope"*. El atributo `value` contiene el texto que tendrá la variable en caso de que se seleccione ese botón. Por ejemplo, si el visitante selecciona la película de *The Return of the Jedi*, recibiremos un e-mail con esta línea:

```
peli=jedi
```

Quizás parezca un poco lioso, pero una vez hayas hecho un par de formularios, todo irá bien.

8.8. Listas de selección

Las **listas de selección** tienen una función parecida a los radio buttons, en tanto que se nos presentan **múltiples opciones** agrupadas en las que escogemos una[6]. La diferencia es que la lista aparece replegada y no ocupa apenas espacio en la web, así que son muy útiles cuando tenemos muchas opciones a elegir.

La etiqueta que las crea es `<select>` y tiene etiqueta de cierre. Entre ellas, insertamos las opciones que tenemos con la etiqueta `<option>`. Pongamos el mismo ejemplo de antes, el de elegir la película preferida de *Star Wars*.

```
<p>
<label for="peli">Película preferida:</label>
<br />

<select name="peli" id="peli">
  <option value="hope">A New Hope</option>

  <option value="empire">The Empire
  Strikes Back</option>

  <option value="jedi" selected="selected">
  The Return of the Jedi</option>
```

[6]Mentira, hay un atributo que permite que seleccionemos varias opciones de la lista, pulsando la tecla `Ctrl`. No seas un pequeño bastardo y lo pongas, que es muy incómodo.

```
</select>
</p>
```

Con *selected="selected"* indicamos cuál es la **opción por defecto**, en este caso *The Return of the Jedi*. Como ves, aquí no tenemos el lío aquel con `id` `name`.

8.9. Botones de enviar y reestablecer

Ya hemos visto todos los campos de formulario que podemos crear, ahora sólo nos falta comentar dos **botones** especiales: el de **enviar** (*submit*) y el de **reestablecer** (*reset*).

Ambos se crean con `<input>`. El atributo `id` no tiene mucho sentido, a menos que usemos CSS para cambiar su aspecto de un modo concreto y exclusivo. Asímismo, `name` tampoco nos será útil si no empleamos algún tipo de script para tratar la información. Como nosotros enviamos el formulario por correo, no debemos preocuparnos. En `value` indicaremos el texto que aparecerá en el botón.

El botón de **enviar** se encarga de mandar la información del formulario, según lo que hayamos especificado en `<form>`. Para crear el botón, simplemente indicamos `type="submit"`:

```
<input type="submit" value="Enviar" />
```

El botón de **reestablecer** borra el formulario y vuelve a poner los valores por defecto. Útil para formularios largos. Lo conseguimos con `type="reset"`:

```
<input type="reset" value="Borrar" />
```

Ni que decir tiene que hay que **diferenciar bien cuál botón es cuál** y no poner textos extraños como título de los botones. Si te sientes creativo, vete a jugar al Lego. Fastidia mucho que se te borre todo el formulario por error, creyendo haberle dado al botón de enviar.

Parte II

CSS

Capítulo 9

Introducción al CSS

> Fuera del mundo de los sueños, la vida
> puede ser dura, incluso cruel. Pero es vida.
>
> Final Fantasy X
> Auron, guardián legendario

En la parte de XHTML hemos comentado muchas veces que podemos cambiar el aspecto de una página web mediante **CSS** (*Cascading Style Sheets)*. Ahora es el momento de aprender cómo hacerlo.

Me ha costado decidirme sobre cómo explicar las diferentes propiedades del CSS, pero creo que la mejor manera de aprender sin que se haga muy pesado es poniendo ejemplos de problemas comunes con los que nos topamos, junto con la manera de solucionarlo en CSS. Después de todo, esto no pretende ser una guía de referencia con todas las propiedades y todos sus posibles valores[1].

9.1. ¿Dónde escribo el código CSS?

Hay dos opciones para insertar CSS en un documento XHTML. Lo más normal es hacerlo en un **archivo externo** (que se llama "hoja de estilos") y enlazarlo desde nuestro documento XHTML.

[1] Googleando un poco, podrás encontrar muchas. Y siempre te queda consultar las especificaciones oficiales en la web del W3C. Es un PDF muy majo de 359 páginas.

Esto tiene una ventaja enorme, y es que podemos tener muchas páginas enlazando a la misma hoja de estilos. Si más adelante queremos cambiar algo, sólo tenemos que modificar un único archivo (la hoja de estilos) y afectará a todas las páginas. Podemos enlazar una o más hojas poniendo esto dentro de la cabecera (`<head>`):

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

La otra opción es escribir la información referente a los estilos incrustada **en el mismo archivo XHTML**. Lo podemos hacer escribiendo entre las etiquetas `<style>` y `</style>`, que también deben ir en la cabecera.

Si hacemos las dos cosas a la vez, siempre tienen prioridad las reglas que estén dentro de `<style>`.

9.2. Vale, ¿pero cómo funciona?

En una hoja de estilos utilizamos **reglas** que consisten en elegir **selectores** a los que asignamos una serie de **propiedades**. Por ejemplo, si queremos que nuestra página web tenga el fondo blanco, las letras grises, y una fuente Arial de 10 puntos de tamaño, escribimos esta regla:

```
body {
  background-color: #fff;
  color: #666;
  font-family: Arial, sans-serif;
  font-size: 10pt;
}
```

Como ves, las líneas terminan en un **punto y coma**. Es muy importante que no se nos olvide. Ah, y si te lo estás preguntando, `#fff` representa al color blanco, y `#666` a un gris oscuro.

9.3. ¿Selectores?

Los **selectores** los utilizamos para elegir a qué elementos se aplican las propiedades que escribimos. Hay diferentes tipos de selectores, los más importantes son los que veremos ahora.

Si queremos elegir una **etiqueta**, simplemente escribimos su nombre. Por ejemplo, si queremos establecer las propiedades para los enlaces:

```
a {
  ...
}
```

También podemos elegir un **elemento único** utilizando su atributo `id`. Para ello, empleamos la almohadilla:

```
#menu {
  ...
}
```

Otra cosa que podemos hacer es definir una **clase** y hacer que muchos elementos la utilicen, escribiendo un punto antes del nombre. Por ejemplo, si queremos destacar algo:

```
.importante {
  ...
}
```

Luego podríamos emplear esa clase en los párrafos que queramos (o cualquier otro elemento), usando el atributo `class` de este modo:

```
<p class="importante">Bla bla bla</p>
```

También podemos seleccionar ciertos elementos, pero sólo cuando estén contenidos dentro de otros. Por ejemplo, si queremos seleccionar los `<li>`, pero sólo de las listas sin ordenar:

```
ul li {
  ...
}
```

9.4. ¿Qué significa eso de "cascading"?

Cascading significa cascada, y tiene que ver con la **herencia**. En CSS, los elementos hijos heredan todas las propiedades de sus padres. Por ejemplo, si establecemos una regla para el elemento `table`, sus hijos (`td` entre otros) también tendrán esas mismas reglas.

Es por esto que si, por ejemplo, queremos establecer un tipo de fuente para todo el documento, debemos indicarlo en el elemento `body` para que se propague por toda nuestra página.

9.5. ¿Puedo poner comentarios?

Claro, pero son distintos a los de XHTML, que van entre `<!-` y `->`.

```
/* Esto es un comentario */
```

9.6. ¿Por qué #fff significa blanco?

En CSS hay varias maneras de indicar un color. Podemos hacerlo en inglés, por ejemplo. En lugar de `#fff` escribimos `white` y lo solucionamos. El problema es que así es muy incómodo porque tenemos que aprendernos los nombres de cada color, y puede que exista algún color que queramos poner y que no tenga nombre... además de que supone una falta de precisión absoluta[2]. Así que normalmente se usa la **notación en hexadecimal**.

Los colores en nuestro monitor está formado por tres haces de luz: rojo, verde y azul. Se llama **sistema RGB** (*Red Green Blue*). Si cogemos un programa de dibujo, vemos que podemos eleigr un color indicando el valor de sus componentes rojo, verde y azul por separado. Este valor puede variar de `0` a `255` (se usan 8 bits para representar este rango). Por ejemplo, si queremos amarillo puro, ponemos `255` para el rojo, `255` para el verde y `0` para el azul.

¿Qué tiene que ver esto con el sistema hexadecimal? Pues que este rango de valores (`0..255`) puede ser representado con dos dígitos hexadecimales, que van desde el `00` hasta el `FF`. La forma de escribir un color completo es `#RRGGBB`, donde la almohadilla indica que se trata de un color hexadecimal, y `RR`, `GG` y `BB` son los dígitos correspondientes al rojo, verde y azul, respectivamente. Entonces el amarillo sería `#ffff00`.

La mayoría de los colores que usaremos tendrán por cada componente los mismos dígitos. Por ejemplo: `#ff0000` (rojo), `#0000aa`

[2] ¿Recuerdas esas discusiones en parvulario sobre si cierto color era lila, morado, o fucsia? ¿Quién te dice que tu concepto de morado sea el mismo que el de tu navegador?

(azul marino) o `#000000` (negro). Estos colores los podemos abreviar y dejarlos en tres cifras. Por ejemplo, `#f00` es completamente equivalente a `#ff0000`. Sin embargo, colores como `#a0a0a0` no pueden ser abreviados.

Por tanto, para conseguir el color blanco, debemos poner poner todas las componentes a `255`, que en hexadecimal nos quedaría `#ffffff`, y abreviado en `#fff`.

La mayoría de editores de código (X)HTML o de programas de dibujo nos mostrarán en la paleta el valor del color en hexadecimal. En cualquier caso, siempre podemos obtener el valor de sus componentes por separado y convertirlo a hexadecimal con una calculadora.

Capítulo 10

Trasteando por primera vez

> Eh, yo he visto esto antes. No... lo he jugado antes. No... ¡lo he diseñado antes!
>
> Post sobre "Piratas del Caribe"
> Ron Gilbert, desarrollador de videojuegos

Para empezar a meter mano al CSS, conviene conocer algunas propiedades sencillas. ¡Empezamos!

10.1. El color de primer plano

La propiedad `color` hace referencia al *foreground color*, es ecir, al **color** que está por encima del fondo. Hablando en plata, viene a ser el color del texto. Si queremos que nuestra página tenga las letras de gris oscuro, lo conseguimos con esto:

```
body { color: #666; }
```

Muy fácil, ¿no? Que pase el siguiente...

10.2. El fondo

Podemos moificar el fondo de un elemento con la propiedad `background`, que tiene la siguiente sintaxis:

```
background: color | image | repeat |
            attachment | position;
```

El primer parámetro corresponde al **color** de fondo, los siguientes son relativos a la **imagen** de fondo:

- **image**: aquí indicamos la ruta a la imagen que pondremos. Por ejemplo, `url(fondo.gif)`.
- **repeat**: con esto establecemos si queremos que la imagen se repita o no, tanto horizontal como verticalmente. Con `repeat` se repite siempre en ambos sentidos (valor por defecto), mientras que con `no-repeat` no se repite nunca. Con `repeat-x` se repite sólo en horizontal, y con `repeat-y` sólo en vertical.
- **attachment**: sirve para indicar si el fondo se queda fijo en el sitio o se desplaza con scroll. Es un poco difícil de explicar, así que lo mejor es que lo probéis vosotros mismos: con `scroll` (valor por defecto) el fondo se desplaza, y con `fixed` se queda siempre en el mismo sitio.
- **position**: indica la posición del fondo. Indicamos tanto la posicion desde la izquierda como desde arriba (ya sea en píxeles, porcentajes, o incluso palabras[1]). Los valores por defecto son `0% 0%`, que sitúan al fondo en la esquina superior izquierda. Si lo quisiéramos en las coordenadas 20,30 (tomando como el origen a la esquina superior izquierda), escribiríamos `20px 30px`. Si queremos el fondo centrado, pues `50% 50%`.

Podemos omitir alguna propiedad si queremos. Además, podemos establecer los valores de forma individual, usando las propiedades `background-color`, `background-repeat`, etc.

Veamos algunos ejemplos para poner fondo a nuestra página:

```
/* sólo color de fondo */
body { background: #fff; }

/* color de fondo e imagen en mosaico */
body { background: #fff url(fondo.gif); }

/* imagen fija, centrada y sin repetir */
/* a modo de marca de agua */
body {
```

[1] En este caso, si podemos usar palabras, es preferible que lo hagamos. Disponemos de `top` (arriba), `bottom` (abajo), `left` (izquierda) y `right` (derecha).

```
  background-color: #fff;
  background-image: url(fondo.gif);
  background-attachment: fixed;
  background-repeat: no-repeat;
  background-position: 50% 50%;
}
```

Hemos explicado de forma bastante detallada cómo funciona la propiedad `background`, pero no te malacostumbres. Ahora iremos mucho más deprisa, y seguramente te toque buscar más información por tu cuenta[2].

10.3. Fuente

Hay varias propiedades que nos permiten jugar con el aspecto del texto. Podemos englobar todas bajo `font`, pero primero veremos algunas subpropiedades.

Antes de nada, debemos considerar que **no todos los ordenadores tienen las mismas fuentes instaladas**. Es decir, que mi wonderfulosa fuente llamada `chachi.ttf` no tiene por qué tenerla mi vecino. Es más, lo normal es que no la tenga[3]. ¿Entonces qué hacemos? Usar sólo fuentes "estándar", que tengan la mayoría de ordenadores. Además, podemos especificar varias, de forma que si no se tiene la primera, se muestre la segunda, si no se tiene la segunda, pues la tercera, etc.

A lo que íbamos. La propiedad que sirve para cambiar la fuente es `font-family`. Como acabamos de ver, podemos indicar varias fuentes, por orden de preferencia. Si el nombre de una fuente tiene espacios en blanco, hay que ponerla entre comillas. Por ejemplo[4]:

```
font-family: "Comic Sans MS", Arial, sans-serif;
```

EL **tamaño** de la fuente lo controlamos con `font-size`. Podemos indicar medidas en píxeles (`px`) o en puntos (`pt`)[5].

[2] ¿Recuerdas ese librito majo de 359 páginas?

[3] La Ley de Murphy, como en cualquier otro aspecto de la vida, es aplicable al diseño web también.

[4] Haz esto y muere. La irrupción de la Comic Sans por doquier sólo es comparable a la aparición de Jar Jar Binks en Star Wars.

[5] Mentira cochina. Podemos (y mucha gente opina que *debemos*) usar medidas relativas, pero el tema de la herencia es muy importante en estos casos y... Dejémoslo en que no es *newbie-friendly*.

Para **alinear** el texto tenemos a `text-align`, que puede tomar los valores `right` (derehca), `left` (izquierda), `center` (centrado) o `justify` (justificado)[6]. Es muy normal tener una clase así:

```
\texttt{.centrar} { text-align: center; }
```

Si queremos `indentar` los párrafos, pues usamos `text-indent`:

```
p { text-indent: 2em; }
```

¿Qué significa `em`? Es una **unidad relativa**. Lo más fácil es trabajar con píxeles o porcentajes, pero en algunos casos conviene utilizar otras unidades. `1 em` equivale al tamaño de la fuente[7]. Así, si por herencia o por cualquier otra cosa el texto se muestra en un tamaño de fuente diferente, la proporción de indentado sería siempre la misma. Si pusiéramos `20px`, siempre sería esa, aunque el tamaño de la fuente fuera de `20` o de `32` píxeles.

También podemos cambiar la **decoración** del texto meidante `text-decoration`. Puede tomar diversos valores, como `none` (sin adornos), `underline` (subrayado)[8], `overline` (subrayado superior) o `line-through` (tachado). Si queremos que nuestros links no tengan subrayado, pondremos:

```
a { text-decoration: none; }
```

Podemos **transformar** las mayúsculas y minúsculas con la propiedad `text-transform`. Si ponemos `lowercase`, todo se mostrará en minúsculas; con `uppercase`, en mayúsculas; y con `capitalize` pondremos una letra capital al principio de cada palabra.

En cuanto al **espaciado**, para la distancia entre palabras usamos `word-spacing`; para el de las letras, `letter-spacing`; y para el **interlineado**, `line-height`.

Un ejemplo con todo esto, para la etiqueta `<strong>`:

```
strong {
  color: #000;
  letter-spacing: 0.25em;
  text-trasform: uppercase;
}
```

[6] A mucha gente le gusta montar Guerras Sagradas acerca de la conveniencia o no de usar texto justificado. Todavía no se han puesto de acuerdo, así que probablemente es un asunto de gustos.

[7] Bueno, exactamente equivale a la anchura de la letra "M" mayúscula.

[8] ¡Cuidado! Los humanos tenemos la estúpida manía de pensar que cualquier texto subrayado es un enlace.

¿Demasiada información? Sí, pero ten en cuenta que no hay que memorizar todo esto. Ni de coña. Lo importante es saber que **existe** una propiedad que hace tal cosa, no cómo se llama ni qué parámetros tiene. Para eso ya están las guías de referencia rápida y los editores de CSS.

Capítulo 11

El modelo de caja

> No consumo drogas, mis sueños ya son lo suficientemente escalofriantes
>
> M.C. Escher, artista

Es hora de conocer uno de los fundamentos más importantes del CSS: el **modelo de caja** (*box model)*. Es muy fácil, pero entenderlo bien es vital para poder realizar una buena maquetación de la web.

11.1. ¿Cómo es el modelo de caja?

En realidad, todos los elementos de una web (párrafos, enlaces, imágenes, tablas, etc.) son **cajas** rectangulares. Los navegadores sitúan estas cajas de la forma que nosotros les hayamos indicado para maquetar la página.

Hay dos tipos de cajas: **block** e **inline**. Los elementos **block** rompen el flujo de maquetación. Esto es, aparecen solos, insertando "saltos de línea". Los elementos **inline** siguen el flujo, y están contenidos dentro de elementos de bloque.

Por ejemplo, un párrafo sería un elemento **block** (no podemos tener un párrafo al lado del otro, sino que dos párrafos seguidos aparecerán uno **abajo del otro**. En cambio, un enlace es un elemento **inline**, ya que no "corta" el texto donde está metido.

Estas dos diferencias son importantes, pero hay que tener en mente que ambos tipos comparten el modelo de caja, que es el que aparece en la figura 11.1.

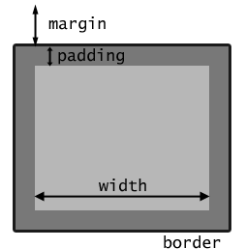

Figura 11.1: Modelo de caja

Las propiedades más importantes de una caja son: `width` (ancho), `height` (alto), `padding` (relleno), `border` (borde) y **margin** (margen).

11.2. Ancho y alto

La propiedad `width` es un poco confusa, y durante mucho tiempo era horroroso trabajar con ella debido a que Ya-Sabes-Quién no la implementaba correctamente[1]. Afortunadamente, desde la versión 6 del IE, `width` funciona como debería, así que es un quebradero de cabeza menos.

A lo que íbamos, `width` representa el **ancho de la caja**. Pero es el **ancho interior**, es decir, si bordes, márgenes, ni *padding*. Podemos indicar este ancho en medidas absolutas (normalmente píxeles) o relativas (normalmente %).

Aunque los elementos **inline** tienen `width`, si la modificamos con CSS no veremos ningún resultado visual. Esto es porque el ancho de estos elementos se establece automáticamente para que se ajuste a las dimensiones del elemento inline. Por ejemplo, si tenemos un enlace que consiste en un texto de cinco caracteres, el ancho (`width`) de este elemento será lo que ocupen esos cinco caracteres.

[1] Lo que ha dado lugar a competiciones de a ver quién conseguía el chanchullo más bonito para sortear el bug. Si estás interesado, googlea buscando "box model hack".

Sobre el **alto** de la caja, se controla con la propiedad `height`, y todo lo que hemos dicho antes sobre el ancho, también se aplica aquí.

11.3. Padding

Con `padding` establecemos la distancia de **"relleno"** entre el límite interior de la caja y el exterior (borde). Es una definición muy mala, pero se entiende a la perfección en el dibujito.

Si queremos poner un padding de 20 píxeles para toda la caja, lo haríamos así:

```
padding: 20px;
```

Podemos establecer un padding distinto para cada lado, usando los sufijos `-top` (superior), `-bottom` (inferior), `left` (izquierda) y `right` (derecha):

```
padding-top: 10px;
padding-bottom: 5px;
padding-left: 30px;
padding-right: 20px;
```

Podemos abreviar lo anterior en una sola línea, indicando primero el padding superior y luego siguiendo el orden de las agujas del reloj. Es decir, nos quedaría: arriba, derecha, abajo, izquierda. El ejemplo anterior se acortaría así:

```
padding: 10px 20px 5px 30px;
```

Otro atajo útil es especificar sólo dos medidas: una corresponderían al padding superior e inferior, y la otra al lateral. Si queremos que los paddings superior e inferior sean de 10 píxeles, y los laterales (izquierdo y derecho) de 20 píxeles, escribimos:

```
padding: 10px 20px;
```

11.4. Bordes

Si queremos que nuestra caja tenga **bordes**, lo conseguimos con la propiedad `border`. Tiene la siguiente sintaxis:

```
border: width | style | color
```

Como habrás supuesto, `width` especifica el **grosor** del borde. Normalmente es una medida en píxeles, pero también podemos utilizar las palabras `thin` (fino), `medium` (normal) y `thick` (grueso). Por supuesto, cómo de gordo es `thick` queda a interpretación del navegador.

En cuanto a `style`, es el **tipo** de borde. Hay bastantes, pero los más comunes son: `solid` (línea continua), `dashed` (línea discontinua), `dotted` (línea de puntos) y `double` (línea continua doble).

Por último, `color` indica el **color** del borde[2].

Podemos escoger un tipo de borde diferente para cada lado con los sufijos `-top`, `-bottom`, `-left` y `-right`. Por ejemplo, para poner que algo tenga el borde inferior de 1 píxel a puntitos rojos:

```
border-bottom: 1px dotted #f00;
```

Para **eliminar el borde**, simplemente ponemos que tiene de grosor 0 píxeles o que el estilo del borde es `none`. Esto es muy importante con las imágenes, ya que si tenemos una imagen enlazando a algo, los navegadores la ponen con un reborde muy feo. Así que esto se ha convertido ya en un fijo de las hojas de estilos:

```
img { border: none; }
```

11.5. Márgenes

Los **márgenes** se controlan con la propiedad `margin`, y es la distancia entre el borde de la caja y los elementos que la rodean.

En cuanto a la forma de usarla, es igual que con la propiedad `padding`, así que la forma de escribir y los atajos es exactamente la misma. Por ejemplo, si queremos márgenes superior e inferior de 20 píxeles, y laterales de 5 píxeles:

```
margin: 20px 5px;
```

¡Truco del almendruco! Para **centrar un elemento de bloque**, podemos hacer uso de `auto`:

```
margin: 0px auto;
```

[2] ¡Wow!

11.6. Capas

Vamos a hablar de una etiqueta XHTML que nos quedó por ver y está estrechamente ligada con CSS: `<div>`. Esta etiqueta se encarga de crear una **capa**, que es un elemento de **bloque** que sirve de contenedor a otros elementos de bloque e inline.

¿Para qué nos sirven? Primero, para organizar **semánticamente** nuestra página. El atributo `id` tiene carga semántica, así que si queremos poner en la cabecera[3] de nuestra web el título y el menú, haríamos esto:

```
<div id="header">
  <h1>Mi blog</h1>
  <ul id="menu">
  <li><a href="..." title="...">Principal</a></li>
  <li><a href="..." title="...">Acerca de</a></li>
  <li><a href="..." title="...">Enlaces</a></li>
  </ul>
</div>
```

El otro uso de las capas es el de **maquetar**. Por ejemplo, el layout típico de un blog tiene cuatro capas: la cabecera, la del contenido principal, la de la barra lateral[4] y la del pie de página. Mediante CSS, podemos controlar la disposición de estas capas, y así conseguir el diseño que queramos.

11.7. Floats

Los **floats** son probablemente una de las cosas que más cuesta dominar. En inglés los califican como *"tricky"*. Yo los califico como "pequeños cabroncetes".

Lo que hacen los **floats** es **alterar el flujo** de la página, "incrustando" en él un elemento de bloque. El ejemplo más típico es el de si queremos poner una imagen acompañando a un texto, y que el texto "envuelva" a la imagen. Esto lo conseguimos creando una clase como la siguiente:

[3] ¡Ojo! Cabecera en cuanto a maquetación. ¡No tiene nada que ver con la etiqueta head!

[4] Muy útil y socorrida para poner pijadas, como la wishlist de Amazon. Y si cuela, cuela.

```
.izquierda {
  float: left;
}
```

Podemos indicar tanto `left` (izquierda) como `right` (derecha). ¿Fácil, rápido y para toda la familia? No. Después de un float, suelen ocurrir **sucesos paranormales**[5]. La mayoría de ellos suelen desaparecer mediante la propiedad `clear`, que se encarga de "anular" los floats. Los valores que admiten son `left`, `right` y `both` (que significa "ambos").

Volviendo al ejemplo de layouts de blogs, lo normal es poner el contenido y la barra lateral mediante floats. Lo que suele pasar es que una de estas dos columnas pasa por encima del pie de página, en lugar de quedar el pie al final de todo. Esto se suele arreglar así:

```
#footer {
  clear: both;
}
```

Buena suerte, y que la Fuerza te acompañe.

[5] Ya te darás cuenta, ya.

Capítulo 12

Algunos truquitos

> He visto cosas que nunca creeríais...
>
> Blade Runner
> Roy Batty, replicante

A partir de ahora, los capítulos de CSS serán totalmente prácticos. Este primero mostrará diferentes "trucos" para conseguir ciertas cosas. Asegúrate de probarlos y de comprender todos ellos antes de continuar.

12.1. Links que cambian

Los enlaces tienen tres estados: **sin visitar** ("normales"), **hover** (cuando pasamos el cursor del ratón por encima), **activos** (cuando hacemos clic) y **visitados** (cuando ya hemos ido a esa dirección).

Estos estados se corresponden a **pseudoelementos**, y podemos cambiar la apariencia de los enlaces con CSS. Sin embargo, algunos navegadores tienen un bug que, dependiendo del orden en el que escribamos las reglas, se mostrará el resultado correctamente o no. ¿Cómo lo solucionamos? Pues **LoVe/HAte**. Es un mnemotécnico que nos ayudará a recordar el orden en el que estas reglas funcionan bien en cualquier navegador. Quedaría así:

```
/* links normales */
a:link {
  text-decoration: none;
```

```
  color: #00a;
}

/* visitados */
a:visited {
  color: #a00;
}

/* hover */
a:hover {
  text-decoration: underline;
}

/* activos */
a:active {
  font-weight: bold;
}
```

Lo que hace el código anterior es poner los enlaces de color azul marino y sin subrayado. Cuando pasamos el cursor por encima, aparece el subrayado. Al hacer clic, el texto del enlace se pone en negrita. Los enlaces que ya hemos visitado aparecen de color granate.

Fácil, ¿no?

12.2. Links con el subrayado de diferente color

Al usar la propiedad `text-decoration`, el color de la línea de subrayado es el mismo que el del texto del enlace. Podemos hacer un pequeño chanchullo para cambiar esto, y es **quitar el subrayado** y poner en su lugar un **borde**:

```
a {
  color: #fff;
  text-decoration: none;
  border-bottom: 1px solid #f0c;
}

a:hover {
  border: none;
}
```

Esto haría que los enlaces fueran de color blanco, y que la línea de subrayado fuese fucsia. Al pasar el cursor por encima, se eliminaría el subrayado.

El pseudoelemento **hover** funciona para cualquier otro elemento de la web[1]. No dudes en usarlo en celdas de tablas o capas.

12.3. Campos de formulario chulos

Un efecto muy Web 2.0[2] es hacer los **campos de formulario** con fuente gigante y que el fondo del campo cambie cuando el usuario está usándolo. Esto se consigue con el pseudoelemento `focus`:

```
input, textarea {
  font-size: large;
  border: 3px solid #70C332;
  color: #666;
  background: #fff;
}

input:focus, text-area:focus {
  background: #eee;
}
```

Lo anterior pondría los campos de formulario `input` y `textarea` con una fuente grande y gris oscura, borde gordito de color verde y el fondo blanco. Al hacer foco (es decir, cuando el visitante está situado en ese campo), el fondo cambia a un color gris clarito.

12.4. Blockquotes 2.0

Otro legado de la Web 2.0: **blockquotes** con texto gigante y unas comillas gigantes. Quizás es algo difícil de imaginar con esta descripción, pero si echas un vistazo a la figura 12.1, seguro que ya lo has visto en algún sitio antes.

Un ejemplo de código podría ser este:

```
blockquote {
```

[1] A no ser que uses una versión antigua de un navegador subdesarrollado... Sí, ese.

[2] Es decir, pijo.

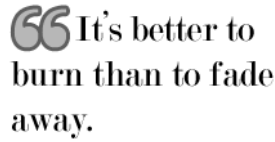

Figura 12.1: Blockquote molón

```
  background: url(comillas.gif) top left;
  background-repeat: no-repeat;
  text-indent: 30px;
  text-align: left;
  font-size: x-large;
  padding: 0px
}
```

Tendrás que ajustar los valores según la fuente y la imagen en sí. En muchos casos, es más conveniente utilizar padding y quitar el indentado.

12.5. Cambiar texto por imagen

Muchas veces, especialmente en headings, queremos **cambiar el texto** y en su lugar que aparezca una **imagen** bonita. Un ejemplo del código sería este (suponiendo que nuestra imagen mide 300x100 píxeles):

```
<!-- XHTML -->
<h3 id="enlaces"><span>Enlaces</span></h3>

/* CSS */
h3#enlaces {
  width: 300px;
  height: 100px;
  padding: 0px;
  background: url(links.png) top left no-repeat;
}

h3 span {
  visibility: hidden;
```

```
}
```

La etiqueta `span` es nueva, y no significa nada (literalmente). Es una etiqueta "vacía", y sólo la usamos para conseguir ciertos efectos CSS. Como semánticamente no tiene ningún valor, hemos de evitar su uso todo lo posible.

Lo que hace el código anterior es hacer coincidir las dimensiones de `<h1>` con las de la imagen. Después, gracias al `<span>` nos encargamos de que el texto del heading sea **invisible**. No obstante, aunque el texto sea invisible, todavía existe. Por eso, si el título es muy largo, es posible que sea más grande que la imagen que usamos de fondo, y es posible que nos descoloque el layout. En este caso, puedes cambiar la fuente de `<h1>` y hacerla enana.

Capítulo 13

Layout tableless a dos columnas

> La frase científica más excitante, la que precede a grandes descubrimientos, no es "¡Eureka!", sino "Qué raro".
>
> Isaac Asimov, bioquímico y escritor

En este capítulo veremos cómo crear un layout sin tablas a dos columnas, muy típico en los blogs. Es bastante sencillo de seguir, y es un ejemplo de que dejar de usar tablas para maquetar no tiene por qué ser un proceso doloroso.

13.1. Características

Esta maquetación tiene las siguientes propiedades:

- Anchura fija
- Centrado
- 2 columnas (una de ellas de barra lateral)
- Cabecera (*header*)
- Pie de página (*footer*)

Puedes echarle una ojeada en la figura 13.1.

Figura 13.1: Layout de 2 columnas

13.2. Código XHTML

El esqueleto del código XHTML de nuestra página es el siguiente (iría dentro del `<body>`):

```
<div id="container">

<div id="header">
<h1>Título</h1>
<h2>Subtítulo</h2>
</div>

<div id="sidebar">
<h3>Sección</h3>
<p>Bla bla...</p>
</div>

<div id="main">
<h3>Sección</h3>
<p>Contenido principal</p>
</div>

<div id="footer">
<p>Pie de página</p>
</div>
```

```
</div>
```

Como ves, tenemos cuatro capas. Ahora veremos el código CSS de cada una de ellas.

13.3. #container

Esta capa es un **contenedor** para el resto de la página. Establece la anchura de todo y es la que centra el contenido. El truco está en usar `auto` dentro de `margin` para lograr el centrado. Esto no funciona en el IE, así que tendremos que echar mano de `text-align: center` en el `body`. El CSS completo es este:

```
body {
  text-align: center;
}

#container {
  width: 700px;
  margin: 0px auto;
  text-align: left;
}
```

13.4. #sidebar

Esta es la **barra lateral** tan famosa en los blogs. En otras páginas web, podemos poner ahí menús, publi, o cualquier otra cosa. La clave en esta capa es usar `float: left`, que la sitúa a la izquierda y hace que todos los demás elementos la rodeen. Aquí va el código:

```
#sidebar {
  width: 200px;
  padding: 10px;
  float: left;
}
```

El `padding` no es obligatorio, pero lo he puesto para después poder explicar bien la siguiente capa. También es necesario especificar el **ancho** en píxeles. Ah, con este método **la barra no se extenderá hasta abajo**, sino que se corta en su final (esto se ve

bien en la figura 13.1). Si pones la barra de otro color, y quieres que llegue hasta abajo, puedes usar la técnica de **Faux Columns**[1].

13.5. #main

Esta es la capa donde irá el **contenido** (en un blog, esto serían los posts). Lo importante en esta capa es indicar con `margin-left` la **distancia** desde el borde del `#container` hasta esta capa, pasando por encima de la barra lateral.

Nuestra barra lateral tiene 200 píxeles de ancho y 10 píxeles de padding a la izquierda y a la derecha. Si hacemos memoria del **modelo de caja**, el margen izquierdo que tenemos que indicar serían **200 + 10 + 10 + X**, donde **X** es la cantidad que nosotros queremos dejar de separación entre la barra lateral y el contenido principal. Para el ejemplo, pondremos 5 píxeles:

```
#main {
  margin-left: 225px;
}
```

13.6. #footer

La capa del **pie de página** viene bien para poner información de copyright/copyleft[2] o cualquier otro tipo de datos misceláneos.

Aquí únicamente hemos de tener cuidado de "anular" el **float** que tiene la barra lateral mediante un `clear`:

```
#footer {
  clear: both;
}
```

¡Ya tenemos nuestro propio layout tablas que cumple los estándares del W3C!

[1] http://www.alistapart.com/articles/fauxcolumns/ Es un chanchullo sencillo e ingenioso. Básicamente consiste en poner una imagen de fondo que simule la columna.

[2] Recordad, chicos, las licencias Creative Commons son buenas para la salud y para el karma.

Capítulo 14

Cabeceras

> Su carencia de fe resulta molesta.
>
> A New Hope
> Darth Vader, sith

Una de las cosas muy útiles que se han puesto de moda, es hacer que la cabecera de una web sea a su vez un enlace hacia la sección principal de una web. El hecho de que si nos perdemos tengamos un enlace bien gordo arriba del todo, es un salvavidas.

14.1. Código XHTML

Los estándares nos dicen que el **título** principal de la página se pone con la etiqueta `<h1>`. No obstante, el texto de esta etiqueta lo cambiaremos por una imagen. Como queremos además meterle un link, también usaremos la etiqueta `<a>`. Dentro del link, metemos una etiqueta `<span>` para poder ocultar con CSS el texto y que sólo se vea la imagen que hará de cabecera... pero nos estamos adelantando. El código XHTML sería este:

```
<div id="header">
  <h1><a href="..." title="...">
  <span>Título</span></a></h1>
</div>
```

14.2. #header

Primero tocaremos el CSS de la capa `#header`. Establecemos el **ancho** y el **alto** de la capa con las medidas que tenga la **imagen** (400x100 en este caso). Además, ponemos la imagen de fondo y quitamos el margin y el padding:

```
#header {
  background: url(bg.png) top left no-repeat;
  width: 400px;
  height: 100px;
  margin: 0px;
  padding: 0px;
```

14.3. El heading

También quitamos los márgenes a la etiqueta `<h1>` (para que la capa mida exactamente lo que la imagen) y hacemos invisible el contenido de la etiqueta `<span>`, para que el texto de la cabecera sólo se muestre en navegadores de texto o aurales:

```
#header h1 {
  margin: 0px;
}

#header a span {
  visibility: hidden;
}
```

14.4. El enlace

Ahora sólo nos falta la etiqueta `<a>`. Lo que queremos conseguir es que se pueda hacer clic en **todo el área** de la cabecera, no sólo en lo que sería el texto (invisible) del enlace. Para eso, tenemos que transformar el enlace en un **elemento de bloque**, y darle las medidas de la imagen.

Además, hemos de quitar los márgenes y el padding, así como la decoración del texto, para que no salga una línea de subrayado:

```
#header a {
  width: 400px;
```

```
  height: 100px;
  display: block;
  padding: 0px;
  margin: 0px;
  text-decoration: none;
}
```

Et voilà! Ya tenemos nuestra cabecera cuca con enlace incorporado.

Capítulo 15

Listas personalizadas

> La Red es una pérdida de tiempo. Y eso es precisamente lo que está bien.
>
> William Gibson, escritor

Ahora vamos a aprender cómo modificar las **listas desordenadas** para que tengan **viñetas personalizadas**. Para ello, necesitaremos una imagen pequeñita que haga de viñeta. En Internet hay muchísimas para descargar, aunque puedes dibujar las tuyas propias. Supondremos que la imagen se llama `bullet.png`.

15.1. Código XHTML

Extremadamente simple. Se trata de una lista sin ordenar, sin más misterio:

```
<ul>
  <li>Sonata Arctica</li>
  <li>Nightwish</li>
  <li>HIM</li>
</ul>
```

15.2. La lista

Ahora la parte de CSS. Disponemos de una propiedad que se llama `list-style-image`, pero da problemas con ciertas medidas

de viñetas. Así que tenemos que guarrear un poco el código. Para el elemento `ul` sería:

```
ul {
  padding-left: 10px;
  margin-left: 10px;
  list-style-type: none;
}
```

Lo más importante es el `list-style-type: none`, que se encarga de quitar esas viñetas feas que pone el navegador por defecto. El `padding` y el `margin` es para sangrar la lista (puedes poner los valores que quieras, o incluso quitarlos).

15.3. Los ítems

¡Ya sólo queda poner nuestra viñeta! Para ello, modificamos el estilo de los `li` que estén en listas desordenadas:

```
ul li {
  padding-left: 12px;
  background: url(bullet.png) 0em 0.5em no-repeat;
  margin-bottom: 1em;
}
```

Lo primero que vemos es el `padding` por la izquierda. Esto **no es el sangrado**, es una distancia de relleno que ponemos para que el texto del ítem **no esté encima** de la viñeta. Este valor lo tienes que modificar dependiendo de las dimensiones de la imagen de tu viñeta.

Después, nos encontramos con la propiedad `background`. Después de establecer la imagen, debemos indicar en qué **posición** (recuerda: primero horizontal, luego vertical) se encuentra la viñeta. ¡Aquí hay truco! Como ves, trabajamos con **em** y no con medidas absolutas en píxeles. ¿Por qué? Porque así nos vale para cualquier tamaño del texto[1]. Lógicamente, es muy difícil dar con el valor correcto a la primera, así que habrá que probar varias veces hasta topar con el que mejor quede. Ah, no se nos puede olvidar el `no-repeat` para evitar el **mosaico**.

Por último, `margin-bottom` se encarga de establecer la **separación** entre un ítem y otro de la lista.

[1] Recuerda que 1em equivale a la anchura de la letra M.

Muy sencillo, ¿no?

Capítulo 16

Menús verticales

> Dije que era un adicto. No dije que tenía un problema.
>
> House M.D.
> Greg House, médico

¿Sabías que con listas se pueden hacer **menús verticales** chulos? ¿Que **por qué listas**? Porque un menú vertical es una serie de elementos relacionados, y lo más semántico que podemos hacer es meterlo en una lista.

El principal problema es que las listas son feas. En el capítulo anterior vimos cómo hacerlas más bonitas, pero quizás queramos algo un poco más diferente para nuestro menú. CSS nos da la solución, así que ya no tendrás excusa para hacer un menú en Flash[1].

En el menú que haremos ahora, hacemos apaños con los colores de fondo y los bordes para conseguir efectos cuando el ratón pase por encima. Puedes ver cómo quedaría en la figura 16.1. Como va con CSS, podríamos fácilmente incorporar imágenes, pero eso queda como ejercicio[2].

[1] Que, por cierto, es de lo peor que se te podría ocurrir. Si alguien no tiene instalado el plugin de Flash (*nunca* des por supuesto que el vistante tenga instalado un plugin), ni siquiera podrá navegar por tu página.

[2] No tiene gracia si te lo dan todo mascadito.

Figura 16.1: Menú vertical

16.1. Código XHTML

Al grano, necesitamos una lista tal que así:

```
<div id="menu">
<ul>
  <li><a href="..." title="...">Home</a></li>
  <li><a href="..." title="...">Archivos</a></li>
  <li><a href="..." title="...">Enlaces</a></li>
  <li><a href="..." title="...">Acerca de</a></li>
</ul>
</div>
```

16.2. La lista

¡A meterle mano al CSS! Empezaremos primero con darle la **anchura** que queramos a la lista (que será la anchura del menú), poner una `fuente` maja y **quitar las viñetas y márgenes** de la lista.

```
#menu ul {
  list-style-type: none;
  margin: 0px;
  padding: 0px;
  width: 200px;
  font-family: Arial, sans-serif;
  font-size: 11pt;
}
```

Ponemos a continuación un **color de fondo** para los ítems de la lista (`<li>`). Lo normal sería poner aquí los efectos de `hover`, para que se activen cuando el ratón pase por encima de todo el bloque, no sólo del texto del enlace; pero Quien-Tú-Sabes en sus versiones 6 e inferiores no soporta hovers en cosas que no sean un enlace, así que vamos a tener que emplear una artimaña más adelante. Por ahora, el CSS para el elemento `li` sería así de sencillo:

```
#menu ul li {
  background-color: #666;
}
```

16.3. Enlaces

Y aquí viene el chanchullo del que os hablaba antes para hacer el hover[3]: poner el enlace como si fuera un **bloque**, y así ocupará todo el `li` y podremos manipular sus dimensiones.

```
#menu ul li a {
  color: #ccc;
  text-decoration: none;
  text-transform: uppercase;
  display: block;
  padding: 10px 10px 10px 20px;
}
```

Sólo nos queda hacer los cambios para el **hover**:

```
#menu ul li a:hover {
  background: #000;
  border-left: 10px solid #333;
  color: #fff;
}
```

Si lo pruebas, verás que las letras se desplazan al hacer el hover, debido a que aparece el borde izquierdo. Si no os gusta este efecto, podéis añadir la línea siguiente al link cuando está normal. Lo que hace es poner un borde del mismo color que el fondo de los `li`, y así parece que no existe:

```
border-left: 10px solid #666;
```

[3]Haz memoria: es lo mismo que empleamos en un capítulo anterior para poner un enlace en la cabecera de la web.

Merece la pena trastear con esta técnica, puede dar lugar a menús muy logrados.

Apéndice A

Por qué en IE se ve "bien" y en Firefox se ve "mal"

> Sólo hay dos cosas infinitas: el Universo y la estupidez humana, y no estoy muy seguro acerca del primero.
>
> ALBERT EINSTEIN, físico

Un día una amiga me comentó que una compañera suya estaba haciendo una web, y que **"en el Explorer se ve bien y en Firefox mal"**, y me preguntó que a qué podría ser debido. Le di una respuesta deprisa y corriendo y creo que no me expresé bien, así que aquí va la *full version*. La comparto aquí con vosotros porque seguramente nos esperen muchas más preguntas de este estilo.

La clave para entenderlo bien es darle la vuelta a la tortilla. La realidad es que en IE se ve "mal", y en **el resto de los navegadores** (no sólo Firefox) se ve "bien". Pongo "bien" y "mal" entre comillas, porque son apreciaciones que hacemos los humanos, subjetivas.

Los navegadores **no son adivinos** con bolas de cristal que se conectan a la mente del maquetador web e interpretan su voluntad. El maquetador tiene que dejar escrito, detalladamente, el contenido y apariencia de la página web: esto se consigue con los lenguajes XHTML y CSS.

Estos lenguajes se encuentran bien definidos como **estándar** en el WWW Consortium (W3C para los colegas). Este organismo

se encarga de describir con precisión cómo deben interpretar el XHTML y el CSS los navegadores.

Ahora bien, los navegadores no siempre cumplen a pies juntillas lo que dice el W3C. Es más, cierto navegador[1] no hace ni puto caso.

Pongamos un ejemplo ficticio: imaginemos que el Explorer "confunde" los colores rojo y blanco, y los intercambia, debido a un error de programación (no entraremos a valorar si por descuido o deliberadamente). Es decir, que donde pone `#fff` IE lo interpreta como `#f00`, y viceversa. En este ejemplo, supondremos que el resto de navegadores interpretan los colores correctamente.

¿Qué pasaría si quisiéramos hacer una página web con fondo blanco? En nuestro código CSS, pondríamos algo así...

```
body { background: #f00};
```

... que en IE se mostraría blanco. Entonces cuando vamos alegremente a mirar la web con otros navegadores, vemos que se muestra de color rojo fosforito. *"En Firefox se ve mal"*. Pues no. Por mucho que se empeñe el Explorer en hacer creer a los desarrolladores web, el número `#f00` significa rojo, y *prou*.

Lo que ha pasado es que una página se ha **desarrollado mal** (a menudo inconscientemente) para **forzar** a que se vea "bien" en IE. Lo que obtenemos es que en IE la página se visualiza incorrectamente, pero por **casualidades místicas**, esa visualización coincide con los deseos del diseñador.

Un caso real y muy gráfico de cómo IE visualiza mal las páginas lo podemos encontrar en el **Acid Test**[2]. Es un ejemplo de página web que construye mediante código estándar y válido un dibujito de una cara sonriente. Según lo bien programado que esté el navegador, veremos ese dibujo más o menos bien.

Los únicos navegadores "mayoritarios" que lo muestran correctamente son **Safari** (Mac), **Konqueror** (GNU/Linux) y **Opera** (multiplataforma). Firefox se queda a medio camino, pero lo de Internet Explorer clama al cielo.

En resumen, lo más fácil para que una web se vea más o menos bien en todos los navegadores es **hacerla primero para Firefox** o cualquier otro navegador, y luego "apañarla" como buenamente podamos para Internet Explorer.

---

[1] Quien-Tú-Ya-Sabes

[2] http://www.webstandards.org/files/acid2/test.html

Apéndice B

Migración rápida a XHTML

Bienvenido al Mundo Real.

The Matrix
Morfeo, capitán de la Nabucodonosor

Si ya usabas `HTML 4`, puedes aprender muy rápido XHTML, puesto que las bases son las mismas. Sólo hay que tener en cuenta que XHTML deja de lado todo aspecto estético y **se centra en el contenido y en la semántica**. En lugar de usar una etiqueta para dar un aspecto concreto, usamos etiquetas para hacer que las palabras signifiquen una cosa u otra. Ahora veremos una serie de "reglas" a seguir para pasarnos a XHTML. También es recomendable leer el capítulo 2, donde se explica la estructura de un documento XHTML. Evidentemente, no se muestran aquí todas las diferencias, pero sirve para hacernos una idea de qué es lo que nos espera[1].

B.1. Minúsculas y comillas, por favor

Antes era una práctica muy común escribir las etiquetas en mayúscula, para diferenciarlas del código. Por compatibilidad con XML, en XHTML todas las etiquetas deben ir en minúsculas.

[1] Diego Lafuente, aka Mini-D, ha escrito artículos muy buenos en su blog sobre el tema. Destaca el post titulado *"13 pasos para dar el salto"*. Podéis encontrarlo en `www.minid.net`

Además, todos los atributos tienen que estar entre comillas dobles. Por ejemplo, si antes teníamos:

```
<IMG SRC=icono.gif ALT=Icono>
```

Ahora escribimos:

```
<img src="icono.gif" alt="Icono" />
```

B.2. Todas las etiquetas se cierran

Con HTML podíamos crear párrafos con la etiqueta `<p>` sin necesidad de cerrarla con `</p>`. Lo mismo ocurría con `<li>`, por ejemplo. En XHTML **ninguna etiqueta puede quedar sin cerrar**.

Lo de "ninguna" es literal, y etiquetas que no tienen una de cierre, como `<img>`, `<br>` deben ser cerradas también. ¿Cómo lo hacemos? Insertando la barra `/`, de forma que el salto de línea ahora nos queda `<br />`. El espacio en blanco que hay entre el nombre y la barra es necesario para que los navegadores antiguos reconozcan la etiqueta.

B.3. FONT y ciertos atributos desaparecen

Hemos dicho que XHTML deja de lado la apariencia del documento, ya que eso es controlado por CSS. Entonces, las etiquetas `<font>` y `<basefont>` carecen de sentido.

Además, esos atributos de algunas etiquetas que hacen referencia al **color** de las cosas, imágenes de fondo, etc. también desaparecen por este motivo: son sustituidos por reglas CSS. Así que *sayônara* a `bgcolor` y compañía. Lo mismo para el atributo `align` usado en párrafos e imágenes.

B.4. B y amigos también se van

Ciertas etiquetas de formato, como `<b>` (negrita), `<i>` (cursiva), etc. ya no se usan porque hacen referencia exclusivamente a la **apariencia** de las palabras. Si queremos dar énfasis, utilizamos `<em>`, y para dar énfasis más fuerte `<strong>`. Los navegadores suelen mostrarlas como cursiva y negrita, respectivamente, aunque esto es lo de menos porque podemos cambiarlo con CSS.

B.5. Hay que usar alt y title

XHTML hace hincapié en la **accesibilidad** de un documento, y por eso debemos facilitar atributos de "apoyo" a algunas etiquetas. La etiqueta `<img>` dispone del atributo `alt`, que se muestra cuando la imagen no se puede cargar (a veces también cuando se pasa el cursor del ratón por encima). Hay que utilizarlo siempre.

Existe un atributo muy similar llamado `title`, que se utiliza en la etiqueta `<a>` (entre otras) y sirve para mostrar una descripción del sitio al que nos dirigimos. Por ejemplo:

```
<a href="http://www.google.es" title="Buscador">
Google</a>
```

B.6. Cuidado al anidar etiquetas

XHTML es muy estricto en cuanto a la **anidación** de etiquetas. Básicamente, hay dos tipos de elementos: los **block** y los **inline**. Los block son etiquetas como párrafos, headings, listas, etc. Así a ojo los distinguimos porque siempre van solos e insertan saltos de línea. Los inline no interrumplen el flujo del texto. Son las etiquetas de formato, los enlaces, y demás. **No podemos meter un elemento block dentro de uno inline**.

Por ejemplo, si queremos hacer que el heading principal de nuestra página sea un enlace, esto sería **incorrecto**:

```
<a href=".." title="..."><h1>Texto</h1></a>
```

Hay que hacerlo así:

```
<h1><a href="..." title="...">Texto</a></h1>
```

Además, hay ciertas etiquetas que no admiten otras dentro. Si tienes dudas, lo mejor es utilizar el **validador** de código del WWW Consortium[2].

B.7. No existen los frames

Aunque la especificación XHTML 1.0 Frameset permite el uso de frames (marcos) en una página, tanto la Transitional como la

[2]http://www.w3c.org

Strict los prohíben[3]. Así que ya no podemos usar **ni frames ni inline frames**.

De todos modos, aunque se pudieran emplear, no deberíamos, ya que los frames son un atentado contra la usabilidad[4].

B.8. No se puede utilizar target

Antes, la etiqueta `<a>` tenía un parámetro llamado `target` que permitía especificar en qué frame debía cargarse el destino de un link. Como ya no hay frames, este atributo es innecesario.

A los amantes del `target="blank"` para abrir webs en ventanas nuevas les digo que si el visitante quiere abrir un ventanuco nuevo, lo hará con el botón derecho del ratón. Y para los que usamos navegación con pestañas[5] es bastante molesto que se abran nuevas instancias del navegador. Gracias.

B.9. Las tablas no se usan para maquetar

Aunque puedes crear un documento XHTML con tablas para maquetar que pase el análisis del validor del W3C, va contra la filosofía de dejar XHTML sólo para el contenido. Las tablas están hechas para representar **información tabulada**, no para diseñar layouts.

De todos modos, el paso para un layout `tableless` no suele ser fácil, así que creo que es conveniente familiarizarse primero con el resto de XHTML y CSS, y dejar la maquetación sin tablas para más tarde[6].

B.10. Los ampersands dan por saco

Si tienes URL's que contengan el **ampersand** (&), te llevarás la sorpresa de que el validador del W3C se pelea con ellas. Esto es debido a las `entities`.

---

[3] En realidad esto es como la Cherry Coke. Existir, existe, pero nadie la compra.

[4] Y a Google no le gustan, de paso.

[5] Ahora mismo prácticamente toda la humanidad con acceso a Intenet, ya que la nueva versión del Maligno por fin las trae

[6] "Tarde" no significa "nunca".

¿Qué demonios son? Pues la manera que tenemos de insertar caracteres de forma "segura"[7]. Por ejemplo, el carácter `á` se escribiría `&aacute;`. Como ves, las entities comienzan por un ampersand y acaban en punto y coma.

Por eso, cuando se leen las URLs que contienen ampersands, hay confusión porque lo que sigue no es una entity... Así que lo que tenemos que hacer es sustituir el ampersand por su propia entity, que es `&`. Es decir, que esta URL:

```
http://alianza.net/p.php?nick=leia&show=all
```

Se quedaría en:

```
http://alianza.net/p.php?nick=leia&amp;show=all
```

Aunque hemos estado hablando de URL's, también debemos emplear la entity `&` en nuestro texto normal y corriente cuando queramos escribir un ampersand.

[7]Es decir, que se vean cuando usamos una codificación muy limitada, como el ASCII estándar. Si usamos Unicode no deberíamos tener ningún problema.

Apéndice C

A favor del Unicode

> Un idiota no se curará hasta que se muera.
>
> Proverbio japonés

A la hora de almacenar archivos de texto, podemos elegir entre bastantes **codificaciones**, que viene a ser qué numerito (y de qué tamaño en bits) se asigna a cada carácter.

¿Qué es lo que pasa cuando abrimos un archivo en una codificación (por ejemplo, la Shift-JIS que es una japonesa) y le decimos que la lea con otra (la ISO 8850-1, de caracteres latinos)? Pues que obtenemos un motón de **símbolos extraños**. Esto pasa, por ejemplo, cuando alguien me envía un trackback[1] con una codificación distinta a la que uso en el blog.

¿Pero qué pasa si se quiere hacer un documento **multilingüe** con distintos alfabetos? ¿Qué pasa si desde una aplicación quiero leer cualquier documento en cualquier lengua? Para esto existe una **codificación de caracteres estándar**, que es la **Unicode**. Sacado de su web[2]:

> Unicode proporciona un número único para cada carácter, sin importar la plataforma, ni el programa, ni la lengua.

[1] Los trackback son un tipo especial de "comentarios" en los blogs. Cuando alguien de otro blog te enlaza al tuyo, si te manda un trackback, aparecerá en tu post un comentario con la URL del blog de la otra persona. Es útil para saber quién habla sobre cosas que has dicho.

[2] http://www.unicode.org

Existen varios formatos de Unicode, el más usado es el **UTF-8** y está bastante extendido en la Red. De hecho, en la blogosfera hispana las dos codificaciones más usadas son la ISO 8859-1 y la UTf-8. Usar Unicode es bueno y es la codificación **estándar**. Usadla.