Curso de DHTML: CSS y Layers

Lola Cárdenas Luque
http://rinconprog.metropoli2000.com
lornacl@iname.com
Última actualización: 3 de febrero de 2001

Con este curso se pretende completar los conocimientos de HTML, aprendiendo cosas sobre hojas de estilo y capas, las dos grandes categorías que constituyen el HTML Dinámico. Es altamente recomendable saber desenvolverse con el lenguaje HTML, puesto que estos temas son una ampliación a lo que ya se conoce de este lenguaje.

Índice General

1 Introducción **1**

2 Definición de los estilos **2**

3 Herencia de estilos **6**

4 Jerarquía de definición de estilos. Clases. Pseudoclases **8**

5 Propiedades de las hojas de estilo **11**
5.1 Propiedades de formato de bloque 12
5.2 Propiedades de las fuentes 13
5.3 Propiedades de texto 14
5.4 Propiedades de color y fondo 14
5.5 Propiedades de clasificación (listas) 15

6 Layers: Creación **16**
6.1 Diferencias entre ambos tipos de posicionamiento 17
6.1.1 Posicionamiento absoluto 17
6.1.2 Posicionamiento relativo 17

7 Propiedades de las capas **19**

8 La etiqueta <LAYER> de Netscape **22**

1 Introducción

DHTML son las siglas de Dynamic HyperText Markup Language (conocido como HTML Dinámico); se trata de una nueva especificación que viene a dar respuesta a la demanda de interactividad en las páginas web.

El mundo de la red hasta ahora había estado lleno de páginas con un contenido más o menos interesante, pero estático: una vez la página había cargado, ya no había modificación posible que la dotara de la interactividad que el gran público pedía. Y llegó DHTML

La especificación DHTML se podría dividir en tres grandes categorías: las hojas de estilo, las capas (layers) y las fuentes de letra cargables. Además, el uso de algún lenguaje de script, como JavaScript, contribuye en buena medida a la interactividad de las páginas, pues estos lenguajes son los que nos permiten cambiar las propiedades de forma dinámica. Si quieres saber algo de JavaScript, puedes consultar este curso.

Las hojas de estilo vienen a intentar volver a separar en un documento el estilo lógico del estilo físico, dejando este último en bloques de definición de estilos separados de la estructura del documento.

El estilo lógico se refiere a la lógica del documento: cabeceras, párrafos, ... no se preocupa de la apariencia final, sino de la estructura del documento. Por el contrario, el estilo físico no se preocupa de la estructura del documento, sino por la apariencia final: párrafos con un cierto tipo de letra, tablas con un determinado color de fondo, ...

La finalidad de las hojas de estilo es crear unos estilos físicos, separados de las etiquetas HTML (en lugar de como parámetros de las etiquetas), y aplicarlos en los bloques de texto en los que se quieran aplicar. Estos estilos podrán ser modificados en algunas ocasiones desde JavaScript, y esto empieza a darnos un poco más de interactividad.

Por otra parte, tenemos las capas, que vienen a darnos la solución al problema de poner elementos justo en la posición que queramos, evitándonos tener que hacer artificios para obtener el resultado buscado. Una capa será una parte más del documento que puede ser situada en cualquier posición del mismo, consiguiendo que se solape sobre algunos elementos si es lo que necesitamos, adecuando sus márgenes y otras propiedades a lo que queramos hacer...

Por último, las fuentes cargables intentan solucionar el problema de que quien esté viendo nuestro documento no tenga en su ordenador la fuente que nosotros consideramos más apropiada para nuestra página, incrustando de alguna manera la fuente en la propia página.

Estas tres grandes categorías serán las que se irán viendo poco a poco, con ejemplos claros, a lo largo de este curso.

2 Definición de los estilos

CSS son las siglas de "Cascade StyleSheet". Se trata de una especificación sobre los estilos físicos aplicables a un documento HTML, y trata de dar la separación definitiva de la lógica (estructura) y el físico (presentación) del documento.

Además, se pretende dar solución a los problemas que da HTML para formatear el texto, y que tratan de arreglarse usando tablas sin borde, imágenes transparentes de 1 pixel a las que se les da el tamaño deseado para hacer espacios en blanco, ...

Tenemos varias posibilidades para definir un estilo: especificarlo directamente en la etiqueta en la que queremos usarlo, definirlo aparte y aplicarlo en las etiquetas que queramos, o definir estilos globales para las etiquetas (que podrán ser cambiados en las que no se desee aplicarlos).

Para aplicar un estilo a una etiqueta concreta, usaremos la sintaxis:

```
<etiqueta STYLE=''propiedad1:valor;...;propiedadN:valor''> ... </etiqueta>
```

Etiqueta es la etiqueta de HTML en la que queremos dar una apariencia concreta (<P>, <B>, <I>, ...). **STYLE** es el parámetro que indica que vamos a aplicar el estilo definido a continuación a la etiqueta en la que se encuentra. La definición del estilo son pares `propiedad:valor` separados por punto y coma. **Propiedad** será la característica de la etiqueta que queramos modificar (el color, el tamaño de la fuente, el tipo de letra, ...) y **valor** es el valor que queremos darle (color negro, 8 puntos de tamaño de letra, ...).

Por ejemplo, si tenemos un texto en negrita y queremos que salga en color rojo, haremos:

```
La negrita que vemos <B STYLE="font-size:14pt;color:red">es
mas grande y esta en rojo</B>
```

Sin embargo, lo que suele ocurrir es que queremos definir estilos que se apliquen a todas las etiquetas del documento, es decir, queremos que todo el documento tenga un cierto tipo de letra, que las tablas tengan otro, que las cabeceras tengan un color determinado, ..., para ello, definiremos estilos globales por medio de la etiqueta <STYLE> ... <STYLE> como sigue:

```
<STYLE TYPE="text/css">
<!--
 Etiqueta1,...,EtiquetaN : {propiedad1:valor;...;propiedadM:valor}
 ...
 Etiquetas1,...,EtiquetasR : {propiedad1:valor;...;propiedadS:valor}
 Otros: {propiedad1:valor;...;propiedadT:valor}
//-->
</STYLE>
```

Podemos aplicar el mismo estilo a varias etiquetas, escribiéndolas separadas por comas y, a continuación, la especificación del estilo según pares `propiedad:valor` separados por punto y coma y encerrados entre llaves {}. En un bloque de estilo global podremos definir cuantos estilos queramos. Aparece un *Otros*; se refiere a las llamadas *clases*, que nos permitirán que una etiqueta concreta tenga una apariencia distinta a la definida como global.

Es recomendable que definamos estos estilos globales dentro de la cabecera del documento (entre <HEAD> ... </HEAD>) para asegurarnos de que se aplicarán a todas las etiquetas para las que se haya definido un estilo. Veamos un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo con bloque de estilo </TITLE>
<STYLE TYPE="text/css">
<!--
BODY {font-family:Verdana,sans-serif;font-size:x-small;
      margin-left:0.25in; margin-right:0.25in}

H2 {font-family:Verdana,sans-serif;font-size:14pt;color:red}

B, TD {font-family:Verdana,sans-serif;font-size:x-small;
       color:olive}
TH {font-family:Verdana,sans-serif;font-size:x-small;
    color:white;background-color:#0080C0}
PRE, TT, CODE  {font-family:Courier New,Courier;
                font-size:9pt;color:maroon}
//-->
</STYLE>
</HEAD>

<BODY BGCOLOR=white>

<H2>Prueba de definicion de estilos en un bloque aparte</H2>

Como puede verse, la apariencia de esta pagina queda completamente
definida por los estilos que hemos especificado en el bloque STYLE
en la cabecera del documento. Los margenes son mas amplios de lo
habitual, la <B>negrita</B> tiene un tamano y un color fijos,
los trozos de texto en teletipo como <TT>este fragmento</TT>
tambien tienen definida su fuente, tamano y color, y vamos a ver como
quedan las tablas, para finalizar el ejemplo: <P>

<CENTER>
```

```
<TABLE BORDER=1 CELLSPACING=2 CELLPADDING=2>
 <TR> <TH>Cabecera 1</TH> <TH>Cabecera 2</TH> </TR>
 <TR> <TD>Celda (1,1)</TD> <TD>Celda (1,2)</TD> </TR>
</TABLE>
</CENTER>

</BODY>
</HTML>
```

Como sabemos definir estilos globales, sería interesante tenerlos definidos en un archivo aparte, pues si queremos dotar a todas las páginas de los mismos estilos, no es tarea grata copiar y pegar la definición de los estilos en cada una de las páginas.

Afortunadamente, sí podemos definir los estilos en un fichero distinto al documento HTML, y después referenciarlo desde el propio documento HTML. Esto lo haremos con la siguiente etiqueta, dentro de la cabecera del documento (entre <HEAD> ... </HEAD>):

```
<LINK REL=''stylesheet'' TYPE=''text/css'' HREF=''URL_Hoja.css''>
```

Veamos un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo con hoja de estilo externa </TITLE>
<LINK REL="stylesheet" TYPE="text/css" HREF="ejemplo.css">
</HEAD>

<BODY BGCOLOR=white>

<H2>Prueba de definicion de estilos en una hoja de
estilo externa</H2>

Esto es igual que el ejemplo de definicion de un bloque en
el documento, pero ahora con las definiciones en una hoja
externa. <P>

Como puede verse, la apariencia de esta pagina queda completamente
definida por los estilos que hemos especificado en el bloque STYLE
en la cabecera del documento. Los margenes son mas amplios de lo
habitual, la <B>negrita</B> tiene un tamano y un color fijos, los
trozos de texto en teletipo como <TT<este fragmento</TT> tambien
tienen definida su fuente, tamano y color, y vamos a ver como quedan
las tablas, para finalizar el ejemplo: <P>
```

```
<CENTER>
<TABLE BORDER=1 CELLSPACING=2 CELLPADDING=2>
 <TR> <TH>Cabecera 1</TH> <TH>Cabecera 2</TH> </TR>
 <TR> <TD>Celda (1,1)</TD> <TD>Celda (1,2)</TD> </TR>
</TABLE>
</CENTER>

Si quisieramos que otros documentos tuvieran el mismo estilo, solo
tendriamos que usar la etiqueta LINK para aplicarlos. Esa es la
ventaja de las hojas de estilo externas.

</BODY>
</HTML>
```

El fichero **ejemplo.css** tiene el siguiente contenido:

```
/* Definicion de estilos en un archivo aparte */

/* Estilo para el documento */
BODY {font-family:Verdana,sans-serif;font-size:x-small;
      margin-left:0.25in; margin-right:0.25in}

/* Estilo para la cabecera de nivel 2 */
H2 {font-family:Verdana,sans-serif;font-size:14pt;color:red}

/* Estilos para otras etiquetas */
B, TD {font-family:Verdana,sans-serif;font-size:x-small;
       color:olive}
TH {font-family:Verdana,sans-serif;font-size:x-small;
    color:white;background-color:#0080C0}
PRE, TT, CODE  {font-family:Courier New,Courier;
                font-size:9pt;color:maroon}
```

Esto nos muestra, además, una característica interesante: podemos usar comentarios al estilo de C en las definiciones del estilo. Tal y como vemos, en el fichero css no es necesario especificar la etiqueta <STYLE> ... </STYLE>, basta con ir definiendo los estilos que queramos aplicar.

3 Herencia de estilos

En el conjunto de las etiquetas HTML podemos establecer una jerarquía de etiquetas que contienen a otras, para darnos una relación de herencia. En primer lugar, tendríamos la etiqueta <BODY> ... </BODY>, que hace referencia a **todo** el documento, y podemos considerarla como la etiqueta "padre" de todas las demás etiquetas de formato, puesto que todas ellas se encuentran contenidas en el *cuerpo* (body) del documento.

Después, tenemos las etiquetas de párrafo (<P>...</P>, <DIV>...</DIV>, cabeceras, ...) y etiquetas de *elementos insertados en línea* (<B>...</B>, <I>...</I>, <SPAN>...</SPAN>, ...). Las etiquetas de párrafo serán contenedoras de las etiquetas de elementos insertados en línea (en el sentido que les estamos dando), estableciéndose así una nueva relación "padre-hijo".

Esto es interesante porque la mayoría de los estilos que se definen se heredan, es decir, si definimos un cierto estilo para una etiqueta, este estilo será heredado por las etiquetas "hijas", con lo que no tendremos que volver a definirlo para ellas. Por ejemplo, si definimos un tipo de letra y un color para la fuente para la etiqueta <BODY> ... </BODY>, este estilo será heredado por todas las etiquetas del documento y no tendremos que definirlo para las otras etiquetas.

Sin embargo, si tenemos definido un estilo para una etiqueta "padre", podremos definir un estilo distinto para una etiqueta "hija", es decir, un estilo heredable se hereda a no ser que especifiquemos lo contrario. Se heredarán aquellas características que no pongamos, y se aplicarán aquellas que definamos para la etiqueta que no tenga la etiqueta "padre".

Veamos un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo 1 de herencia </TITLE>
<LINK REL="stylesheet" TYPE="text/css" HREF="ejemplo.css">
</HEAD>

<BODY BGCOLOR=white>
Todo el texto tiene definido el estilo Fuente:Verdana,
Tamano:x-small, Margen izquierdo:0.25in, Margen derecho:0.25in,
<SPAN STYLE="color:red">pero este trozo de linea es de un color
distinto, conservando el resto de propiedades</SPAN>, y eso hace
interesante la herencia y la posibilidad de cambiar en partes
concretas los estilos heredados.
</BODY>
</HTML>
```

Otro tema a comentar aquí es la posibilidad de definir, en lugar de un estilo para una etiqueta, sin más, es definir un *estilo en función del contexto*. Por ejemplo, es posible que sólamente queramos que el texto en negrita sea de color verde cuando se encuentre en una celda de una tabla, o que sea de color púrpura cuando forme parte de una lista. Esto lo definiríamos como sigue:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo de estilos segun el contexto </TITLE>
<LINK REL="stylesheet" TYPE="text/css" HREF="ejemplo.css">
<STYLE TYPE="text/css">
<!--
  TD B {color:green}
  UL B {color:purple}
//-->
</STYLE>
</HEAD>

<BODY BGCOLOR=white>
En este ejemplo, seguimos con los estilos de la hoja externa, pero
vamos a comprobar que se verifican los estilos definidos en
funcion del contexto: la negrita de una celda cualquiera de una
tabla debe ser de color verde, y la negrita de una lista debe
ser de color purpura. <P>

<UL TYPE=DISC>
 <LI>Un elemento cualquiera</LI>
 <LI>Un elemento con una palabra en <B>negrita</B></LI>
 <LI>Otro elemento cualquiera</LI>
</UL><P>

<CENTER>
<TABLE BORDER=1 CELLSPACING=2 CELLPADDING=2>
 <TR><TH>Cabecera 1</TH><TH>Cabecera 2</TH></TR>
 <TR><TD>Celda (1,1)</TD><TD>Celda (1,2)</TD></TR>
 <TR><TD><B>Celda (2,1) en negrita</B></TD><TD>Celda (2,2)</TD></TR>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

(Notar que las etiquetas **no** están separadas por comas).

4 Jerarquía de definición de estilos. Clases. Pseudo-clases

Ya hemos visto varias formas de definir estilos (bien mediante una hoja externa, un bloque de estilos o directamente en una etiqueta). Estas posibilidades de definición inducen una jerarquía que decide qué estilo se aplicará a una etiqueta en el caso de que haya varias posibilidades a elegir para ella.

Supongamos, por ejemplo, que tenemos en un documento una hoja externa que define un estilo para <P>, en el mismo documento un bloque de estilo que define otro estilo para <P>, y luego, para una <P> concreta del documento, se define otro estilo.

¿Cuál de todos es el que se debe aplicar? Con todo lo que hemos visto, ya sabemos que será siempre el más restrictivo, en el sentido del ámbito en el que se define el estilo. Así, un bloque de estilo definido en el documento es más restrictivo que una hoja externa, y una definición en una etiqueta concreta es más restrictiva que un bloque de estilo. Por tanto, a nuestra <P> concreta se le aplicará el estilo definido localmente para ella, resolviendo así el conflicto de definición de varios estilos a la vez para una misma etiqueta.

Esta posibilidad no es en absoluto extraña; es normal querer definir unos estilos globales en hojas externas que homogeneicen el aspecto de nuestras páginas, y luego, en una página concreta querer variar el estilo en alguna etiqueta concreta. Como ya sabemos, esto podemos hacerlo definiendo el estilo localmente en **esa** etiqueta.

Pero también puede suceder que esta definición de un estilo concreto queramos aplicarla a otra etiqueta. Lo primero que se nos ocurre es copiar esta definición del estilo a la otra etiqueta en la que también queremos aplicarlo. Sin embargo, este estilo concreto que queremos aplicar a algunas etiquetas concretas puede ser definido en un bloque de estilo global o, incluso, en la hoja externa, y aplicarlo, gracias a un identificador, a las etiquetas concretas en las que queramos que se aplique dicho estilo. Con este fin se definen las **clases**.

Una *clase* es una definición de un estilo que en principio no está asociado a alguna etiqueta HTML, pero que podemos asociar, en el documento, a etiquetas concretas.

Para ello, en primer lugar definimos la clase (en el bloque de estilos o en la hoja externa) como un estilo más, de esta forma:

```
.Nombre_de_la_Clase {propiedad1:valor;...;propiedadN:valor}
```

es decir, escribiendo un punto seguido del nombre que le queramos dar a la clase, y definiendo el estilo como lo definimos para cualquier otra etiqueta: pares `propiedad:valor` separados por punto y coma y encerrados entre llaves. Además, podremos definir cuantas clases necesitemos.

Ahora, para aplicar el estilo de una clase a una etiqueta concreta, utilizaremos el parámetro CLASS como sigue:

`<etiqueta CLASS=''Nombre_de_la_Clase''> ... </etiqueta>`

donde `Nombre_de_la_Clase` es el nombre que le hemos dado a la clase, **sin el punto**.

Por ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo de uso de clases </TITLE>
<STYLE TYPE="text/css">
<!--
 BODY   {font-family:Verdana,sans-serif;font-size:x-small}
 P,A,B  {color:red}
 .BAzul {color:blue}
//-->
</STYLE>
</HEAD>

<BODY BGCOLOR=white>

En este ejemplo vamos a ver como se aplican las clases.
Por ejemplo, <B>esta negrita</B> utiliza el estilo definido
en el bloque, pero <B CLASS="BAzul">esta otra negrita</B>
tiene un color distinto. Y no solo podemos usar la clase
para la negrita. Por ejemplo, <SPAN CLASS="BAzul">este trozo
de linea tambien utiliza la clase para su estilo particular</SPAN>.

</BODY>
</HTML>
```

Vamos ahora a estudiar el atributo ID de una etiqueta HTML en relación a la definición de estilos. Cualquier etiqueta HTML puede tener como parámetro la etiqueta ID seguida de un nombre, por ejemplo:

`<etiqueta ID=''NombreReferencia''> ... </etiqueta>`

Este "NombreReferencia" debe ser único en el documento HTML (es decir, no debe haber dos etiquetas con el mismo ID), puesto que nos servirá para tratarla (si lo necesitamos) desde JavaScript, y por esto no debe haber confusión con el nombre como referencia.

Para definir un estilo mediante un ID, usaremos la siguiente notación (en un bloque de estilo o en la hoja externa):

`#Nombre_del_ID {propiedad1:valor;...;propiedadN:valor}`

es decir, escribiendo # seguido del nombre que le queramos dar al ID, y definiendo el estilo como ya sabemos: pares `propiedad:valor` separados por punto y coma y encerrados entre llaves. Podremos definir todos los ID que queramos, **pero cada ID sólo debe ser asociado a una única etiqueta concreta** de la siguiente forma:

```
<etiqueta ID=''Nombre_del_ID''> ... </etiqueta>
```

Así identificaremos de forma unívoca a esa etiqueta concreta, asignándole la definición del estilo hecha en el bloque o en la hoja para ese ID, y además nos permitirá tratarlo (por ejemplo, cambiando algunas características del estilo definido) desde JavaScript, que usará ese identificador para saber sobre quién ha de actuar, suponiendo que quisiéramos hacerlo.

Por último, en este capítulo vamos a hablar de *pseudoclases*.

Hay ocasiones en las que HTML da a algunas etiquetas un estilo propio: por ejemplo, los enlaces aparecen por defecto de otro color y subrayados. Estos elementos son las pseudoclases. Por ahora, sólo están definidas para la etiqueta <A>.

La forma de definir un estilo para una pseudoclase es la siguiente:

```
etiqueta:pseudoclase {propiedad1:valor;...;propiedadN:valor}
```

y las pseudoclases de que disponemos son:

- `link`. Nos dice el estilo de un enlace que no ha sido visitado.
- `visited`. Nos dice el estilo de un enlace que ha sido visitado.
- `active`. Nos dice el estilo de un enlace que está siendo pulsado.
- `hover`. Nos dice el estilo de un enlace sobre el que está pasando el ratón.

Por ejemplo, en esta web todos los enlaces aparecen sin subrayar, esto lo consigo definiendo los estilos:

```
A:link,A:visited,A:active {text-decoration:none}
```

Las pseudoclases pueden usarse de forma conjunta con las clases, para aplicar ese estilo sólo en casos concretos, siguiendo la notación:

```
A.NombreClase:pseudoclase
```

y también se pueden usar en función del contexto.

5 Propiedades de las hojas de estilo

Ahora que ya hemos visto cómo se definen estilos en un documento HTML, así como todas las posibilidades en cuanto a jerarquías, clases, ..., nos vamos a centrar en qué es lo que podemos poner en cada una de esas parejas `propiedad:valor` que decíamos que definen un estilo.

Para facilitar su identificación, los vamos a dividir en las siguientes categorías:

1. Propiedades de formato de bloque
2. Propiedades de las fuentes
3. Propiedades de texto
4. Propiedades de color y fondo
5. Propiedades de clasificación (listas)

Hay propiedades en las que necesitaremos especificar alguna longitud (por ejemplo, en los márgenes). Para ello, usaremos esta notación:

[-]NN*tipo*

– es el signo, y es opcional (es lo que quieren decir los corchetes, que se trata de algo opcional, los corchetes no hay que ponerlos). NN es la cantidad, y *tipo* es la magnitud. Esta última puede ser relativa o absoluta. Las magnitudes relativas son em (el alto de la M mayúscula), ex (la mitad de la altura de la M mayúscula, que viene a ser aproximadamente la altura de la x minúscula), px (píxel). Las magnitudes absolutas son pt (puntos), pc(picas), in (inches, es decir, pulgadas), mm (milímetros), cm (centímetros).

Hay otras propiedades en las que tendremos que especificar un color; para ello hay tres posibilidades: escribiéndolo de la misma forma que en HTML, con la notación #RRGGBB, siendo RR, GG, BB los valores en hexadecimal de las componentes roja, verde y azul del color, usando algún nombre predefinido, o usando la función rgb(R,G,B), donde R, G, B son los valores en decimal de las componentes roja, verde y azul del color.

A continuación mostramos algunos de los nombres predefinidos para los colores:

aqua, black, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow, blue

Un último detalle a comentar antes de pasar al estudio de las propiedades y sus posibles valores, es el siguiente: desde el punto de vista de las hojas de estilo, existen tres tipos de elementos HTML: de **bloque** (que son los que hacen empezar línea nueva, como <P>, las cabeceras, ...), **incrustados en línea** (que no alteran la línea en la que se encuentran, como <B>, <I>, ...) y de **lista** (que son los elementos de una lista delimitados por <LI>).

Todo elemento de bloque se considera rodeado por una caja, con propiedades de márgenes, borde, padding y fondo. Además, la caja que lo rodea tiene un cierto ancho, y una cierta alineación con respecto al documento.

5.1 Propiedades de formato de bloque

Nombre	margin-left, margin-right, margin-top, margin-bottom, margin	padding-left, padding-right, padding-top, padding-bottom, padding	border-left-width, border-right-width, border-top-width, border-bottom-width, border-width	border-style
Valores posibles	longitud, porcentaje, `auto`	longitud, porcentaje	longitud	`none, solid, double, inset, outset, groove, ridge`
Valor por defecto	0	0	0	none
Aplica a	todos	todos	todos	todos
Heredado	No	No	No	No
Valor porcentual	Relativo al ancho del padre	Relativo al ancho del padre	–	–

Nombre	border-color	width	float	clear
Valores posibles	`none`, color	longitud, porcentaje, `auto`	`left, right, none`	`none, left, right, both`
Valor por defecto	none	auto	none	none
Aplica a	todos	elementos a nivel de bloque	todos	todos
Heredado	No	No	No	No
Valor porcentual	–	Relativo al ancho del padre	–	–

Aclaraciones sobre algunos valores:

Longitud: Debe ser un número seguido de una unidad de medida, tal y como se comentaba unos párrafos más arriba. Por ejemplo, un valor posible es 12pt, ó 0.2in. Porcentaje: Fija el tamaño en valor porcentual respecto al padre.

5.2 Propiedades de las fuentes

Nombre	font-size	font-family	font-weight	font-style
Valores posibles	absoluto, relativo, porcentaje, longitud	*familia*	`bolder`, `lighter`, 100-900	`normal`, `bold`, `italic`
Valor por defecto	medium	user preferences	normal	normal
Aplica a	todos	todos	todos	todos
Heredado	Sí	Sí	Sí	Sí
Valor porcentual	Relativo al padre	–	–	–

Aclaraciones sobre algunos valores:

En `font-size` hemos dicho que puede tomar un valor absoluto; los posibles valores absolutos son: `xx-large`, `x-large`, `large`, `medium`, `small`, `x-small`, `xx-small`.

En `font-family` hemos dicho que el valor será *family*; esto hace referencia a una lista de nombres de familias de fuentes, separadas por comas. El navegador buscará por orden cada una de las fuentes especificadas, y usará la primera que encuentre instalada en el sistema. Por ejemplo, podemos poner `font-family:Verdana,Arial`; esto significará que el navegador buscará la fuente Verdana, y si no está instalada en el ordenador de quien ve la página, buscará la fuente Arial, por ser la siguiente en la lista. Así continuaríamos si hubiera más fuentes en la lista. Si no encuentra ninguna, usará la fuente por defecto. Es importante saber que existen unas familias de fuentes genéricas que están en todos los ordenadores, a fin de ponerlas como última alternativa en la lista. Estas familias son: `serif`, `sans-serif`, `monospace`, `cursive`, `fantasy`.

En `font-weight` teníamos las posibilidades `bolder`, `lighter` y 100-900. Las dos primeras son relativas al valor actual. 100-900 quiere decir decir que podemos especificar un valor numérico entre 100 y 900. 100 será el más ligero y 900 el más pesado.

5.3 Propiedades de texto

Nombre	line-height	text-decoration	text-transform	text-align	text-indent
Valores posibles	número, longitud, porcentaje, `normal`	`none`, `underline`, `line-through`, `blink`	`capitalize`, `uppercase`, `lowercase`	`left`, `right`, `center`, `justify`, `none`	longitud, porcentaje
Valor por defecto	normal	none	none	left	0
Aplica a	elementos a nivel de bloque	todos	todos	elementos a nivel de bloque	elementos a nivel de bloque
Heredado	Sí	No	Sí	Sí	Sí
Valor porcentual	Relativo al tamaño de la fuente	–	–	–	Relativo al ancho del elemento padre

Aclaraciones sobre algunos valores:

`line-height` sólo se aplica a elementos a nivel de bloque. Si se indica un número o porcentaje, esta distancia será el producto de este número por el tamaño de la fuente del elemento actual. La diferencia está en que, al indicar un número, los elementos hijos heredan el factor, mientras que, en otro caso, heredan el tamaño de la separación.

5.4 Propiedades de color y fondo

Nombre	color	background-color	background-image
Valores posibles	color	color	URL entre paréntesis
Valor por defecto	black	ninguno	ninguno
Aplica a	todos	todos	todos
Heredado	Sí	No	No
Valor porcentual	–	–	–

5.5 Propiedades de clasificación (listas)

Nombre	display	list-style-type	white-space
Valores posibles	`block`, `inline`, `list-item`, `none`	`disc`, `circle`, `square`, `decimal`, `upper-roman`, `lower-roman`, `upper-alpha`, `lower-alpha`	`normal`, `pre`, `none`
Valor por defecto	según HTML	disc	según HTML
Aplica a	todos	elementos cuya propiedad display es list-item	elementos a nivel de bloque
Heredado	No	Sí	Sí
Valor porcentual	–	–	–

Aclaraciones sobre algunos valores:

En display se indica si un elemento es de nivel de bloque (block), incrustado (inline) o de nivel de lista (list-item). Si se ajusta el valor a none, el elemento no será mostrado, lo que se extiende a sus "hijos" y la caja que lo rodea. Los elementos de bloque no responden si se les ajusta como inline.

Son muchas las posibilidades que tenemos para variar propiedades, pero al final terminaremos aprendiéndonos las seis o siete que más usemos. Es un interesante ejercicio practicar aplicando los posibles estilos en trozos del documento para ir adquiriendo más práctica. Aprendida la técnica de las hojas de estilo, en el siguiente bloque de capítulos nos centraremos en los layers.

6 Layers: Creación

Como ya se comentó en la introducción a DHTML, una de las motivaciones por las que surgió esta nueva especificación fue para conseguir que los diseñadores de webs tuvieran el mayor control posible sobre su trabajo, evitando los "trucos" habituales de insertar espacios, tablas con bordes invisibles o imágenes GIF de un píxel que se ajustan al tamaño necesitado.

En la primera parte del curso se han estudiado las formas de definir estilos que controlan por completo la presentación de los distintos elementos HTML, pero aún queda un problema por resolver, que estudiamos en este bloque: ¿cómo ponemos la información exactamente en el sitio que queremos?

Es para este fin para lo que existen las **capas**.

Una *capa* es un bloque HTML (un párrafo, varias líneas, ...) sobre el que se define un estilo especial que nos indica la posición que va a tomar en el documento y al que se le pueden dar unas características propias que nos dirán si es o no visible, si está superpuesto a otros elementos, ...

Para crear una capa existen dos posibilidades, mediante un estilo CSSP (CSS Positioning; posicionamiento mediante hojas de estilo) o mediante la etiqueta `<LAYER> ... </LAYER>`. El inconveniente de este último método es que se trata de una solución propietaria que Netscape adoptó a partir de la versión 4 de su navegador, y por tanto no funciona con Explorer, mientras que si usamos CSSP el resultado obtenido podrá verse en cualquier navegador con soporte para CSSP.

Para definir un layer mediante CSSP usaremos la propiedad `position` del parámetro STYLE siguiendo la sintaxis vista en los capítulos de CSS:

```
<etiqueta STYLE=''position:valor;propiedad1:valor;...; propiedadN:valor''>
... </etiqueta>
```

donde **valor**, para **position**, puede ser:

- `absolute`: Indica que su posición será absoluta respecto de un origen, la esquina superior izquierda de la capa contenedora.

- `relative`: Su posicionamiento sigue el flujo del HTML, aunque podremos darle unas coordenadas para situarlo de forma relativa a este flujo.

La propiedad `position` puede aplicarse a cualquier etiqueta HTML, aunque lo más normal es usarla en las siguientes: `<DIV>`, `<SPAN>`, `<P>`.

6.1 Diferencias entre ambos tipos de posicionamiento

6.1.1 Posicionamiento absoluto

Nos permite colocar un elemento de forma absoluta respecto a unas coordenadas fijas (las de la capa contenedora, que puede ser el navegador u otro elemento que lo contenga). Al estar situado de forma absoluta, podrá solaparse con otros elementos. Veamos un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo de capas </TITLE>
<STYLE TYPE="text/css">
<!--
 BODY   {font-family:Verdana,sans-serif;font-size:x-small}
//-->
</STYLE>
</HEAD>

<BODY BGCOLOR=white>

<DIV STYLE="position:absolute;left:50px;top:10px">
Aqui tenemos un texto dentro de una capa, al que le ha
sucedido una pequena desgracia: otro texto, que
proviene de una capa desalmada, se le ha superpuesto,
con lo que no se le puede leer completamente.

<DIV STYLE="position:absolute;left:150px;top:20px;
            color:red;font-size:11pt;font-weight:bold">
Capa malvada
</DIV>
</DIV>

</BODY>
</HTML>
```

6.1.2 Posicionamiento relativo

Coloca un elemento siguiendo el flujo natural del HTML. Esto nos permite poner un elemento respecto de la posición de elemento anterior en el flujo del HTML: las coordenadas que le demos lo situarán tomando como origen las coordenadas del elemento anterior. Por ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo de capas </TITLE>
<STYLE TYPE="text/css">
<!--
 BODY   {font-family:Verdana,sans-serif;font-size:x-small}
//-->
</STYLE>
</HEAD>

<BODY BGCOLOR=white>

Esta frase tiene una
<SPAN STYLE="position:relative;top:10px">palabra</SPAN>
mas baja que las demas.

</BODY>
</HTML>
```

Es especialmente importante asignar nombres a las capas mediante el parámetro ID de las etiquetas (cosa que yo no estoy haciendo en los ejemplos O:-)), pues éstas tienen bastantes propiedades que pueden modificarse desde JavaScript y así conseguir efectos interesantes. Para poder conseguir estos efectos, JavaScript debe saber a qué capa nos estamos refiriendo, y para ello lo más cómodo es asignarles un nombre vía ID. Además, podemos hacer, al igual que vimos en el bloque de CSS, lo siguiente: definir un estilo con ID que incluya position, y luego asignárselo a alguna etiqueta concreta. Esto le asignará el ID y el estilo que la convierte en capa.

Y eso es todo en cuanto a definición. En el siguiente capítulo estudiamos qué características podemos añadir a nuestras capas.

7 Propiedades de las capas

Las capas añaden unas propiedades adicionales que se deben especificar en el parámetro STYLE de las etiquetas. Las primeras propiedades que vemos especifican la posición y el tamaño de las capas:

- `left`: Separación izquierda de la capa con respecto a su origen.
- `top`: Separación superior de la capa con respecto a su origen.
- `width`: Especificamos el ancho de la capa.
- `height`: Especificamos la altura de la capa.

Como valor podemos poner medidas absolutas o porcentajes.

Las propiedades que vamos a estudiar ahora tienen que ver con la siguiente cuestión: ¿qué sucede si el tamaño del contenido de la capa excede el tamaño del contenedor? Para dar una respuesta a este problema, CSSP nos da la propiedad 'clip'.

Por defecto, si no la ponemos, a pesar de haber dado unos límites a la capa en cuanto a ancho y alto (con width y height), se mostrará el contenido completo del layer, como podemos ver en este ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo de capas </TITLE>
<STYLE TYPE="text/css">
<!--
 BODY   {font-family:Verdana,sans-serif;font-size:x-small}
//-->
</STYLE>
</HEAD>

<BODY BGCOLOR=white>

<DIV STYLE="position:absolute;left:100px;top:50px;
     width:100px;height:100px;text-align:justify">
Aqui tenemos una capa suficientemente estrecha como
para observar que sucede si no ponemos nada al
respecto de 'clip' o no 'clip'. En el siguiente
ejemplo pondremos casi el mismo texto pero con 'clip'
y veremos la diferencia.
```

```
</DIV>

</BODY>
</HTML>
```

Ahora veamos qué podemos hacer con clip: esta propiedad nos permite fijar los límites del área de visibilidad del contenido de la capa. Todo lo que quede fuera de esta área se verá cortado. Esta área de visibilidad se fija dando valores separados por comas que especificarán, en este orden, el límite superior, el límite derecho, el límite inferior y el límite izquierdo, de la siguiente forma:

```
clip:rect(superior,derecho,inferior,izquierdo)
```

Por ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo de capas </TITLE>
<STYLE TYPE="text/css">
<!--
 BODY   {font-family:Verdana,sans-serif;font-size:x-small}
//-->
</STYLE>
</HEAD>

<BODY BGCOLOR=white>

<DIV STYLE="position:absolute;left:100px;top:50px;
     width:100px;height:100px;text-align:justify";
     clip:rect(7,97,139,3)">
Aqui tenemos una capa suficientemente estrecha como
para observar que sucede si ponemos algo al
respecto de 'clip'. Es la continuacion
del ejemplo anterior, casi el mismo texto pero
ahora con 'clip'.
</DIV>

</BODY>
</HTML>
```

Vamos a finalizar el capítulo con dos propiedades que nos van a permitir, por una parte, apilar las capas en un orden determinado, de forma que sea A la que esté sobre B si eso es

lo que nos interesa, y por otra parte la propiedad que nos va a dar la posibilidad de hacer invisible o visible una cierta capa según nos convenga.

En un documento, podemos poner tantas capas como necesitemos. Estas capas se irán apilando de acuerdo a un cierto orden, que es recogido en la propiedad 'z-index'. Esta propiedad toma como valor un entero que puede ser 0,1,2,..., y que nos indica que capa está sobre qué otra capa, de acuerdo a lo siguiente: la capa 0 es la que está más abajo de todas, estando todas las demás sobre ella. La capa que va a continuación será la 1, que tiene sobre ella a las capas con z-index de 2 en adelante, luego está la capa con z-index 2, que tiene sobre ella a las capas con z-index de 3 en adelante... y así sucesivamente.

Podemos asignar este valor a la hora de crear la capa, o podemos no poner nada. Si no ponemos nada para z-index, es el propio navegador quien va asignando este orden, empezando con el 0, después el 1, el 2, etc.

Por otro lado, tenemos la propiedad 'visible', que nos va a decir si la capa es visible o no. Puede tomar los siguientes valores:

- `visible`: La capa es visible.
- `hidden`: La capa es invisible.

Finalizaremos en el siguiente capítulo el tema de las capas, tratando la etiqueta <LAYER> que incorporó Netscape, a pesar de tratarse de una solución propietaria que sólo funciona en estos navegadores a partir de su versión 4.

8 La etiqueta <LAYER> de Netscape

Netscape propuso, con la aparición de la versión 4 de su navegador, el uso de la etiqueta pareada <LAYER> ... </LAYER> para implementar el concepto de capa, en lugar de definir un estilo que tuviera la propiedad position para que ese elemento fuese una capa. Todo lo que esté encerrado entre esa etiqueta será la capa, y también proveyó de una <NOLAYER> ... </NOLAYER>, con lo que este navegador ignoraría todo lo que hubiera encerrado entre esas etiquetas, pensadas para ofrecer una alternativa a los navegadores que no soporten la etiqueta <LAYER>.

La forma de definir las propiedades de la capa (propiedades que ya estudiamos en el capítulo anterior) es la forma usual de añadir parámetros dentro de una etiqueta HTML: PARAMETRO_1=VALOR ... PARAMETRO_N=VALOR, por ejemplo:

```
<LAYER ID="MiLayer" TOP="100" LEFT="50">
 Aqui dentro va lo que quiera que lleve la capa
</LAYER>
```

También podemos, como con cualquier otra etiqueta HTML, asignarle un estilo mediante CLASS. Por ejemplo, si tenemos definido en alguna parte un estilo (una hoja externa, un bloque de estilos) cuyo nombre es 'EstiloNumberOne', podremos aplicarlo a la capa definida con LAYER simplemente añadiendo el parámetro 'CLASS="EstiloNumberOne"', es decir:

```
<LAYER ID="MiLayer" CLASS="EstiloNumberOne" TOP="100" LEFT="50">
 Aqui dentro va (de nuevo) lo que quiera que lleve la capa
</LAYER>
```

Sin embargo, las capas definidas con esta etiqueta son equivalentes a las que definíamos con CSSP usando 'position:absolute', ¿cómo definimos una capa con posicionamiento relativo? La solución que ha adoptado Netscape al respecto es otra etiqueta pareada: <ILAYER> ... </ILAYER>. Todo lo que pongamos dentro de esa etiqueta será una capa, pero cuyo posicionamiento es relativo.

¿Cómo especificamos el área de visibilidad que definíamos con 'clip', usando CSSP? Con el parámetro CLIP, dentro de la etiqueta, y dándole los valores, entre comillas y separados por comas, del límite superior, límite izquierdo, límite inferior y límite derecho. Por ejemplo:

```
<LAYER ID="MiLayer" TOP="100" LEFT="50" CLIP="0,0,150,100">
 Aqui dentro va lo que quiera que lleve la capa
</LAYER>
```

Equivaldría a 'clip:rect(0,150,0,100)' del CSSP.

Para especificar el valor de z-index, tenemos un parámetro Z-INDEX. Además, con esta etiqueta podemos decirle, gracias a otros dos parámetros extra, qué capa tiene por encima y qué capa tiene por debajo. Esto lo hacemos con ABOVE (sobre) y BELOW (bajo), por ejemplo:

```
<LAYER ID="MiLayer" TOP="100" LEFT="50" ABOVE="OtraCapa">
 Aqui dentro va lo que quiera que lleve la capa
</LAYER>
<LAYER ID="MiLayer" TOP="100" LEFT="50">
 Esta capa esta por debajo de la anterior
</LAYER>
```

Las capas que proporciona Netscape tienen un parámetro adicional que hace su uso muy interesante y que, por desgracia, no se ha adoptado más que en este navegador; se trata del parámetro SRC, en el que podemos especificar el nombre de un fichero con el contenido. Esto sería muy útil en páginas escritas en varios idiomas, en páginas con noticias renovables con una misma estructura... porque así, únicamente habría que cambiar el fichero a incluir en SRC, y una vez diseñada la estructura no habría que meterse en el documento a buscar dónde hacer los cambios, únicamente en esos pequeños ficheros externos que podríamos incluir posteriormente.

Finalizamos el capítulo con un ejemplo que muestra cómo definir capas con Netscape, y cómo esta definición hace luego posible interactuar fácilmente con JavaScript para cambiar las propiedades de la capa. En este ejemplo, desplazamos la capa. Sólo podrán verlo los que usen Netscape 4.*. Lo siento por los demás usuarios :-(

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE> Ejemplo etiqueta LAYER </TITLE>
<STYLE TYPE="text/css">
<!--
 BODY   {font-family:Verdana,sans-serif;font-size:x-small}
//-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function MoverLayer(direccion) {
  switch(direccion) {
    case "N": document.Prueba.top -= 5; break;
    case "S": document.Prueba.top += 5; break;
    case "E": document.Prueba.left += 5; break;
```

```
    case "O": document.Prueba.left -= 5; break;
    }
  }
//-->
</SCRIPT>
</HEAD>

<BODY BGCOLOR=white>

<LAYER ID="Prueba" LEFT="50" TOP="100" WIDTH="300">
Aqui tenemos una capa definida segun Netscape. Pulsa los
botones para moverla.<P>
<CENTER><IMG SRC="gato_peq.gif"></CENTER>
</LAYER>

<FORM NAME="Botones">
<TABLE BORDER=0>
<TR><TD COLSPAN="3" ALIGN=CENTER>
<INPUT TYPE="BUTTON" VALUE="Arriba" onClick="MoverLayer('N');">
</TD></TR>
<TR><TD>
<INPUT TYPE=BUTTON VALUE="Izquierda" onClick="MoverLayer('O');">
</TD><TD>
<INPUT TYPE=BUTTON VALUE="Abajo" onClick="MoverLayer('S');">
</TD><TD>
<INPUT TYPE=BUTTON VALUE="Derecha" onClick="MoverLayer('E');">
</TD></TR>
</TABLE>
</FORM>

</BODY>
</HTML>
```