CSS

avanzado

Javier Eguíluz Pérez

Sobre este libro...

- **Esta versión impresa se creó el 2 de enero de 2009 y todavía está incompleta**. La versión más actualizada de los contenidos de este libro se puede encontrar en http://www.librosweb.es/css_avanzado

- Si quieres aportar sugerencias, comentarios, críticas o informar sobre errores, puedes enviarnos un mensaje a **contacto@librosweb.es**

Capítulo 1. Técnicas imprescindibles

El estándar CSS 2.1 incluye más de 100 propiedades de todo tipo para diseñar el aspecto de las páginas HTML. No obstante, los diseños web más actuales muestran recursos gráficos que no se pueden realizar con CSS, como sombras, transparencias, esquinas redondeadas y tipografía avanzada. Por ese motivo, es preciso que los diseñadores web profesionales conozcan las técnicas imprescindibles para crear diseños web avanzados.

En las próximas secciones se muestran las siguientes técnicas imprescindibles:

- Propiedades *shorthand* para crear hojas de estilos concisas.
- La propiedad `hasLayout` de Internet Explorer, imprescindible para solucionar muchos errores de ese navegador.
- *Limpiar floats*, para trabajar correctamente con los elementos posicionados de forma flotante.
- Cómo crear elementos de la misma altura, imprescindible para el *layout* o estructura de las páginas.
- Sombras, transparencias y esquinas redondeadas, que no se pueden crear con CSS 2.1.
- Sustitución de texto por imágenes y por Flash, para utilizar cualquier tipografía en el diseño de las páginas.
- Rollovers y *sprites CSS* para mejorar el tiempo de respuesta de las páginas.
- Técnicas para trabajar con el texto y la tipografía.

1.1. Propiedades shorthand

Algunas propiedades del estándar CSS 2.1 son especiales, ya que permiten establecer simultáneamente el valor de varias propiedades diferentes. Este tipo de propiedades se denominan *"propiedades shorthand"* y todos los diseñadores web profesionales las utilizan.

La gran ventaja de las *propiedades shorthand* es que permiten crear hojas de estilos mucho más concisas y por tanto, mucho más fáciles de leer. A continuación se incluye a modo de referencia la definición formal de las seis *propiedades shorthand* disponibles en el estándar CSS 2.1.

font	Tipografía
Valores	`( ( <font-style> || <font-variant> || <font-weight> )? <font-size> ( / <line-height> )? <font-family> ) | caption | icon | menu | message-box | small-caption | status-bar | inherit`
Se aplica a	Todos los elementos
Valor inicial	-
Descripción	Permite indicar de forma directa todas las propiedades de la tipografía de un texto

margin	Margen
Valores	`( <medida> | <porcentaje> | auto ) {1, 4} | inherit`
Se aplica a	Todos los elementos salvo algunos casos especiales de elementos mostrados como tablas
Valor inicial	-
Descripción	Establece de forma directa todos los márgenes de un elemento

padding	Relleno
Valores	`( <medida> | <porcentaje> ){1, 4} | inherit`
Se aplica a	Todos los elementos excepto algunos elementos de tablas como grupos de cabeceras y grupos de pies de tabla
Valor inicial	-
Descripción	Establece de forma directa todos los rellenos de los elementos

border	Estilo completo de todos los bordes
Valores	`( <border-width> || <border-color> || <border-style> ) | inherit`
Se aplica a	Todos los elementos
Valor inicial	-
Descripción	Establece el estilo completo de todos los bordes de los elementos

background	Fondo de un elemento
Valores	`( <background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position> ) | inherit`
Se aplica a	Todos los elementos
Valor inicial	-
Descripción	Establece todas las propiedades del fondo de un elemento

list-style	Estilo de una lista
Valores	`( <list-style-type> || <list-style-position> || <list-style-image> ) | inherit`
Se aplica a	Elementos de una lista
Valor inicial	-
Descripción	Propiedad que permite establecer de forma simultanea todas las opciones de una lista

Si se considera la siguiente hoja de estilos:

```
p {
  font-style: normal;
  font-variant: small-caps;
  font-weight: bold;
  font-size: 1.5em;
  line-height: 1.5;
  font-family: Arial, sans-serif;
}
div {
  margin-top: 5px;
  margin-right: 10px;
  margin-bottom: 5px;
  margin-left: 10px;
  padding-top: 3px;
  padding-right: 5px;
  padding-bottom: 10px;
  padding-left: 7px;
}
h1 {
  background-color: #FFFFFF;
  background-image: url("imagenes/icono.png");
  background-repeat: no-repeat;
  background-position: 10px 5px;
}
```

Utilizando las *propiedades shorthand* es posible convertir las 24 líneas que ocupa la hoja de estilos anterior en sólo 10 líneas, manteniendo los mismos estilos:

```
p {
  font: normal small-caps bold 1.5em/1.5 Arial, sans-serif;
}
div {
  margin: 5px 10px;
  padding: 3px 5px 10px 7px;
}
h1 {
  background: #FFF url("imagenes/icono.png") no-repeat 10px 5px;
}
```

1.2. La propiedad hasLayout de Internet Explorer

El navegador Internet Explorer 7 y sus versiones anteriores incluyen decenas de errores relacionados con CSS. La mayoría de esos errores se pueden solucionar con trucos y técnicas que aprovechan otros errores o características del navegador. Además, muchos errores se solucionan gracias a la propiedad `hasLayout` de Internet Explorer.

En efecto, muchas soluciones de los errores de Internet Explorer consisten en *"forzar a un elemento a que tenga un layout"*. El motivo es que para mostrar los elementos de una página, el navegador Internet Explorer divide a todos los elementos en dos grupos:

- Los elementos cuyo tamaño y posición dependen de su elemento contenedor.
- Los elementos que establecen su propio tamaño y posición.

La mayoría de elementos de una página son del primer tipo, ya que sus elementos contenedores (normalmente el elemento `<body>`) determinan su tamaño y posición. Los elementos del segundo tipo son los que Internet Explorer considera que tienen un *layout*.

Los elementos HTML que por defecto tienen un *layout* en Internet Explorer son:

- `<html>`, `<body>`
- `<table>`, `<tr>`, `<th>`, `<td>`
- `<img>`
- `<hr>`
- `<input>`, `<button>`, `<select>`, `<textarea>`, `<fieldset>`, `<legend>`
- `<iframe>`, `<embed>`, `<object>`, `<applet>`
- `<marquee>`

No obstante, algunas propiedades CSS provocan que el elemento tenga un *layout* y por tanto, activan la propiedad `hasLayout`. La siguiente tabla muestra todas las propiedades CSS y valores que hacen que un elemento tenga un *layout*:

Propiedad	**Valores que activan la propiedad hasLayout**	**Comentarios**
`position`	`absolute`, `fixed`	`fixed` sólo en Internet Explorer 7
`float`	`left`, `right`	
`display`	`inline-block`	
`width`	Cualquier valor que no sea `auto`	
`min-width`	Cualquier valor	Sólo en Internet Explorer 7
`max-width`	Cualquier valor	Sólo en Internet Explorer 7
`height`	Cualquier valor que no sea `auto`	
`min-height`	Cualquier valor	Sólo en Internet Explorer 7
`max-height`	Cualquier valor	Sólo en Internet Explorer 7
`zoom`	Cualquier valor que no sea `normal`	
`writing-mode`	`tb-rl`	
`overflow`	`hidden`, `scroll`, `auto`	Sólo en Internet Explorer 7

Las propiedades `zoom` y `writing-mode` no se definen en ningún estándar de CSS porque son propietarias del navegador Internet Explorer. Si se utilizan estas dos propiedades, la hoja de estilos no pasa satisfactoriamente el proceso de validación.

Para quitar el *layout* a un elemento, es necesario establecer el valor por defecto de todas las propiedades de la tabla anterior que hayan sido modificadas:

- `width: auto`, `height: auto`
- `max-width: auto`, `min-width: auto`

- `position: static`
- `float: none`
- `overflow: visible`
- `zoom: normal`
- `writing-mode: lr-tb`

Utilizando alguna de las propiedades CSS anteriores se han ideado numerosas soluciones para *forzar a que un elemento tenga un layout*. Uno de los trucos más conocidos desde hace muchos años es el famoso *Holly hack* (http://www.communitymx.com/content/article.cfm?page=2&cid=C37E0) que soluciona un problema con un elemento posicionado de forma flotante aplicando la siguiente regla:

```
.selector {
  height: 1%;
}
```

En el navegador Internet Explorer 6 se puede utilizar el truco del guión bajo y la propiedad `height` para forzar a un elemento a que tenga *layout*:

```
.selector {
  _height: 1%;
}
```

En Internet Explorer 7 se puede utilizar la propiedad `min-height` utilizando cualquier valor, incluso el `0`:

```
.selector {
  min-height: 0;
}
```

Otra propiedad muy utilizada en Internet Explorer 7 es `zoom`, aunque no es una propiedad estándar de CSS:

```
.selector {
  zoom: 1;
}
```

En las próximas secciones se presentan algunas técnicas que requieren forzar a que un elemento tenga *layout* en Internet Explorer. Si quieres acceder a decenas de técnicas que hacen uso de la propiedad `hasLayout`, puedes consultar el artículo *On having layout* (http://www.satzansatz.de/cssd/onhavinglayout.html) . Microsoft también ha publicado un artículo llamado *HasLayout Overview* (http://msdn.microsoft.com/en-us/library/bb250481(VS.85).aspx) que explica la propiedad `hasLayout` y sus implicaciones en el diseño de sitios web.

1.3. Limpiar floats

La principal característica de los elementos posicionados de forma flotante mediante la propiedad `float` es que desaparecen del flujo normal del documento. De esta forma, es posible que algunos o todos los elementos flotantes se salgan de su elemento contenedor.

La siguiente imagen muestra un elemento contenedor que encierra a dos elementos de texto. Como los elementos interiores están posicionados de forma flotante y el elemento contenedor no dispone de más contenidos, el resultado es el siguiente:

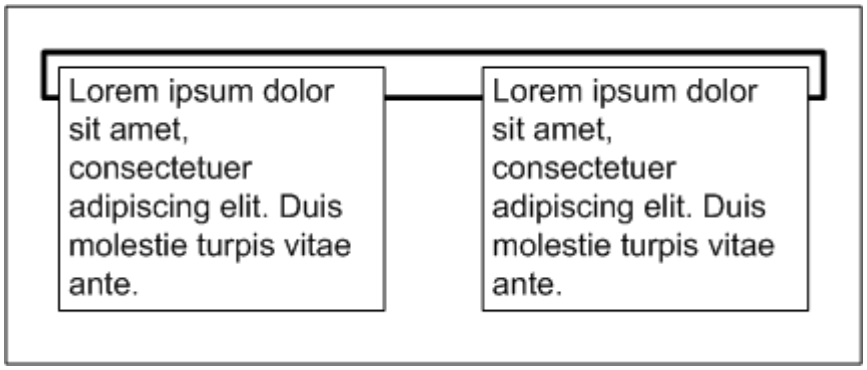


Figura 1.1. Los elementos posicionados de forma flotante se salen de su elemento contenedor

El código HTML y CSS del ejemplo anterior se muestra a continuación:

```
<div id="contenedor">
  <div id="izquierda">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis
molestie turpis vitae ante.</div>
  <div id="derecha">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla
bibendum mi non lacus.</div>
</div>
#contenedor {
  border: thick solid #000;
}

#izquierda {
  float: left;
  width: 40%;
}

#derecha {
  float: right;
  width: 40%;
}
```

La solución tradicional de este problema consiste en añadir un elemento invisible después de todos los elementos posicionados de forma flotante para forzar a que el elemento contenedor tenga la altura suficiente. Los elementos invisibles más utilizados son `<div>`, `<br>` y `<p>`.

De esta forma, si se añade un elemento `<div>` invisible con la propiedad `clear` de CSS en el ejemplo anterior:

```
<div id="contenedor">
  <div id="izquierda">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis
molestie turpis vitae ante.</div>
  <div id="derecha">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla
bibendum mi non lacus.</div>
  <div style="clear: both"></div>
</div>
```

Ahora el elemento contenedor se visualiza correctamente porque encierra a todos sus elementos:

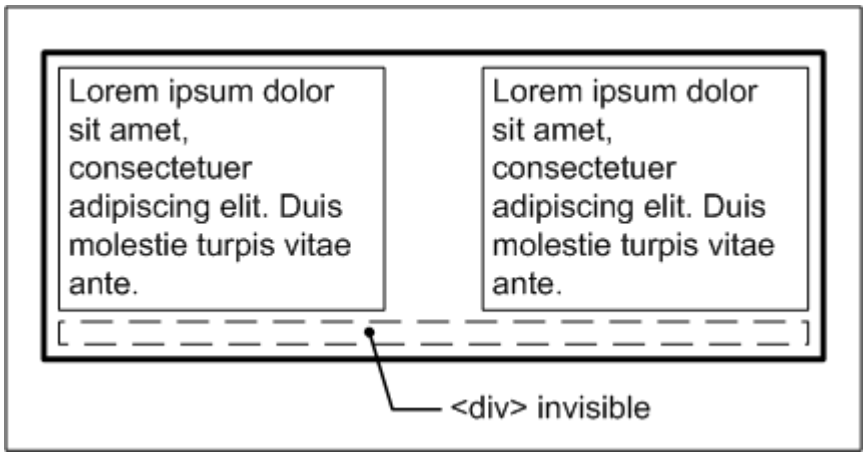


Figura 1.2. Solución tradicional al problema de los elementos posicionados de forma flotante

La técnica de corregir los problemas ocasionados por los elementos posicionados de forma flotante se suele denominar *"limpiar los float"*.

Aunque añadir un elemento invisible corrige correctamente el problema, se trata de una solución poco elegante e incorrecta desde el punto de vista semántico. No tiene ningún sentido añadir un elemento vacío en el código HTML, sobre todo si ese elemento se utiliza exclusivamente para corregir el aspecto de los contenidos.

Afortunadamente, existe una solución alternativa para *limpiar los float* que no obliga a añadir nuevos elementos HTML y que además es elegante y muy sencilla. La solución consiste en utilizar la propiedad `overflow` de CSS sobre el elemento contenedor. El autor de la solución es el diseñador Paul O'Brien (http://pmob.co.uk/) y se publicó por primera vez en el artículo *Simple Clearing of Floats* (http://www.sitepoint.com/blogs/2005/02/26/simple-clearing-of-floats/) .

Si se modifica el código CSS anterior y se incluye la siguiente regla:

```
#contenedor {
  border:   thick solid #000;
  overflow: hidden;
}
```

Ahora, el contenedor encierra correctamente a los dos elementos `<div>` interiores y no es necesario añadir ningún elemento adicional en el código HTML. Además del valor `hidden`, también es posible utilizar el valor `auto` obteniendo el mismo resultado.

Esta solución funciona correctamente en todas las versiones recientes de los navegadores, incluyendo Internet Explorer 7 y 8. No obstante, en las versiones anteriores de Internet Explorer es necesario añadir cualquier propiedad CSS que obligue al navegador a considerar que el elemento contenedor ocupa sitio en la página.

Técnicamente, esta estrategia se conoce como *forzar a que Internet Explorer active la propiedad hasLayout sobre el elemento*. A continuación se muestran algunas técnicas sencillas para conseguirlo:

```
/* Funciona correctamente con cualquier navegador */
#contenedor {
  border:   thick solid #000;
  overflow: hidden;
  width:    100%;
}

/* Funciona correctamente con cualquier navegador */
#contenedor {
  border:   thick solid #000;
  overflow: hidden;
  height:   1%;
}

/* La propiedad zoom no es parte del estándar CSS, por lo
   que esta hoja de estilos no validaría */
#contenedor {
  border:   thick solid #000;
  overflow: hidden;
  zoom:     1;
}

/* El truco del guión bajo delante de las propiedades CSS no
   lo interpreta correctamente la versión 7 de Internet Explorer */
#contenedor {
  border:   thick solid #000;
  overflow: hidden;
  _height:  1%;
}
```

De todas las soluciones anteriores, la más utilizada es la que establece la propiedad `height: 1%`, ya que normalmente es la que menos afecta al aspecto del elemento contenedor. Considerando todo lo anterior, a continuación se muestra la solución definitiva para *limpiar los floats*, que es compatible con todos los navegadores y que hace que la hoja de estilos sea válida:

```
#contenedor {
  border:   thick solid #000;
  overflow: hidden;
  height:   1%;
}

#izquierda {
  float: left;
  width: 40%;
}

#derecha {
  float: right;
  width: 40%;
}
```

1.4. Elementos de la misma altura

Hasta hace unos años, la estructura de las páginas web complejas se creaba mediante tablas HTML. Aunque esta solución presenta muchos inconvenientes, su principal ventaja es que todas las columnas que forman la página son de la misma altura.

Normalmente, cuando se crea la estructura de una página compleja, se desea que todas las columnas que la forman tengan la misma altura. De hecho, cuando algunas o todas las columnas tienen imágenes o colores de fondo, esta característica es imprescindible para obtener un diseño correcto.

Sin embargo, como el contenido de cada columna suele ser variable, no es posible determinar la altura de la columna más alta y por tanto, no es posible hacer que todas las columnas tengan la misma altura directamente con la propiedad `height`.

Cuando se utiliza una tabla para crear la estructura de la página, este problema no existe porque cada columna de la estructura se corresponde con una celda de datos de la tabla. Sin embargo, cuando se diseña la estructura de la página utilizando sólo CSS, el problema no es tan fácil de solucionar. Afortunadamente, existen varias soluciones para asegurar que dos elementos tengan la misma altura.

La primera solución es la más sencilla y la publicó el diseñador Alex Robinson en su artículo *Equal Height Columns - revisited* (http://www.positioniseverything.net/articles/onetruelayout/equalheight) . El truco consiste en añadir un espacio de relleno inferior (`padding-bottom`) muy grande a todas las columnas y después añadirles un margen inferior negativo (`margin-bottom`) del mismo tamaño.

```
#contenedor {
  overflow: hidden;
}

#columna1, #columna2, #columna3 {
  padding-bottom: 32767px;
  margin-bottom: -32767px;
}
```

El valor utilizado en el espacio de relleno y en el margen inferior de las columnas debe ser tan grande como la altura esperada para la columna más alta. Para evitar quedarse corto, se recomienda utilizar valores a partir de `10.000` píxeles.

Los dos principales problemas que presenta esta solución son los siguientes:

- Se pueden producir errores al imprimir la página con el navegador Internet Explorer.
- Si se utilizan enlaces de tipo ancla en cualquier columna, al pulsar sobre el enlace las columnas se desplazan de forma ascendente y *desaparecen* de la página.

Otra solución al problema de los elementos de la misma altura es la que presentó el diseñador Dan Cederholm en su célebre artículo *Faux Columns* (http://www.alistapart.com/articles/fauxcolumns/) . Si la solución anterior consitía en *engañar* al navegador, esta segunda solución se basa en *engañar* al ojo del usuario.

La solución de las columnas falsas consiste en establecer una imagen de fondo repetida verticalmente en el elemento contenedor. Como el contenedor es tan alto como la columna más alta, su imagen de fondo da la sensación de que todas las columnas son de la misma altura.

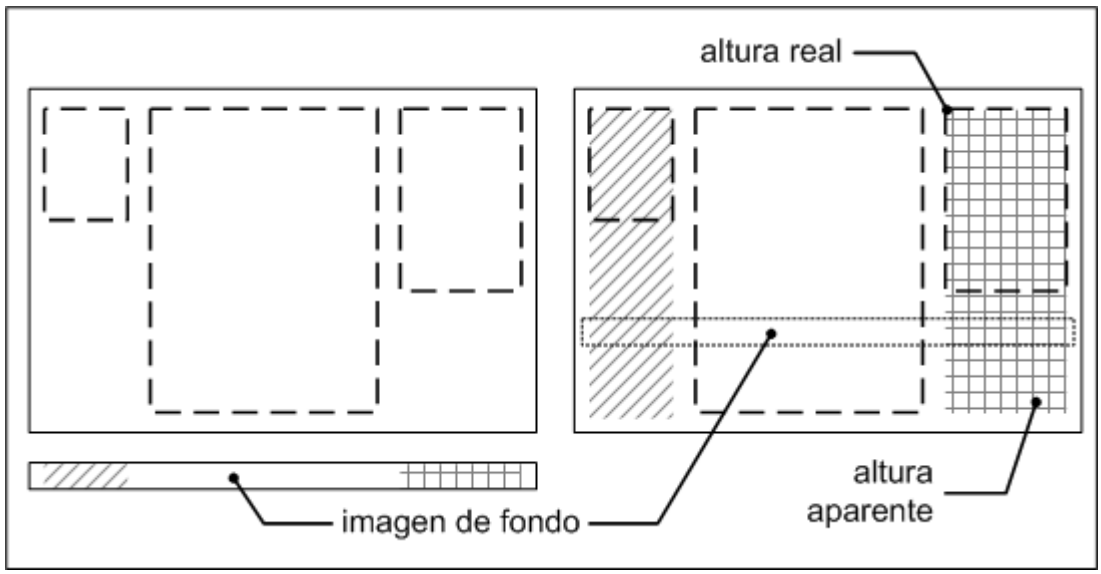


Figura 1.3. Las columnas parecen de la misma altura porque el elemento contenedor muestra una imagen de fondo repetida verticalmente

El principal inconveniente de esta técnica es que sólo se puede emplear cuando la estructura de la página es de anchura fija, es decir, cuando su diseño no es líquido y no se adapta a la anchura de la ventana del navegador.

Si el fondo de las columnas es simplemente un color y no una imagen, se puede utilizar una solución alternativa planteada por el diseñador Douglas Livingstone en su técnica *Bordered Colors* (http://www.redmelon.net/tstme/3cols2/) y que se basa en el uso de las propiedades `border-left` y `border-right` sobre la columna central de la estructura de la página. Una versión alternativa y mejorada de la técnica original se puede encontrar en *Ordered Borders Layout* (http://www.positioniseverything.net/ordered-borders-center.html) .

Las dos soluciones planteadas hasta el momento consisten en trucos para engañar a los navegadores y a los usuarios. A continuación se presenta la única solución técnicamente correcta para forzar a que dos elementos muestren la misma altura.

La solución fue propuesta por el diseñador Roger Johansson en su artículo *Equal height boxes with CSS* (http://www.456bereastreet.com/archive/200405/equal_height_boxes_with_css/) y se basa en el uso avanzado de la propiedad `display` de CSS.

En primer lugar, es necesario añadir un elemento adicional (`<div id="contenidos">`) en el código HTML de la página:

```
<div id="contenedor">
  <div id="contenidos">
    <div id="columna1"></div>
    <div id="columna2"></div>
    <div id="columna3"></div>
  </div>
</div>
```

A continuación, se utiliza la propiedad `display` de CSS para mostrar los elementos `<div>` anteriores como si fueran celdas de una tabla de datos:

```
#contenedor {
  display: table;
}

#contenidos {
  display: table-row;
}

#columna1, #columna2, #columna3 {
  display: table-cell;
}
```

Gracias a la propiedad `display` de CSS, cualquier elemento se puede comportar como una tabla, una fila de tabla o una celda de tabla, independientemente del tipo de elemento que se trate.

De esta forma, los elementos `<div>` que forman las columnas de la página en realidad se comportan como celdas de tabla, lo que permite que el navegador las muestre con la misma altura.

El único inconveniente de la solución anterior es que el navegador Internet Explorer 7 y sus versiones anteriores no son capaces de manejar los valores más avanzados de la propiedad `display`, por lo que en esos navegadores la página no muestra correctamente su estructura.

1.5. Sombras

Una de las carencias del estándar CSS 2.1 más demandadas por los diseñadores es la posibilidad de mostrar sombras tipo *"drop shadow"* sobre cualquier elemento de la página. Por este motivo, la futura versión CSS 3 incluirá una propiedad llamada `box-shadow` para crear este tipo de sombras.

A continuación se muestra la regla CSS 3 necesaria para crear una sombra gris muy difuminada y que se visualice en la esquina inferior derecha de un elemento:

```
.elemento {
  box-shadow: 2px 2px 5px #999;
}
```

Desafortunadamente, esta propiedad sólo está disponible en los navegadores que más se preocupan por los estándares. El navegador Safari 3 incluye la propiedad con el nombre `-webkit-box-shadow` y Firefox 3.1 la incluye con el nombre `-moz-box-shadow`.

La sintaxis completa de la propiedad `box-shadow` es muy compleja y se define en el borrador de trabajo del módulo de fondos y bordes de CSS3 (http://dev.w3.org/csswg/css3-background/#the-box-shadow) . A continuación se muestra su sintaxis simplificada habitual:

```
box-shadow: <medida> <medida> <medida>? <medida>? <color>
```

- La primera medida es obligatoria e indica el desplazamiento horizontal de la sombra. Si el valor es positivo, la sombra se desplaza hacia la derecha y si es negativo, se desplaza hacia la izquierda.
- La segunda medida también es obligatoria e indica el desplazamiento vertical de la sombra. Si el valor es positivo, la sombra se desplaza hacia abajo y si es negativo, se desplaza hacia arriba.
- La tercera medida es opcional e indica el radio utilizado para difuminar la sombra. Cuanto más grande sea su valor, más borrosa aparece la sombra. Si se utiliza el valor `0`, la sombra se muestra como un color sólido.
- La cuarta medida también es opcional e indica el radio con el que se expande la sombra. Si se establece un valor positivo, la sombra se expande en todas direcciones. Si se utiliza un valor negativo, la sombra se comprime.
- El color indicado es directamente el color de la sombra que se muestra.

La siguiente regla CSS muestra una sombra en los navegadores Firefox y Safari:

```
.elemento {
  -webkit-box-shadow: 2px 2px 5px #999;
  -moz-box-shadow: 2px 2px 5px #999;
}
```

Por su parte, el navegador Internet Explorer dispone de su propio mecanismo para crear sombras. La solución se basa en el uso de los filtros, un mecanismo que no forma parte del estándar de CSS y que permiten aplicar operaciones complejas a los elementos de la página. Entre los filtros de Internet Explorer (http://msdn.microsoft.com/en-us/library/ms532853(VS.85).aspx) , se encuentra el filtro shadow (http://msdn.microsoft.com/en-us/library/ms533086(VS.85).aspx) , que permite mostrar una sombra en un elemento de la página.

Las opciones del filtro `shadow` son mucho más limitadas que las de la propiedad `box-shadow`. Su sintaxis es la habitual de los filtros de Internet Explorer y las opciones son:

- `color`, establecido mediante el formato hexadecimal (ejemplo: `#CC0000`).
- `direction`, dirección hacia la que se desplaza la sombra. Su valor se indica en grados y sólo se permiten los valores `0`, `45`, `90`, `135`, `180`, `225`, `270` y `315`.
- `strength`, distancia en píxeles hasta la que se extiende la sombra.

A continuación se modifica la regla CSS anterior para incluir el filtro de Internet Explorer que muestra una sombra similar:

```
.elemento {
  -webkit-box-shadow: 2px 2px 5px #999;
  -moz-box-shadow: 2px 2px 5px #999;
  filter: shadow(color=#999999, direction=135, strength=2);
}
```

Lamentablemente, hasta que todos los navegadores más utilizados no incluyan la propiedad `box-shadow`, la única forma de mostrar una sombra sobre un elemento de la página consiste en utilizar imágenes que simulan una sombra.

A continuación se detallan los pasos necesarios para mostrar una sombra sencilla sobre una imagen:

1. Se crea una imagen del mismo tamaño que la imagen original y cuyo aspecto sea el de la sombra que se quiere mostrar.
2. Se añade un espacio de relleno a la imagen original en la posición en la que se quiere mostrar la sombra. Si por ejemplo se quiere mostrar una sombra en la esquina inferior derecha, se añade `padding-right` y `padding-bottom`.
3. Utilizando la propiedad `background`, se añade la imagen de la sombra como imagen de fondo de la imagen original. La imagen de fondo se coloca en la posición en la que se quiere mostrar la sombra. En el caso de la sombra inferior derecha, la posición de la imagen de fondo es `bottom right`.

La siguiente imagen muestra el resultado final y las imágenes utilizadas en el proceso:

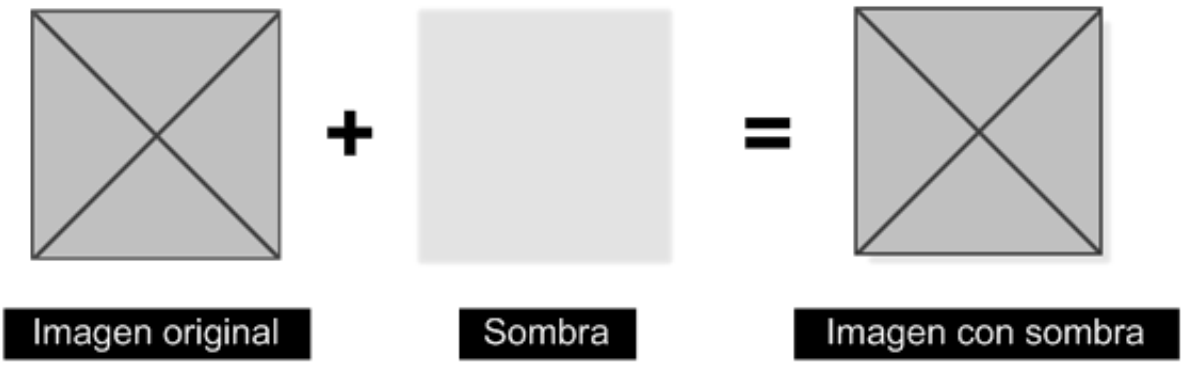


Figura 1.4. Aplicando una sombra a una imagen

El código CSS necesario para conseguir este efecto se muestra a continuación:

```
img {
  background: url("imagenes/sombra.png") no-repeat bottom right;
  padding-right:  10px;
  padding-bottom: 10px;
}
```

El principal inconveniente de esta técnica sencilla es que se deben crear tantas imágenes de sombra como tamaños diferentes de imágenes haya en el sitio web. La solución a este problema consiste en crear una imagen de sombra muy grande y aplicarla a todas las imágenes. Esta nueva sombra debe tener un tamaño al menos tan grande como la imagen más grande que se vaya a utilizar.

El problema de utilizar una imagen de sombra muy grande es que los bordes de la sombra no quedan tan bien como cuando se utiliza una imagen de sombra del mismo tamaño que la imagen original:

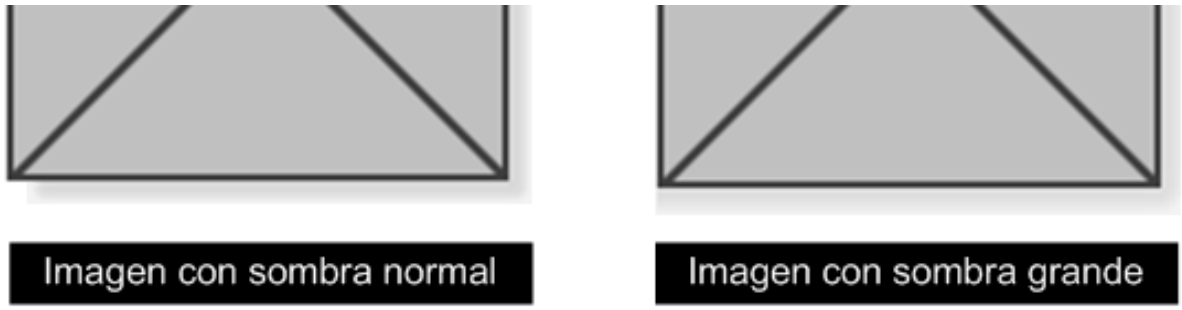


Figura 1.5. Las imágenes de sombra muy grande producen bordes más feos

La solución completa para crear sombras de cualquier tamaño y con bordes correctos implica el uso de más imágenes. Los siguientes recursos muestran cómo crear ese tipo de sombras:

- *Shadowed Image* (http://www.cssdesignpatterns.com/Chapter%2014%20-%20IMAGES/Shadowed%20Image/example.html) : utiliza tres imágenes y tres elementos `<div>` para cada imagen.
- *CSS Drop Shadows II: Fuzzy Shadows* (http://www.alistapart.com/articles/cssdrop2/) : utiliza dos imágenes y dos elementos `<div>` para cada imagen.
- *My contribution to the CSS shadow kerfuffle* (http://theshapeofdays.com/2005/11/29/my-contribution-to-the-css-shadow-kerfuffle.html) : el resultado es idéntico al obtenido mediante el *drop shadow* de Photoshop, pero requiere cinco imágenes y cinco elementos `<div>` para cada imagen.

Si se quiere mostrar una sombra sobre elementos que no sea imágenes, la solución consiste en encerrar los contenidos con dos elementos `<div>` y aplicar la imagen de sombra sobre el primero de ellos:

```
#sombra {
  background: url("imagenes/sombra_grande.png") no-repeat 100% 100%;
  padding-right: 15px;
  padding-bottom: 15px;
  width: 200px;
}

#sombra div {
  background: #FFF;
  width: 200px;
}

<div id="sombra">
  <div>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut diam metus,
venenatis ullamcorper, consequat sit amet, volutpat at, nulla. Nulla sollicitudin
metus.</div>
</div>
```

Las soluciones basadas en imágenes tienen la ventaja de que funcionan correctamente en cualquier navegador. Sin embargo, complican innecesariamente el código HTML de la página y tienen limitaciones como la de tener que crear una nueva imagen cada vez que se quiere cambiar el color de la sombra.

1.6. Transparencias

El estándar de CSS 2.1 no incluye ninguna propiedad para controlar la opacidad de los elementos de la página. Sin embargo, la futura versión CSS 3 incluye una propiedad llamada `opacity`, definida en el módulo de colores de CSS 3 (http://www.w3.org/TR/css3-color/) y que permite incluir transparencias en el diseño de la página.

A pesar de que falta mucho tiempo hasta que se publique la versión definitiva del estándar CSS 3, los navegadores que más se esfuerzan en cumplir los estándares (Firefox, Safari y Opera) ya incluyen la propiedad `opacity` en sus últimas versiones.

El valor de la propiedad `opacity` se establece mediante un número decimal comprendido entre `0.0` y `1.0`. La interpretación del valor numérico es tal que el valor `0.0` es la máxima transparencia (el elemento es invisible) y el valor `1.0` se corresponde con la máxima opacidad (el elemento es completamente visible). De esta forma, el valor `0.5` corresponde a un elemento semitransparente y así sucesivamente.

En el siguiente ejemplo, se establece la propiedad `opacity` con un valor de `0.5` para conseguir una transparencia del `50%` sobre dos de los elementos `<div>`:

```
#segundo, #tercero {
  opacity: 0.5;
}

#primero {
  background-color: blue;
}

#segundo {
  background-color: red;
}

#tercero {
  background-color: green;
}
```

Si se visualiza el ejemplo anterior en el navegador Firefox, Safari u Opera, los elementos `<div>` rojo y verde aparecen semitransparentes, tal y como se muestra en la siguiente imagen:

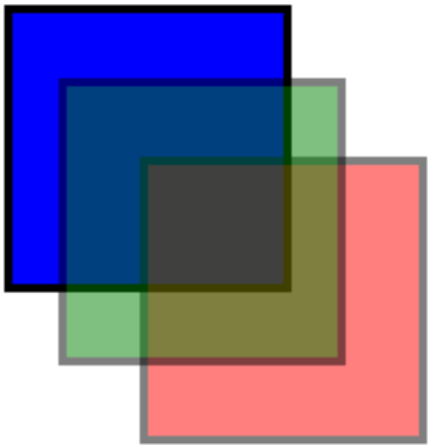


Figura 1.6. Creando elementos semitransparentes con la propiedad opacity

Desafortunadamente, la solución anterior no funciona en el navegador Internet Explorer 7 y sus versiones anteriores porque no soportan la propiedad `opacity`. No obstante, el navegador Internet Explorer incluye un mecanismo llamado filtros, que no forman parte de ningún estándar de CSS pero que permiten solucionar esta limitación.

Los filtros permiten aplicar operaciones complejas a los elementos de la página. Los filtros de Internet Explorer (http://msdn.microsoft.com/en-us/library/ms532853(VS.85).aspx) se dividen en estáticos y de transición. Los filtros estáticos (http://msdn.microsoft.com/en-us/library/ms673539(VS.85).aspx) se utilizan básicamente para crear efectos gráficos sobre los elementos, normalmente imágenes. Entre los filtros estáticos se encuentra el filtro alpha (http://msdn.microsoft.com/en-us/library/ms532967.aspx) , que permite controlar la opacidad de un elemento de la página.

La sintaxis completa del filtro alpha es muy compleja porque sus posibilidades son numerosas. No obstante, la sintaxis necesaria para establecer solamente el nivel de transparencia de un elemento es muy sencilla:

```
#segundo, #tercero {
  filter: alpha(opacity=50);
}
```

El valor de la opción `opacity` del filtro `alpha` se establece mediante un número entero comprendido entre `0` (el elemento es invisible) y `100` (el elemento es completamente opaco). El valor `50` del ejemplo anterior hace que el elemento sea semitransparente.

A continuación se muestra la solución compatible con todos los navegadores para que un elemento de la página sea semitransparente:

```
selector {
  opacity: 0.5;
  filter:  alpha(opacity=50);
}
```

Por último, recuerda que la propiedad `filter` es propietaria de Internet Explorer y no forma parte de ningún estándar de CSS, por lo que regla anterior no pasa satisfactoriamente el proceso de validación CSS.

1.7. Sustitución de texto por imágenes

La limitación más frustrante para los diseñadores web que cuidan especialmente la tipografía de sus páginas es la imposibilidad de utilizar cualquier tipo de letra para mostrar los contenidos de texto. Como se sabe, las fuentes que utiliza una página deben estar instaladas en el ordenador o dispositivo del usuario para que el navegador pueda mostrarlas.

Por lo tanto, si el diseñador utiliza en la página una fuente especial poco común entre los usuarios normales, el navegador no encuentra esa fuente y la sustituye por una fuente por defecto. El resultado es que la página que ve el usuario y la que ve el diseñador se diferencian completamente en su tipografía.

La consecuencia directa de esta limitación es que todos los diseñadores se limitan a utilizar las pocas fuentes que tienen casi todos los usuarios del mundo:

- Sistemas operativos tipo Windows: `Arial`, `Verdana`, `Tahoma`, `Courier New`, `Times New Roman`, `Georgia`. También está disponible una lista con todas las fuentes que incluye por defecto cada versión de Windows (http://www.ampsoft.net/webdesign-l/windows-fonts-by-version.html) .
- Sistemas operativos tipo Mac: `Arial`, `Helvetica`, `Verdana`, `Monaco`, `Times`.
- Sitemas operativos tipo Linux: incluyen decenas de fuentes, muchas de ellas versiones libres de las fuentes comerciales. También es posible instalar las fuentes más populares de Windows mediante el paquete Core Font (http://corefonts.sourceforge.net/) .

Debido a la presencia mayoritaria de los sistemas operativos Windows y la disponibilidad de muchas de sus fuentes en otros sistemas operativos, la mayoría de diseñadores utilizan exclusivamente las fuentes más populares de Windows.

Afortunadamente, existen varias técnicas que permiten utilizar cualquier tipo de letra en el diseño de una página con la seguridad de que todos los usuarios la verán correctamente.

1.7.1. La directiva @font-face

La única solución técnicamente correcta desde el punto de vista de CSS es el uso de la directiva `@font-face`. Esta directiva se definió en el estándar CSS 2, pero desapareció en el estándar CSS 2.1 que utilizan todos los navegadores de hoy en día. La futura versión de CSS 3 volverá a incluir la directiva `@font-face` en el módulo llamado *Web Fonts* (http://www.w3.org/TR/css3-webfonts/) .

La directiva `@font-face` permite describir las fuentes que utiliza una página web. Como parte de la descripción se puede indicar la dirección o URL desde la que el navegador se puede descargar la fuente utilizada si el usuario no dispone de ella. A continuación se muestra un ejemplo de uso de la directiva `@font-face`:

```
@font-face {
  font-family: "Fuente muy rara";
  src: url("http://www.ejemplo.com/fuentes/fuente_muy_rara.ttf");
}
```

```
h1 {
  font-family: "Fuente muy rara", sans-serif;
}
```

La directiva `@font-face` también permite definir otras propiedades de la fuente, como su formato, grosor y estilo. A continuación se muestran otros ejemplos:

```
@font-face {
  font-family: Fontinsans;
  src: url("fonts/Fontin_Sans_SC_45b.otf") format("opentype");
  font-weight: bold;
  font-style: italic;
}

@font-face {
  font-family: Tagesschrift;
  src: url("fonts/YanoneTagesschrift.ttf") format("truetype");
}
```

Los ejemplos anteriores han sido extraídos de la página *10 Great Free Fonts for @font-face embedding* (http://opentype.info/demo/webfontdemo.html) . Para ver el efecto de la directiva `@font-face`, debes acceder a esa página utilizando un navegador como Safari.

1.7.2. Soluciones FIR

Las siglas FIR (*Fahrner Image Replacement*) abarcan decenas de técnicas similares que tratan de resolver el problema de utilizar cualquier tipo de letra en una página web. El propósito de estas técnicas es sustituir el texto normal por una imagen que contenga el mismo texto escrito con la tipografía deseada. La primera técnica se presentó en el artículo *Using Background-Image to Replace Text* (http://www.stopdesign.com/articles/replace_text/) .

Si se dispone de un título de sección `<h1>` que se quiere mostrar con una tipografía muy especial, se puede utilizar el siguiente código HTML y CSS para sustituirlo por una imagen:

```
<h1><span>Lorem ipsum dolor sit amet</span></h1>

h1 {
  width: 450px;
  height: 100px;
  background: #FFF url("/imagenes/titular.png") no-repeat top left;
}

h1 span {
  display: none;
}
```

El código anterior muestra como imagen de fondo del elemento `<h1>` la imagen que contiene el titular escrito con la tipografía deseada. Para sustituir el texto por la imagen, simplemente se oculta el texto mediante la propiedad `display: none`.

Aunque se trata de una de las técnicas FIR más sencillas, su principal problema es que el texto oculto mediante `display` no lo leen correctamente los dispositivos lectores de páginas que utilizan las personas con discapacidades visuales.

La evolución de la técnica anterior consiste en reemplazar la propiedad `display` por `text-indent`:

```
<h1>Lorem ipsum dolor sit amet</h1>

h1 {
  width: 450px;
  height: 100px;
  background: #FFF url("/imagenes/titular.png") no-repeat top left;
  text-indent: -5000px;
}
```

Utilizando un valor negativo muy grande en la propiedad `text-indent` del elemento que se quiere reemplazar, el texto no se oculta pero se desplaza fuera de la pantalla. Los navegadores normales no muestran el texto y los lectores de páginas lo leen correctamente porque el texto no está oculto.

El problema de la solución anterior es que si el navegador tiene activado CSS y desactivada la carga de las imágenes, no se muestra nada. La solución consiste en volver a utilizar otro elemento `<span>` dentro del elemento que se quiere reemplazar:

```
<h1><span></span>Lorem ipsum dolor sit amet</h1>

h1 {
  width: 450px;
  height: 100px;
  position: relative;
}

h1 span {
  background: #FFF url("/imagenes/titular.png") no-repeat top left;
  position: absolute;
  width: 100%;
  height: 100%;
}
```

En esta solución, la imagen se muestra por delante del texto, por lo que aunque el texto no se oculta ni se desplaza, el usuario no puede verlo. El principal problema de esta técnica es que no se pueden utilizar imágenes con transparencias.

El artículo *Revised Image Replacement* (http://www.mezzoblue.com/tests/revised-image-replacement/) presenta otras técnicas FIR y discute sus problemas y limitaciones.

Independientemente de los problemas técnicos relacionados con CSS y los navegadores, el principal problema de las técnicas FIR anteriores es que se deben crear tantas imágenes como elementos se quieran reemplazar. Aunque el proceso de creación de imágenes sea automático, es un proceso pesado cuando se quiere modificar por ejemplo el tamaño o tipo de letra utilizado.

1.7.3. Soluciones avanzadas

Las soluciones basadas exclusivamente en CSS suelen presentar problemas con los lectores de páginas que utilizan las personas discapacitadas para navegar. Por ese motivo se han ideado otras soluciones basadas en diferentes lenguajes de programación.

El programador Peter-Paul Koch ha creado una solución basada en JavaScript y que explica en su artículo *Image Replacement* (http://www.quirksmode.org/dom/fir.html) . Esta solución muestra las imágenes en todos aquellos navegadores que las soporten y muestra sólo el texto en cualquier otro caso. De esta forma, el texto original no se oculta de ninguna manera que pueda impedir a los lectores de páginas acceder a los contenidos.

Otras soluciones permiten crear de forma dinámica las imágenes que sustituyen al texto. El proceso es muy complejo porque hay que considerar aspectos como el espacio máximo que puede ocupar la imagen y los posibles saltos de línea en el texto. Existen multitud de soluciones de este tipo para diferentes lenguajes de programación, como por ejemplo el proyecto pcdtr (http://code.google.com/p/pcdtr/) para PHP.

1.8. Sustitución de texto por Flash

Como se explica en la sección anterior, las soluciones basadas en CSS para sustituir el texto por imágenes presentan problemas técnicos no resueltos: los lectores de páginas pueden no leer el texto y los navegadores normales tienen problemas cuando no se pueden cargar las imágenes.

Además, estas soluciones basadas en CSS siempre tienen el inconveniente de que se deben crear todas las imágenes que sustituyen al texto. Si las imágenes se crean a mano, el proceso es tedioso y poco flexible. Si las imágenes se generan dinámicamente, la aplicación web puede sufrir una penalización apreciable en su rendimiento.

La solución definitiva de todos los problemas de las soluciones basadas en CSS es la técnica sIFR (http://www.mikeindustries.com/blog/sifr/) (*Scalable Inman Flash Replacement*), que combina CSS, JavaScript y Flash para mostrar correctamente cualquier texto con cualquier tipografía deseada.

La versión de sIFR recomendada para su uso en sitios web es la versión sIFR 2 (http://www.mikeindustries.com/blog/sifr/) , ya que la versión sIFR 3 (http://wiki.novemberborn.net/sifr3) todavía se encuentra en período de pruebas.

A continuación se muestran los pasos que hay que seguir para sustituir el texto con la técnica sIFR:

1) Descarga la última versión disponible de los archivos de sIFR. Actualmente, la versión estable es 2.0.2 y se puede descargar mediante el archivo comprimido sIFR2.0.2.zip (http://www.mikeindustries.com/blog/files/sifr/2.0/sIFR2.0.2.zip) . Descomprime este archivo en cualquier carpeta del sistema.

2) El segundo paso consiste en crear un archivo Flash que incluya la fuente que se va a utilizar en la sustitución. Abre el archivo `sifr.fla` con un editor de archivos de tipo Flash (como por ejemplo el programa Flash Professional (http://www.adobe.com/es/products/flash/)). Si no

dispones de ningún editor de archivos Flash, más adelante se muestran varias herramientas y utilidades equivalentes.

Cuando se abre el archivo `sifr.fla`, sólo se ve un rectángulo blanco. Pincha dos veces sobre ese rectángulo de forma que se muestre un texto. Selecciona el texto, modifica su fuente por la tipografía que quieres utilizar en el diseño de tu página y guarda los cambios.

Si el texto que vas a sustituir contiene caracteres *especiales* o caracteres propios de algunos idiomas, debes añadir todos esos caracteres mediante la paleta de `Propiedades` del texto.

3) Exporta el archivo Flash mediante la opción `File > Export > Export Movie` (o `Archivo > Exportar`). Una buena recomendación consiste en guardar el archivo con el mismo nombre del tipo de letra que incluye (`arial.swf`, `verdana.swf`, etc.). Una vez creado el archivo, guárdalo en un directorio del servidor en el que guardes todas las fuentes de sIFR (puedes llamar a este directorio `fuentes/` por ejemplo).

4) sIFR necesita varios archivos CSS y JavaScript para funcionar. Copia el archivo `sifr.js` en la carpeta de archivos JavaScript de tu servidor web (normalmente este directorio se llama `js/`). Copia los archivos CSS `sIFR-screen.css` y `sIFR-print.css` en la carpeta de archivos CSS de tu servidor web (normalmente este directorio se llama `css/`).

5) Todas las páginas en las que se sustituye el texto por Flash deben incluir los archivos JavaScript y CSS. Para ello, añade lo siguiente dentro de la sección `<head>` de la página:

```
<head>
  ...
  <script src="js/sifr.js" type="text/javascript"></script>
  ...
  <link rel="stylesheet" href="css/sIFR-screen.css" type="text/css" media="screen" />
  <link rel="stylesheet" href="css/sIFR-print.css" type="text/css" media="print" />
  ...
</head>
```

El valor de los atributos `src` y `href` debe contener la ruta completa (absoluta o relativa) hasta los archivos JavaScript y CSS respectivamente.

También es posible añadir las reglas de los archivos CSS de sIFR en los archivos CSS propios de la página y así no tener que enlazar otros dos archivos CSS en cada página.

6) A partir de este momento ya es posible sustituir cualquier texto por Flash utilizando instrucciones JavaScript sencillas. A continuación se muestra el código JavaScript necesario para mostrar todos los titulares de la página con una tipografía propia:

```
if(typeof sIFR == "function"){
  sIFR.replaceElement("h1", named({sFlashSrc: "../fuentes/mifuente.swf"}));
};
```

El código JavaSript anterior lo puedes colocar en cualquier parte de la página. Algunos diseñadores prefieren colocarlo en la sección `<head>` donde se encuentran el resto de elementos de sIFR y otros diseñadores prefieren colocarlo justo antes de la etiqueta `</body>` de cierre de la página.

La sustitución requiere que en la función `replaceElement()` se indique el selector CSS de los elementos que se van a sustituir y una serie de opciones que se deben tener en cuenta en la sustitución. La única opción obligatoria se denomina `sFlashSrc` e indica la ruta completa hasta el archivo `.swf` que contiene la tipografía deseada.

A continuación se muestra una tabla con todas las propiedades de sFIR:

Opción	**Descripción**
`sSelector`	El selector del elemento o elementos que se quieren reemplazar. Se puede indicar mediante esta opción o como primer argumento de la función `replaceElement()`. Se pueden indicar varios selectores separándolos por comas. Los selectores soportados son los de etiqueta, clase, id y el selector de hijos.
`sFlashSrc`	Ruta completa hasta el archivo `.swf` que contiene la fuente utilizada. Si indicas la ruta de forma relativa, el origen es la propia página HTML.
`sColor`	Color del texto indicado en formato hexadecimal, como por ejemplo `#000000` o `#CC0000`
`sLinkColor`	Color de los enlaces que puede contener el texto
`sHoverColor`	Color del estado `:hover` de los enlaces que puede contener el texto
`sBgColor`	Color de fondo del texto indicado en formato hexadecimal

`nPaddingTop`
`nPaddingRight`
`nPaddingBottom`
`nPaddingLeft`
	Relleno superior, izquierdo, inferior e izquierdo del texto

`sCase`	Transformación del texto. El valor `upper` transforma el texto a mayúsculas; el valor `lower` lo transforma a minúsculas
`sWmode`	Esta opción causa problemas en algunas versiones de algunos navegadores, por lo que no se recomienda su uso. Si se le asigna el valor `transparent`, el Flash que sustituye al texto tiene un fondo transparente. Si se emplea el valor `opaque`, el Flash tiene un color de fondo sólido.
`sFlashVars`	Variables adicionales que pueden modificar el aspecto del texto sustituido. `textalign=center` centra el texto de forma horizontal `offsetleft=5` desplaza el texto hacia la derecha los píxeles indicados por el número `offsetTop=5` desplaza el texto hacia abajo los píxeles indicados por el número `underline=true` muestra los enlaces subrayados en el estado `:hover`

El siguiente ejemplo muestra los titulares de sección `<h1>`, `<h2>` y `<h3>` con el mismo tipo de letra y color negro, mientras que sus posibles enlaces se muestran de color azul que cambia a rojo cuando se pasa el ratón por encima del texto:

```
if(typeof sIFR == "function"){
  sIFR.replaceElement("h1, h2, h3", named({sFlashSrc: "./sifr/vandenkeere.swf", sColor:
"#000", sLinkColor: "#336699", sHoverColor: "#CC0000"}));
};
```

El principal inconveniente de la técnica sIFR es la creación del archivo `.swf` con la fuente que se quiere mostrar. Además de ser un proceso manual, es necesario disponer de un editor de

archivos Flash. Afortunadamente, debido a la popularidad de sIFR, se han publicado numerosas herramientas para crear los archivos `.swf` necesarios:

- sIFR Generator (http://www.sifrgenerator.com/wizard.html) : aplicación web que genera de forma rápida y sencilla el archivo `.swf` a partir de una fuente de tipo TrueType (extensión `.ttf`).
- sIFR Font Embedder (http://digitalretrograde.com/Projects/sifrFontEmbedder/) : aplicación de escritorio que genera el archivo `.swf` a partir de la fuente seleccionada. Permite añadir fácilmente todos los caracteres especiales deseados. Requiere el uso de un programa externo llamado `swfmill` y del framework .NET versión 2.0.
- Extensión para integrar sIFR en DreamWeaver (http://www.tecnorama.org/document.php?id_doc=70) : extensión para el conocido entorno de desarrollo de sitios web Dreamweaver. Permite configurar todas las opciones de sIFR de forma visual y crea automáticamente los enlaces a los archivos CSS y JavaScript necesarios.

1.9. Esquinas redondeadas

El actual estándar CSS 2.1 obliga a que todos los bordes de los elementos sean rectangulares. Esta limitación es una de las más criticadas por los diseñadores, ya que les impide crear bordes curvos o redondeados que se adapten mejor a sus diseños.

La futura versión CSS 3 incluye varias propiedades para definir bordes redondeados. La propiedad `border-radius` establece la misma curvatura en todas las esquinas y también se definen las propiedades `border-top-right-radius`, `border-bottom-right-radius`, `border-bottom-left-radius`, `border-top-left-radius` para establecer la curvatura de cada esquina.

En cada propiedad se debe indicar obligatoriamente una medida y se puede indicar opcionalmente una segunda medida. Cuando se indica una sola medida, la esquina es circular y su radio es igual a la medida indicada. Cuando se indican dos medidas, la esquina es elíptica, siendo la primera medida el radio horizontal de la elipse y la segunda medida su radio vertical.

Aunque faltan muchos años hasta que se publique la versión definitiva de CSS 3, los navegadores que más se preocupan de los estándares ya incluyen soporte para crear esquinas redondeadas. El siguiente ejemplo muestra cómo crear esquinas redondeadas con los navegadores Safari y Firefox:

```
div {
  -webkit-border-radius: 5px 10px;  /* Safari  */
  -moz-border-radius: 5px 10px;     /* Firefox */
}
```

La solución basada en CSS 3 es la más sencilla y la mejor técnicamente, pero hasta que todos los navegadores no incluyan soporte para CSS 3, no es posible utilizar esta técnica para crear esquinas redondeadas.

Afortunadamente, las esquinas redondeadas son uno de los recursos más solicitados por los diseñadores web y por eso se han definido decenas de técnicas para mostrarlas. Las técnicas se clasifican en:

- Soluciones basadas en CSS y que no utilizan imágenes.
- Soluciones basadas en CSS y que utilizan imágenes.
- Soluciones basadas en JavaScript.

Las soluciones basadas exclusivamente en CSS y que no utilizan imágenes combinan HTML y CSS para *engañar* al ojo del usuario y hacerle creer que la esquina es redondeada.

El truco consiste en añadir varios elementos cuya longitud disminuye progresivamente para crear un perfil curvo. La siguiente imagen muestra el resultado final de esta técnica (izquierda), el número de elementos necesarios para conseguirlo (centro) y un detalle ampliado de una esquina (derecha):

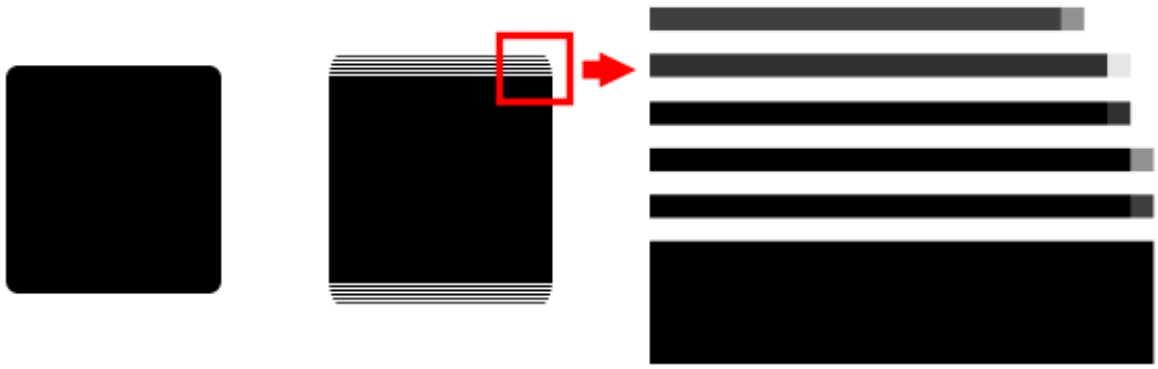


Figura 1.7. Esquinas redondeadas creadas con CSS y sin imágenes (resultado final y detalle de cómo se consigue)

A continuación se muestra el código HTML y CSS necesarios para crear esquinas redondeadas con CSS y sin imágenes:

```
<div id="contenedor">
<b class="esquinas_superiores">
  <b class="r1"></b>
  <b class="r2"></b>
  <b class="r3"></b>
  <b class="r4"></b>
</b>

<!-- Aquí se incluyen los contenidos -->

<b class="esquinas_inferiores">
  <b class="r4"></b>
  <b class="r3"></b>
  <b class="r2"></b>
  <b class="r1"></b>
```

```
</b>
</div>
.esquinas_superiores, .esquinas_inferiores {
  display: block;
}
.esquinas_superiores *, .esquinas_inferiores * {
  display: block;
  height: 1px;
  overflow: hidden;
}
.r1 { margin: 0 5px; }
.r2 { margin: 0 3px; }
.r3 { margin: 0 2px; }
.r4 { margin: 0 1px; height: 2px; }
```

Para crear una esquina redondeada con esta técnica es necesario incluir muchos elementos HTML adicionales. Por ese motivo se utiliza la etiqueta `<b>`, que hace que el código HTML siga siendo válido y su nombre sólo tiene una letra, por lo que aumenta lo mínimo posible el tamaño del código HTML.

Las reglas CSS anteriores hacen que los elementos `<b>` tengan sólo `1px` de altura y que cada vez se hagan más cortos, ya que aumentan sus márgenes laterales de forma progresiva. Modificando ligeramente los `margin` de cada elemento se pueden crear esquinas con más o menos radio de cuvatura. Además, también se pueden crear curvas sólo en una de las esquinas.

El principal problema de esta técnica es que no es sencillo cambiar la forma de la esquina redondeada y que la curva es tan escalonada que los usuarios pueden apreciarlo.

La solución al segundo problema consiste en crear curvas que utilicen la técnica del *anti-aliasing* para suavizar sus bordes. Esta técnica no es sencilla, ya que se debe tener en cuenta el radio de curvatura, el color de la curva y el color de fondo del elemento contenedor.

Algunas aplicaciones web generan automáticamente el código HTML y CSS necesarios a partir de los colores y el radio de curvatura deseado. A continuación se muestra el código HTML y CSS generados por la técnica Spiffy Corners (http://www.spiffycorners.com/) :

```
<div>
  <b class="spiffy">
  <b class="spiffy1"><b></b></b>
  <b class="spiffy2"><b></b></b>
  <b class="spiffy3"></b>
  <b class="spiffy4"></b>
  <b class="spiffy5"></b></b>

  <div class="spiffyfg">
    <!-- Aquí se incluyen los contenidos -->
  </div>

  <b class="spiffy">
  <b class="spiffy5"></b>
  <b class="spiffy4"></b>
  <b class="spiffy3"></b>
  <b class="spiffy2"><b></b></b>
```

```
  <b class="spiffy1"><b></b></b></b>
</div>
.spiffy{display:block}
.spiffy *{
  display:block;
  height:1px;
  overflow:hidden;
  font-size:.01em;
  background:#000000}
.spiffy1{
  margin-left:3px;
  margin-right:3px;
  padding-left:1px;
  padding-right:1px;
  border-left:1px solid #919191;
  border-right:1px solid #919191;
  background:#3f3f3f}
.spiffy2{
  margin-left:1px;
  margin-right:1px;
  padding-right:1px;
  padding-left:1px;
  border-left:1px solid #e5e5e5;
  border-right:1px solid #e5e5e5;
  background:#303030}
.spiffy3{
  margin-left:1px;
  margin-right:1px;
  border-left:1px solid #303030;
  border-right:1px solid #303030;}
.spiffy4{
  border-left:1px solid #919191;
  border-right:1px solid #919191}
.spiffy5{
  border-left:1px solid #3f3f3f;
  border-right:1px solid #3f3f3f}
.spiffyfg{
  background:#000000}
```

RoundedCornr (http://www.roundedcornr.com/) es otra aplicación web que permite generar automáticamente el código HTML y CSS necesarios para crear esquinas redondeadas avanzadas.

Además de las esquinas redondeadas sencillas, muchos diseñadores quieren utilizar en sus trabajos esquinas complejas creadas a partir de imágenes. Una vez más, la futura versión CSS 3 incluirá una propiedad llamada `border-image` para utilizar cualquier imagen como borde de un elemento.

Hasta que los navegadores no soporten CSS 3, la solución consiste en aplicar las imágenes mediante CSS y algunos elementos HTML especialmente preparados.

Esta técnica requiere en primer lugar crear las imágenes de cada una de las cuatro esquinas del elemento. Una vez creadas las imágenes, se añaden elementos en el código HTML. Utilizando CSS, se muestran las esquinas redondeadas como imágenes de fondo de esos elementos HTML.

Los elementos HTML que se añaden pueden variar de una solución a otra, pero en general se añaden elementos <b> porque es una etiqueta de una sola letra:

```
<div id="contenedor">
  <b class="superior">
    <b></b>
  </b>
  <!-- Aquí se incluyen los contenidos -->
  <b class="inferior">
    <b></b>
  </b>
</div>
```

Con el código HTML anterior, las reglas CSS necesarias para mostrar las imágenes de cada esquina son muy sencillas:

```
b.superior   { background:url("imagenes/esquina_superior_izquierda.png") no-repeat; }
b.superior b { background:url("imagenes/esquina_superior_derecha.png") no-repeat; }

b.inferior   { background:url("imagenes/esquina_inferior_izquierda.png") no-repeat; }
b.inferior b { background:url("imagenes/esquina_inferior_derecha.png") no-repeat; }
```

Por último, cuando las esquinas redondeadas no utilizan imágenes, es más conveniente utilizar soluciones basadas en JavaScript. La principal ventaja de esta técnica es que no es necesario *ensuciar* de forma permanente el código HTML con decenas de elementos de tipo o <b>. Cuando el usuario carga la página, el código JavaScript crea en ese momento todos los elementos necesarios y los añade de forma dinámica al código HTML de la página.

Además, la otra gran ventaja de las soluciones basadas en JavaScript es que añaden decenas de elementos para crear bordes redondeados tan perfectos que son indistinguibles de los que se pueden crear con imágenes.

Algunas de las soluciones basadas en JavaScript más conocidas son *jQuery Corner* (http://methvin.com/jquery/jq-corner.html) , *jQuery Curvy Corners* (http://blue-anvil.com/archives/anti-aliased-rounded-corners-with-jquery) , *jQuery Corners* (http://www.atblabs.com/jquery.corners.html) y *Nifty Corners Cube* (http://www.html.it/articoli/niftycube/index.html) .

El artículo *CSS Rounded Corners Roundup* (http://www.smileycat.com/miaow/archives/000044.php) compara decenas de técnicas para crear esquinas redondeadas basadas en CSS y JavaScript.

1.10. Rollovers y sprites

Según varios estudios realizados por Yahoo!, hasta el 80% de la mejora en el rendimiento de la descarga de páginas web depende de la parte del cliente. En el artículo *Performance Research, Part 1: What the 80/20 Rule Tells Us about Reducing HTTP Requests* (http://yuiblog.com/blog/2006/11/28/performance-research-part-1/) Yahoo! explica que generar dinámicamente el código HTML de la página y servirla ocupa el 20% del tiempo total de descarga de la página. El 80% del tiempo restante los navegadores descargan las imágenes, archivos JavaScript, hojas de estilos y cualquier otro tipo de archivo enlazado.

Además, en la mayoría de páginas web *normales*, la mayor parte de ese 80% del tiempo se dedica a la descarga de las imágenes. Por tanto, aunque los mayores esfuerzos siempre se centran en reducir el tiempo de generación dinámica de las páginas, se consigue más y con menos esfuerzo mejorando la descarga de las imágenes.

La idea para mejorar el rendimiento de una página que descarga por ejemplo 15 imágenes consiste en crear una única imagen grande que incluya las 15 imágenes individuales y utilizar las propiedades CSS de las imágenes de fondo para mostrar cada imagen. Esta técnica se presentó en el artículo *CSS Sprites: Image Slicing's Kiss of Death* (http://www.alistapart.com/articles/sprites) y desde entonces se conoce con el nombre de *sprites CSS*.

El siguiente ejemplo explica el uso de los *sprites CSS* en un sitio web que muestra la previsión meteorológica de varias localidades utilizando iconos:

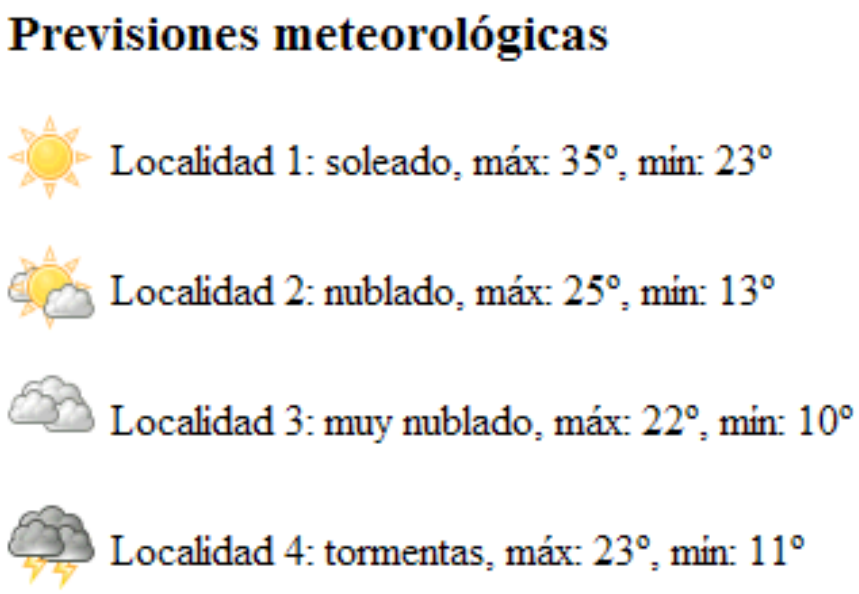


Figura 1.8. Aspecto de la previsión meteorológica mostrada con iconos

La solución tradicional para crear la página anterior consiste en utilizar cuatro elementos `<img>` en el código HTML y disponer de cuatro imágenes correspondientes a los cuatro iconos:

```
<h3>Previsiones meteorológicas</h3>
<p id="localidad1"><img src="imagenes/sol.png" /> Localidad 1: soleado, máx: 35º, mín:
23º</p>
<p id="localidad2"><img src="imagenes/sol_nubes.png" /> Localidad 2: nublado, máx: 25º,
mín: 13º</p>
<p id="localidad3"><img src="imagenes/nubes.png" /> Localidad 3: muy nublado, máx: 22º,
mín: 10º</p>
<p id="localidad4"><img src="imagenes/tormentas.png" /> Localidad 4: tormentas, máx:
23º, mín: 11º</p>
```

Aunque es una solución sencilla y que funciona muy bien, se trata de una solución completamente ineficiente. El navegador debe descargar cuatro imágenes diferentes para mostrar la página, por lo que debe realizar cuatro peticiones al servidor.

Después del tamaño de los archivos descargados, el número de peticiones realizadas al servidor es el factor que más penaliza el rendimiento en la descarga de páginas web. Utilizando la técnica

de los *sprites CSS* se va a rehacer el ejemplo anterior para conseguir el mismo efecto con una sola imagen y por tanto, una sola petición al servidor.

El primer paso consiste en crear una imagen grande que incluya las cuatro imágenes individuales. Como los iconos son cuadrados de tamaño `32px`, se crea una imagen de `32px` x `128px`:

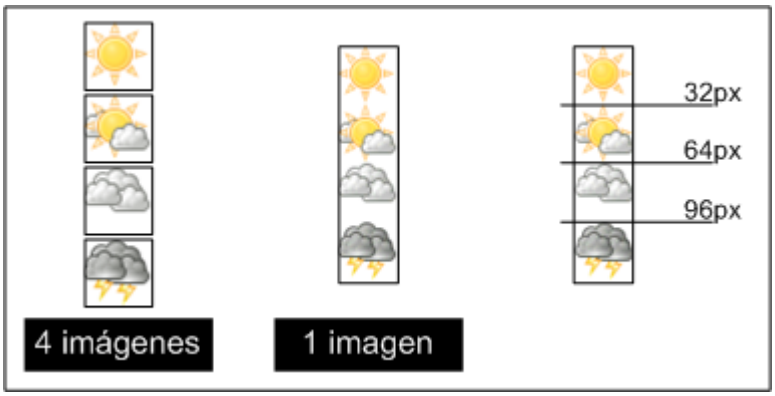


Figura 1.9. Creando un sprite de CSS a partir de varias imágenes individuales

Para facilitar el uso de esta técnica, todas las imágenes individuales ocupan el mismo sitio dentro de la imagen grande. De esta forma, los cálculos necesarios para desplazar la imagen de fondo se simplifican al máximo.

El siguiente paso consiste en simplificar el código HTML:

```
<h3>Previsiones meteorológicas</h3>
<p id="localidad1">Localidad 1: soleado, máx: 35º, mín: 23º</p>
<p id="localidad2">Localidad 2: nublado, máx: 25º, mín: 13º</p>
<p id="localidad3">Localidad 3: muy nublado, máx: 22º, mín: 10º</p>
<p id="localidad4">Localidad 4: tormentas, máx: 23º, mín: 11º</p>
```

La clave de la técnica de los *sprites CSS* consiste en mostrar las imágenes mediante la propiedad `background-image`. Para mostrar cada vez una imagen diferente, se utiliza la propiedad `background-position` que desplaza la imagen de fondo teniendo en cuenta la posición de cada imagen individual dentro de la imagen grande:

```
#localidad1, #localidad2, #localidad3, #localidad4 {
  padding-left: 38px;
  height: 32px;
  line-height: 32px;
  background-image: url("imagenes/sprite.png");
  background-repeat: no-repeat;
}

#localidad1 {
  background-position: 0 0;
}
#localidad2 {
  background-position: 0 -32px;
}
#localidad3 {
  background-position: 0 -64px;
```

```
}
#localidad4 {
  background-position: 0 -96px;
}
```

La siguiente imagen muestra lo que sucede con el segundo párrafo:

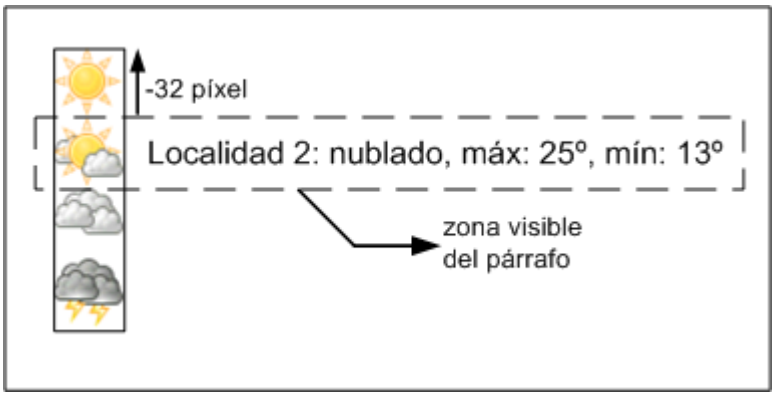


Figura 1.10. Funcionamiento de la técnica de los sprites CSS

El párrafo tiene establecida una altura de `32px`, idéntica al tamaño de los iconos. Después de añadir un `padding-left` al párrafo, se añade la imagen de fondo mediante `background-image`. Para cambiar de una imagen a otra, sólo es necesario desplazar de forma ascendente o descendente la imagen grande.

Si se quiere mostrar la segunda imagen, se desplaza de forma ascendente la imagen grande. Para desplazarla en ese sentido, se utilizan valores negativos en el valor indicado en la propiedad `background-position`. Por último, como la imagen grande ha sido especialmente preparada, se sabe que el desplazamiento necesario son `32` píxel, por lo que la regla CSS de este segundo elemento resulta en:

```
#localidad2 {
  padding-left: 38px;
  height: 32px;
  line-height: 32px;
  background-image: url("imagenes/sprite.png");
  background-repeat: no-repeat;
  background-position: 0 -32px;
}
```

La solución original utilizaba cuatro imágenes y realizaba cuatro peticiones al servidor. La solución basada en *sprites CSS* sólo realiza una conexión para descargar una sola imagen. Además, los iconos del proyecto Tango (http://tango.freedesktop.org/) que se han utilizado en este ejemplo ocupan 7.441 bytes cuando se suman los tamaños de los cuatro iconos por separado. Por su parte, la imagen grande que contiene los cuatro iconos sólo ocupa 2.238 bytes.

Los *sprites* que incluyen todas sus imágenes verticalmente son los más fáciles de manejar. Si en el ejemplo anterior se emplea un *sprite* con las imágenes dispuestas también horizontalmente:

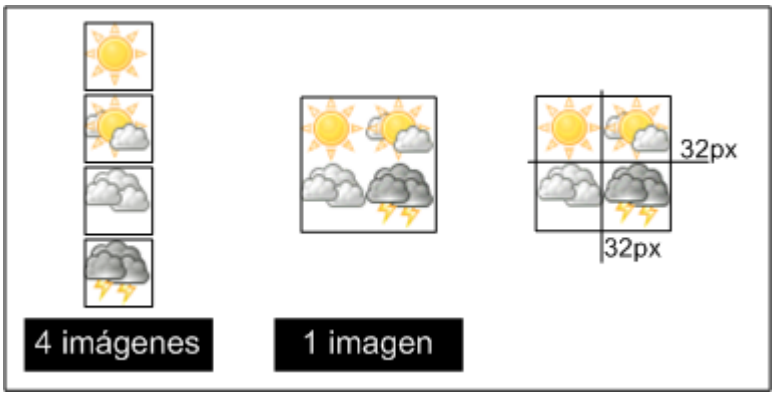


Figura 1.11. Sprite complejo que dispone las imágenes de forma vertical y horizontal

Aparentemente, utilizar este nuevo *sprite* sólo implica que la imagen de fondo se debe desplazar también horizontalmente:

```
#localidad1, #localidad2, #localidad3, #localidad4 {
  padding-left: 38px;
  height: 32px;
  line-height: 32px;
  background-image: url("imagenes/sprite.png");
  background-repeat: no-repeat;
}

#localidad1 {
  background-position: 0 0;
}
#localidad2 {
  background-position: -32px 0;
}
#localidad3 {
  background-position: 0 -32px;
}
#localidad4 {
  background-position: -32px -32px;
}
```

El problema del *sprite* anterior es que cuando una imagen tiene a su derecha o a su izquierda otras imágenes, estas también se ven:

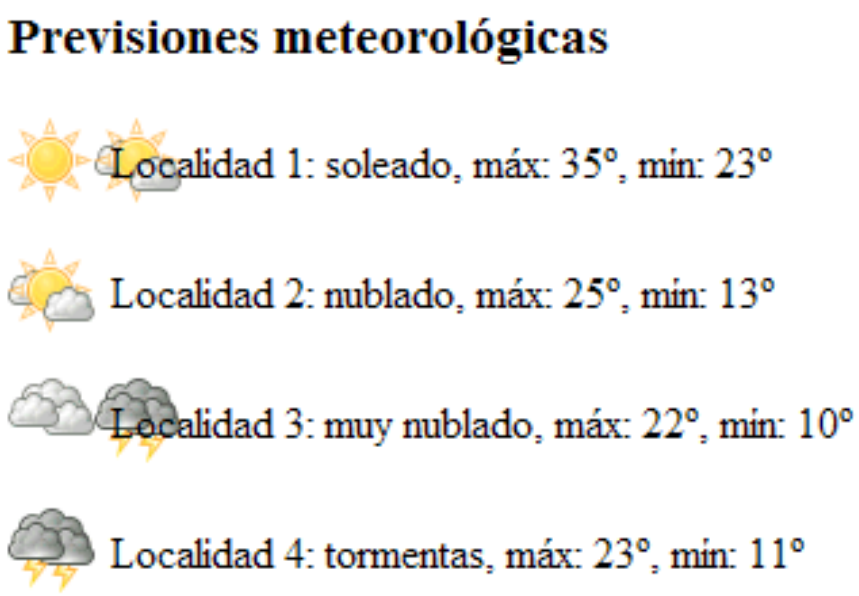


Figura 1.12. Errores producidos por utilizar un sprite complejo

La solución de este problema es sencilla, aunque requiere algún cambio en el código HTML:

```
<h3>Previsiones meteorológicas</h3>
<p id="localidad1"><img src="imagenes/pixel.gif" /> Localidad 1: soleado, máx: 35º,
mín: 23º</p>
<p id="localidad2"><img src="imagenes/pixel.gif" /> Localidad 2: nublado, máx: 25º,
mín: 13º</p>
<p id="localidad3"><img src="imagenes/pixel.gif" /> Localidad 3: muy nublado, máx: 22º,
mín: 10º</p>
<p id="localidad4"><img src="imagenes/pixel.gif" /> Localidad 4: tormentas, máx: 23º,
mín: 11º</p>
```

El código anterior muestra uno de los trucos habituales para manejar *sprites* complejos. En primer lugar se añade una imagen transparente de `1px` x `1px` a todos los elementos mediante una etiqueta `<img>`. A continuación, desde CSS se establece una imagen de fondo a cada elemento `<img>` y se limita su tamaño para que no deje ver las imágenes que se encuentran cerca:

```
#localidad1 img, #localidad2 img, #localidad3 img, #localidad4 img {
  height: 32px;
  width: 32px;
  background-image: url("imagenes/sprite2.png");
  background-repeat: no-repeat;
  vertical-align: middle;
}

#localidad1 img {
  background-position: 0 0;
}
#localidad2 img {
  background-position: -32px 0;
}
#localidad3 img {
  background-position: 0 -32px;
}
#localidad4 img {
```

```
  background-position: -32px -32px;
}
```

Utilizar *sprites CSS* es una de las técnicas más eficaces para mejorar los tiempos de descarga de las páginas web complejas. La siguiente imagen muestra un *sprite* complejo que incluye 241 iconos del proyecto Tango (http://tango.freedesktop.org/) y sólo ocupa 42 KB:

Figura 1.13. Sprite complejo que incluye 210 iconos en una sola imagen

La mayoría de sitios web profesionales utilizan *sprites* para mostrar sus imágenes de adorno. La siguiente imagen muestra el *sprite* del sitio web YouTube:

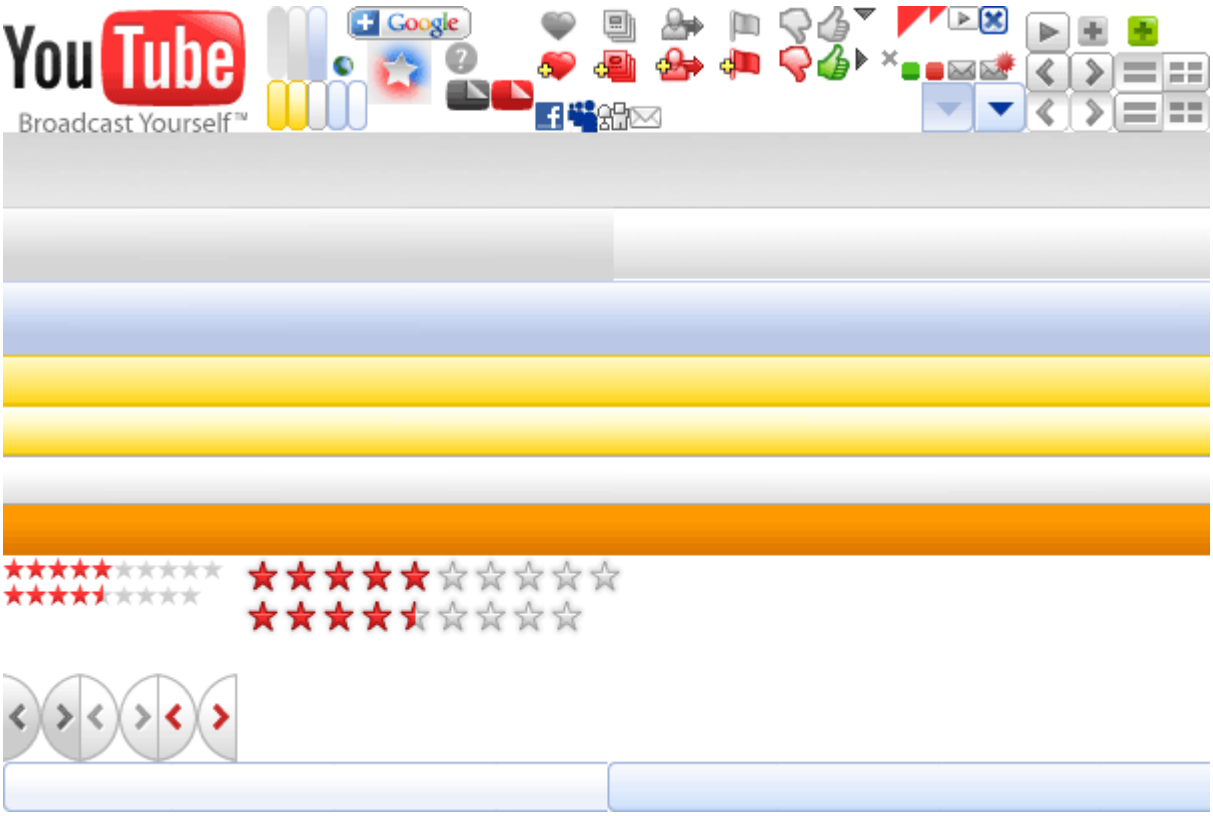


Figura 1.14. Sprite utilizado por el sitio web de YouTube

Los principales inconvenientes de los *sprites CSS* son la poca flexibilidad que ofrece (añadir o modificar una imagen individual no es inmediato) y el esfuerzo necesario para crear el *sprite*.

Afortunadamente, existen aplicaciones web como *CSS Sprite Generator* (http://spritegen.website-performance.org/) que generan el *sprite* a partir de un archivo comprimido en formato ZIP con todas las imágenes individuales.

1.11. Texto

1.11.1. Tamaño de letra

La recomendación más importante cuando se trabaja con las propiedades tipográficas de CSS está relacionada con el tamaño de la letra. La propiedad `font-size` permite utilizar cualquiera de las nueve unidades de medida definidas por CSS para establecer el tamaño de la letra. Sin embargo, la recomendación es utilizar únicamente las unidades relativas `%` y `em`.

De hecho, el documento *CSS Techniques for Web Content Accessibility Guidelines 1.0* (http://www.w3.org/TR/WCAG10-CSS-TECHS/) elaborado por el organismo W3C recomienda utilizar siempre esas unidades de medida para mejorar la accesibilidad de los contenidos web. La siguiente versión del documento (*Techniques for WCAG 2.0* (http://www.w3.org/TR/WCAG20-GENERAL/)) aún se encuentra en proceso de elaboración, pero su borrador de trabajo recoge exactamente las mismas recomendaciones en lo que se refiere al texto.

Además de mejorar la accesibilidad de los contenidos de texto, las unidades de medida relativas `%` y `em` hacen que las páginas sean más flexibles y se adapten a cualquier medio y dispositivo sin apenas tener que realizar modificaciones. Además, si se utilizan las unidades de medida relativas es posible modificar todos los tamaños de letra del sitio de forma consistente e inmediata.

Aunque todos los diseñadores web profesionales conocen esta recomendación y utilizan sólo las unidades `%` y `em` para establecer todos sus tamaños de letra, los diseñadores que comienzan a trabajar con CSS encuentran dificultades para comprender estas unidades y prefieren utilizar la unidad `px`.

Si tienes dificultades para comprender la unidad `em` y prefieres establecer los tamaños de letra mediante píxeles, puedes utilizar el siguiente truco. Como todos los navegadores establecen un tamaño de letra por defecto equivalente a `16px`, si se utiliza la siguiente regla CSS:

```
body {
  font-size: 62.5%;
}
```

El tamaño de letra del elemento `<body>`, y por tanto el tamaño de letra base del resto de elementos de la página, se establece como el `62.5%` del tamaño por defecto. Si se calcula el resultado de `16px` x `62.5%` se obtienen `10px`.

La ventaja de establecer el tamaño de letra del `<body>` de esa forma es que ahora se pueden utilizar `em` mientras se piensa en `px`. En efecto, las siguientes reglas muestran el truco en la práctica:

```
body {
  font-size: 62.5%;
}

h1 {
  font-size: 2em;   /* 2em = 2 x 10px = 20px */
}
p {
  font-size: 1.4em; /* 1.4em x 10px = 14px */
}
```

Como el tamaño base son `10px`, cualquier valor de `em` cuya referencia sea el elemento `<body>` debe multiplicarse por `10`, por lo que se puede trabajar con `em` mientras se piensa en `px`.

1.11.2. Efectos gráficos

1.11.2.1. Texto con sombra

Mostrar texto con sombra es otra de las limitaciones de CSS que más irritan a los diseñadores. En realidad, la versión CSS 2 incluía una propiedad llamada `text-shadow` para mostrar textos con sombra. La versión CSS 2.1 que utilizan todos los navegadores de hoy en día elimina esta propiedad, aunque se vuelve a recuperar en la futura versión CSS 3.

En los navegadores que más se preocupan por los estándares ya es posible utilizar la propiedad `text-shadow` de CSS 3:

```
h1 {
  color: #000;
  text-shadow: #555 2px 2px 3px;
}
```

La sintaxis de la propiedad `text-shadow` obliga a indicar dos medidas y permite establecer de forma opcional una tercera medida y un color. Las dos medidas obligatorias son respectivamente el desplazamiento horizontal y vertical de la sombra respecto del texto. La tercera medida opcional indica lo nítido o borroso que se ve la sombra y el color establece directamente el color de la sombra.

Las últimas versiones de los navegadores Firefox, Safari y Opera ya soportan la propiedad `text-shadow`, aunque no siempre de forma fiel a la descripción del futuro estándar CSS 3.

Por otra parte, el navegador Internet Explorer no incluye la propiedad `text-shadow`, pero incluye un mecanismo propio que se puede utilizar para crear un efecto parecido. Una vez más, la solución se basa en el uso de los filtros de Internet Explorer (http://msdn.microsoft.com/en-us/library/ms532853(VS.85).aspx) . Concretamente, el filtro shadow (http://msdn.microsoft.com/en-us/library/ms533086(VS.85).aspx) se puede emplear para crear una sombra sobre cualquier elemento, por ejemplo un contenido de texto.

La siguiente regla CSS muestra el filtro necesario para crear un efecto similar al del ejemplo anterior:

```
h1 {
  filter: shadow(color=#555555, direction=135, strength=2);
```

```
  zoom: 1;  /* necesario para activar la propiedad hasLayout */
}
```

1.11.2.2. Texto con relleno gradiente o degradado

Combinando el texto con imágenes semitransparentes, se pueden lograr fácilmente efectos gráficos propios de los programas de diseño. A continuación se detalla cómo crear un texto que muestra su color en forma de degradado o gradiente.

El truco consiste en mostrar por encima del texto una imagen semitransparente que simule el efecto degradado. La siguiente imagen muestra el esquema de la solución:

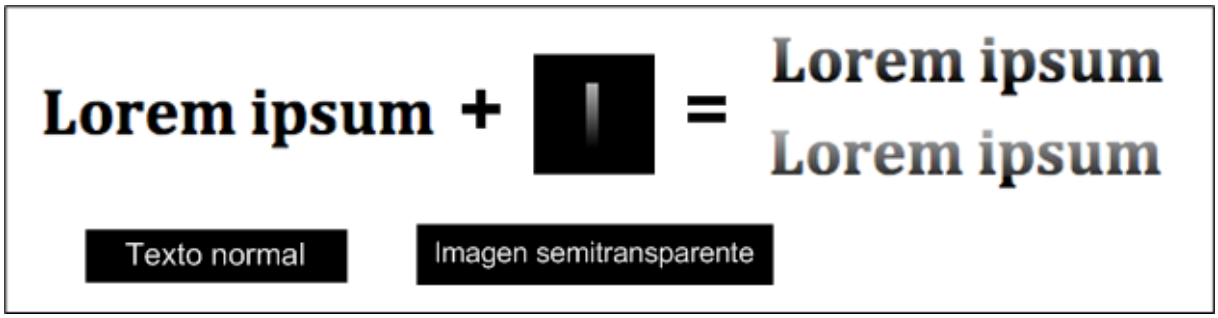


Figura 1.15. Mostrando texto avanzado gracias a una imagen semitransparente

En el esquema anterior, la imagen semitransparente se muestra en el interior de un cuadrado de color negro para poder visualizar correctamente su aspecto real.

Si se dispone por ejemplo de un titular de sección `<h1>`, el primer paso consiste en añadir un elemento HTML adicional (normalmente un `<span>`) para mostrar la imagen semitransparente:

```
<!-- Elemento original -->
<h1>Lorem Ipsum</h1>

<!-- Elemento preparado para mostrar texto avanzado -->
<h1><span></span>Lorem Ipsum</h1>
```

Una vez preparado el código HTML, el truco consiste en mostrar la imagen semitransparente como imagen de fondo del elemento `<span>`. Además, ese elemento `<span>` se muestra por delante del contenido de texto del elemento `<h1>` y ocupando toda su longitud:

```
h1 {
  position: relative;
}

h1 span {
  position: absolute;
  display: block;
  background: url("imagenes/gradiente.png") repeat-x;
  width: 100%;
  height: 31px;
}
```

Para conseguir diferentes acabados en el degradado del texto, se modifica la posición de la imagen de fondo mediante las propiedades `background` o `background-position`.

Utilizando este mismo truco pero con otras imágenes, se pueden conseguir efectos tan espectaculares como los que se pueden ver en los ejemplos del artículo *CSS Gradient Text* (http://www.webdesignerwall.com/demo/css-gradient-text/) .

Capítulo 2. Buenas prácticas

2.1. Inicializar los estilos

Cuando los navegadores muestran una página web, además de aplicar las hojas de estilo de los diseñadores, siempre aplican otras dos hojas de estilos: la del navegador y la del usuario.

La hoja de estilos del navegador se utiliza para establecer el estilo inicial por defecto a todos los elementos HTML: tamaños de letra, decoración del texto, márgenes, etc. Esta hoja de estilos siempre se aplica a todas las páginas web, por lo que cuando una página no incluye ninguna hoja de estilos propia, el aspecto con el que se muestra en el navegador se debe a esta hoja de estilos del navegador.

Por su parte, la hoja de estilos del usuario es la que puede aplicar el usuario mediante su navegador. Aunque la inmensa mayoría de usuarios no utiliza esta característica, en teoría es posible que los usuarios establezcan el tipo de letra, color y tamaño de los textos y cualquier otra propiedad CSS de los elementos de la página que muestra el navegador.

El orden en el que se aplican las hojas de estilo es el siguiente:

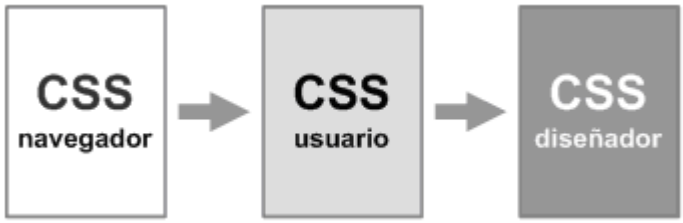


Figura 2.1. Orden en el que se aplican las diferentes hojas de estilos

Por tanto, las reglas que menos prioridad tienen son las del CSS de los navegadores, ya que son las primeras que se aplican. A continuación se aplican las reglas definidas por los usuarios y por último se aplican las reglas CSS definidas por el diseñador, que por tanto son las que más prioridad tienen.

Nota
CSS define la palabra reservada `!important` para controlar la prioridad de las declaraciones de las diferentes hojas de estilos. Las reglas CSS que incluyen la palabra `!important` tienen prioridad sobre el resto de las reglas CSS, independientemente del orden en el que se incluyan o definan las reglas.

En caso de igualdad, las reglas `!important` de los usuarios son más importantes que las reglas `!important` del diseñador. Gracias a esta característica, si un usuario sufre deficiencias visuales, puede crear una hoja de estilos CSS con reglas de tipo `!important` con la seguridad de que el navegador siempre aplicará esas reglas por encima de cualquier otra regla definida por los diseñadores.

El principal problema de las hojas de estilo de los navegadores es que los valores que aplican por defecto son diferentes en cada navegador. Aunque todos los navegadores coinciden en algunos valores importantes (tipo de letra `serif`, color de letra negro, etc.) presentan diferencias en valores tan importantes como los márgenes verticales (`margin-bottom` y `margin-top`) de los

títulos de sección (`<h1>`, ... `<h6>`), la tabulación izquierda de los elementos de las listas (`margin-left` o `padding-left` según el navegador) y el tamaño de línea del texto (`line-height`).

A continuación se muestra el código HTML de una página de ejemplo y seguidamente, una imagen que muestra cómo la visualizan los navegadores Internet Explorer y Firefox:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Reset</title>
</head>

<body>
<h1>Lorem ipsum dolor sit amet</h1>
<h2>Consectetuer adipiscing elit</h2>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
euismod tincidunt ut consectetuer adipiscing elit</p>
<ul>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>Elemento 3</li>
</ul>
<table summary="Lorem Ipsum">
  <caption>Lorem Ipsum</caption>
  <tr>
    <th>Celda 1-1</th>
    <th>Celda 1-2</th>
  </tr>
  <tr>
    <td>Celda 2-1</td>
    <td>Celda 2-2</td>
  </tr>
</table>
</body>
</html>
```

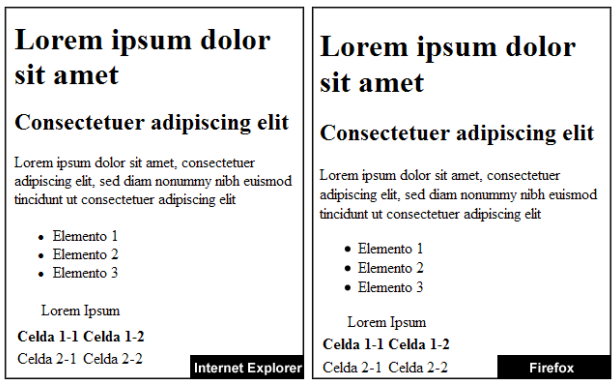


Figura 2.2. Visualización de una misma página en los navegadores Internet Explorer y Firefox

Como todas las hojas de estilo de los navegadores son diferentes, cuando un diseñador prueba sus estilos sobre diferentes navegadores, es común encontrarse con diferencias visuales

apreciables. La solución a este problema es muy sencilla y consiste en borrar o *resetear* los estilos que aplican por defecto los navegadores.

Una de las formas más sencillas de neutralizar algunos de los estilos de los navegadores consiste en eliminar el margen y relleno a todos los elementos de la página para establecerlos posteriormente de forma individual:

```
* {
  margin:  0;
  padding: 0;
}
```

Aunque la regla CSS anterior se ha utilizado desde hace muchos años, se trata de una solución muy rudimentaria y limitada. La solución completa consiste en crear una hoja de estilos CSS que neutralice todos los estilos que aplican por defecto los navegadores y que pueden afectar al aspecto visual de las páginas. Este tipo de hojas de estilos se suelen llamar *"reset CSS"*.

A continuación se muestra la hoja de estilos `reset.css` propuesta por el diseñador Eric Meyer (http://meyerweb.com/) :

```
/* v1.0 | 20080212 */

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;
  font-size: 100%;
  vertical-align: baseline;
  background: transparent;
}
body {
  line-height: 1;
}
ol, ul {
  list-style: none;
}
blockquote, q {
  quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
  content: '';
  content: none;
}
```

```
/* No olvides definir estilos para focus */
:focus {
  outline: 0;
}

/* No olvides resaltar de alguna manera el texto insertado/borrado */
ins {
  text-decoration: none;
}
del {
  text-decoration: line-through;
}

/* En el código HTML es necesario añadir cellspacing="0" */
table {
  border-collapse: collapse;
  border-spacing: 0;
}
```

El propio Eric Meyer recuerda que la hoja de estilos anterior es sólo un punto de partida que debe ser adaptado por cada diseñador hasta obtener los resultados deseados. Utilizar una hoja de estilos de tipo *reset* es una de las buenas prácticas imprescindibles para los diseñadores web profesionales.

2.2. Comprobar el diseño en varios navegadores

2.2.1. Principales navegadores

Una de las prácticas imprescindibles de los diseñadores web profesionales consiste en probar su trabajo en varios navegadores diferentes. De esta forma, el diseñador puede descubrir los errores de su trabajo y también los errores causados por los propios navegadores.

El número de navegadores y versiones diferentes que se deben probar depende de cada diseñador. En el caso ideal, el diseñador conoce estadísticas de uso de los navegadores que utilizan los usuarios para acceder al sitio o aplicación web que está diseñando. Una buena práctica consiste en probar los diseños en aquellos navegadores y versiones que sumen un 90% de cuota de mercado.

Cuando no se dispone de estadísticas de uso o el diseño es para un sitio web completamente nuevo, se debe probar el diseño en los navegadores más utilizados por los usuarios en general. Aunque no existe ninguna estadística completamente fiable y los resultados varían mucho de un país a otro, en general los siguientes navegadores y versiones suman más del 90% de cuota de mercado: Internet Explorer 6, Internet Explorer 7, Firefox 2, Firefox 3, Safari 3 y Opera 9.

En primer lugar, los diseñadores web profesionales disponen de todos los navegadores principales instalados en sus equipos de trabajo. Por lo tanto, si no lo has hecho ya, descarga e instala los siguientes navegadores:

- Firefox (http://www.mozilla.com/en-US/firefox/all.html) : disponible para sistemas operativos Windows, Mac, Linux y en más de 45 idiomas.

- Opera (http://www.opera.com/download/) : disponible para sistemas operativos Windows, Mac, Linux y en múltiples idiomas.
- Safari (http://www.apple.com/es/safari/) : disponible solamente para sistemas operativos Windows y Mac.

Respecto al navegador Internet Explorer, la mayoría de diseñadores trabajan en entornos Windows en los que ya está instalado por defecto. Si tienes la fortuna de trabajar con un sistema operativo tipo Linux, puedes instalar varias versiones de Internet Explorer mediante la aplicación IEs4Linux (http://www.tatanka.com.br/ies4linux/page/Main_Page) . También es posible instalar varias versiones de Internet Explorer en los sistemas operativos Mac OS X gracias a la aplicación ies4osx (http://www.kronenberg.org/ies4osx/) .

2.2.2. Probar el diseño en todos los navegadores

En algunas ocasiones no es suficiente con probar los diseños en los navegadores más utilizados, ya que el cliente quiere que su sitio o aplicación web se vea correctamente en muchos otros navegadores. Además, en otras ocasiones el diseñador ni siquiera dispone de los navegadores más utilizados por los usuarios, de forma que no puede probar correctamente sus diseños.

Afortunadamente, existe una aplicación web gratuita que permite solucionar todos estos problemas. La aplicación Browsershots (http://browsershots.org) prueba la página indicada en varias versiones diferentes de cada navegador, crea una imagen de cómo se ve la página en cada uno de ellos y nos muestra esa imagen.

Aunque el proceso es lento y mucho menos flexible que probar la página realmente en cada navegador, el resultado permite al diseñador comprobar si su trabajo se ve correctamente en multitud de navegadores y sistemas operativos. En la actualidad, Browsershots comprueba el aspecto de las páginas en 4 sistemas operativos y 72 navegadores diferentes.

2.2.3. Integrar Internet Explorer en Firefox

Algunos diseñadores prueban continuamente sus diseños en los navegadores Internet Explorer y Firefox y después los comprueban en el resto de navegadores para corregir los últimos errores. Si este es tu caso, puedes mejorar tu productividad gracias a una extensión de Firefox.

IE Tab (https://addons.mozilla.org/es-ES/firefox/addon/1419) es una extensión que se instala en el navegador Firefox y hace que Internet Explorer se integre en Firefox. Una vez instalada, esta extensión permite ver las páginas con Internet Explorer dentro de Firefox.

Aunque resulta sorprendente, IE Tab hace que las páginas en Firefox se puedan ver mediante Internet Explorer, de forma que los diseñadores no tienen que cambiar constantemente de navegador y por tanto aumenta considerablemente su productividad.

2.2.4. Diferentes versiones de Internet Explorer

El principal problema de los diseñadores web que quieren probar su trabajo en diferentes navegadores y versiones es la imposibilidad de instalar varias versiones del navegador Internet Explorer en el mismo sistema operativo.

Aunque la empresa Microsoft, creadora de Internet Explorer, sigue sin resolver este problema, se han publicado soluciones no oficiales para disponer de varias versiones de Internet Explorer en el mismo sistema operativo.

La primera solución propuesta fue el *Browser Archive* (http://browsers.evolt.org/) , un repositorio de navegadores desde el que se pueden descargar versiones antiguas de decenas de navegadores diferentes, entre ellos Internet Explorer. Lamentablemente hace mucho tiempo que no se añaden nuevas versiones, por lo que la última versión disponible de Internet Explorer en la 6.

Más recientemente se ha presentado IETester (http://www.my-debugbar.com/wiki/IETester/HomePage) , una aplicación descargable gratuitamente y que permite disponer de Internet Explorer 5.5, 6, 7 y 8 en un mismo sistema operativo. De esta forma, IETester es una de las herramientas imprescindibles de los diseñadores web profesionales.

2.3. Mejora progresiva

La mejora progresiva (*"progressive enhancement"*) es uno de los conceptos más importantes del diseño web y a la vez uno de los más desconocidos. Su origen proviene de su concepto contrario, la degradación útil o *"graceful degradation"*.

La degradación útil es un concepto propuesto hace décadas por el psicólogo inglés David Courtenay Marr. Aplicada al diseño web, la degradación útil significa que un sitio web sigue funcionando correctamente cuando el usuario accede con un navegador limitado o antiguo en el que no funcionan las características más avanzadas.

La mejora progresiva toma ese concepto y lo aplica de forma inversa. De esta forma, aplicada al diseño web la mejora progresiva significa que el sitio web dispone de características más avanzadas cuanto más avanzado sea el navegador con el que accede el usuario.

Muchos diseñadores web y muchos de sus clientes están obsesionados con que sus diseños se vean exactamente igual en cualquier versión de cualquier navegador. Aunque resulta prácticamente imposible conseguirlo, este tipo de diseñadores prefiere sacrificar cualquier característica interesante de CSS de manera que las páginas se vean igual en cualquier navegador.

La idea propuesta por la técnica de la mejora progresiva consiste en que el diseño web permita el acceso completo y correcto a toda la información de la página independientemente del tipo de navegador utilizado por el usuario. Además, propone utilizar las características más modernas de CSS 2 e incluso de CSS 3, aunque sólo los usuarios con navegadores más modernos sean capaces de disfrutarlas.

La técnica de la mejora progresiva es mucho mejor que las soluciones alternativas que utilizan algunos diseñadores:

- Utilizar sólo las características de CSS que soporte correctamente el navegador obsoleto Internet Explorer 6, porque un gran número de usuarios siguen utilizándolo.
- Utilizar sólo las características de CSS que soporten correctamente navegadores limitados como Internet Explorer 7, ya que es el navegador más utilizado por los usuarios.

- Olvidarse completamente de navegadores limitados como Internet Explorer 6 y 7, diseñando los sitios web sólo para los navegadores más modernos.

A continuación se muestra la mejora progresiva en la práctica mediante un ejemplo publicado en el artículo *Progressive Enhancement with CSS 3: A better experience for modern browsers* (http://dev.opera.com/articles/view/progressive-enhancement-with-css-3-a-be/) .

El propósito del ejemplo es crear un menú de navegación que se ve más bonito cuanto más moderno sea tu navegador. Como es habitual, el código HTML del menú se basa en una lista de tipo `<ul>`:

```
<ul>
  <li><a href="">Lorem Ipsum</a></li>
  <li><a href="">Lorem Ipsum</a></li>
  <li><a href="">Lorem Ipsum</a></li>
  <li><a href="">Lorem Ipsum</a></li>
  <li><a href="">Lorem Ipsum</a></li>
</ul>
```

El primer paso consiste en aplicar los estilos CSS básicos que interpretan correctamente todas las versiones de todos los navegadores. Aunque estos estilos hacen que el menú tenga un aspecto muy básico, permiten el acceso correcto a todos los contenidos.

```
ul {
  background-color: blue;
  border-bottom: 1px dotted #999;
  list-style: none;
  margin: 15px;
  width: 150px;
  padding-left: 0;
}

li {
  background-color: #FFF;
  border: 1px dotted #999;
  border-bottom-width: 0;
  font: 1.2em/1.333 Verdana, Arial, sans-serif;
}

li a {
  color: #000;
  display: block;
  height: 100%;
  padding: 0.25em 0;
  text-align: center;
  text-decoration: none;
}

li a:hover { background-color: #EFEFEF; }
```

Las reglas CSS anteriores hacen que el menú de navegación tenga el siguiente aspecto en cada navegador:

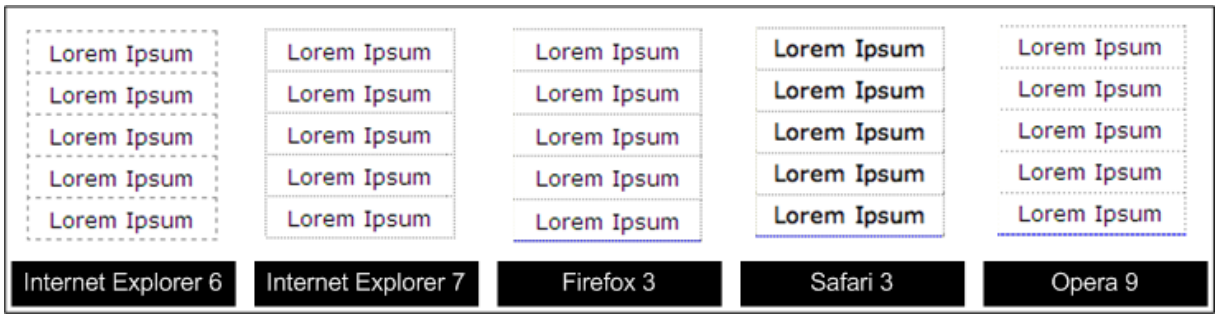


Figura 2.3. Aspecto del menú en los navegadores Internet Explorer 6 y 7, Firefox 3, Safari 3 y Opera 9

Como se ve en la imagen anterior, incluso con unos estilos CSS tan básicos se producen diferencias visuales entre los navegadores. El motivo es que Internet Explorer 6 no es capaz de mostrar un borde punteado de `1px` de anchura y lo sustituye por un borde discontinuo.

El siguiente paso consiste en utilizar el selector de hijos (uno de los selectores avanzados de CSS) para añadir nuevos estilos:

```
body > ul { border-width: 0; }

ul > li {
  border: 1px solid #FFF;
  border-width: 1px 0 0 0;
}

li > a {
  background-color: #666;
  color: white;
  font-weight: bold;
}

li:first-child a { color: yellow; }

li > a:hover{ background-color: #999; }
```

Ahora el primer elemento del menú de navegación se muestra con otro estilo y cuando el usuario pasa su ratón por encima de un elemento se muestra destacado:

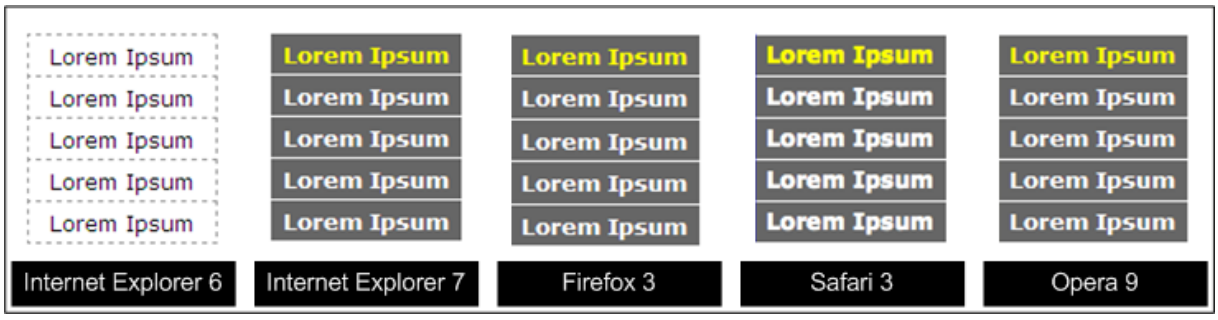


Figura 2.4. Aspecto del menú en los navegadores Internet Explorer 6 y 7, Firefox 3, Safari 3 y Opera 9

El navegador Internet Explorer 6 se queda atrás y sigue mostrando el menú con el mismo aspecto, ya que no es capaz de entender el selector de hijos. La mejora progresiva permite que los usuarios de Internet Explorer 6 sigan accediendo a todos los contenidos y que el resto de usuarios vean un menú más avanzado.

A continuación se modifica la opacidad de los elementos del menú, de forma que el elemento seleccionado se vea más claramente:

```
li { opacity: 0.9; }

li:hover{ opacity: 1; }
```

En esta ocasión, el navegador que se queda atrás es Internet Explorer 7, ya que no incluye soporte para esa propiedad de CSS. En el resto de navegadores se muestra correctamente el efecto:

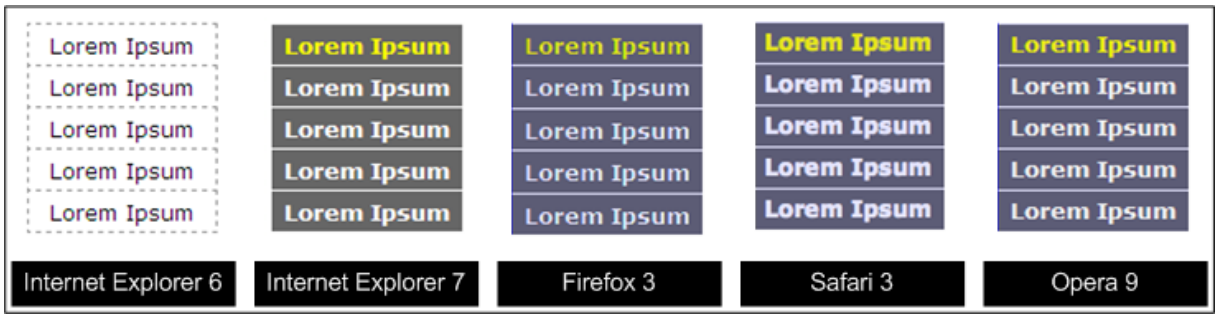


Figura 2.5. Aspecto del menú en los navegadores Internet Explorer 6 y 7, Firefox 3, Safari 3 y Opera 9

Otra posible mejora del menú consiste en añadir una sombra al texto del elemento seleccionado mediante la propiedad `text-shadow` de CSS:

```
li a:hover { text-shadow: 2px 2px 4px #333; }
```

Solamente los navegadores Safari y Opera soportan la propiedad `text-shadow`, por lo que el navegador Firefox se queda atrás y no muestra esta mejora:

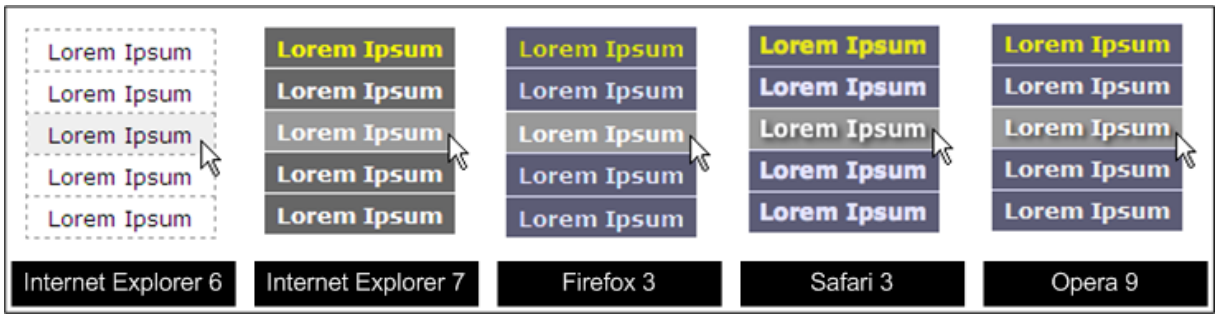


Figura 2.6. Aspecto del menú en los navegadores Internet Explorer 6 y 7, Firefox 3, Safari 3 y Opera 9

Por último, utilizando los selectores de CSS 3 se va a alternar el color de fondo de los elementos del menú para mejorar su visualización:

```
li:nth-child(2n+1) a {
  background-color: #333;
}

li:nth-child(n) a:hover {
  background-color: #AAA;
  color: #000;
}

li:first-child > a:hover{ color: yellow; }
```

Solamente los navegadores Opera y Safari incluyen todos los selectores de CSS 3, por lo que el resultado final del menú en cada navegador es el que muestra la siguiente imagen:

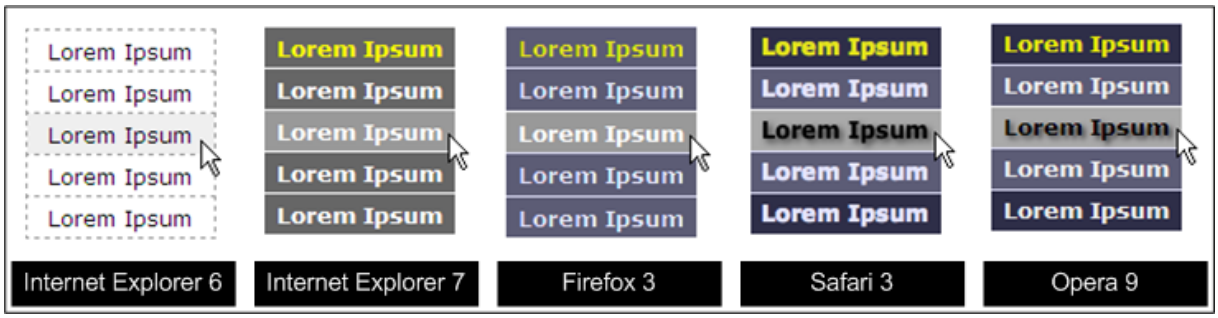


Figura 2.7. Aspecto del menú en los navegadores Internet Explorer 6 y 7, Firefox 3, Safari 3 y Opera 9

Como se ha visto en este ejemplo, la mejora progresiva permite aprovechar todas las posibilidades de CSS sin perjudicar a los navegadores obsoletos o limitados. Los usuarios de Internet Explorer 6 visualizan un menú muy básico adecuado para un navegador obsoleto, pero que les permite el acceso a todos los contenidos. Los usuarios de Internet Explorer 7 visualizan un menú normal adecuado a las limitaciones de su navegador pero que también permite el acceso a todos los contenidos. Por último, los usuarios de los navegadores más avanzados (Opera y Safari) visualizan un menú avanzado que aprovecha todas las características disponibles en CSS.

2.4. Depuración

Inevitablemente, todos los diseñadores web cometen errores en los trabajos que realizan. En la mayoría de las ocasiones, los errores se descubren al probar el diseño en diferentes navegadores. Además de mostrar los errores, los principales navegadores disponen de herramientas que permiten descubrir de forma sencilla la causa concreta del error.

Antes de que existieran estas herramientas avanzadas, el trabajo del diseñador era mucho más complicado, ya que no era fácil descubrir la causa exacta del error entre todas las posibles:

- El selector está mal escrito.
- Las propiedades están mal escritas o tienen valores no permitidos.
- Otros selectores tienen más prioridad y están sobreescribiendo una propiedad y/o valor.
- Las reglas y valores están bien escritos, pero los elementos no ocupan el espacio que a simple vista parece que están ocupando en la pantalla.
- El navegador tiene un error que impide mostrar correctamente la página.

Los diseñadores web idearon hace mucho tiempo soluciones ingeniosas para cada uno de los problemas anteriores. En primer lugar, cuando no se está seguro de si todas las reglas CSS están bien escritas, lo mejor es validar la hoja de estilos utilizando el validador CSS del W3C (http://jigsaw.w3.org/css-validator/) .

Una vez descartado el error de sintaxis, el siguiente problema a resolver es por qué una regla CSS no se aplica correctamente a un elemento. Una estrategia muy utilizada consistía en añadir alguna propiedad que sea visualmente significativa para comprobar si realmente el selector se

está aplicando. Poner todo el texto del elemento en negrita, aumentar mucho su tamaño de letra y cambiar el color de fondo eran algunas de las estrategias habituales. Cuando lo anterior no resultaba, se utilizaba directamente la palabra reservada `!important` para aumentar la prioridad de esa propiedad CSS.

Otro de los problemas habituales en el diseño web está relacionado con el espacio que ocupa cada elemento en pantalla. Como los elementos por defecto no muestran ningún borde y su color de fondo es transparente, no es posible conocer a simple vista el espacio que ocupa cada elemento. Por lo tanto, cuando se posicionan elementos de forma flotante o cuando se establecen márgenes, rellenos, alturas y anchuras máximas/mínimas, no es posible visualizar si el navegador está mostrando correctamente todos los elementos.

Para solucionar este problema la técnica habitual consistía en añadir un borde visible a los elementos. Como los bordes visibles ocupan sitio en pantalla, el problema de esta solución es que modifica el propio diseño. La alternativa consistía en añadir un color de fondo diferente para cada elemento. Otra posible alternativa es el uso de la propiedad `outline` de CSS, que añade un perfil en el contorno de un elemento pero no ocupa sitio y por tanto no modifica el diseño de la página.

Los navegadores modernos como Safari, Opera y Firefox incluyen el soporte de la propiedad `outline`, mientras que el navegador Internet Explorer 7 no es capaz de mostrar perfiles en los elementos de la página. El diseñador Chris Page ha publicado un artículo llamado *A Handy CSS Debugging Snippet* (http://homepage.mac.com/chrispage/iblog/C42511381/E20060806095030/index.html) en el que muestra unas reglas CSS que hacen uso de `outline` y permiten depurar fácilmente cualquier diseño sin utilizar herramientas avanzadas:

```
* { outline: 2px dotted red }
* * { outline: 2px dotted green }
* * * { outline: 2px dotted orange }
* * * * { outline: 2px dotted blue }
* * * * * { outline: 1px solid red }
* * * * * * { outline: 1px solid green }
* * * * * * * { outline: 1px solid orange }
* * * * * * * * { outline: 1px solid blue }
```

2.4.1. Firebug

Al margen de soluciones manuales y técnicas más o menos ingeniosas, los diseñadores web profesionales de hoy en día utilizan herramientas avanzadas para averiguar con precisión la causa de los errores de diseño. De todas las herramientas disponibles, la mejor con mucha diferencia es Firebug (https://addons.mozilla.org/es-ES/firefox/addon/1843) , una extensión del navegador Firefox.

Firebug dispone de todas las utilidades que necesitan los diseñadores y programadores web en su trabajo. Firebug es una herramienta gratuita, completa, fácil de utilizar y para la que se publican nuevas versiones de forma continua.

Después de instalar Firebug, en la esquina inferior derecha del navegador Firefox aparece un nuevo icono. Para abrir Firebug, solamente hay que pinchar con el ratón sobre ese icono o pulsar directamente la tecla `F12` después de cargar la página que se quiere depurar.

Firebug muestra su información dividida en los siguientes paneles:

- Consola: muestra mensajes de error, notificaciones y otros tipos de mensajes. No es muy útil para los diseñadores web.
- HTML: muestra el código HTML de la página y permite seleccionar los elementos, modificar sus contenidos y ver las reglas CSS que se le están aplicando.
- CSS: muestra todas las hojas de estilos incluidas en la página y permite modificar sus valores.
- Guión y DOM: paneles relacionados con la programación JavaScript. No son muy útiles para los diseñadores web.
- Red: muestra toda la información de todos los elementos descargados por la página (HTML, JavaScript, CSS, imágenes).

El primero de los paneles importantes para los diseñadores web es el panel HTML:

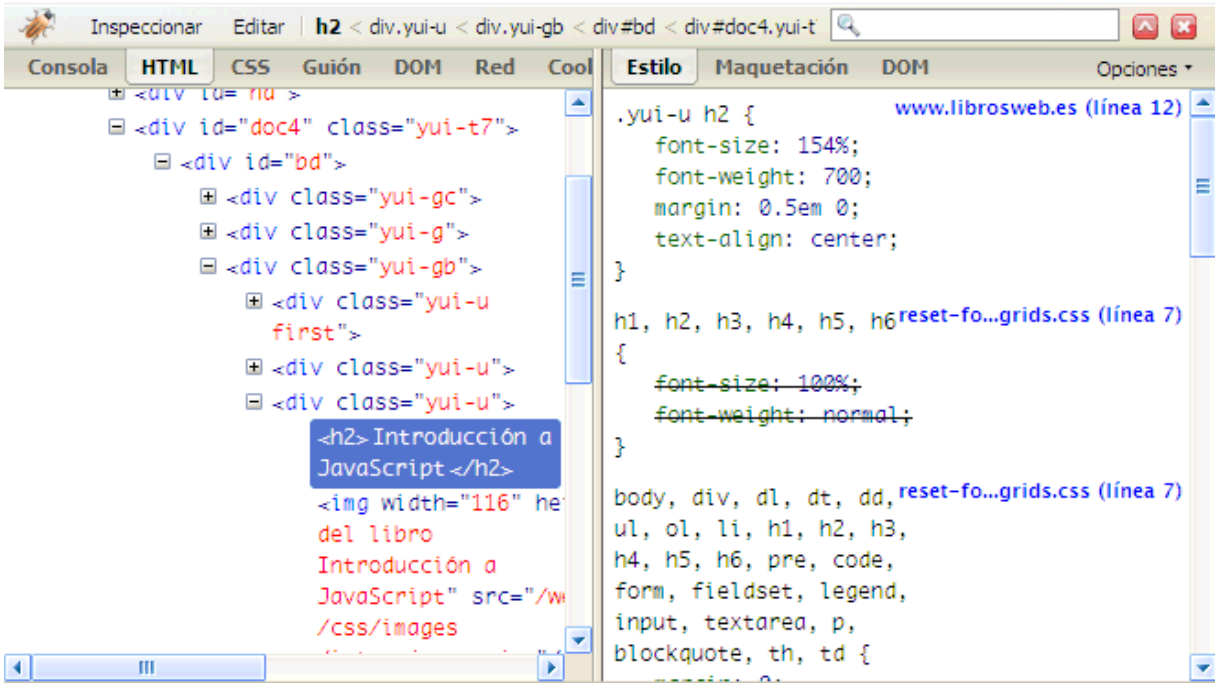


Figura 2.8. Panel HTML de Firebug

El panel HTML es el más utilizado por los diseñadores web, ya que muestra toda la información de la página relacionada con HTML y CSS. En la parte izquierda del panel se muestra el código HTML de la página y en la parte derecha la información CSS.

Si se pulsa el botón "Inspeccionar" de la parte superior de Firebug, es posible seleccionar con el ratón un elemento de la página web. Después de pinchar sobre cualquier elemento, en la parte izquierda se muestra el código HTML de ese elemento y en la parte derecha todas las reglas CSS que se le aplican.

Si se pulsa sobre el contenido de un elemento en la parte izquierda del panel HTML, se puede modificar su valor y ver el resultado en tiempo real sobre la propia página web. Desde este mismo panel también es posible eliminar el elemento de la página.

La parte derecha del panel HTML es la más útil, ya que siempre muestra todas las reglas CSS que se aplican a un elemento de la página. Gracias a esta información es imposible dudar si un selector está bien escrito o si una regla CSS realmente se está aplicando a un elemento.

Además, como normalmente varias reglas CSS diferentes aplican valores diferentes a las mismas propiedades de un mismo elemento, Firebug muestra tachadas todas las propiedades que en teoría se deben aplicar al elemento pero que no lo hacen porque existen otras reglas CSS con más prioridad.

La parte derecha del panel HTML incluye otras utilidades interesantes como cuando se pasa el ratón por encima de un color definido en formato hexadecimal y que hace que se vea realmente cuál es el color. Igualmente, al pasar el ratón por encima de una `url()` utilizada para incluir una imagen, Firebug muestra de qué imagen se trata. Las reglas CSS que se muestran en la parte derecha del panel HTML también se pueden modificar, eliminar y bloquear temporalmente. También es posible añadir nuevas propiedades a cualquier regla CSS.

Firebug muestra por defecto el valor de las reglas CSS tal y como se han establecido en las hojas de estilos. Sin embargo, muchas veces estos valores originales no son prácticos. ¿cuál es el tamaño de letra de un elemento con `font-size: 1em`? Sin más información es imposible saberlo. ¿cuál es la anchura en píxeles de un elemento con la propiedad `width: 60%`? Imposible saberlo sin conocer las anchuras de sus elementos contenedores.

Por todo ello, Firebug permite mostrar los valores que realmente utiliza Firefox para dibujar cada elemento en pantalla. Pulsando sobre el texto `Opciones` de la parte derecha del panel HTML, se puede activar la opción `Show Computed Style`:

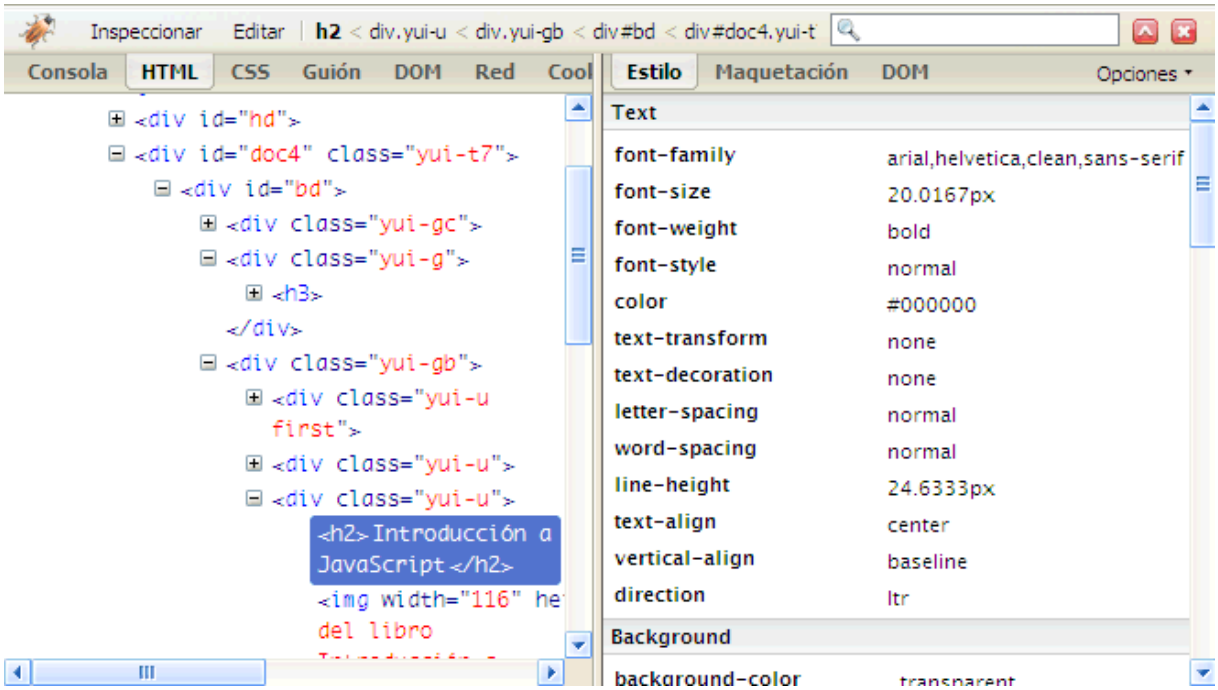


Figura 2.9. Panel HTML de Firebug con la opción Show Computed Style

Después de activar esta opción, los tamaños de letra y anchuras se muestran en píxeles y se muestra el valor de todas las propiedades CSS del elemento, independientemente de si se han establecido de forma explícita o de si se trata de los valores por defecto que aplica el navegador.

Otra de las utilidades más interesantes del panel HTML es la información sobre la maquetación del elemento, que se puede mostrar pinchando sobre la pestaña `Maquetación` de la parte derecha del panel:

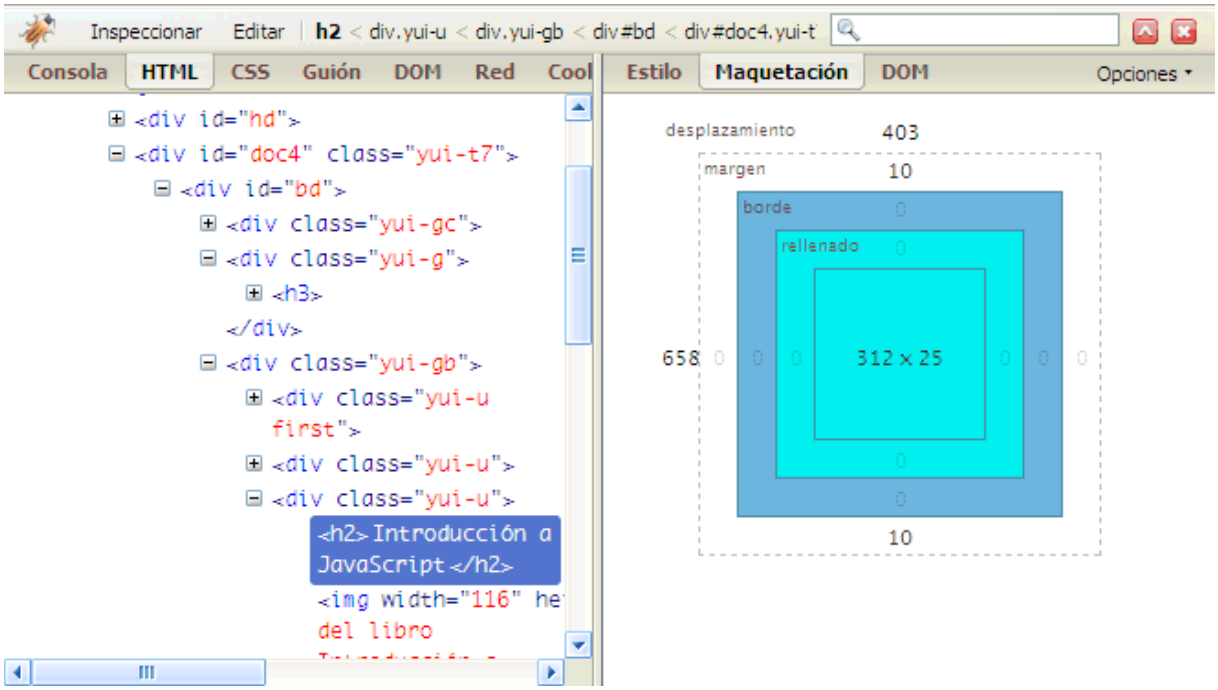


Figura 2.10. Pestaña Maquetación del panel HTML de Firebug

La opción `Maquetación` muestra la información completa del *"box model"* o modelo de cajas de un elemento: anchura, altura, rellenos, bordes y márgenes.

El otro panel más utilizado por los diseñadores web es el panel CSS, que muestra el contenido de todas las hojas de estilos que se están aplicando en la página y permite realizar cualquier modificación sobre cualquier regla CSS viendo el resultado en tiempo real en la propia página:

Figura 2.11. Panel CSS de Firebug

Por último, el panel Red de Firebug permite ver toda la información sobre todos los elementos que se descarga el navegador para mostrar la página:

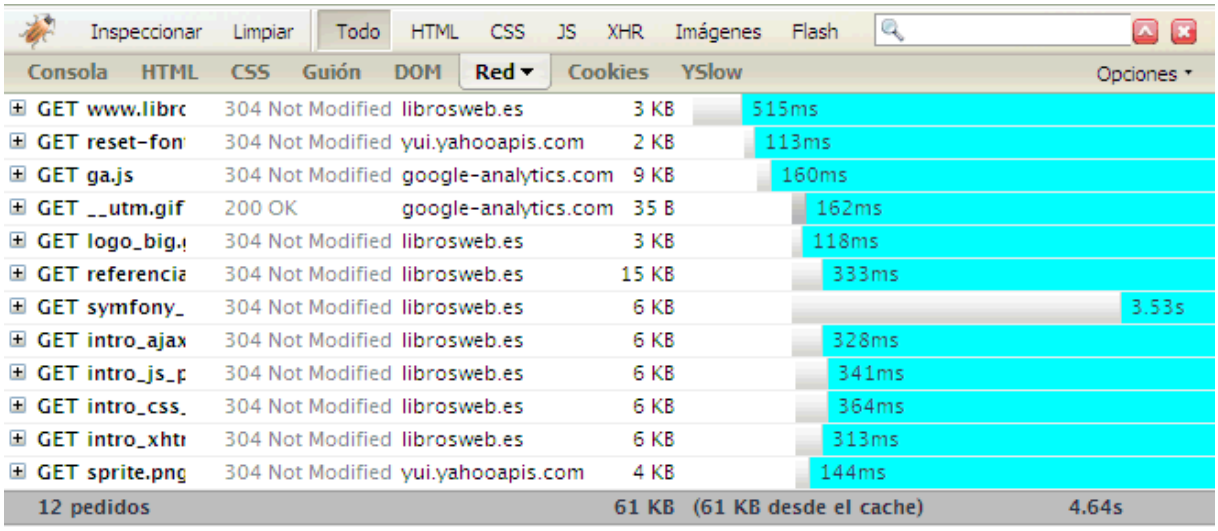


Figura 2.12. Panel Red de Firebug

Desde el punto de vista del diseñador, este panel se puede utilizar para mejorar el rendimiento de la página reduciendo el número de archivos CSS, reduciendo el número de imágenes de adorno mediante los *sprites CSS*, reduciendo el número de peticiones HTTP realizadas al servidor y reduciendo el tamaño de los archivos.

2.4.2. Otras herramientas de depuración

Firebug es la mejor herramienta para depurar el diseño de los sitios web, pero sólo está disponible para el navegador Firefox. Como la mayoría de errores en el diseño web sólo se producen en los navegadores de la familia Internet Explorer, Firebug no se puede utilizar.

Afortunadamente, los creadores de Firebug han publicado una versión reducida y simplificada de Firebug compatible con el resto de navegadores. La versión reducida se denomina Firebug Lite (http://getfirebug.com/lite.html) y requiere el uso de JavaScript. Aunque se puede descargar Firebug Lite para utilizarlo desde nuestros propios servidores, la forma más sencilla de probarla es añadir el siguiente código en la página que se quiere depurar:

```
<script type="text/javascript" src="http://getfirebug.com/releases/lite/1.2/
firebug-lite-compressed.js"></script>
```

El código anterior se puede colocar en cualquier zona de la página, aunque normalmente se incluye dentro de la sección `<head>`.

Por último, como las empresas que desarrollan los navegadores consideran que Firebug es insuperable, desde hace un tiempo se dedican a copiar todas sus características. Internet Explorer 8, Safari 3 y Opera 9 disponen de herramientas de depuración que son una réplica de Firebug.

Internet Explorer 8 ha denominado a su herramienta *Developer Tools* (http://msdn.microsoft.com/en-us/library/cc848894(VS.85).aspx) , mientras que Opera la ha denominado *Dragonfly* (http://www.opera.com/products/dragonfly/) y en Safari está disponible desde el menú `Desarrollo`.

2.5. Hojas de estilos

Las hojas de estilos reales de los sitios web profesionales suelen contener cientos de reglas y ocupan miles de líneas. Por este motivo, es imprescindible seguir unas buenas prácticas al crear las hojas de estilos para mejorar la producitivdad y reducir los posibles errores. A continuación se muestran algunas de las recomendaciones más útiles para crear hojas de estilos profesionales.

2.5.1. Llaves

Uno de los elementos básicos que los diseñadores web deben acordar es el tipo de llaves que se utilizan para encerrar la declaración de cada regla CSS. Aunque se utilizan muchos modelos diferentes, a continuación se muestran los más populares.

Llave de apertura en la misma línea del selector y llave de cierre en una nueva línea para separar unas reglas de otras:

```
selector {
  propiedad1: valor1;
  propiedad2: valor2;
}
```

Una variante del modelo anterior consiste en mostrar cada llave en su propia línea, para separar aún más el selector de su declaración. Este modelo lo utilizan normalmente los programadores web:

```
selector
{
  propiedad1: valor1;
```

```
  propiedad2: valor2;
}
```

Por último, existe un modelo compacto que no crea nuevas líneas para las llaves. Aunque es mucho más compacto, normalmente es más difícil de leer:

```
selector {
  propiedad1: valor1;
  propiedad2: valor2; }
```

2.5.2. Tabulaciones

Tabular el código facilita significativamente su lectura, por lo que tabular las propiedades CSS es una de las mejores prácticas de los diseñadores web profesionales. Como conocen la mayoría de programadores, no es recomendable insertar tabuladores en el código, sino que se deben emplear 2 o 4 espacios en blanco.

```
/* Propiedades sin tabular */
selector {
propiedad1: valor1;
propiedad2: valor2;
}

/* Propiedades tabuladas con 2 espacios */
selector {
  propiedad1: valor1;
  propiedad2: valor2;
}
```

Extendiendo la práctica de tabular las propiedades, algunos diseñadores recomiendan tabular también las reglas CSS relacionadas. El siguiente ejemplo muestra tres reglas CSS relacionadas entre sí:

```
ul {
  margin: 1em 0;
  padding: 0;
}
ul li {
  list-style-type: square;
}
ul li ul {
  font-style: italic;
  list-style-type: disc;
}
```

La recomendación consiste en tabular las reglas CSS que están relacionadas porque sus selectores están anidados. Por tanto, la regla cuyo selector es `ul li` debería estar tabulada respecto de la regla `ul` y de la misma forma la regla cuyo selector es `ul li ul` debería estar tabulada respecto de `ul li`:

```
ul {
  margin: 1em 0;
  padding: 0;
}
```

```
  ul li {
    list-style-type: square;
  }
    ul li ul {
      font-style: italic;
      list-style-type: disc;
    }
```

2.5.3. Propiedades

Cuando se establece la misma propiedad en varios selectores diferentes pero relacionados, se recomienda escribir las reglas CSS en una única línea, para facilitar su lectura:

```
#contenedor #principal h1 { font-size: 2em;   }
#contenedor #principal h2 { font-size: 1.8em; }
#contenedor #principal h3 { font-size: 1.6em; }
```

Cuando en el caso anterior las propiedades y/o selectores no ocupan el mismo sitio, se pueden utilizar espacios en blanco para crear una estructura similar a las columnas de una tabla:

```
h1                  { color: #000;                      text-align: center;  }
#principal h2       { color: #C0C0C0; font-size: 1.8em;                      }
#lateral blockquote {                 font-size: 0.9em; text-align: justify; }
```

Respecto al orden en el que se indican las propiedades en la declaración, algunos diseñadores recomiendan agruparlas por su funcionalidad:

```
selector {
  position: absolute;  /* propiedades de posicionamiento */
  right: 0;
  bottom: 10px;

  width: 300px;        /* propiedades de tamaño */
  height: 250px;

  color: #000;        /* propiedades tipográficas */
  font-size: 2em;
}
```

Otros diseñadores recomiendan ordenar alfabéticamente las propipedades para facilitar su búsqueda y evitar duplicidades:

```
selector {
  bottom: 10px;
  color: #000;
  font-size: 2em;
  height: 250px;
  position: absolute;
  right: 0;
  width: 300px;
}
```

2.5.4. Selectores

Los selectores deben ser descriptivos para facilitar la lectura de la hoja de estilos, por lo que hay que poner especial cuidado en elegir el nombre de los atributos `id` y `class`. Además, aunque aumenta el tamaño total de la hoja de estilos, se recomienda utilizar selectores lo más específicos posible para facilitar el mantenimiento de la hoja de estilos:

```
p.especial { ... }                          /* poco específico */
#contenedor #principal p.especial { ... }  /* muy específico */
```

Por otra parte, cuando una regla CSS tiene varios selectores largos, es mejor colocar cada selector en su propia línea:

```
#contenedor h1,
#contenedor #principal h2,
#contenedor #lateral blockquote {
  color: #000;
  font-family: arial, sans-serif;
}
```

2.5.5. Organización

Cuando una hoja de estilos tiene cientos de reglas, es recomendable dividirla en secciones para facilitar su mantenimiento. Aunque no existe ninguna organización seguida por todos los diseñadores, en general se utilizan las siguientes secciones:

- Estilos básicos (estilos de `<body>`, tipo de letra por defecto, márgenes de `<ul>`, `<ol>` y `<li>`, estilos de los enlaces, etc.)
- Estilos de la estructura o *layout* (anchura, altura y posición de la cabecera, pie de página, zonas de contenidos, menús de navegación, etc.)
- Estilos del menú de navegación
- Estilos de cada una de las zonas (elementos de la cabecera, titulares y texto de la zona de contenidos, enlaces, listas e imágenes de las zonas laterales, etc.)

Si la hoja de estilos tiene muchas secciones, algunos diseñadores incluyen un índice o tabla de contenidos al principio del todo. En la hoja de estilos del sitio web mozilla.org (http://www.mozilla.org/css/base/content.css) utilizan la siguiente tabla de contenidos:

```
/* TOC:
   Random HTML Styles
     Forms
   General Structure
   Navigation
   Quotations
   Comments and Other Asides
   Emphasis
   Computers - General
   Code
   Examples and Figures
   Q and A (FAQ)
   Tables
```

```
   Headers
   Meta
   Specific to Products Pages
*/
```

Otra recomendación relacionada con la organización de las hojas de estilos es la de utilizar siempre los mismos nombres para los mismos elementos. Si observas las hojas de estilos de los diseñadores profesionales, verás que siempre llaman `#cabecera` o `#header` o `#hd` a la cabecera de la página, `#contenidos` o `#principal` a la zona principal de contenidos y así sucesivamente.

Algunos de los atributos `id` más utilizados en los diseños web son: `cabecera`, `cuerpo`, `pie`, `contenidos`, `principal`, `secundario`, `lateral`, `buscador`, `contacto`, `logo`.

2.5.6. Comentarios

Separar las secciones de la hoja de estilos y otros bloques importantes es mucho más fácil cuando se incluyen comentarios. Por este motivo se han definido decenas de tipos diferentes de comentarios separadores.

El modelo básico sólo añade un comentario destacado para el inicio de cada sección importante:

```
/* ----------------------------------------*/
/* ---------->>> CONTENEDOR <<<-----------*/
/* ----------------------------------------*/
```

Otros diseñadores emplean diferentes comentarios para indicar el comienzo de las secciones y el de las partes importantes dentro de una sección:

```
/*  ------------------------------------------------------------
  CABECERA
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

/*  Logotipo
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

/*  Buscador
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

Combinando los comentarios y la tabulación de reglas, la estructura de la hoja de estilos está completamente ordenada:

```
/*  ------------------------------------------------------------
  CABECERA
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
#cabecera {
  ...
}

  /*  Logotipo
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  #cabecera #logo {
    ...
  }
```

```
/*  Buscador
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
#cabecera #buscador {
  ...
}
```

2.6. Rendimiento

Antes de su explicación detallada, se muestra a continuación la lista de estrategias útiles para mejorar el rendimiento de la parte de CSS de las páginas web:

- Utilizar *sprites CSS* para reducir el número de imágenes de adorno a una única imagen.
- No utilizar expresiones (`expression()`) en las hojas de estilos. El navegador Internet Explorer revalúa continuamente el valor de las expresiones y puede penalizar el rendimiento de la página.
- No utilizar los filtros de Internet Explorer, ya que algunos filtros como `AlphaImageLoader` bloquean la carga de la página hasta que no se descarga la imagen utilizada por el filtro.
- Enlazar hojas de estilos externas en vez de incluir los estilos en la propia página.
- Enlazar las hojas de estilos mediante `<link>` en vez de `@import` (en Internet Explorer las reglas `@import` tienen el mismo efecto que enlazar los archivos CSS al final de la página).
- Reducir el número de archivos CSS de la página.
- Combinar si es posible todos los archivos CSS individuales en un único archivo CSS.
- Reducir el tamaño de las hojas de estilos comprimiendo los archivos con las herramientas disponibles (CSS Tidy, YUI Compressor).
- Comprimir los archivos CSS en el servidor web antes de enviarlos al usuario.

Según los estudios realizados por Yahoo! y publicados en el artículo *Performance Research, Part 1: What the 80/20 Rule Tells Us about Reducing HTTP Requests* (http://yuiblog.com/blog/2006/11/28/performance-research-part-1/) , del tiempo total que el usuario espera hasta que la página solicitada se carga completamente, el 20% del tiempo corresponde a la parte del servidor (normalmente generar la página HTML de forma dinámica utilizado información de una base de datos) y el 80% restante corresponde a la parte del cliente (normalmente descargar hojas de estilos CSS, archivos JavaScript e imágenes).

De esta forma, es mucho más fácil mejorar el tiempo de respuesta de la página mejorando el rendimiento de la parte del cliente. Como uno de los principales elementos de la parte del cliente está formado por las hojas de estilos CSS, a continuación se indican algunas de las técnicas para mejorar su rendimiento.

En primer lugar, en los estudios realizados por Yahoo! se demuestra que aproximadamente el 20% de las páginas vistas de un sitio web no disponen de sus elementos guardados en la cache del navegador del usuario. Esto supone que en el 20% de las páginas vistas, los navegadores de los usuarios se descargan todos sus elementos: imágenes, archivos JavaScript y hojas de estilos CSS.

La conclusión de lo anterior es que resulta mucho mejor reducir el número de archivos diferentes y no reducir el tamaño de cada archivo individual. El objetivo es reducir el número de peticiones HTTP que la página realiza al servidor para descargar todos los contenidos.

Los navegadores limitan el número de peticiones HTTP simultáneas que se pueden realizar a un mismo servidor. Aunque el número máximo de peticiones varía de un navegador a otro, el límite se encuentra entre 2 y 8. Por tanto, si el límite del navegador son 2 peticiones HTTP simultáneas, cuando el navegador realice 2 peticiones, la descarga de la página se bloquea hasta que alguna de las peticiones concluya.

Como demuestran los estudios realizados por Yahoo!, la descarga de las páginas web se realiza *a saltos*, descargando cada vez 2, 4 o 6 elementos de la página. Por este motivo es fundamental reducir el número de elementos de la página, entre ellos el número de hojas de estilos CSS, para reducir el número de peticiones HTTP necesarias.

La recomendación de reducir el número de hojas de estilos CSS entra en conflicto con la recomendación de modularizar el diseño CSS del sitio web separando de forma lógica todos sus estilos en varios archivos individuales.

La solución consiste en combinar todos los archivos CSS individuales en un único archivo CSS. Aunque la hoja de estilos resultante tiene un tamaño muy grande, el número de peticiones HTTP requeridas para la parte de CSS se reduce al mínimo posible.

El siguiente paso consiste en reducir el tamaño de esa única hoja de estilos grande. Como los archivos CSS normales incluyen muchos espacios en blanco, nuevas líneas, comentarios y propiedades no optimizadas, es muy sencillo reducir su tamaño de forma considerable eliminando todos los elementos sobrantes.

Yahoo! ha desarrollado una herramienta llamada *YUI compressor* (http://developer.yahoo.com/yui/compressor/) que permite reducir de forma segura el tamaño de los archivos JavaScript y CSS. Su uso no es muy cómodo para los diseñadores web, ya que requiere el uso de Java y la línea de comandos.

Otra herramienta similar a la anterior es *CSSTidy* (http://csstidy.sourceforge.net/) , que permite reducir el tamaño de los archivos CSS eliminando los elementos sobrantes (comentarios, espacios en blanco, nuevas líneas) y optimizando las reglas CSS (sustituir propiedades individuales por propiedades *shorthand*, utilizar la notación abreviada de números y colores, etc.) *CSS Tidy* es una herramienta gratuita disponible para todos los sitemas operativos, se puede utilizar mediante la línea de comandos y también se puede integrar con el lenguaje de programación PHP.

Además de las herramientas descargables, existen aplicaciones online que permiten reducir fácilmente el tamaño del código CSS indicado. Una de las aplicaciones más conocidas es *Clean CSS* (http://www.cleancss.com/) , que utiliza internamente *CSS Tidy*.

2.7. Recursos imprescindibles

El trabajo de los diseñadores web se visualiza a través de un navegador web la mayoría de las ocasiones. Por este motivo, los diseñadores siempre están limitados por las capacidades de los

navegadores. Además, los diseñadores también están expuestos a los errores que comete cada versión de cada navegador.

De esta forma, es imprescindible disponer de recursos para conocer las limitaciones y errores de cada navegador. A continuación se indica la localización de la documentación técnica oficial de cada navegador.

Información técnica general para diseñadores y programadores web:

- Internet Explorer (http://msdn.microsoft.com/en-us/ie/default.aspx)
- Firefox (http://developer.mozilla.org/en/Main_Page)
- Safari (http://developer.apple.com/internet/safari/)
- Opera (http://www.opera.com/docs/specs/)

Información técnica específica de CSS:

- Internet Explorer (http://msdn.microsoft.com/en-us/library/cc351024(VS.85).aspx)
- Firefox (http://developer.mozilla.org/en/CSS_Reference)
- Safari (http://developer.apple.com/internet/safari/safari_css.html)
- Opera (http://www.opera.com/docs/specs/css/)

Al margen de las referencias oficiales, existen sitios como http://positioniseverything.net/ que publican artículos técnicos sobre los errores más conocidos de cada navegador.

Capítulo 3. Selectores

Conocer y dominar todos los selectores de CSS es imprescindible para crear diseños web profesionales. El estándar de CSS 2.1 incluye una docena de tipos diferentes de selectores, que permiten seleccionar de forma muy precisa elementos individuales o conjuntos de elementos dentro de una página web.

Utilizando solamente los cinco selectores básicos de CSS 2.1 (universal, de tipo, descendente, de clase y de id) es posible diseñar cualquier página web. No obstante, los selectores avanzados de CSS 2.1 permiten simplificar las reglas CSS y también el código HTML.

Desafortunadamente, los navegadores obsoletos como Internet Explorer 6 y sus versiones anteriores no soportan este tipo de selectores avanzados, por lo que su uso no era común hasta hace poco tiempo. Hoy en día, todos los navegadores más utilizados soportan los selectores avanzados de CSS 2.1 y algunos de ellos también soportan la mayoría o todos los selectores propuestos por la futura versión CSS 3.

3.1. Selector de hijos

Se trata de un selector similar al selector descendente, pero muy diferente en su funcionamiento. Se utiliza para seleccionar un elemento que es *hijo* de otro elemento y se indica mediante el *"signo de mayor que"* (>).

Mientras que en el selector descendente sólo importa que un elemento esté dentro de otro, independientemente de lo profundo que se encuentre, en el selector de hijos el elemento debe ser *hijo directo* de otro elemento.

```
p > span { color: blue; }

<p>
  <span>Texto1</span>
</p>

<p>
  <a href="#">
    <span>Texto2</span>
  </a>
</p>
```

En el ejemplo anterior, el selector `p > span` se interpreta como *"cualquier elemento que sea hijo directo de un elemento <p>"*, por lo que el primer elemento `<span>` cumple la condición del selector. Sin embargo, el segundo elemento `<span>` no la cumple porque es descendiente pero no es hijo directo de un elemento `<p>`.

Utilizando el mismo ejemplo anterior se pueden comparar las diferencias entre el selector descendente y el selector de hijos:

```
p a { color: red; }
p > a { color: red; }
```

```
<p>
  <a href="#">Enlace1</a>
</p>

<p>
  <span>
    <a href="#">Enlace2</a>
  </span>
</p>
```

El primer selector es de tipo descendente (`p a`) y por tanto se aplica a todos los elementos `<a>` que se encuentran dentro de elementos `<p>`. En este caso, los estilos de este selector se aplican a los dos enlaces.

El segundo selector es de hijos (`p > a`) por lo que obliga a que el elemento `<a>` sea hijo directo de un elemento `<p>`. Por tanto, los estilos del selector `p > a` no se aplican al segundo enlace del ejemplo anterior.

3.2. Selector adyacente

El selector adyacente se emplea para seleccionar elementos que son *hermanos* (su elemento padre es el mismo) y están seguidos en el código HTML. Este selector emplea en su sintaxis el símbolo +. Si se considera el siguiente ejemplo:

```
h1 + h2 { color: red }

<body>
<h1>Titulo1</h1>

<h2>Subtítulo</h2>
...

<h2>Otro subtítulo</h2>
...
</body>
```

Los estilos del selector `h1 + h2` se aplican al primer elemento `<h2>` de la página, pero no al segundo `<h2>`, ya que:

- El elemento padre de `<h1>` es `<body>`, el mismo padre que el de los dos elementos `<h2>`. Así, los dos elementos `<h2>` cumplen la primera condición del selector adyacente.
- El primer elemento `<h2>` aparece en el código HTML justo después del elemento `<h1>`, por lo que este elemento `<h2>` también cumple la segunda condición del selector adyacente.
- Por el contrario, el segundo elemento `<h2>` no aparece justo después del elemento `<h1>`, por lo que no cumple la segunda condición del selector adyacente y por tanto no se le aplican los estilos de `h1 + h2`.

El siguiente ejemplo puede ser útil para los textos que se muestran como libros:

```
p + p { text-indent: 1.5em; }
```

En muchos libros es habitual que la primera línea de todos los párrafos esté indentada, salvo la primera línea del primer párrafo. El selector `p + p` selecciona todos los párrafos que están dentro de un mismo elemento padre y que estén precedidos por otro párrafo. En otras palabras, el selector `p + p` selecciona todos los párrafos de un elemento salvo el primer párrafo.

El selector adyacente requiere que los dos elementos sean *hermanos*, por lo que su elemento *padre* debe ser el mismo. Si se considera el siguiente ejemplo:

```
p + p { color: red; }

<p>Lorem ipsum dolor sit amet...</p>
<p>Lorem ipsum dolor sit amet...</p>
<div>
  <p>Lorem ipsum dolor sit amet...</p>
</div>
```

En el ejemplo anterior, solamente el segundo párrafo se ve de color rojo, ya que:

- El primer párrafo no va precedido de ningún otro párrafo, por lo que no cumple una de las condiciones de `p + p`
- El segundo párrafo va precedido de otro párrafo y los dos comparten el mismo padre, por lo que se cumplen las dos condiciones del selector `p + p` y el párrafo muestra su texto de color rojo.
- El tercer párrafo se encuentra dentro de un elemento `<div>`, por lo que no se cumple ninguna condición del selector `p + p` ya que ni va precedido de un párrafo ni comparte padre con ningún otro párrafo.

3.3. Selector de atributos

El último tipo de selectores avanzados lo forman los selectores de atributos, que permiten seleccionar elementos HTML en función de sus atributos y/o valores de esos atributos.

Los cuatro tipos de selectores de atributos son:

- `[nombre_atributo]`, selecciona los elementos que tienen establecido el atributo llamado `nombre_atributo`, independientemente de su valor.
- `[nombre_atributo=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` con un valor igual a `valor`.
- `[nombre_atributo~=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` y cuyo valor es una lista de palabras separadas por espacios en blanco en la que al menos una de ellas es exactamente igual a `valor`.
- `[nombre_atributo|=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` y cuyo valor es una serie de palabras separadas con guiones, pero que comienza con `valor`. Este tipo de selector sólo es útil para los atributos de tipo `lang` que indican el idioma del contenido del elemento.

A continuación se muestran algunos ejemplos de estos tipos de selectores:

```
/* Se muestran de color azul todos los enlaces que tengan
   un atributo "class", independientemente de su valor */
a[class] { color: blue; }

/* Se muestran de color azul todos los enlaces que tengan
   un atributo "class" con el valor "externo" */
a[class="externo"] { color: blue; }

/* Se muestran de color azul todos los enlaces que apunten
   al sitio "http://www.ejemplo.com" */
a[href="http://www.ejemplo.com"] { color: blue; }

/* Se muestran de color azul todos los enlaces que tengan
   un atributo "class" en el que al menos uno de sus valores
   sea "externo" */
a[class~="externo"] { color: blue; }

/* Selecciona todos los elementos de la página cuyo atributo
   "lang" sea igual a "en", es decir, todos los elementos en inglés */
*[lang=en] { ... }

/* Selecciona todos los elementos de la página cuyo atributo
   "lang" empiece por "es", es decir, "es", "es-ES", "es-AR", etc. */
*[lang|="es"] { color : red }
```

3.4. Pseudo-clases

3.4.1. La pseudo-clase :first-child

La pseudo-clase `:first-child` selecciona el primer elemento hijo de un elemento. Si se considera el siguiente ejemplo:

```
p em:first-child {
  color: red;
}

<p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim. Praesent nulla
ante, <em>ultricies</em> id, porttitor ut, pulvinar quis, dui.</p>
```

El selector `p em:first-child` selecciona el primer elemento `<em>` que sea hijo de un elemento y que se encuentre dentro de un elemento `<p>`. Por tanto, en el ejemplo anterior sólo el primer `<em>` se ve de color rojo.

La pseudo-clase `:first-child` también se puede utilizar en los selectores simples, como se muestra a continuación:

```
p:first-child { ... }
```

La regla CSS anterior aplica sus estilos al primer párrafo de cualquier elemento. Si se modifica el ejemplo anterior y se utiliza un selector compuesto:

```
p:first-child em {
  color: red;
```

```
}

<body>
<p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim.</p>

<p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim.</p>

<div>
  <p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim.</p>
</div>
</body>
```

El selector `p:first-child em` selecciona todos aquellos elementos `<em>` que se encuentren dentro de un elemento `<p>` que sea el primer hijo de cualquier otro elemento.

El primer párrafo del ejemplo anterior es el primer hijo de `<body>`, por lo que sus `<em>` se ven de color rojo. El segundo párrafo de la página no es el primer hijo de ningún elemento, por lo que sus elementos `<em>` interiores no se ven afectados. Por último, el tercer párrafo de la página es el primer hijo del elemento `<div>`, por lo que sus elementos `<em>` se ven de la misma forma que los del primer párrafo.

3.4.2. Las pseudo-clases :link y :visited

Las pseudo-clases `:link` y `:visited` se pueden utilizar para aplicar diferentes estilos a los enlaces de una misma página:

- La pseudo-clase `:link` se aplica a todos los enlaces que todavía no han sido visitados por el usuario.
- La pseudo-clase `:visited` se aplica a todos los enlaces que han sido visitados al menos una vez por el usuario.

El navegador gestiona de forma automática el cambio de enlace no visitado a enlace visitado. Aunque el usuario puede borrar la cache y el historial de navegación de forma explícita, los navegadores también borran de forma periódica la lista de enlaces visitados.

Por su propia definición, las pseudo-clases `:link` y `:visited` son mutuamente excluyentes, de forma que un mismo enlace no puede estar en los dos estados de forma simultánea.

Como los navegadores muestran por defecto los enlaces de color azul y los enlaces visitados de color morado, es habitual modificar los estilos para adaptarlos a la guía de estilo del sitio web:

```
a:link    { color: red;   }
a:visited { color: green; }
```

3.4.3. Las pseudo-clases :hover, :active y :focus

Las pseudo-clases `:hover`, `:active` y `:focus` permiten al diseñador web variar los estilos de un elemento en respuesta a las acciones del usuario. Al contrario que las pseudo-clases `:link` y

`:visited` que sólo se pueden aplicar a los enlaces, estas pseudo-clases se pueden aplicar a cualquier elemento.

A continuación se indican las acciones del usuario que activan cada pseudo-clase:

- `:hover`, se activa cuando el usuario pasa el ratón o cualquier otro elemento apuntador por encima de un elemento.
- `:active`, se activa cuando el usuario activa un elemento, por ejemplo cuando pulsa con el ratón sobre un elemento. El estilo se aplica durante un espacio de tiempo prácticamente imperceptible, ya que sólo dura desde que el usuario pulsa el botón del ratón hasta que lo suelta.
- `:focus`, se activa cuando el elemento tiene el foco del navegador, es decir, cuando el elemento está seleccionado. Normalmente se aplica a los elementos `<input>` de los formularios cuando están activados y por tanto, se puede escribir directamente en esos campos.

De las definiciones anteriores se desprende que un mismo elemento puede verse afectado por varias pseudo-clases diferentes de forma simultánea. Cuando se pulsa por ejemplo un enlace que fue visitado previamente, al enlace le afectan las pseudo-clases `:visited`, `:hover` y `:active`.

Debido a esta característica y al comportamiento en cascada de los estilos CSS, es importante cuidar el orden en el que se establecen las diferentes pseudo-clases. El siguiente ejemplo muestra el único orden correcto para establecer las cuatro pseudo-clases principales en un enlace:

```
a:link    { ... }
a:visited { ... }
a:hover   { ... }
a:active  { ... }
```

Los navegadores obsoletos como Internet Explorer 6 y sus versiones anteriores no son capaces de aplicar estas pseudo-clases a los elementos que no sean enlaces.

Por último, también es posible aplicar estilos combinando varias pseudo-clases compatibles entre sí. La siguiente regla CSS por ejemplo sólo se aplica a aquellos enlaces que están seleccionados y en los que el usuario pasa el ratón por encima:

```
a:focus:hover { ... }
```

3.4.4. La pseudo-clase :lang

La pseudo-clase `:lang` se emplea para seleccionar elementos en función de su idioma. Los navegadores utilizan los atributos `lang`, las etiquetas `<meta>` y la información de la respuesta del servidor para determinar el idioma de cada elemento.

Si se considera el siguiente ejemplo:

```
p { color: blue; }
p:lang(es) { color: red; }
```

Los párrafos del ejemplo anterior se ven de color azul, salvo los párrafos cuyo contenido esté escrito en español, que se ven de color rojo.

Como los navegadores actuales no son capaces de inferir el idioma de un elemento a partir de su contenido, el uso de esta clase está muy limitado salvo que la página utilice de forma explícita los atributos `lang`:

```
<p lang="en">Lorem ipsum dolor sit amet...</p>
<div lang="fr">
<p>Lorem ipsum dolor sit amet...</p>
<p lang="es_ES">Lorem ipsum dolor sit amet...</p>
</div>
<p lang="en">Lorem ipsum dolor sit amet...</p>
<ul>
  <li lang="fr">Lorem ipsum dolor sit amet...</li>
</ul>
```

La pseudo-clase `:lang(xx)` es muy diferente al selector de atributos `[lang|=xx]`, tal y como muestran las siguientes reglas:

```
*[lang|=es] { ... }  /* selector de atributo */
*:lang(es)  { ... }  /* pseudo-clase */

<body lang="es">
  <p>Lorem ipsum dolor sit amet...</p>
</body>
```

El selector `*[lang|=es]` selecciona todos los elementos de la página que tengan un atributo llamado `lang` cuyo valor empiece por `es`. En el ejemplo anterior, solamente el elemento `<body>` cumple con la condición del selector.

Por otra parte, el selector `*:lang(es)` selecciona todos los elementos de la página cuyo idioma sea el español, sin tener en cuenta el método empleado por el navegador para averiguar el idioma de cada elemento. En este caso, tanto el elemento `<body>` como el elemento `<p>` cumplen esta condición.

3.5. Pseudo-elementos

Los selectores de CSS, las pseudo-clases y todos los elementos HTML no son suficientes para poder aplicar estilos a algunos elementos especiales. Si se desea por ejemplo cambiar el estilo de la primera línea de texto de un elemento, no es posible hacerlo con las utilidades anteriores.

La primera línea del texto normalmente es variable porque el usuario puede aumentar y disminuir la ventana del navegador, puede disponer de más o menos resolución en su monitor y también puede aumentar o disminuir el tamaño de letra del texto.

La única forma de poder seleccionar estos elementos especiales es mediante los pseudo-elementos definidos por CSS para este propósito.

3.5.1. El pseudo-elemento :first-line

El pseudo-elemento `:first-line` permite seleccionar la primera línea de texto de un elemento. Así, la siguiente regla CSS muestra en mayúsculas la primera línea de cada párrafo:

```
p:first-line { text-transform: uppercase; }
```

Este pseudo-elemento sólo se puede utilizar con los elementos de bloque y las celdas de datos de las tablas.

Se pueden combinar varios pseudo-elementos de tipo `:first-line` para crear efectos avanzados:

```
div:first-line { color: red; }
p:first-line { text-transform: uppercase; }

<div>
  <p>Lorem ipsum dolor sit amet...</p>
  <p>Lorem ipsum dolor sit amet...</p>
  <p>Lorem ipsum dolor sit amet...</p>
</div>
```

En el ejemplo anterior, la primera línea del primer párrafo también es la primera línea del elemento `<div>`, por lo que se le aplican las dos reglas CSS y su texto se ve en mayúsculas y de color rojo.

3.5.2. El pseudo-elemento :first-letter

El pseudo-elemento `:first-letter` permite seleccionar la primera letra de la primera línea de texto de un elemento. De esta forma, la siguiente regla CSS muestra en mayúsculas la primera letra del texto de cada párrafo:

```
p:first-letter { text-transform: uppercase; }
```

Los signos de puntuación y los caracteres como las comillas que se encuentran antes y después de la primera letra también se ven afectados por este pseudo-elemento.

Este pseudo-elemento sólo se puede utilizar con los elementos de bloque y las celdas de datos de las tablas.

3.5.3. Los pseudo-elementos :before y :after

Los pseudo-elementos `:before` y `:after` se utilizan en combinación con la propiedad `content` de CSS para añadir contenidos antes o después del contenido original de un elemento.

Las siguientes reglas CSS añaden el texto `Capítulo -` delante de cada título de sección `<h1>` y el carácter `.` detrás de cada párrafo de la página:

```
h1:before { content: "Capítulo - "; }
p:after   { content: "."; }
```

El contenido insertado mediante los pseudo-elementos `:before` y `:after` se tiene en cuenta en los otros pseudo-elementos `:first-line` y `:first-letter`.

3.6. Selectores de CSS 3

La futura versión CSS 3 incluye todos los selectores de CSS 2.1 y añade otras decenas de selectores, pseudo-clases y pseudo-elementos. La lista provisional de novedades y su explicación detallada se puede encontrar en el módulo de selectores de CSS 3 (http://www.w3.org/TR/css3-selectors/) .

En primer lugar, CSS 3 añade tres nuevos selectores de atributos:

- `elemento[atributo^="valor"]`, selecciona todos los elementos que disponen de ese atributo y cuyo valor comienza exactamente por la cadena de texto indicada.
- `elemento[atributo$="valor"]`, selecciona todos los elementos que disponen de ese atributo y cuyo valor termina exactamente por la cadena de texto indicada.
- `elemento[atributo*="valor"]`, selecciona todos los elementos que disponen de ese atributo y cuyo valor contiene la cadena de texto indicada.

De esta forma, se pueden crear reglas CSS tan avazadas como las siguientes:

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

/* Selecciona todos los enlaces que apuntan a una página HTML */
a[href$=".html"] { ... }

/* Selecciona todos los títulos h1 cuyo atributo title contenga la palabra "capítulo" */
h1[title*="capítulo"] { ... }
```

Otro de los nuevos selectores de CSS 3 es el *"selector general de elementos hermanos"*, que generaliza el selector adyacente de CSS 2.1. Su sintaxis es `elemento1 ~ elemento2` y selecciona el `elemento2` que es hermano de `elemento1` y se encuentra detrás en el código HTML. En el selector adyacente la condición adicional era que los dos elementos debían estar uno detrás de otro en el código HTML, mientras que ahora la única condición es que uno esté detrás de otro.

Si se considera el siguiente ejemplo:

```
h1 + h2 { ... }  /* selector adyacente */
h1 ~ h2 { ... }  /* selector general de hermanos */

<h1>...</h1>
<h2>...</h2>
<p>...</p>
<div>
  <h2>...</h2>
</div>
<h2>...</h2>
```

El primer selector (`h1 + h2`) sólo selecciona el primer elemento `<h2>` de la página, ya que es el único que cumple que es hermano de `<h1>` y se encuentra justo detrás en el código HTML. Por su parte, el segundo selector (`h1 ~ h2`) selecciona todos los elementos `<h2>` de la página salvo el segundo. Aunque el segundo `<h2>` se encuentra detrás de `<h1>` en el código HTML, no son elementos hermanos porque no tienen el mismo elemento padre.

Los pseudo-elementos de CSS 2.1 se mantienen en CSS 3, pero cambia su sintaxis y ahora se utilizan `::` en vez de `:` delante del nombre de cada pseudo-elemento:

- `::first-line`, selecciona la primera línea del texto de un elemento.
- `::first-letter`, selecciona la primera letra del texto de un elemento.
- `::before`, selecciona la parte anterior al contenido de un elemento para insertar nuevo contenido generado.
- `::after`, selecciona la parte posterior al contenido de un elemento para insertar nuevo contenido generado.

CSS 3 añade además un nuevo pseudo-elemento:

- `::selecion`, selecciona el texto que ha seleccionado un usuario con su ratón o teclado.

Las mayores novedades de CSS 3 se producen en las pseudo-clases, ya que se añaden 12 nuevas, entre las cuales se encuentran:

- `elemento:nth-child(numero)`, selecciona el elemento indicado pero con la condición de que sea el hijo enésimo de su padre. Este selector es útil para seleccionar el segundo párrafo de un elemento, el quinto elemento de una lista, etc.
- `elemento:nth-last-child(numero)`, idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.
- `elemento:empty`, selecciona el elemento indicado pero con la condición de que no tenga ningún hijo. La condición implica que tampoco puede tener ningún contenido de texto.
- `elemento:first-child` y `elemento:last-child`, seleccionan los elementos indicados pero con la condición de que sean respectivamente los primeros o últimos hijos de su elemento padre.
- `elemento:nth-of-type(numero)`, selecciona el elemento indicado pero con la condición de que sea el enésimo elemento hermano de ese tipo.
- `elemento:nth-last-of-type(numero)`, idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.

Algunas pseudo-clases como `:nth-child(numero)` permiten el uso de expresiones complejas para realizar selecciones avanzadas:

```
li:nth-child(2n+1) { ... }   /* selecciona todos los elementos impares de una lista */
li:nth-child(2n)   { ... }   /* selecciona todos los elementos pares de una lista */

/* Las siguientes reglas alternan cuatro estilos diferentes para los párrafos */
p:nth-child(4n+1) { ... }
p:nth-child(4n+2) { ... }
p:nth-child(4n+3) { ... }
p:nth-child(4n+4) { ... }
```

Empleando la pseudo-clase `:nth-of-type(numero)` se pueden crear reglas CSS que alternen la posición de las imágenes en función de la posición de la imagen anterior:

```
img:nth-of-type(2n+1) { float: right; }
img:nth-of-type(2n)   { float: left;  }
```

Otro de los nuevos selectores que incluirá CSS 3 es `:not()`, que se puede utilizar para seleccionar todos los elementos que no cumplen con la condición de un selector:

```
:not(p) { ... }  /* selecciona todos los elementos de la página que no sean párrafos */
:not(#especial) { ... }  /* selecciona cualquier elemento cuyo atributo id no sea
"especial" */
```

Aunque todavía faltan muchos años hasta que la versión CSS 3 sustituya a la actual versión CSS 2.1, los navegadores que más se preocupan por los estándares (Opera, Safari y Firefox) incluyen soporte para varios o casi todos los selectores de CSS 3.

Existe una herramienta llamada CSS Selectors test (http://www.css3.info/selectors-test/) que permite comprobar los selectores que soporta el navegador con el que se hace la prueba.

Capítulo 4. Propiedades avanzadas

El estándar CSS 2.1 incluye 115 propiedades que abarcan el modelo de cajas (*box model*), la tipografía, las tablas, las listas, el posicionamiento de los elementos, la generación de contenidos y los medios impresos y auditivos.

Aunque la mayoría de diseñadores web conocen y utilizan casi todas las propiedades de CSS 2.1, no siempre hacen uso de todas sus posibilidades. Algunas propiedades de CSS 2.1 han sido infrautilizadas hasta hace poco tiempo porque los navegadores no las soportaban.

Afortunadamente, los navegadores que más se preocupan por los estándares (Firefox, Safari y Opera) ya incluyen soporte completo de casi todas las propiedades de CSS 2.1, con la única excepción de las propiedades relacionadas con los medios auditivos. Por su parte, Internet Explorer 8 promete que incluirá soporte completo de todas las propiedades de CSS 2.1.

4.1. Propiedad white-space

Definición	Establece el tratamiento de los espacios en blanco
Valores permitidos	Uno y sólo uno de los siguientes valores: ▪ normal ▪ pre ▪ nowrap ▪ pre-wrap ▪ pre-line ▪ inherit
Valor inicial	normal
Se aplica a	Todos los elementos
Válida en	medios visuales
Se hereda	si
Definición en el estándar	http://www.w3.org/TR/CSS21/text.html#propdef-white-space

El tratamiento de los espacios en blanco en el código HTML es una de las características más desconcertantes para los diseñadores web que comienzan a crear páginas. A continuación se muestra cómo visualizan los navegadores dos párrafos de ejemplo:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.

Aunque los dos párrafos anteriores se visualizan de la misma forma, en realidad su código HTML es completamente diferente:

```
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus
vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum
tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis
nunc in leo.</p>

<p>Lorem ipsum    dolor    sit amet,   consectetuer adipiscing     elit.
Sed non sem   quis tellus vulputate       lobortis. Vivamus fermentum, tortor
id ornare    ultrices, ligula ipsum tincidunt       pede, et blandit
sem pede suscipit     pede. Nulla cursus    porta sem. Donec mollis nunc in leo.</p>
```

El segundo párrafo contiene numerosos espacios en blanco y saltos de línea. Sin embargo, como los navegadores eliminan automáticamente todos los espacios en blanco sobrantes, los dos párrafos se ven exactamente igual.

En el estándar HTML un *"espacio en blanco"* puede ser un salto de línea, un tabulador y un espacio en blanco normal. Los navegadores eliminan de forma automática todos los espacios en blanco sobrantes salvo el espacio en blanco que separa las palabras del texto.

La única excepción de este comportamiento es la etiqueta `<pre>` de HTML, utilizada para mostrar texto que ya tiene formato (su nombre viene de *preformateado*) y que por tanto respeta todos los espacios en blanco y todos los saltos de línea:

```
Lorem ipsum    dolor    sit amet,   consectetuer adipiscing     elit.
Sed non sem   quis tellus vulputate       lobortis. Vivamus fermentum, tortor
id ornare    ultrices, ligula ipsum tincidunt       pede, et blandit
sem pede suscipit     pede. Nulla cursus    porta sem. Donec mollis nunc in leo.
```

El código HTML del ejemplo anterior es:

```
<pre>Lorem ipsum    dolor    sit amet,   consectetuer adipiscing     elit.
Sed non sem   quis tellus vulputate       lobortis. Vivamus fermentum, tortor
id ornare    ultrices, ligula ipsum tincidunt       pede, et blandit
sem pede suscipit     pede. Nulla cursus    porta sem. Donec mollis nunc in leo.</pre>
```

La propiedad `white-space` permite variar el comportamiento de los espacios en blanco. El estándar CSS 2.1 define cinco modelos diferentes de tratamiento de espacios en blanco:

- `normal`: los espacios en blanco sobrantes y los saltos de línea se eliminan. No obstante, el texto se muestra en tantas líneas como sea necesario para que sus contenidos no se salgan del elemento contenedor.
- `pre`: no se eliminan los espacios en blanco sobrantes y sólo se muestran los saltos de línea incluidos en el texto original. Este comportamiento puede provocar que los contenidos de texto se salgan de su elemento contenedor.
- `nowrap`: se comporta igual que `normal` en los espacios en blanco, pero no añade saltos de línea en el texto original, por lo que los contenidos se pueden salir de su elemento contenedor.

- `pre-wrap`: se comporta igual que `pre`, pero se introducen los saltos de línea que sean necesarios para que los contenidos de texto nunca se salgan de su elemento contenedor.
- `pre-line`: se eliminan los espacios en blanco sobrantes, pero se respetan los saltos de línea originales y se crean tantos saltos de línea como sean necesarios para que el contenido de texto no se salga de su elemento contenedor.

Como las explicaciones incluídas en el estándar CSS 2.1 pueden llegar a ser confusas, la siguiente tabla resume el comportamiento de cada valor:

Valor	Respeta espacios en blanco	Respeta saltos de línea	Ajusta las líneas
`normal`	no	no	si
`pre`	si	si	no
`nowrap`	no	no	no
`pre-wrap`	si	si	si
`pre-line`	no	si	si

A continuación se muestra el efecto de cada modelo de tratamiento de espacios en blanco sobre un mismo párrafo que contiene espacios en blanco y saltos de línea y que se encuentra dentro de un elemento contenedor de anchura limitada:

[white-space: normal] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vestibulum dictum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

[white-space: pre] Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Vestibulum dictum. Class aptent taciti sociosqu ad litora torquent per
conubia nostra, per inceptos hymenaeos.

[white-space: pre-wrap] Lorem ipsum dolor
sit amet, consectetuer adipiscing elit.
Vestibulum dictum. Class aptent taciti
sociosqu ad litora torquent per
conubia nostra, per inceptos hymenaeos.

[white-space: nowrap] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vestibulum dictum. Clas

[white-space: pre-line] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vestibulum dictum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

Los valores más utilizados son `normal`, `pre` y `pre-wrap`. El valor `normal` se puede emplear por ejemplo en un elemento `<pre>` que se quiere mostrar como si fuera un párrafo:

```
<pre>Lorem ipsum    dolor    sit amet,   consectetuer adipiscing     elit.
Sed non sem   quis tellus vulputate       lobortis. Vivamus fermentum, tortor
id ornare    ultrices, ligula ipsum tincidunt       pede, et blandit
sem pede suscipit     pede. Nulla cursus    porta sem. Donec mollis nunc in leo.</pre>

<pre style="white-space: normal">Lorem ipsum dolor sit amet, consectetuer
adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus
fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit
sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in
leo.</pre>
```

De la misma forma, el valor `pre` se puede emplear en un párrafo de texto que se quiere mostrar como si fuera un elemento `<pre>`:

```
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus
vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede,
et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.</p>


<p style="white-space: pre">Lorem ipsum    dolor    sit amet,   consectetuer adipiscing     elit.
Sed non sem   quis tellus vulputate       lobortis. Vivamus fermentum, tortor
id ornare    ultrices, ligula ipsum tincidunt       pede, et blandit
sem pede suscipit     pede. Nulla cursus    porta sem. Donec mollis nunc in leo.</p>
```

Por último, el valor `pre-wrap` es muy útil cuando se quiere mostrar un texto de la forma más fiel posible a su formato original (respetando espacios en blanco y saltos de línea) pero sin que el contenido de texto se salga de su elemento contenedor.

4.2. Propiedad display

Definición	Establece el tipo de caja generada por un elemento
Valores permitidos	Uno y sólo uno de los siguientes valores: ▪ inline ▪ block ▪ list-item ▪ run-in ▪ inline-block ▪ table ▪ inline-table ▪ table-row-group ▪ table-header-group ▪ table-footer-group ▪ table-row ▪ table-column-group ▪ table-column ▪ table-cell ▪ table-caption ▪ none ▪ inherit
Valor inicial	inline
Se aplica a	Todos los elementos
Válida en	all
Se hereda	no
Definición en el estándar	http://www.w3.org/TR/CSS21/visuren.html#propdef-display

La propiedad `display` es una de las propiedades CSS más infrautilizadas. Aunque todos los diseñadores conocen esta propiedad y utilizan sus valores `inline`, `block` y `none`, las posibilidades de `display` son mucho más avanzadas.

De hecho, la propiedad `display` es una de las más complejas de CSS 2.1, ya que establece el tipo de la caja que genera cada elemento. La propiedad `display` es tan compleja que casi ningún navegador es capaz de mostrar correctamente todos sus valores.

El valor más sencillo de `display` es `none` que hace que el elemento no genere ninguna caja. El resultado es que el elemento desaparece por completo de la página y no ocupa sitio, por lo que los elementos adyacentes ocupan su lugar. Si se utiliza la propiedad `display: none` sobre un elemento, todos sus descendientes también desaparecen por completo de la página.

Si se quiere hacer un elemento invisible, es decir, que no se vea pero que siga ocupando el mismo sitio, se debe utilizar la propiedad `visibility`. La propiedad `display: none` se utiliza habitualmente en aplicaciones web dinámicas creadas con JavaScript y que muestran/ocultan contenidos cuando el usuario realiza alguna acción como pulsar un botón o un enlace.

Los otros dos valores más utilizados son `block` e `inline` que hacen que la caja de un elemento sea de bloque o en línea respectivamente. El siguiente ejemplo muestra un párrafo y varios enlaces a los que se les ha añadido un borde para mostrar el espacio ocupado por cada caja:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Lorem ipsum (#) Donec mollis nunc in leo (#) Vivamus fermentum (#)

Como el párrafo es por defecto un elemento de bloque (*"block element"*), ocupa todo el espacio disponible hasta el final de su línea, aunque sus contenidos no ocupen todo el sitio. Por su parte, los enlaces por defecto son elementos en línea (*"inline element"*), por lo que su caja sólo ocupa el espacio necesario para mostrar sus contenidos.

Si se aplica la propiedad `display: inline` al párrafo del ejemplo anterior, su caja se convierte en un elemento en línea y por tanto sólo ocupa el espacio necesario para mostrar sus contenidos:

[display: inline] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Lorem ipsum (#) Donec mollis nunc in leo (#) Vivamus fermentum (#)

Para visualizar más claramente el cambio en el tipo de caja, el siguiente ejemplo muestra un mismo párrafo largo con `display: block` y `display: inline`:

[display: block] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.

[display: inline] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.

De la misma forma, si en los enlaces del ejemplo anterior se emplea la propiedad `display: block` se transforman en elementos de bloque, por lo que siempre empiezan en una nueva línea y siempre ocupan todo el espacio disponible en la línea, aunque sus contenidos no ocupen todo el sitio:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

[display: block] Lorem ipsum (#)

[display: block] Donec mollis nunc in leo (#)

[display: block] Vivamus fermentum (#)

Uno de los valores más curiosos de `display` es `inline-block`, que crea cajas que son de bloque y en línea de forma simultánea. Una caja de tipo `inline-block` se comporta como si fuera de bloque, pero respecto a los elementos que la rodean es una caja en línea.

El enlace del siguiente ejemplo es de tipo `inline-block`, lo que permite por ejemplo establecerle un tamaño mediante la propiedad `width`:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo. **[display: inline-block, width: 25%]** Quisque semper, magna sed pharetra tincidunt, quam urna dapibus dolor, a dignissim sem neque id purus. Etiam luctus viverra nisi. (#) Integer lobortis accumsan felis. Cras venenatis. Morbi cursus, tellus vitae iaculis pulvinar, turpis nibh posuere nisl, sed vehicula massa orci at dui. Morbi pede ipsum, porta quis, venenatis et, ullamcorper in, metus. Nulla facilisi. Quisque laoreet molestie mi. Ut mollis elit eget urna.

Si tu navegador soporta el valor `inline-block`, el ejemplo anterior se debe visualizar tal y como muestra la siguiente imagen:

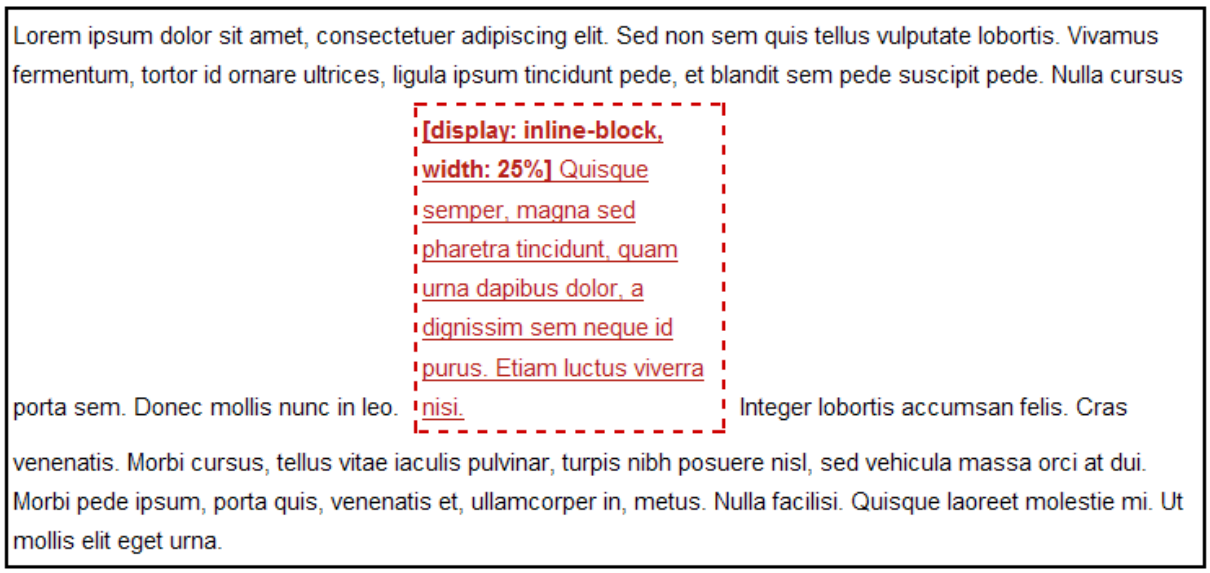


Figura 4.1. Ejemplo del valor inline-block de la propiedad display

Otro de los valores definidos por la propiedad `display` es `list-item`, que hace que cualquier elemento de cualquier tipo se muestre como si fuera un elemento de una lista (elemento `<li>`). El siguiente ejemplo muestra tres párrafos que utilizan la propiedad `display: list-item` para simular que son una lista de elementos de tipo `<ul>`:

- Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
- Sed non sem quis tellus vulputate lobortis.
- Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede.

A continuación se muestra el código HTML del ejemplo anterior:

```
<p style="display: list-item; margin-left: 2em">Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.</p>
<p style="display: list-item; margin-left: 2em">Sed non sem quis tellus vulputate
lobortis.</p>
<p style="display: list-item; margin-left: 2em">Vivamus fermentum, tortor id ornare
ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede.</p>
```

Los elementos con la propiedad `display: list-item` son exactamente iguales que los elementos `<li>` a efectos de su visualización, por lo que se pueden utilizar las propiedades de listas como `list-style-type`, `list-style-image`, `list-style-position` y `list-style`.

Uno de los valores más curiosos de la propiedad `display` es `run-in`, que genera una caja de bloque o una caja en línea dependiendo del contexto, es decir, dependiendo de sus elementos adyacentes. El comportamiento de las cajas de tipo `run-in` se rige por las siguientes reglas:

- Si la caja `run-in` contiene una caja de bloque, la caja `run-in` se convierte en una caja de bloque.
- Si después de la caja `run-in` se encuentra una caja de bloque (que no esté posicionada de forma absoluta y tampoco esté posicionada de forma flotante), la caja `run-in` se convierte en una caja en línea en el interior de la caja de bloque.
- En cualquier otro caso, la caja `run-in` se convierte en una caja de bloque.

El siguiente ejemplo muestra una misma caja de tipo `run-in` que se visualiza de forma muy diferente en función del tipo de caja que existe a continuación:

[display: run-in] Lorem ipsum **[display: block]** dolor sit amet, consectetuer adipiscing elit.

[display: run-in] Lorem ipsum **[display: inline]** dolor sit amet, consectetuer adipiscing elit.

El código HTML y CSS del ejemplo anterior se muestra a continuación:

```
<p style="display: run-in; border: 2px dashed #C00;"><strong>[display: run-in]</strong>
Lorem ipsum</p>
<p style="display: block; border: 2px solid #000;"><strong>[display: block]</strong>
dolor sit amet, consectetuer adipiscing elit.</p>

<p style="display: run-in; border: 2px dashed #C00;"><strong>[display: run-in]</strong>
Lorem ipsum</p>
<p style="display: inline; border: 2px solid #000;"><strong>[display: inline]</strong>
dolor sit amet, consectetuer adipiscing elit.</p>
```

En la actualidad sólo la última versión del navegador Opera es capaz de mostrar correctamente el ejemplo anterior, tal y como muestra la siguiente imagen:

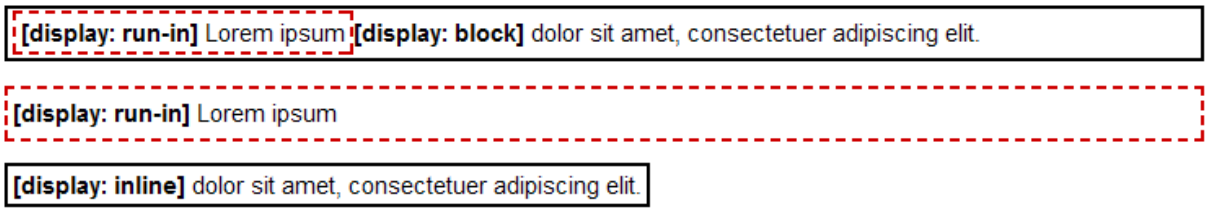


Figura 4.2. Ejemplo del valor run-in de la propiedad display

El estándar CSS 2.1 incluye un ejemplo del posible uso del valor `run-in`. En este ejemplo, un título de sección `<h3>` crea una caja `run-in`, de forma que cuando va seguido de un párrafo, el titular *se mete* dentro del párrafo:

```
<h3 style="display: run-in">Lorem ipsum dolor sit amet</h3>
<p>Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare
ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus
porta sem. Donec mollis nunc in leo. Integer lobortis accumsan felis.</p>
```

El resto de valores de la propiedad `display` están relacionados con las tablas y hacen que un elemento se muestre como si fuera una parte de una tabla: fila, columna, celda o grupos de filas/columnas. Los valores definidos por la propiedad `display` son `inline-table`, `table-row-group`, `table-header-group`, `table-footer-group`, `table-row`, `table-column-group`, `table-column`, `table-cell`, `table-caption`.

Aunque los valores relacionados con las tablas son los más avanzados, también son los que peor soportan los navegadores. A continuación se muestra un ejemplo con tres párrafos de texto que establecen la propiedad `display: table-cell`:

[display: table-cell] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas non tortor. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Sed fermentum lorem a velit.

[display: table-cell] In molestie suscipit libero. Cras sem. Nunc non tellus et urna mattis tempor. Nulla nec tellus a quam hendrerit venenatis. Suspendisse pellentesque odio et est. Morbi sed nisl sed dui consequat sodales.

[display: table-cell] Morbi sed nisl sed dui consequat sodales. Vivamus ornare felis nec est. Phasellus massa justo, ornare sed, malesuada a, dignissim a, nibh. Vestibulum vitae nunc at lectus euismod feugiat. Nullam eleifend. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. In ut ipsum.

La propiedad `display: table-cell` hace que cualquier elemento se muestre como si fuera una celda de una tabla. Como en el ejemplo anterior los tres elementos `<p>` utilizan la propiedad `display: table-cell`, el resultado es visualmente idéntico a utilizar una tabla y tres elementos `<td>`.

Si utilizas un navegador con soporte completo de CSS 2.1, el ejemplo anterior se visualiza tal y como muestra la siguiente imagen:

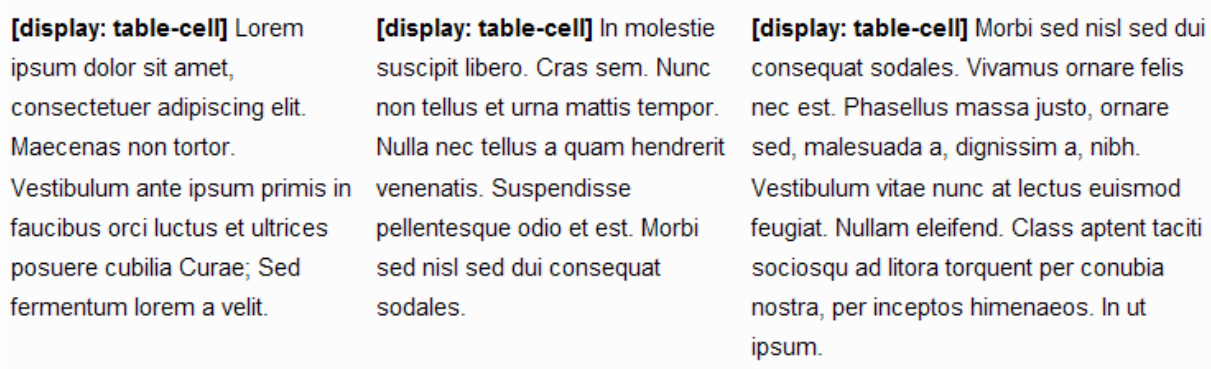


Figura 4.3. Ejemplo del valor table-cell de la propiedad display

Como los valores relacionados con las tablas hacen que cualquier elemento que no sea una tabla se muestre y comporte como si lo fuera, se pueden utilizar para crear los *layouts* de las páginas. Hace años, la estructura de las páginas se definía mediante tablas, filas y columnas. Esta solución tiene innumerables desventajas y por eso todos los diseñadores web profesionales crean la estructura de sus páginas mediante CSS y elementos `<div>`.

No obstante, las tablas tienen algunas ventajas en su comportamiento respecto a los elementos `<div>` posicionados de forma absoluta o flotante. La principal ventaja es que todas las celdas de una fila siempre tienen la misma altura, por lo que si se utilizan tablas no se sufre el problema de las columnas de página con diferente altura.

Además, la estructura creada con una tabla nunca se rompe, ya que las celdas de datos nunca se visualizan una debajo de otra cuando la ventana del navegador se hace muy pequeña. Sin embargo, cuando se define la estructura mediante elementos `<div>` posicionados es posible que la página se rompa y alguna columna se muestre debajo de otros contenidos.

Utilizando la propiedad `display` de forma avanzada es posible crear una estructura de página que sea semánticamente correcta, esté diseñada exclusivamente con CSS y que se comporte exactamente igual que como lo hacen las tablas.

El siguiente código HTML corresponde a la estructura de una página con tres columnas:

```
...
<div id="contenedor">
  <div id="contenidos">
    <div id="secundario">
      Curabitur rutrum...
    </div>
    <div id="principal">
      Lorem ipsum dolor sit amet...
    </div>
```

```
    <div id="lateral">
      Nam erat massa...
    </div>
  </div>
</div>
...
```

Utilizando las siguientes reglas CSS y la propiedad `display` es posible hacer que los elementos `<div>` anteriores se comporten como si fueran elementos `<tr>` y `<td>`:

```
#contenedor {
  display: table;
  border-spacing: 5px;
}
#contenidos {
  display: table-row;
}
#principal, #secundario, #lateral {
  display: table-cell;
}
#principal {
  width: 60%;
}
#secundario, #lateral {
  width: 20%;
}
```

El elemento `#contenedor` se visualiza como una tabla porque se le aplica la propiedad `display: table`. De esta forma, se pueden aplicar al elemento `#contenedor` propiedades exclusivas de las tablas como `border-spacing`. El elemento `#contenidos` se visualiza como si fuese una fila de tabla (etiqueta `<tr>`). En su interior se encuentran las tres columnas de la página que se visualizan como si fueran tres elementos `<td>` gracias a la propiedad `display: table-cell`.

A continuación se muestra el resultado obtenido al aplicar estas reglas CSS al código HTML anterior:

Curabitur rutrum eros a risus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas non tortor. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Sed fermentum lorem a velit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin sodales, enim in volutpat vehicula, leo turpis vehicula magna, ut rutrum arcu lorem ac pede. Curabitur rutrum eros a risus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In molestie suscipit libero. Cras sem. Nunc non	Nam erat massa, blandit in, dapibus sit amet, vestibulum et, augue. Suspendisse ac risus.

molestie suscipit libero. Cras sem. Nunc non tellus et urna mattis tempor. Nulla nec tellus a quam hendrerit venenatis. Suspendisse pellentesque odio et est.	tellus et urna mattis tempor. Nulla nec tellus a quam hendrerit venenatis. Suspendisse pellentesque odio et est. Morbi sed nisl sed dui consequat sodales. Donec porta porta ligula. Nam erat massa, blandit in, dapibus sit amet, vestibulum et, augue. Suspendisse ac risus. Duis semper fringilla sem. Praesent augue arcu, scelerisque nec, ornare malesuada, posuere a, neque. Nullam nulla nisi, ultrices quis, adipiscing non, varius ut, dui. Nulla viverra pellentesque sem.	

La estructura de la página del ejemplo anterior está diseñada exclusivamente con CSS pero se comporta como si fuera una tabla. Todas las columnas de la página tienen la misma altura sin necesidad de recurrir a ningún truco y la página nunca se rompe por muy pequeña que se haga la ventana del navegador.

Si visualizas esta página con un navegador que soporte correctamente la propiedad `display` de CSS 2.1, el ejemplo anterior se ve tal y como muestra la siguiente imagen:

Curabitur rutrum eros a risus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In molestie suscipit libero. Cras sem. Nunc non tellus et urna mattis tempor. Nulla nec tellus a quam hendrerit venenatis. Suspendisse pellentesque odio et est.	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas non tortor. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Sed fermentum lorem a velit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin sodales, enim in volutpat vehicula, leo turpis vehicula magna, ut rutrum arcu lorem ac pede. Curabitur rutrum eros a risus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In molestie suscipit libero. Cras sem. Nunc non tellus et urna mattis tempor. Nulla nec tellus a quam hendrerit venenatis. Suspendisse pellentesque odio et est. Morbi sed nisl sed dui consequat sodales. Donec porta porta ligula. Nam erat massa, blandit in, dapibus sit amet, vestibulum et, augue. Suspendisse ac risus. Duis semper fringilla sem. Praesent augue arcu, scelerisque nec, ornare malesuada, posuere a, neque. Nullam nulla nisi, ultrices quis, adipiscing non, varius ut, dui. Nulla viverra pellentesque sem.	Nam erat massa, blandit in, dapibus sit amet, vestibulum et, augue. Suspendisse ac risus.

Figura 4.4. Ejemplo de los valores table, table-row y table-cell de la propiedad display

Por último, aunque el estándar CSS 2.1 establece que el valor por defecto de la propiedad `display` es `inline`, todos los navegadores obvian esta recomendación y asignan por defecto el valor `block` a los elementos de bloque y el valor `inline` a los elementos en línea.

4.3. Propiedad outline

Definición	Establece algunas o todas las propiedades de todos los perfiles de los elementos
Valores permitidos	Alguno o todos los siguientes valores y en cualquier orden: ▪ Color de perfil (./outline-color.html) ▪ Estilo de perfil (./outline-style.html) ▪ Anchura de perfil (./outline-width.html)
Valor inicial	Cada propiedad define su propio valor por defecto
Se aplica a	Todos los elementos
Válida en	medios visuales, medios interactivos (./medios.html)
Se hereda	no
Definición en el estándar	http://www.w3.org/TR/CSS21/ui.html#propdef-outline

La propiedad `outline` es una de las *"propiedades shorthand"* que define CSS y que se utilizan para establecer de forma abreviada el valor de una o más propiedades individuales. En este caso, se utiliza para establecer el mismo grosor, estilo y/o anchura de todos los perfiles de un elemento.

Aunque es cierto que guarda muchas similitudes con la propiedad `border`, en realidad se diferencian en algunos aspectos muy importantes:

1. Los perfiles no ocupan espacio, mientras que los bordes normales sí.
2. Los perfiles pueden tener formas no rectangulares.

Desde el punto de vista del diseño, la primera característica es la más importante. Los perfiles u *outline* siempre se dibujan "por encima del elemento", por lo que no modifican la posición o el tamaño total de los elementos.

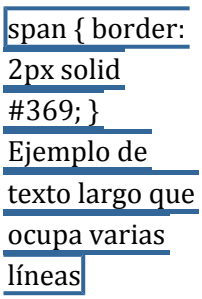
En el siguiente ejemplo se muestran dos cajas; la primera muestra un borde muy grueso y la segunda muestra un perfil de la misma anchura:

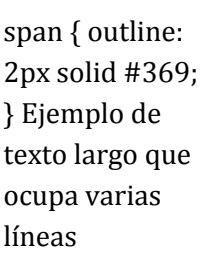
```
div { border: 5px solid #369; }

div { outline: 5px solid #369; }
```

El perfil de la segunda caja se dibuja justo por el exterior de su borde. Aunque visualmente no lo parece, el perfil y el borde no se encuentran en el mismo plano. De esta forma, el perfil no se tiene en cuenta para calcular la anchura total de un elemento y no influye en el resto de elementos cercanos.

La segunda característica importante de los perfiles pueden tener formas no rectangulares. En el siguiente ejemplo se muestra un texto muy largo encerrado en una caja muy estrecha, lo que provoca que el texto se muestre en varias líneas:





El primer texto se encierra con un borde que produce un resultado estético poco afortunado. Cada línea muestra un borde superior e inferior, aunque sólo la primera y última líneas cierran el borde lateralmente.

Mientras tanto, el segundo texto se encierra con un perfil. Como indica su nombre, la propiedad `outline` perfila los contenidos del elemento y los encierra con una forma no rectangular. No obstante, si visualizas esta misma página en diferentes navegadores, verás que todos los navegadores dibujan el perfil de forma diferente, al contrario de lo que sucede con el borde.

Por este motivo, al contrario que los bordes, no existe el concepto de *perfil izquierdo* o *perfil superior*. El perfil de un elemento es único y sus propiedades son idénticas para cada uno de los cuatro lados.

Como sucede con muchas propiedades de tipo *shorthand*, el orden en el que se indican los valores de la propiedd `outline` es indiferente:

```
div { outline: 1px solid #C00; }
div { outline: solid 1px #C00; }
div { outline: #C00 1px solid; }
div { outline: #C00 solid 1px; }
```

La propiedad `outline` no requiere que se indiquen las tres propiedades que definen el estilo de los perfiles. Si no se indica alguna propiedad, su valor se obtiene mediante el valor por defecto de esa propiedad.

En el siguiente ejemplo sólo se indica el estilo del perfil, por lo que el navegador asigna automáticamente el valor `medium` a su grosor y el color se escoge para que tenga el máximo contraste con el fondo de la página:

```
div { outline: solid; }
```

En el siguiente ejemplo se omite el grosor del perfil, por lo que el navegador le asigna automáticamente el valor `medium`:

```
div { outline: solid blue; }
```

No obstante, como el valor por defecto del estilo de un perfil es `none`, si no se indica explícitamente el estilo del perfil, el resultado es que el navegador no muestra ese perfil:

```
div { outline: 2px blue; }
```

El grosor del perfil se puede establecer de cualquiera de las diferentes formas de indicar una medida en CSS. El color también se puede establecer de alguna de las diferentes formas de indicar un color en CSS, con la particularidad del valor `invert` que selecciona el color que tenga más contraste con el color de fondo. Por último, el estilo del perfil se establece con los mismos valores que los que se utilizan en la propiedad `border-style`.

Aunque la propiedad `outline` es infinitamente menos utilizada que la propiedad `border`, la has visto muchas más veces de las que crees. Si pulsas repetidamente la tecla del tabulador en una página web, el navegador va seleccionando de forma secuencial todos los elementos *pinchables* o *seleccionables*: enlaces, botones, controles de formulario, etc.

Para indicar el elemento que está seleccionado, el navegador muestra un perfil muy fino de `1px` de ancho, de estilo punteado y del color que más contrasta con el color de fondo de la página. En la mayoría de las páginas web, el perfil que se muestra por defecto es `outline: 1px dotted #000`.

Los navegadores más avanzados incluyen una *pseudo-clase* llamada `:focus` que permite establecer el estilo de los elementos seleccionados. Utilizando la propiedad `outline` junto con `:focus` se puede modificar el estilo por defecto del navegador:

```
<style type="text/css">
:focus { outline: 2px solid red; }
</style>
```

Si pruebas a pulsar repetidamente la tecla `Tabulador` en una página que incluya la regla CSS anterior, verás que el navegador selecciona secuencialmente los enlaces de la página y muestra un perfil continuo de color rojo para indicar el elemento que está seleccionado en cada momento.

De hecho, según el estándar oficial, ese es el propósito para el que se crearon los perfiles: indicar de forma clara el elemento (botón, enlace, otros controles de formulario) que está seleccionado en cada momento.

4.4. Propiedad quotes

Definición	Establece los caracteres utilizados para mostrar las comillas
Valores permitidos	Uno y sólo uno de los siguientes valores: ▪ Un número par de cadenas de texto ▪ none ▪ inherit
Valor inicial	Depende de cada navegador
Se aplica a	Todos los elementos
Válida en	medios visuales
Se hereda	si
Definición en el estándar	http://www.w3.org/TR/CSS21/generate.html#propdef-quotes

La propiedad `quotes` sólo tiene sentido cuando se utiliza junto con la propiedad `content` y los *pseudo-elementos* `:after` y `:before`. La propiedad `quote` se puede emplear para establecer los caracteres que se utilizan al mostrar las comillas en un elemento.

Por defecto, cuando se utiliza la propiedad `content` junto con los valores `open-quote` y `close-quote`, el navegador muestra unas determinadas comillas. Por tanto, si se considera el siguiente código HTML y CSS:

```
blockquote:before { content: open-quote; }
blockquote:after  { content: close-quote; }

<blockquote>Lorem ipsum dolor sit amet...</blockquote>
```

El navegador muestra el ejemplo anterior de la siguiente forma:

> "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo."

Si utilizas un navegador moderno con soporte de la propiedad `quotes`, el ejemplo anterior se visualiza como muestra la siguiente imagen:

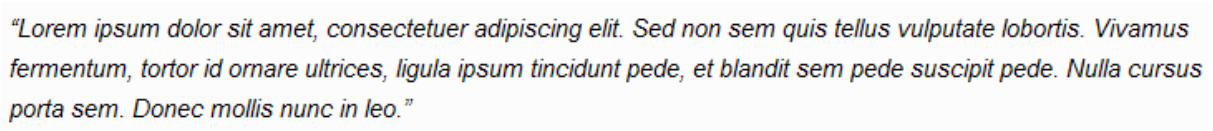


Figura 4.5. Ejemplo de la propiedad quotes

La imagen anterior corresponde al navegador Firefox. Como el estándar CSS 2.1 no indica las comillas que se deben utilizar por defecto, se pueden producir diferencias visuales en cada navegador.

La propiedad `quotes` se utiliza para establecer de forma explícita las comillas que debe utilizar el navegador y por tanto, neutraliza las posibles diferencias entre navegadores. Las comillas siempre se indican por pares, siendo la primera la comilla de apertura y la segunda la comilla de cierre.

El siguiente ejemplo modifica el anterior para utilizar las comillas « y »:

```
blockquote {
  quotes: "«" "»";
}
blockquote:before { content: open-quote; }
blockquote:after  { content: close-quote; }

<blockquote>Lorem ipsum dolor sit amet...</blockquote>
```

Ahora, el navegador muestra el ejemplo anterior de la siguiente forma:

> "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo."

Si se visualiza esta página con un navegador que soporte la propiedad `quotes`, el ejemplo anterior se visualiza como muestra la siguiente imagen:

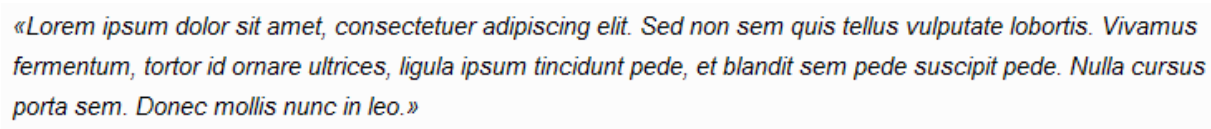


Figura 4.6. Ejemplo de la propiedad quotes

La propiedad `quotes` permite indicar un número ilimitado de pares de comillas. Cuando sólo se define un par, se muestran siempre que se utilice `open-quote` y `close-quote`. Cuando se definen varios pares, se van alternando cada vez que se muestran unas comillas dentro de otro elemento que ya tiene comillas. Cada comilla se separa de las demás mediante un espacio en blanco, por lo que no debe utilizarse una coma o cualquier otro caracter de separación.

El siguiente ejemplo define dos pares de comillas en cada elemento de modo que se alternan las comillas cada vez que un elemento se encuentra dentro de otro elemento que también muestra comillas:

```
blockquote, span {
  quotes: "«" "»" '"' '"';
}

blockquote:before, span:before {
```

```
  content: open-quote;
}
blockquote:after, span:after {
  content: close-quote;
}

<blockquote>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis
tellus vulputate lobortis. <span>Vivamus fermentum</span>, tortor id ornare ultrices,
ligula ipsum tincidunt pede, et blandit <span>sem pede suscipit pede.</span> Nulla
cursus porta sem. Donec mollis nunc in leo.</blockquote>
```

Las reglas CSS anteriores indican que el navegador debe utilizar diferentes comillas cuando el nivel de anidamiento sea diferente:

> “Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.”

En el ejemplo anterior, tanto `<blockquote>` como `<abbr>` establecen dos pares de comillas. Cuando uno de estos dos elementos se encuentre dentro de otro elemento que ya muestra comillas, la propiedad `quotes` indica que se debe utilizar el segundo par de comillas en vez del primero. Por lo tanto, si utilizas un navegador que soporta la propiedad `quotes`, el ejemplo anterior se visualiza de la siguiente forma:

«Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. "Vivamus fermentum", tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit "sem pede suscipit pede." Nulla cursus porta sem. Donec mollis nunc in leo.»

Figura 4.7. Ejemplo de la propiedad quotes

Los valores de la propiedad `quotes` se indican mediante cadenas de texto. Una cadena de texto es un conjunto de uno o más caracteres encerrados por las comillas dobles (") o las comillas simples ('). Si la cadena contiene comillas dobles, se encierra con las comillas simples y viceversa. Si una cadena de texto tiene tanto comillas simples como dobles, las comillas problemáticas se modifican y se les añade la barra invertida \ por delante:

```
/* Comillas simples encerradas por comillas dobles */
.selector { quotes: "'" "'"; }

/* Comillas dobles encerradas por comillas simples */
.selector { quotes: '"' '"'; }

/* Comillas simples y dobles encerradas por comillas dobles */
.selector { quotes: "\"" "\"" "'" "'"; }

/* Comillas simples y dobles encerradas por comillas simples */
.selector { quotes: '"' '"' '\'' '\''; }
```

Como las comillas se indican mediante cadenas de texto, no están limitadas a un solo carácter. En el siguiente ejemplo se utilizan unas comillas poco habituales formadas por varios caracteres:

```
blockquote, span {
  quotes: "««««" "»»»»" "--" "--";
}

blockquote:before, span:before {
  content: open-quote;
}
blockquote:after, span:after {
  content: close-quote;
}

<blockquote>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis
tellus vulputate lobortis. <span>Vivamus fermentum</span>, tortor id ornare ultrices,
ligula ipsum tincidunt pede, et blandit <span>sem pede suscipit pede.</span> Nulla
cursus porta sem. Donec mollis nunc in leo.</blockquote>
```

Las reglas CSS anteriores producen el siguiente resultado en el navegador:

> “Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.”

En un navegador con soporte de la propiedad `quotes`, el ejemplo anterior se visualiza de la siguiente forma:

««««Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. -- Vivamus fermentum--, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit --sem pede suscipit pede.-- Nulla cursus porta sem. Donec mollis nunc in leo.»»»»

Figura 4.8. Ejemplo de la propiedad quotes

Utilizando el *pseudo-selector* `:lang()` es posible incluso definir diferentes tipos de comillas en función del idioma del contenido:

```
blockquote:lang(en) { quotes: '"' '"' "'" "'" } /* comillas para inglés */
blockquote:lang(es) { quotes: "«" "»" '"' '"' } /* comillas para español */
```

La propiedad `quotes` define otro valor llamado `none` que hace que no se muestre ninguna comilla cuando se utiliza el valor `open-quote` y `close-quote` de la propiedad `content`.

Por último, el estándar de CSS 2.1 recomienda utilizar directamente los códigos definidos en el estándar ISO 10646 para indicar la comilla que se quiere utilizar.

De esta forma, las dos siguientes reglas CSS son equivalentes:

```
blockquote, abbr {
  quotes: "«" "»" "'" "'";
}
```

```
blockquote, abbr {
  quotes: "\00AB" "\00BB" "\0027" "\0027";
}
```

La siguiente tabla recoge las comillas más utilizadas y sus códigos correspondientes:

Carácter tipográfico de la comilla	Cómo la muestran los navegadores	Código ISO 10646
"	"	0022
'	'	0027
‹	<	2039
›	>	203A
«	«	00AB
»	»	00BB
‘	`	2018
’	'	2019
“	``	201C
”	''	201D
„	,,	201E

4.5. Propiedad counter-reset

Definición	Inicializa el valor de uno o más contadores
Valores permitidos	Uno y sólo uno de los siguientes valores: ▪ Uno o más nombres de contadores seguidos opcionalmente por un número entero cada uno ▪ none ▪ inherit
Valor inicial	none
Se aplica a	Todos los elementos
Válida en	all
Se hereda	no
Definición en el estándar	http://www.w3.org/TR/CSS21/generate.html#propdef-counter-reset

La propiedad `counter-reset` se emplea para controlar la numeración automática de CSS 2.1 que se utiliza en las funciones `counter()` y `counters()` de la propiedad `content`.

El funcionamiento básico de la propiedad `counter-reset` es sencillo, pero su flexibilidad y la combinación con otras propiedades y funciones de CSS pueden complicar mucho su interpretación.

En el caso más sencillo, la propiedad `counter-reset` indica el nombre de un contador seguido opcionalmente por un número entero que indica el valor al que se inicializa el contador:

```
body {
  counter-reset: numero_capitulo 0;
}
```

La regla CSS anterior hace que se inicialice a `0` un contador llamado `numero_capitulo` cuando el navegador encuentre el elemento `<body>`. Si el contador no existía, el estándar CSS 2.1 indica que se crea cuando se encuentre el primer elemento `<body>`, es decir, al principio de la página.

La regla CSS anterior se puede emplear para crear el contador que se utiliza posteriormente en las propiedades `counter-increment` y `content`:

```
body {
  counter-reset: numero_capitulo 0;
}
h1:before {
  content: "Capítulo " counter(numero_capitulo);
  counter-increment: numero_capitulo 1;
}
```

Cuando se inicializa y se actualiza el valor de un contador en un mismo elemento, primero se inicializa su valor y después se actualiza según la propiedad `counter-increment`.

Si no se indica el número entero opcional, los navegadores suponen que vale `0`, por lo que las siguientes reglas CSS son equivalentes a las anteriores:

```
body {
  counter-reset: numero_capitulo;
}
h1:before {
  content: "Capítulo " counter(numero_capitulo);
  counter-increment: numero_capitulo 1;
}
```

La propiedad `counter-reset` también cumple con el comportamiento en cascada de las propiedades CSS, por lo que en el siguiente ejemplo:

```
p {
  counter-reset: parrafos;
}
...
p {
  counter-reset: elementos;
}
```

Las reglas CSS anteriores provocan un colisión en el valor de la propiedad `counter-reset`. Según las normas de resolución de colisiones, en este caso gana la última regla y los elementos de tipo `<p>` sólo inicializan el valor del contador `elementos`.

Además de inicializar el valor de los contadores a `0`, también es posible inicializarlos a cualquier otro valor entero positivo o negativo, tal y como muestra el siguiente ejemplo:

```
body {
  counter-reset: numero_capitulo -1;
}
h1:before {
  content: "Capítulo " counter(numero_capitulo);
  counter-increment: numero_capitulo;
  counter-reset: numero_seccion -1;
}
```

La propiedad `counter-reset` también permite indicar varios contadores para inicializarlos de forma simultánea:

```
h1 {
  counter-reset: numero_seccion numero_tabla numero_imagen;
}
```

La regla CSS anterior inicializa a `0` los contadores `numero_seccion`, `numero_tabla` y `numero_imagen` cada vez que se encuentra un elemento `<h1>` en la página. También es posible indicar diferentes valores iniciales para cada contador:

```
h1 {
  counter-reset: numero_seccion 0 numero_tabla -1 numero_imagen -1;
}
```

Además, también es posible indicar el mismo contador más de una vez en la propiedad `counter-reset`. En este caso, se realizan todas las inicializaciones en el orden indicado:

```
p {
  counter-reset: parrafos parrafos -1 parrafos 3;
}

p {
  counter-reset: parrafos parrafos -1 parrafos 3 parrafos;
}
```

La primera regla del ejemplo anterior inicializa el valor del contador `parrafos` a `3`, mientras que la segunda regla inicializa el contador `parrafos` a `0`. Como el navegador procesa las inicializaciones en el mismo orden indicado, la única que se tiene en cuenta es la última.

Los contadores tienen en cuenta el anidamiento de los elementos HTML. Si se consideran las siguientes reglas CSS:

```
div {
  counter-reset: numero_parrafo;
}
div p:before {
  content: "Parrafo " counter(numero_parrafo) " ";
  counter-increment: numero_parrafo;
}
```

La propiedad `counter-reset` crea o inicializa el valor del contador `numero_parrafo` cada vez que encuentra un elemento `<div>`. De esta forma, si la regla CSS anterior se aplica al siguiente código HTML:

```
<div>
  <p>Lorem ipsum dolor sit amet</p>
  <p>Lorem ipsum dolor sit amet</p>
  <p>Lorem ipsum dolor sit amet</p>
  <div>
    <p>Lorem ipsum dolor sit amet</p>
    <p>Lorem ipsum dolor sit amet</p>
    <p>Lorem ipsum dolor sit amet</p>
    <div>
      <p>Lorem ipsum dolor sit amet</p>
      <p>Lorem ipsum dolor sit amet</p>
      <p>Lorem ipsum dolor sit amet</p>
    </div>
  </div>
</div>
```

En el ejemplo anterior, el navegador crea automáticamente tres contadores diferentes pero con el mismo nombre (`numero_parrafo`). Cada vez que se encuentra un elemento `<div>`, el navegador crea o inicializa un contador llamado `numero_parrafo`, por lo que todos los párrafos del ejemplo anterior disponen de la misma numeración (`1`, `2` y `3`).

Si se visualiza el ejemplo anterior con un navegador que soporte completamente las propiedades `counter-reset`, `counter-increment` y `content`, el resultado es el que muestra la siguiente imagen:

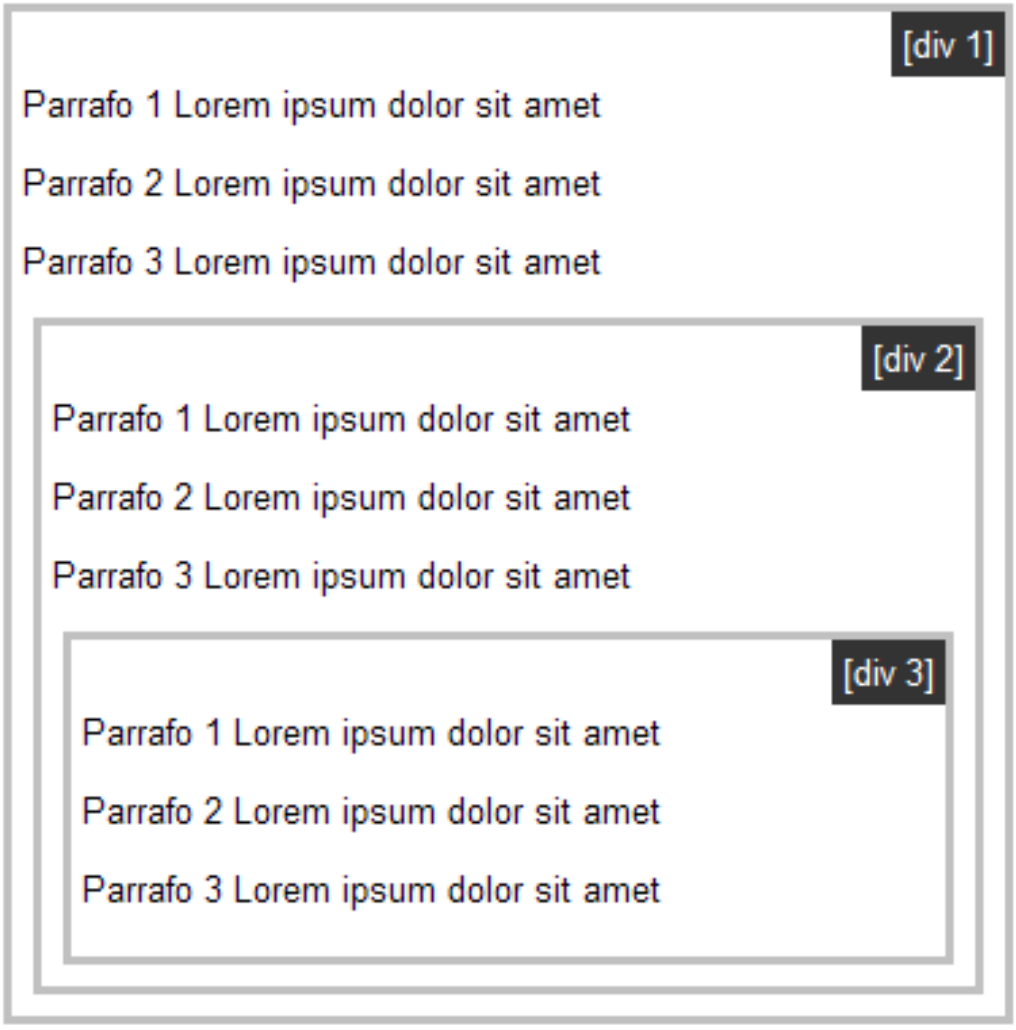


Figura 4.9. Ejemplo de las propiedades counter-reset, counter-increment y content

Los elementos ocultos mediante la propiedad `display` (`display: none`) no pueden modificar el valor de los contadores, mientras que los elementos ocultos mediante la propiedad `visibility` (`visibility: hidden`) si que los actualizan:

```
p.oculto {
  display: none;
  counter-reset: parrafos;  /* No se inicializa porque el elemento
                               se ha ocultado mediante display: none */
}

p.invisible {
  visibility: hidden;
  counter-reset: parrafos;  /* Se inicializa a 0 porque el elemento
                               se ha hecho invisible con visibility: hidden */
}
```

4.6. Propiedad counter-increment

Definición	Actualiza el valor de uno o más contadores
Valores permitidos	Uno y sólo uno de los siguientes valores: ▪ Uno o más nombres de contadores seguidos opcionalmente por un número entero cada uno ▪ none ▪ inherit
Valor inicial	none
Se aplica a	Todos los elementos
Válida en	all
Se hereda	no
Definición en el estándar	http://www.w3.org/TR/CSS21/generate.html#propdef-counter-increment

La propiedad `counter-increment` se emplea para controlar la numeración automática de CSS 2.1 que se utiliza en las funciones `counter()` y `counters()` de la propiedad `content`.

El funcionamiento básico de la propiedad `counter-increment` es sencillo, pero su flexibilidad y la combinación con otras propiedades y funciones de CSS pueden complicar mucho su interpretación.

En el caso más sencillo, la propiedad `counter-increment` indica el nombre de un contador seguido opcionalmente por un número entero que indica la cantidad en la que se incrementa:

```
h1 {
  counter-increment: numero_capitulo 1;
}
```

La regla CSS anterior hace que se incremente en una unidad el valor de un contador llamado `numero_capitulo` cada vez que la página incluya un elemento `<h1>`. Si el contador no existía, el estándar CSS 2.1 indica que cuando se encuentre el primer elemento `<h1>`, se crea el contador, se inicializa al valor `0` y posteriormente se actualiza su valor según la propiedad `counter-increment` indicada.

Este contador básico se puede emplear para añadir una numeración automática a los elementos `<h1>` de la página:

```
h1:before {
  content: "Capítulo " counter(numero_capitulo);
  counter-increment: numero_capitulo 1;
}
```

Cuando se actualiza y se utiliza el valor de un contador en un mismo elemento, primero se actualiza su valor y después se utiliza en la propiedad `content`.

Si no se indica el número entero opcional, los navegadores suponen que vale `1`, por lo que la siguiente regla CSS es equivalente a la anterior:

```
h1:before {
  content: "Capítulo " counter(numero_capitulo);
  counter-increment: numero_capitulo;
}
```

Además de incrementar el valor de los contadores, también es posible decrementarlo o no modificarlo. El siguiente ejemplo amplía el anterior para no incrementar el valor del contador en algunos elementos `<h1>` especiales:

```
h1:before {
  content: "Capítulo " counter(numero_capitulo);
  counter-increment: numero_capitulo;
}
h1.especial {
  counter-increment: numero_capitulo 0;
}
```

No obstante, si se quiere inicializar un contador para que vuelva a valer `0` o cualquier otro valor numérico, es preciso utilizar la propiedad `counter-reset`.

La propiedad `counter-increment` también permite indicar varios contadores para actualizarlos de forma simultánea:

```
p {
  counter-increment: elementos parrafos especial 2;
}
```

La regla CSS anterior incrementa en una unidad el valor de los contadores `elementos` y `parrafos` cada vez que se encuentra un elemento `<p>` en la página. Además, incrementa en `2` unidades el valor de otro contador llamado `especial` cada vez que encuentra un elemento `<p>`.

También es posible indicar el mismo contador más de una vez en la propiedad `counter-increment`:

```
p {
  counter-increment: elementos parrafos especial 2 parrafos elementos 3;
}
```

En el ejemplo anterior, cada vez que se encuentra un elemento `<p>` en la página se incrementa `4` unidades el valor del contador `elementos`, `2` unidades el valor del contador `parrafos` y `2` unidades el valor del contador `especial`.

Los contadores tienen en cuenta el anidamiento de los elementos HTML. Si se consideran las siguientes reglas CSS:

```
div {
  counter-reset: numero_parrafo;
}
div p:before {
  content: "Parrafo " counter(numero_parrafo) " ";
```

```
  counter-increment: numero_parrafo;
}
```

La propiedad `counter-reset` crea o inicializa el valor del contador `numero_parrafo` cada vez que encuentra un elemento `<div>`. De esta forma, si la regla CSS anterior se aplica al siguiente código HTML:

```
<div>
  <p>Lorem ipsum dolor sit amet</p>
  <p>Lorem ipsum dolor sit amet</p>
  <p>Lorem ipsum dolor sit amet</p>
  <div>
    <p>Lorem ipsum dolor sit amet</p>
    <p>Lorem ipsum dolor sit amet</p>
    <p>Lorem ipsum dolor sit amet</p>
    <div>
      <p>Lorem ipsum dolor sit amet</p>
      <p>Lorem ipsum dolor sit amet</p>
      <p>Lorem ipsum dolor sit amet</p>
    </div>
  </div>
</div>
```

En el ejemplo anterior, el navegador crea automáticamente tres contadores diferentes pero con el mismo nombre (`numero_parrafo`). Cada vez que se encuentra un elemento `<div>`, el navegador crea o inicializa un contador llamado `numero_parrafo`, por lo que todos los párrafos del ejemplo anterior disponen de la misma numeración (`1`, `2` y `3`).

Si se visualiza el ejemplo anterior con un navegador que soporte completamente las propiedades `counter-increment`, `counter-reset` y `content`, el resultado es el que muestra la siguiente imagen:

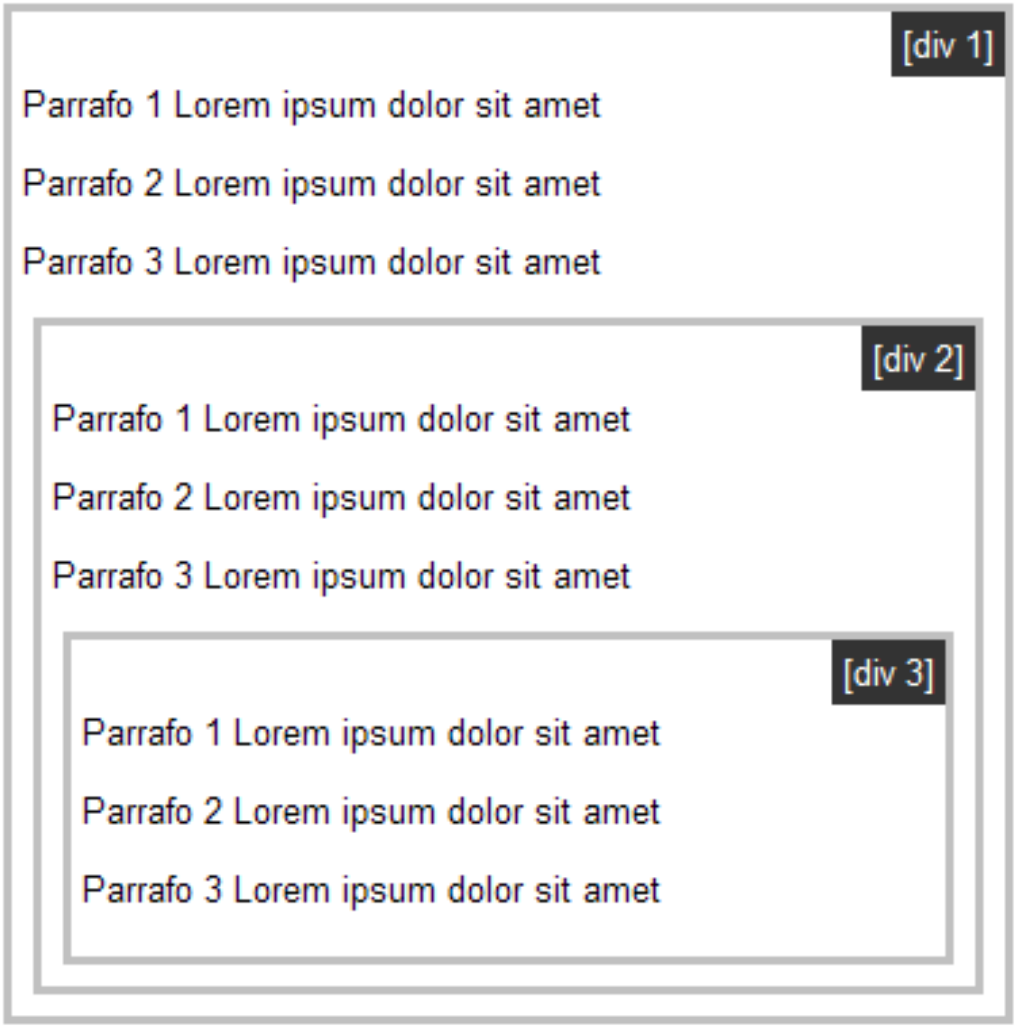


Figura 4.10. Ejemplo de las propiedades counter-increment, counter-reset y content

Los elementos ocultos mediante la propiedad `display` (`display: none`) no modifican el valor de los contadores, mientras que los elementos ocultos mediante la propiedad `visibility` (`visibility: hidden`) si que los actualizan:

```
p.oculto {
  display: none;
  counter-increment: parrafos;  /* No se actualiza porque el elemento
                                   se ha ocultado mediante display: none */
}

p.invisible {
  visibility: hidden;
  counter-increment: parrafos;  /* Se actualiza porque el elemento
                                   se ha hecho invisible con visibility: hidden */
}
```

4.7. Propiedad content

Definición	Genera contenido de forma dinámica
Valores permitidos	Existen cuatro opciones diferentes para establecer el valor o valores de esta propiedad: 1. Uno o más de los siguientes valores: ▪ cadena de texto ▪ URL ▪ contador ▪ valor de un atributo ▪ open-quote ▪ close-quote ▪ no-open-quote ▪ no-close-quote 2. normal 3. none 4. inherit
Valor inicial	normal
Se aplica a	Solamente a los pseudo-elementos :before y :after
Válida en	all
Se hereda	no
Definición en el estándar	http://www.w3.org/TR/CSS21/generate.html#propdef-content

La propiedad `content` es una de las propiedades CSS más poderosas y a la vez más controvertidas. La propiedad `content` se emplea para generar nuevo contenido de forma dinámica e insertarlo en la página HTML. Como CSS es un lenguaje de hojas de estilos cuyo único propósito es controlar el aspecto o presentación de los contenidos, algunos diseñadores defienden que no es correcto generar nuevos contenidos mediante CSS.

En primer lugar, el estándar CSS 2.1 indica que la propiedad `content` sólo puede utilizarse en los *pseudo-elementos* `:before` y `:after`. Como su propio nombre indica, estos *pseudo-elementos* permiten seleccionar (y por tanto modificar) la parte anterior o posterior de un elemento de la página.

El siguiente ejemplo muestra cómo añadir la palabra `Capítulo` delante del contenido de cada título de sección `<h1>`:

```
h1:before {
  content: "Capítulo ";
}
```

Los *pseudo-elementos* `:before` y `:after` se pueden utilizar sobre cualquier elemento de la página. El siguiente ejemplo añade la palabra `Nota:` delante de cada párrafo cuya clase CSS sea `nota`:

```
p.nota:before {
  content: "Nota: ";
}
```

Combinando las propiedades `content` y `quotes` con los *pseudo-elementos* `:before` y `:after`, se pueden añadir de forma dinámica comillas de apertura y de cierre a todos los elementos `<blockquote>` de la página:

```
blockquote:before {
  content: open-quote;
}
blockquote:after {
  content: close-quote;
}
blockquote {
  quotes: "«" "»";
}
```

Los contenidos insertados dinámicamente en un elemento son a todos los efectos parte de ese mismo elemento, por lo que heredan el valor de todas sus propiedades CSS.

Los dos valores más sencillos de la propiedad `content` son `none` y `normal`. En la práctica, estos dos valores tienen el mismo efecto ya que hacen que el *pseudo-elemento* no se genere.

El siguiente valor que se puede indicar en la propiedad `content` es una cadena de texto. En el estándar CSS 2.1, una cadena de texto es un conjunto de uno o más caracteres encerrados por las comillas dobles (") o las comillas simples ('). Si la cadena contiene comillas dobles, se encierra con las comillas simples y viceversa. Si una cadena de texto tiene tanto comillas simples como dobles, las comillas problemáticas se modifican y se les añade la barra invertida `\` por delante:

```
p:before {
  content: "Contenido generado \"dinámicamente\" mediante CSS. ";
}
#ultimo:after {
  content: " Fin de los 'contenidos' de la página.";
}
```

Las cadenas de texto sólo permiten incluir texto básico. Si se incluye alguna etiqueta HTML en la cadena de texto, el navegador muestra la etiqueta tal y como está escrita, ya que no las interpreta. Para incluir un salto de línea en el contenido generado, se utiliza el caracter especial `\A`

El siguiente valor aceptado por la propiedad `content` es una URL, que suele utilizarse para indicar la URL de una imagen que se quiere añadir de forma dinámica al contenido. La sintaxis es idéntica al resto de URL que se pueden indicar en otras propiedades CSS:

```
span.especial:after {
  content: url("imagenes/imagen.png");
}
```

Otros valores que se pueden indicar en la propiedad `content` son `open-quote`, `close-quote`, `no-open-quote` y `no-close-quote`. Los dos primeros indican que se debe mostar una comilla de apertura o de cierre respectivamente. Las comillas utilizadas se establecen mediante la propiedad `quotes`:

```
blockquote { quotes: "«" "»" '"' '"' }
blockquote:before {
```

```
  content: open-quote;
}
blockquote:after {
  content: close-quote;
}
```

Los valores `no-open-quote` y `no-close-quote` se utilizan para no mostrar ninguna comilla en ese elemento, pero incrementando el nivel de anidamiento de las comillas. De esta forma se puede evitar mostrar una comilla en un determinado elemento mientras se mantiene la jerarquía de comillas establecida por la propiedad `quotes`.

Uno de los valores más avanzados de la propiedad `content` es `attr()`, que permite obtener el valor de un atributo del elemento sobre el que se utiliza la propiedad `content`. En el siguiente ejemplo, se modifican los elementos `<abbr>` y `<acronym>` para que muestren entre paréntesis el valor de sus atributos `title`:

```
abbr:after, acronym:after {
  content: " (" attr(title) ")"
}
```

El valor de la propiedad `content` anterior en realidad es la combinación de tres valores:

- Cadena de texto `" ("`, que es el paréntesis de apertura tras el cual se muestra el valor del atributo `title`.
- Atributo `title` del elemento obtenido mediante `attr(title)`
- Cadena de texto `")"`, que es el paréntesis de cierre que se muestra detrás del valor del atributo `title`.

Si el elemento no dispone del atributo solicitado, la función `attr(nombre_del_atributo)` devuelve una cadena de texto vacía. Utilizando `attr()` solamente se puede obtener el valor de los atributos del elemento al que se aplica la propiedad `content`.

La función `attr()` es muy útil por ejemplo para mostrar la dirección a la que apuntan los enlaces de la página:

```
a:after {
  content: " (" attr(href) ")";
}
```

Los últimos valores que se pueden indicar en la propiedad `content` son los contadores creados con las propiedades `counter-increment` y `counter-reset`. Los contadores más sencillos se muestran con la función `counter(nombre_del_contador)`. El siguiente ejemplo crea dos contadores llamado `capitulo` y `seccion` para utilizarlos con los elementos `<h1>` y `<h2>`:

```
body {
  counter-reset: capitulo;
}
h1 {
  counter-reset: seccion;
}
h1:before {
```

```
  content: "Capítulo " counter(capitulo) ". ";
  counter-increment: capitulo;
}
h2:before {
  content: counter(capitulo) "." counter(seccion) " ";
  counter-increment: seccion;
}
```

En el ejemplo anterior, se crea e inicializa su valor a `0` un contador llamado `capitulo` cuando se encuentre el elemento `<body>`, es decir, al comienzo de la página. Además, se crea e inicializa su valor a `0` otro contador llamado `seccion` cada vez que se encuentra un elemento `<h1>` en la página.

Posteriormente, se añade de forma dinámica a los elementos `<h1>` y `<h2>` el contenido generado mediante los contadores. Los elementos `<h1>` utilizan el contador `capitulo` y lo incrementan en una unidad cada vez que lo utilizan. Los elementos `<h2>` utilizan los dos contadores para generar un contenido que muestra su numeración completa. Además, los elementos `<h2>` actualizan el valor del contador `seccion`.

Cuando un mismo elemento inicializa/actualiza un contador y lo utiliza en la propiedad `content`, en primer lugar se inicializa/actualiza y después, el valor actualizado es el que se utiliza mediante `counter()`.

Además de mostrar el valor de un contador básico, la función `counter()` permite indicar el estilo con el que se muestra el valor del contador. La lista de estilos permitidos son los mismos que los de la propiedad `list-style-type`.

El siguiente ejemplo modifica el anterior para mostrar el valor de los contadores en números romanos:

```
body {
  counter-reset: capitulo;
}
h1 {
  counter-reset: seccion;
}
h1:before {
  content: "Capítulo " counter(capitulo, upper-roman) ". ";
  counter-increment: capitulo;
}
h2:before {
  content: counter(capitulo, upper-roman) "." counter(seccion, upper-roman) " ";
  counter-increment: seccion;
}
```

Los estilos de los contadores también se pueden emplear para no mostrar el valor de los contadores en algunos elementos:

```
p {
  counter-increment: parrafos;
}
p:before {
  content: counter(parrafos);
```

```
}
#especial p:before {
  content: counter(parrafos, none);
}
```

Aunque el valor de los contadores siempre es númerico, también es pueden emplear estilos gráficos como `square`, `disc` o `circle`:

```
h2 {
  counter-increment: seccion;
}
h2:before {
  content: counter(seccion, disc);
}
```

La función `counter()` solamente muestra el valor de un contador. Por su parte, la función `counters()` se utiliza para mostrar de forma simultánea el valor de todos los contadores asociados con el elemento. Como se explica en la descripción de las propiedades `counter-increment` y `counter-reset`, los contadores se pueden anidar y un mismo elemento puede tener asociados varios contadores diferentes con el mismo nombre.

El siguiente ejemplo muestra unas reglas CSS que crean un contador para los elementos de una lista `<ol>`:

```
ol {
  counter-reset: elemento;
  list-style-type: none;
}
li:before {
  content: counter(elemento) ". ";
  counter-increment: elemento;
}
```

Si se considera el siguiente código HTML:

```
<ol>
  <li>Elemento</li>
  <li>Elemento</li>
  <li>Elemento
    <ol>
        <li>Elemento</li>
        <li>Elemento
          <ol>
            <li>Elemento</li>
            <li>Elemento</li>
            <li>Elemento</li>
          </ol>
        </li>
        <li>Elemento</li>
    </ol>
  </li>
  <li>Elemento</li>
</ol>
```

Si se aplican las reglas CSS al código HTML anterior, se crean tres contadores diferentes con el mismo nombre (`elemento`) y el resultado es el que muestra la siguiente imagen:

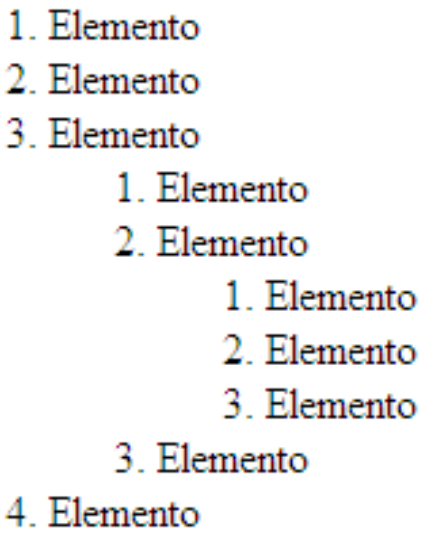


Figura 4.11. Ejemplo de la propiedad content

Sin embargo, si se utiliza la función `counters()` en las reglas CSS anteriores:

```
ol {
  counter-reset: elemento;
  list-style-type: none;
}
li:before {
  content: counters(elemento, '. ') ". ";
  counter-increment: elemento;
}
```

Ahora, el aspecto que muestra la lista de elementos es el de la siguiente imagen:

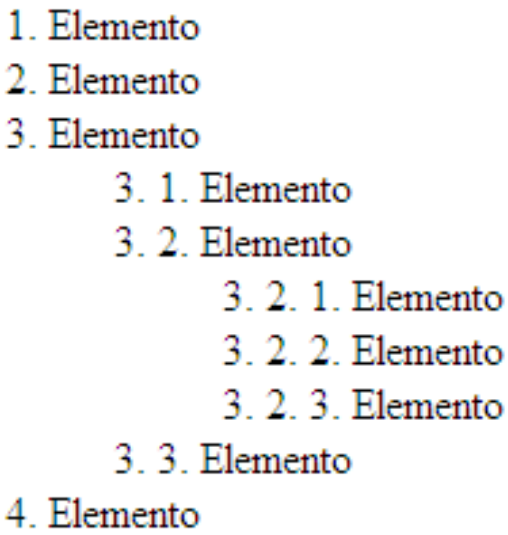


Figura 4.12. Ejemplo de la propiedad content

En el ejemplo anterior, cada vez que se encuentra un elemento `<ol>` se crea un contador llamado `elemento`. Por este motivo, los elementos `<li>` anidados se ven afectados por varios contadores llamados `elemento`. La función `counter()` sólo muestra el valor del contador que afecta más directamente al elemento, mientras que la función `counters()` muestra todos los contadores empezando desde el más externo hasta llegar al más interno.

El segundo argumento de la función `counters()` es una cadena de texto que se emplea para separar los valores de los diferentes contadores. CSS 2.1 no permite utilizar diferentes separadores para cada nivel jerárquico.

Por último, la función `counters()` también permite indicar el estilo con el que se muestra el valor de los contadores. De esta forma, el siguiente ejemplo modifica el anterior para mostrar el valor de todos los contadores en números romanos:

```
ol {
  counter-reset: elemento;
  list-style-type: none;
}
li:before {
  content: counters(elemento, '. ', upper-roman) ". ";
  counter-increment: elemento;
}
```

Capítulo 5. Frameworks

Los *frameworks* se utilizan en el ámbito de la programación de aplicaciones desde hace décadas. Recientemente han comenzado a utilizarse para crear aplicaciones web y ya han aparecido decenas de *frameworks* para CSS.

Genéricamente, un *framework* es un conjunto de herramientas, librerías, convenciones y buenas prácticas que pretenden encapsular las tareas repetitivas en módulos genéricos fácilmente reutilizables.

De la misma forma, un *framework* CSS es un conjunto de herramientas, hojas de estilos y buenas prácticas que permiten al diseñador web olvidarse de las tareas repetitivas para centrarse en los elementos únicos de cada diseño en los que puede aportar valor.

¿Qué aporta a un diseñador descubrir en cada diseño que debe neutralizar los estilos por defecto que aplican los navegadores? ¿Qué aporta un diseñador que se dedica continuamente a resolver los mismos problemas que se producen al crear *layouts* complejos? ¿Por qué el diseñador se dedica a tareas y problemas que han sido resueltos satisfactoriamente hace mucho tiempo?

Los *frameworks* CSS más completos incluyen utilidades para que el diseñador no tenga que trabajar en ningún aspecto genérico del diseño web. Por este motivo, es habitual que los mejores *frameworks* CSS incluyan herramientas para:

- Neutralizar los estilos por defecto que aplican los navegadores. Se trata de la habitual hoja de estilos *reset.css* que todos los diseñadores profesionales utilizan.
- Manejar correctamente el texto, de forma que se todos los contenidos se vean exactamente igual en todos los navegadores y que sean adaptables para mejorar su accesibilidad y permitir su acceso en cualquier medio y/o dispositivo.
- Crear cualquier estructura compleja o *layout* de forma sencilla, con la seguridad de que funciona correctamente en cualquier versión de cualquier navegador.

Actualmente existen decenas de *frameworks* CSS, tal y como se recoge en la página *List of CSS frameworks* (http://en.wikipedia.org/wiki/List_of_CSS_frameworks) de la Wikipedia. En las siguientes secciones se explica detalladamente el *framework* YUI creado por Yahoo!

5.1. El framework YUI

El *framework* o librería YUI (http://developer.yahoo.com/yui/) (*Yahoo! User Interface*) es un conjunto de utilidades y controles escritos en JavaScript que se utilizan para crear aplicaciones web dinámicas complejas. Además, la librería YUI incluye varias utilidades relacionadas con CSS, por lo que también se considera un *framework* CSS.

Yahoo! distribuye gratuitamente la librería YUI en forma de software libre y bajo la licencia BSD (http://developer.yahoo.com/yui/license.html) , que permite utilizar YUI para proyectos de cualquier tipo, incluso comerciales.

Lo mejor de YUI es que cuenta con el respaldo de Yahoo!, que utiliza su librería en muchas de sus miles de millones de páginas vistas diarias. Además, YUI cuenta con una gran documentación que incluye cientos de ejemplos de uso.

YUI se ejecuta correctamente en todos los navegadores modernos e incluso en algún navegador obsoleto. El soporte de los navegadores se puede consultar en la lista de los navegadores soportados por YUI (http://developer.yahoo.com/yui/articles/gbs/) , que se actualiza periódicamente. A continuación se muestra la lista de navegadores soportados en cada sistema operativo:

- Windows XP: Firefox 2 y 3, Internet Explorer 6 y 7, Opera 9.5
- Windows Vista: Firefox 3, Internet Explorer 7
- Mac OS X 10.5: Firefox 2 y 3, Opera 9.5, Safari 3.1

5.2. Primeros pasos con el framework YUI

Para empezar a trabajar con YUI, lo primero que debes hacer es descargar la librería completa. Desde la página principal de YUI (http://developer.yahoo.com/yui/) se puede pulsar directamente sobre el botón *Download YUI* para que nos redirija automáticamente hasta la página de descarga. La versión de YUI que se utiliza actualmente es la 2, aunque la versión en pruebas de YUI 3 ya se ha publicado. YUI 2 se descarga mediante un archivo comprimido que ocupa unos 9 MB.

Una vez descargado, descomprime el archivo en cualquier directorio del sistema y verás que incluye varios directorios. De todos ellos, los más importantes para el programador y diseñador son los siguientes:

- `docs/`: incluye más de 300 páginas de documentación y ayuda sobre todos los componentes de YUI
- `examples/`: incluye casi 300 ejemplos de uso de todos y cada uno de los componentes de YUI
- `build/`: incluye todos los archivos JavaScript y CSS que se deben incluir en las páginas que utilizan YUI

5.2.1. Componentes CSS del framework YUI

La librería YUI incluye los siguientes cuatro componentes de CSS que se pueden utilizar de forma separada, conjunta o combinada de cualquier forma:

- `reset.css`: conjunto de estilos que eliminan y neutralizan todas las inconsistencias en el estilo por defecto que aplican los navegadores a los elementos HTML.
- `base.css`: conjunto de estilos que se pueden aplicar después de neutralizar los estilos por defecto del navegador y que muestran cada elemento HTML de forma adecuada.
- `fonts.css`: conjunto de estilos que permiten mostrar el texto con un tamaño consistente en cualquier navegador.

- `grids.css`: conjunto de estilos que permiten crear cientos de *layouts* o estructuras de páginas comunes.

Los componentes CSS de YUI funcionan tanto en el modo *quirks* como en el modo *standards* de los navegadores. No obstante, YUI recomienda activar el modo *standards* utilizando el siguiente DOCTYPE en las páginas:

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

5.2.2. Enlazando los archivos del framework YUI

Todas las páginas que hacen uso de los componentes de la librería YUI deben enlazar sus archivos. La arquitectura de YUI es tan modular que permite enlazar únicamene los archivos que necesita cada página y no todos los archivos de la librería, lo que mejora el rendimiento de la aplicación.

Si sólo quieres utilizar uno de los componentes CSS de YUI, solamente debes enlazar una hoja de estilos mediante la etiqueta `<link>`. Si por ejemplo sólo se quiere utilizar el componente `reset.css` para inicializar todos los estilos del navegador, los pasos necesarios son los siguientes:

1) Copia el archivo `reset.css` que se encuentra en el directorio `build/reset` al directorio en el que guardas todas tus hojas de estilos CSS.

2) Enlaza la hoja de estilos `reset.css` en tus páginas utilizando la siguiente etiqueta `<link>`:

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

El valor del atributo `href` debe contener la ruta completa (absoluta o relativa) hasta el archivo CSS enlazado.

Aunque esta forma de enlazar los recursos CSS de YUI es la más sencilla, también es la más ineficiente. El motivo es que los archivos CSS originales incluyen comentarios, espacios en blanco, saltos de línea y otros elementos que aumentan innecesariamente el tamaño de los archivos.

Por este motivo, sólo se recomienda el uso de los archivos CSS *normales* cuando el diseñador se encuentra diseñando la página y puede necesitar consultar el contenido de los archivos CSS de YUI. Cuando la página se encuentra terminada, la recomendación es utilizar los archivos CSS comprimidos que proporciona YUI.

Los archivos comprimidos se encuentran en los mismos directorios que los archivos CSS *normales*. La única diferencia es que los archivos comprimidos añaden el prefijo `-min` en su nombre, indicando que se trata de una versión minimizada de los archivos CSS *normales*.

Siguiendo con el ejemplo anterior, para enlazar la versión comprimida de la hoja de estilos `reset.css` se debe utilizar la siguiente etiqueta `<link>`:

```
<link rel="stylesheet" type="text/css" href="css/reset-min.css">
```

Si se utilizan varios componentes CSS de YUI, se sigue el mismo razonamiento para enlazar todas las hojas de estilos. El siguiente ejemplo muestra el caso en el que se utilizan las hojas de estilos `reset.css`, `fonts.css` y `grids.css`:

```
<link rel="stylesheet" type="text/css" href="css/reset-min.css">
<link rel="stylesheet" type="text/css" href="css/fonts-min.css">
<link rel="stylesheet" type="text/css" href="css/grids-min.css">
```

Como es muy habitual enlazar varias hojas de estilos a la vez, YUI proporciona un método alternativo para hacerlo de forma mucho más eficiente. En concreto, YUI proporciona las hojas de estilos `reset-fonts.css` y `reset-fonts-grids.css` por ser las combinaciones de hojas de estilos más habituales. De esta forma, el ejemplo anterior se puede rehacer utilizando una sola etiqueta `<link>`:

```
<link rel="stylesheet" type="text/css" href="css/reset-fonts-grids.css">
```

La hoja de estilos `reset-fonts-grids.css` combina las versiones comprimidas de las tres hojas de estilos individuales. Además de la reducción en el tamaño total del archivo, la gran ventaja es que sólo se realiza una petición al servidor, en vez de las tres peticiones necesarias en el ejemplo anterior.

Por último, existe otra forma aún más eficiente de enlazar los componentes CSS necesarios. Desde hace unos meses, Yahoo! ofrece la posibilidad de enlazar gratuitamente las hojas de estilos CSS desde sus propios servidores. En efecto, nuestras páginas pueden descargar los archivos de YUI directamente desde los servidores de Yahoo! y de forma completamente gratuita.

A continuación se muestra la etiqueta `<link>` necesaria para descargar las mismas hojas de estilos del ejemplo anterior directamente desde los servidores de Yahoo!:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/
reset-fonts-grids/reset-fonts-grids.css">
```

Si sólo se emplean por ejemplo los componentes `reset.css` y `base.css`, las etiquetas necesarias se muestran a continuación:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/reset/
reset-min.css">
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/base/
base-min.css">
```

Aunque deducir las direcciones completas de cada componente CSS es trivial, YUI proporciona una herramienta gráfica llamada *YUI: Configuration and Hosting* (http://developer.yahoo.com/yui/articles/hosting/) que genera automáticamente las etiquetas `<link>` de los componentes CSS seleccionados.

Las ventajas de utilizar los propios servidores de Yahoo! para servir los archivos de YUI son numerosas:

- Yahoo! dispone de miles de servidores repartidos por todo el mundo y la descarga se realiza siempre desde el servidor más cercano al usuario. De hecho, los servidores que proporciona Yahoo! son exactamente los mismos que utiliza Yahoo! en sus aplicaciones.

- Yahoo! sirve todos sus archivos comprimidos mediante el servidor web y con la información precisa para que el navegador los almacene en su cache local. La compresión reduce el tamaño de los archivos CSS hasta un 60% y el control de la cache evita que el usuario tenga que descargar repetidamente el mismo archivo.
- Los servidores de Yahoo! son infinitamente más potentes y fiables que cualquier servidor web propio. Además, la velocidad de transmisión de los archivos también es infinitamente más rápida que la conseguida con cualquier conexión de red propia.
- El servicio es completamente gratuito y permite enlazar cualquier versión actual o pasada de la librería YUI.

5.3. Inicializando estilos con el framework YUI

El primer componente CSS de YUI es `reset.css`, una hoja de estilos utilizada para neutralizar los estilos que aplican por defecto los navegadores a todos los elementos HTML.

A continuación se muestra el contenido completo de `reset.css`:

```
/*
Copyright (c) 2008, Yahoo! Inc. All rights reserved.
Code licensed under the BSD License:
http://developer.yahoo.net/yui/license.txt
version: 2.5.2
*/
html{color:#000;background:#FFF;}
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,code,form,fieldset,legend,input,textarea,p,blockquo
table{border-collapse:collapse;border-spacing:0;}
fieldset,img{border:0;}
address,caption,cite,code,dfn,em,strong,th,var{font-style:normal;font-weight:normal;}
li{list-style:none;}
caption,th{text-align:left;}
h1,h2,h3,h4,h5,h6{font-size:100%;font-weight:normal;}
q:before,q:after{content:'';}
abbr,acronym {border:0;font-variant:normal;}
/* to preserve line-height and selector appearance */
sup {vertical-align:text-top;}
sub {vertical-align:text-bottom;}
input,textarea,select{font-family:inherit;font-size:inherit;font-weight:inherit;}
/*to enable resizing for IE*/
input,textarea,select{*font-size:100%;}
/*because legend doesn't inherit in IE */
legend{color:#000;}
```

Al contrario que las hojas de estilo de tipo *reset* de otras librerías y frameworks, el archivo `reset.css` solamente neutraliza los estilos que aplica por defecto el navegador. El cambio más evidente que provoca el `reset.css` de YUI es que el texto de todos los elementos se muestra con el mismo aspecto. No importa si se trata de un párrafo `<p>`, una etiqueta `<strong>` o un título de sección `<h1>`, ya que el `reset.css` anterior hace que todos se vean igual. Por tanto, las páginas a las que sólo se aplica la hoja de estilos `reset.css` no están preparadas para mostrarlas a los usuarios.

Las hojas de estilo `reset.css` de otros frameworks neutralizan los estilos por defecto y depués aplican unos estilos adecuados para poder mostrar correctamente las páginas: cada título de sección tiene un tamaño mayor que el siguiente, los elementos `<strong>` se ven en negrita, los márgenes son adecuados para separar los contenidos de texto, etc.

YUI dispone de otro componente CSS llamado `base.css` para aplicar unos estilos adecuados a las páginas que han sido neutralizadas mediante `reset.css`. Por lo tanto, una página a la que se le aplican las hojas de estilos `reset.css` y `base.css` ya están listas para mostrarse a los usuarios, a falta de la personalización del aspecto que realice posteriormente el diseñador.

El contenido de la hoja de estilos `base.css` de YUI se muestra a continuación:

```
/*
Copyright (c) 2008, Yahoo! Inc. All rights reserved.
Code licensed under the BSD License:
http://developer.yahoo.net/yui/license.txt
version: 2.5.2
*/
/* base.css, part of YUI's CSS Foundation */
h1 {
  font-size:138.5%;
}
h2 {
  font-size:123.1%;
}
h3 {
  font-size:108%;
}
h1,h2,h3 {
  margin:1em 0;
}
h1,h2,h3,h4,h5,h6,strong {
  font-weight:bold;
}
abbr,acronym {
  border-bottom:1px dotted #000;
  cursor:help;
}
em {
  font-style:italic;
}
blockquote,ul,ol,dl {
  margin:1em;
}
ol,ul,dl {
  margin-left:2em;
}
ol li {
  list-style: decimal outside;
}
ul li {
  list-style: disc outside;
}
dl dd {
```

```
  margin-left:1em;
}
th,td {
  border:1px solid #000;
  padding:.5em;
}
th {
  font-weight:bold;
  text-align:center;
}
caption {
  margin-bottom:.5em;
  text-align:center;
}
p,fieldset,table,pre {
  margin-bottom:1em;
}
input[type=text],input[type=password],textarea{width:12.25em;*width:11.9em;}
```

Después de aplicar `reset.css` todos los contenidos de texto de la página se muestran con un tamaño de letra de `16px`. La hoja de estilos `base.css` hace que los títulos de sección `<h1>`, `<h2>` y `<h3>` se muestren más grandes que el resto de texto y también hace que se muestren en negrita. Otros de los estilos comunes que establece `base.css` son los marcadores que muestran los elementos de las listas, los bordes de las celdas de tabla y los márgenes verticales y laterales de los elementos.

La mayoría de páginas creadas con los componentes CSS de YUI utilizan `reset.css` y `base.css`, por lo que es común que incluyan las siguientes etiquetas `<link>`:

```
<link rel="stylesheet" type="text/css" href="css/reset-min.css">
<link rel="stylesheet" type="text/css" href="css/base-min.css">
```

Recuerda que también puedes enlazar estas mismas hojas de estilos directamente desde los servidores de Yahoo!:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/reset/
reset-min.css">
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/base/
base-min.css">
```

5.4. Texto con el framework YUI

El manejo del texto es una de las grandes ventajas aportadas por el uso del framework. Definir las propiedades del texto de la forma que indica YUI asegura que tus páginas se ven exactamente igual en cualquier versión de cualquier navegador. Además, garantiza que las propiedades del texto de tus páginas serán consistentes y que los contenidos de texto se adaptan perfectamente a cualquier medio y/o dispositivo.

En primer lugar, el tercer componente CSS de YUI es `fonts.css`, una hoja de estilos tan sencilla como se muestra a continuación:

```
/*
Copyright (c) 2008, Yahoo! Inc. All rights reserved.
```

```
Code licensed under the BSD License:
http://developer.yahoo.net/yui/license.txt
version: 2.5.2
*/
body {font:13px/1.231 arial,helvetica,clean,sans-serif;*font-size:small;*font:x-small;}
table {font-size:inherit;font:100%;}
pre,code,kbd,samp,tt {font-family:monospace;*font-size:108%;line-height:100%;}
```

Después de aplicar `reset.css`, `base.css` y `fonts.css`, las páginas muestran el texto normal con un tamaño de letra de `13px` y un interlineado de `16px`. Los títulos `<h1>` se muestran con un tamaño de `18px`, los títulos `<h2>` con un tamaño de `16px` y los títulos `<h3>` con un tamaño de `14px`. Además, todos los elementos de texto se muestran con el tipo de letra `Arial` o similar, salvo los elementos `<code>` y `<pre>` que se muestran con una fuente de ancho fijo.

YUI recomienda establecer todos los tamaños de letra utilizando el porcentaje como unidad de medida, porque asegura que el texto se muestra de forma más consistente que al utilizar la unidad `em`. Como el tamaño de letra base establecido por `fonts.css` es `13px`, esta medida es la que corresponde al valor `100%`. Utilizando una sencilla operación se puede obtener el porcentaje correspondiente a cualquier tamaño en píxeles. La siguiente tabla muestra algunos de los valores más comunes:

Para obtener este tamaño en píxeles...	**...utiliza el siguiente valor de porcentaje**
10px	77%
11px	85%
12px	93%
13px	100%
14px	108%
15px	116%
16px	123.1%
17px	131%
18px	138.5%
19px	146.5%
20px	153.9%
21px	161.6%
22px	167%
23px	174%
24px	182%
25px	189%
26px	197%

Por lo tanto, si se quiere modificar por ejemplo el tamaño de letra de los párrafos y de los títulos de sección `<h1>` y `<h2>`, debes utilizar las siguientes reglas en tu hoja de estilos propia:

```
p {
  font-size: 123.1%;  /* equivale a 16px */
}
h1 {
  font-size: 182%;    /* equivale a 24px */
}
h2 {
  font-size: 167%;    /* equivale a 22px */
}
h3 {
  font-size: 153.9%;  /* equivale a 20px */
}
```

De la misma forma, también es posible modificar el tipo de letra de cualquier elemento de la página. Las siguientes reglas hacen que los párrafos se muestren con un tipo de letra `Georgia` o similar y los títulos de sección con un tipo de letra `Verdana` o similar:

```
p {
  font-family: Georgia, "Times New Roman", Times, serif;
}
h1, h2, h3, h4, h5, h6 {
  font-family: Verdana, Arial, Helvetica, sans-serif;
}
```

Para hacer uso de los estilos proporcionados por el componente `fonts.css` es necesario que las páginas enlacen esa hoja de estilos:

```
<link rel="stylesheet" type="text/css" href="css/fonts-min.css">
```

Recuerda que también puedes enlazar esta misma hoja de estilos directamente desde los servidores de Yahoo!:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/fonts/
fonts-min.css">
```

Por último, como es muy común utilizar todos los componentes CSS de forma simultánea, YUI incluye una hoja de estilos que combina todas las hojas de estilos individuales. Además de reducir el tamaño total de los archivos, disponer de una sola hoja de estilos reduce el número de peticiones realizadas al servidor. Para utilizar la hoja de estilos completa sólo es necesario utilizar la siguiente etiqueta `<link>`:

```
<link rel="stylesheet" type="text/css" href="css/reset-fonts-grids.css">
```

La hoja de estilos completa también se puede enlazar directamente desde los servidores de Yahoo!:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/
reset-fonts-grids/reset-fonts-grids.css">
```

5.5. Layouts con el framework YUI

El último componente CSS de YUI es `grids.css`, una hoja de estilos sencilla que permite crear más de 1.000 *layouts* diferentes. Diseñar *layouts* con YUI es muy sencillo y tiene la ventaja de que todos los diseños se ven exactamente igual en cualquier versión de cualquier navegador.

5.5.1. Primeros pasos

YUI recomienda utilizar un DOCTYPE que asegure que el navegador muestra sus contenidos en el modo *standards* en vez del modo *quirks*. De hecho, Yahoo! utiliza el siguiente DOCTYPE en todos los sitios diseñados con YUI:

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

No obstante, ten en cuenta que el DOCTYPE anterior corresponde a una página HTML y por tanto, no es válido si utilizas XHTML en tus páginas. Si se quiere emplear YUI en páginas XHTML puedes utilizar uno de los dos siguientes DOCTYPE:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Antes de crear *layouts* con el componente `grids.css`, es necesario enlazar su hoja de estilos:

```
<link rel="stylesheet" type="text/css" href="css/grids-min.css">
```

La hoja de estilos también se puede enlazar directamente desde los servidores de Yahoo!:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/grids/
grids-min.css">
```

El componente `grids.css` depende de los valores iniciales establecidos por el componente `fonts.css`, por lo que también es necesario enlazar este último. Como las páginas diseñadas con YUI también utilizan el componente `reset.css`, es habitual utilizar la hoja de estilos global que incluye estos tres componentes individuales:

```
<link rel="stylesheet" type="text/css" href="css/reset-fonts-grids.css">
```

Para obtener el máximo rendimiento también es posible enlazar esta hoja de estilos completa directamente desde los servidores de Yahoo!:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/
reset-fonts-grids/reset-fonts-grids.css">
```

5.5.2. Página, plantilla y rejilla

YUI recomienda utilizar la siguiente estructura general en las páginas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc">
  <div id="hd">
    <!-- contenidos de la cabecera -->
```

```
  </div>
  <div id="bd">
    <!-- contenidos principales de la página -->
  </div>
  <div id="ft">
    <!-- contenidos del pie de página -->
  </div>
</div>
</body>
</html>
```

La página se divide en tres partes:

- Cabecera (`#hd`): zona en la que se incluyen normalmente los elementos invariantes de la página tales como el logotipo, el buscador y el menú principal de navegación.
- Cuerpo (`#bd`): zona en la que se incluyen todos los contenidos específicos de la página. Se trata de la zona más importante y la que suele tener una estructura más compleja.
- Pie (`#ft`): zona en la que se incluyen otros elementos invariantes de la página que son de menor importancia, tales como el aviso de copyright, enlaces a las secciones de contacto, privacidad, términos y condiciones y otros elementos que dependen del tipo de página.

La división anterior de la página no es obligatoria y YUI permite crear *layouts* sin utilizar esa estructura. No obstante, se trata de una estructura muy recomendable porque es válida para la inmensa mayoría de sitios web y ayuda a estructurar los contenidos de la página de forma lógica.

Los nombres utilizados como identificador (`#hd` de *header* o cabecera, `#bd` de *body* o cuerpo y `#ft` de *footer* o pie) tampoco son obligatorios, aunque su uso está muy extendido entre los diseñadores profesionales y al ser tan cortos permiten hacer reglas CSS más concisas.

Antes de diseñar *layouts* con YUI es necesario comprender las partes en las que YUI divide la estructura de una página:

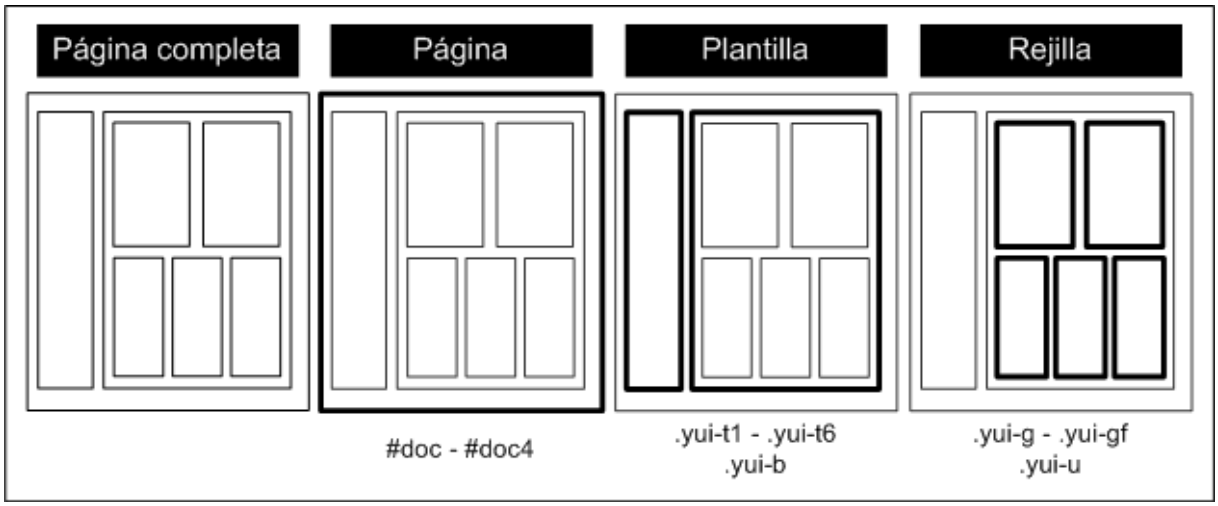


Figura 5.1. Página, plantilla y rejilla en los layouts de YUI

YUI divide la estructura de una página en tres partes:

- Página: establece la anchura total de la página y su comportamiento (anchura fija o variable). Utiliza los selectores `#doc`, `#doc2`, `#doc3` y `#doc4`.
- Plantilla: establece la división en columnas del cuerpo de la página. Utiliza los selectores `.yui-b`, `.yui-t1`, `.yui-t2`, `.yui-t3`, `.yui-t4`, `.yui-t5` y `.yui-t6`.
- Rejilla: establece las divisiones internas de cada columna. Utiliza los selectores `.yui-u`, `.yui-g`, `.yui-gb`, `.yui-gc`, `.yui-gd`, `.yui-ge`, `.yui-gf`.

5.5.3. Página

La primera parte de la estructura de la página es la propia página, concretamente su anchura y su comportamiento. YUI incluye por defecto cuatro tipos de páginas que corresponden a los diseños más utilizados hoy en día:

Selector	Características de la página	Recomendada para
#doc	Centrada, anchura 750px	Resolución de 800x600 o superior
#doc2	Centrada, anchura 950px	Resolución de 1024x768 o superior
#doc3	Anchura 100%	Cualquier resolución
#doc4	Centrada, anchura 974px	Resolución de 1024x768 o superior

Nota
El diseño `#doc3` en realidad no ocupa el 100% de la página, ya que incluye `10px` de margen izquierdo y otros `10px` de margen derecho. De esta forma se evita que los contenidos de la página puedan llegar a invadir el espacio ocupado por el propio navegador. No obstante, si quieres eliminar estos márgenes laterales tan sólo debes añadir la siguiente regla CSS en la hoja de estilos propia de la página: `#doc3 { margin: auto; }`

Para utilizar una u otra página, sólo es necesario cambiar el atributo `id` del elemento `<div>` que encierra todos los contenidos. El ejemplo mostrado anteriormente utiliza el selector `#doc`, correspondiente a una página de anchura fija de `750px`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc">  <!-- 750px, centrada -->
  ...
</div>
</body>
</html>
```

Si se quiere modificar la página anterior para que su anchura sea del `100%`, sólo es necesario modificar ligeramente el selector del elemento `<div>` que encierra a todos los contenidos:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc3">  <!-- 100% -->
  ...
</div>
</body>
</html>
```

Aunque los cuatro tipos de página incluídos en YUI corresponden a los diseños más utilizados, también es posible crear una anchura de página completamente personalizada. Como las anchuras de página en YUI se establecen mediante la unidad de medida `em`, el proceso requiere realizar alguna operación matemática.

Para obtener la anchura en `em`, divide la anchura deseada en píxeles por `13`. Para el navegador Internet Explorer es preciso dividir la anchura en píxeles por `13.3333`. Si por ejemplo se quiere establecer una anchura de página de `1300px`, el cálculo que se debe realizar es el siguiente:

- Navegador Internet Explorer: `1300` / `13.3333` = `97.50`. Por tanto, la anchura de página es `97.50em`.
- Resto de navegadores: `1300` / `13` = `100`. Por tanto, la anchura de página es `100em`.

Una vez calculada la anchura en `em`, se utiliza la siguiente regla CSS para crear el tamaño de página propio:

```
#doc-propio {
  text-align:left;   /* obligatorio */
  margin:auto;       /* para centrar la página */
  width: 100em;      /* resto de navegadores */
  *width: 97.50em;   /* navegador Internet Explorer */
  min-width: 1300px; /* opcional, pero recomendada */
}
```

La propiedad `text-align: left` es obligatoria para mostrar el texto alineado a la izquierda, ya que en la hoja de estilos `grids.css` el elemento `<body>` establece la propiedad `text-align: center;`. La propiedad `margin: auto` sólo es necesaria si se quiere centrar la página respecto de la ventana del navegador.

La anchura de todos los navegadores menos Internet Explorer se establece directamente mediante la propiedad `width`. En el caso de Internet Explorer, su anchura se establece mediante un *hack* muy conocido que hace que sólo Internet Explorer interprete la propiedad `*width`. Esta propiedad `*width` siempre se debe incluir después de la propiedad `width`.

Por último, la propiedad `min-width` es opcional, pero se recomienda establecerla con el mismo valor de la anchura en píxeles. Su función es evitar que la estructura de la página se rompa cuando la ventana del navegador reduce su tamaño.

5.5.4. Plantilla

El siguiente nivel en el que divide YUI la estructura de una página es la plantilla, que está formada por las columnas de contenidos de la página. Normalmente, las páginas disponen de una zona central de contenidos y otra zona lateral con otros contenidos secundarios.

YUI permite crear estructuras de páginas con dos columnas de diferentes anchuras y en diferentes posiciones (izquierda o derecha). En primer lugar se definen las columnas de la página mediante lo que YUI llama *bloques*. Cada bloque se crea mediante un elemento `<div>` con la clase `yui-b`. En el siguiente ejemplo se supone que la página está formada por dos columnas, por lo que se crean dos bloques dentro del cuerpo de la página:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc">
  <div id="hd"><!-- cabecera --></div>
  <div id="bd">
    <!-- cuerpo -->
    <div class="yui-b">...</div>
    <div class="yui-b">...</div>
  </div>
  <div id="ft"><!-- pie --></div>
</div>
</body>
</html>
```

Indicar que la página tiene dos bloques no es suficiente, ya que también es necesario indicar cuál de los dos bloques es el principal. Para ello, se encierra el bloque principal con otro elemento `<div>` cuyo `id` es `yui-main`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc">
  <div id="hd"><!-- cabecera --></div>
  <div id="bd">
    <!-- cuerpo -->
    <div id="yui-main">
      <div class="yui-b">
        <!-- bloque principal -->
      </div>
    </div>
    <div class="yui-b">
```

```
      <!-- bloque secundario -->
    </div>
  </div>
  <div id="ft"><!-- pie --></div>
</div>
</body>
</html>
```

Nota
La forma en la que está diseñado YUI permite que el orden en el que se definen sus bloques en el código HTML sea independiente a su visualización. Algunos diseñadores prefieren colocar el lateral secundario lo antes posible dentro del código HTML, ya que suele incluir el menú de navegación y por razones de accesibilidad y de SEO es recomendable que aparezca lo antes posible. Con YUI es posible incluir el bloque secundario en primer lugar en el código HTML con la seguridad de que se visualizará correctamente en la posición establecida.

Después de crear los bloques de la página y después de indicar cuál es el bloque principal, el siguiente paso consiste en utilizar la plantilla adecuada para establecer las anchuras de cada bloque y su posición. YUI incluye por defecto seis tipos de plantillas, que corresponden a los diseños más comunes y están preparados para mostrar los formatos publicitarios más utilizados:

Selector	Características de los bloques
`.yui-t1`	Lateral de `160px` a la izquierda
`.yui-t2`	Lateral de `180px` a la izquierda
`.yui-t3`	Lateral de `300px` a la izquierda
`.yui-t4`	Lateral de `180px` a la derecha
`.yui-t5`	Lateral de `240px` a la derecha
`.yui-t6`	Lateral de `300px` a la derecha

Nota
Existe una séptima plantilla especial que utiliza el selector `.yui-t7` y que se puede emplear en las páginas que no están divididas en columnas o bloques.

Si la página ya ha definido sus bloques, utilizar una plantilla u otra es tan sencillo como modificar el valor de la clase del elemento `<div>` que encierra a todos los contenidos:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc" class="yui-t"> <!-- plantilla con el lateral izquierdo de 160px -->
  <div id="hd"><!-- cabecera --></div>
  <div id="bd">
```

```
    <!-- cuerpo -->
    <div id="yui-main">
      <div class="yui-b">
        <!-- bloque principal -->
      </div>
    </div>
    <div class="yui-b">
      <!-- bloque secundario -->
    </div>
  </div>
  <div id="ft"><!-- pie --></div>
</div>
</body>
</html>
```

A continuación se modifica la página anterior para utilizar la plantilla que muestra el lateral a la derecha y con una anchura de `300px`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc" class="yui-t6"> <!-- plantilla con el lateral derecho de 300px -->
  <div id="hd"><!-- cabecera --></div>
  <div id="bd">
    <!-- cuerpo -->
    <div id="yui-main">
      <div class="yui-b">
        <!-- bloque principal -->
      </div>
    </div>
    <div class="yui-b">
      <!-- bloque secundario -->
    </div>
  </div>
  <div id="ft"><!-- pie --></div>
</div>
</body>
</html>
```

Las plantillas ofrecidas por YUI son las más utilizadas en el diseño web, pero en muchas ocasiones las páginas presentan otras estructuras más complejas. Cuando la estructura está dividida en más columnas, no se utilizan las plantillas, sino que se define directamente una rejilla.

5.5.5. Rejilla

El tercer elemento que forma la estructura de las páginas en YUI es la rejilla, que permite realizar diseños mucho más complejos que los que se realizan únicamente con las plantillas. La estructura en rejilla se puede utilizar junto con las plantillas o de forma independiente.

Las rejillas permiten crear divisiones muy complejas y se definen mediante dos elementos:

1. El contenedor de la rejilla establece el tipo de división.
2. Las unidades son los elementos individuales en los que está dividida la rejilla.

La rejilla más utilizada y más sencilla es `.yui.g`, que divide un elemento en dos partes iguales. El código HTML necesario es el siguiente:

```
<div class="yui-g">
  <div class="yui-u"></div>
  <div class="yui-u"></div>
</div>
```

El elemento `<div class="yui-g">` crea una nueva división o rejilla. Como se trata de una división en dos partes iguales, en su interior se deben definir dos unidades mediante `<div class="yui-u"></div>`. Además, YUI obliga a indicar explícitamente cuál de las dos unidades es la primera mediante la clase `first`. De esta forma, el código correcto del ejemplo anterior se muestra a continuación:

```
<div class="yui-g">
  <div class="yui-u first"></div>
  <div class="yui-u"></div>
</div>
```

La siguiente tabla muestra las seis rejillas incluidas por defecto en YUI y sus características:

Selector	Tipo de división
`.yui.g`	2 unidades a partes iguales (1/2, 1/2)
`.yui.gb`	3 unidades a partes iguales (1/3, 1/3, 1/3)
`.yui.gc`	2 unidades (2/3, 1/3)
`.yui.gd`	2 unidades (1/3, 2/3)
`.yui.ge`	2 unidades (3/4, 1/4)
`.yui.gf`	2 unidades (1/4, 3/4)

Utilizando varias de las rejillas predefinidas es posible crear estructuras de página muy complejas. El siguiente ejemplo muestra el código de una página cuyo cuerpo está dividido en dos columnas en su parte superior y en tres columnas en su parte inferior:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<html>
<head>
  <title>Página diseñada con YUI</title>
  <link rel="stylesheet" type="text/css" href="reset-fonts-grids.css">
</head>
<body>
<div id="doc" class="yui-t7">
  <div id="hd"><!-- cabecera --></div>
  <div id="bd">
    <!-- cuerpo -->
    <div id="yui-main">
```

```
      <div class="yui-b"><!-- bloque principal -->
        <!-- rejilla 1/2, 1/2 -->
        <div class="yui-g">
          <div class="yui-u first"></div>
          <div class="yui-u"></div>
        </div>

        <!-- rejilla 1/3, 1/3, 1/3 -->
        <div class="yui-gb">
          <div class="yui-u first"></div>
          <div class="yui-u"></div>
          <div class="yui-u"></div>
        </div>
      </div>
    </div>
    <div class="yui-b"><!-- bloque secundario -->
    </div>
  </div>
  <div id="ft"><!-- pie --></div>
</div>
</body>
</html>
```

Combinando las rejillas básicas se pueden obtener otro tipo de rejillas muy utilizadas. La rejilla 1/4, 1/4, 1/4, 1/4 se puede obtener combinando dos rejillas 1/2, 1/2:

```
...
  <div id="bd">
    <!-- cuerpo -->
    <div id="yui-main">
      <div class="yui-b"><!-- bloque principal -->
        <!-- rejilla 1/2, 1/2 -->
        <div class="yui-g">
          <div class="yui-u first">

            <!-- rejilla 1/2, 1/2 -->
            <div class="yui-g">
              <div class="yui-u first"></div>
              <div class="yui-u"></div>
            </div>

          </div>

          <div class="yui-u">

            <!-- rejilla 1/2, 1/2 -->
            <div class="yui-g">
              <div class="yui-u first"></div>
              <div class="yui-u"></div>
            </div>

          </div>
        </div>

      </div>
```

```
    </div>
    <div class="yui-b"><!-- bloque secundario -->
    </div>
  </div>
  ...
</div>
...
```

Crear estructuras complejas con YUI es realmente sencillo aunque requiere de un período de aprendizaje. Por ese motivo, para los diseñadores que están empezando puede ser de utilidad la herramienta *YUI: CSS Grid Builder* (http://developer.yahoo.com/yui/grids/builder/) que permite configurar la estructura de la página de forma gráfica.

5.6. Otros frameworks

YUI (http://developer.yahoo.com/yui/) no es el único *framework* CSS disponible, pero si es el más completo. El motivo es que YUI ni siquiera es un *framework* CSS, sino que es una completa librería de componentes listos para crear aplicaciones web dinámicas. YUI incluye decenas de utilidades de todo tipo para facilitar la programación con Javascript, incluye componentes prediseñados listos para utilizar, herramientas para comprimir el código JavaScript y el código CSS, componentes para facilitar el diseño web con CSS y otros muchos componentes de todo tipo.

Además de YUI, existen muchos otros *frameworks* que facilitan el diseño de páginas web con CSS. Aunque no son tan completos como YUI, cuentan con la ventaja de que sólo son *frameworks* CSS, por lo que están mucho más especializados.

A continuación se indican los tres frameworks más populares al margen de YUI:

- 960 Grid System (http://960.gs/) : su nombre proviene de `960px`, que es la anchura para la que optimiza el diseño de los layouts. Este framework divide los `960px` de anchura de la página en `12` o `16` columnas de `60px` y `40px` de anchura respectivamente y les añade un margen izquierdo y derecho de `20px` a cada columna. El archivo comprimido del framework también incluye una plantilla de papel en formato PDF y plantillas para los programas Fireworks, OmniGraffle, Photoshop y Visio.
- YAML (http://www.yaml.de/) : su nombre proviene de *"Yet Another Multicolumn Layout"* y es uno de los frameworks más completos. Soporta todas las versiones de todos los navegadores, incluyendo navegadores tan obsoletos como Internet Explorer 5. La documentación de YAML está formada por un libro en formato PDF de más de 110 páginas. Al igual que YUI, el framework YAML también dispone de una herramienta llamada YAML Builder (http://builder.yaml.de/) que permite configurar gráficamente la estructura de la página.
- Blueprint (http://blueprintcss.org/) : framework muy similar en concepto a *960 Grid System*, ya que por defecto la página tiene una anchura de `950px` dividida en `24` columnas de `30px` de ancho y `10px` de márgenes laterales.

Capítulo 6. Técnicas avanzadas

6.1. Imágenes embebidas

Según los estudios realizados por Yahoo! el tiempo de carga de una página media depende en un 80% de la parte del cliente y en un 20% de la parte del servidor. Los navegadores de los usuarios dedican la mayor parte del tiempo a descargar imágenes, archivos JavaScript, hojas de estilos CSS y otros recursos externos.

Por este motivo, las mejoras en la parte del cliente generan muchos más beneficios que las mejoras en la parte del servidor. De todos los elementos de la parte del cliente, las imágenes son normalmente las que más penalizan el rendimiento. Además del peso de cada imagen, el rendimiento se resiente porque cada imagen requiere realizar una petición al servidor. Como los navegadores tienen un límite de conexiones simultáneas con el servidor (entre 2 y 8), la descarga de una página con muchas imágenes se bloquea continuamente.

Como ya se explicó en las secciones anteriores, la solución para mejorar el rendimiento de las imágenes consiste en combinarlas en una imagen grande llamada *sprite CSS* y utilizar la propiedad `background-image` en cada elemento HTML.

Además de los *sprites CSS* existe otra técnica que permite *embeber* o incluir las imágenes en la propia página HTML u hoja de estilos CSS. La técnica se suele denominar *"imágenes embebidas"* o *"imágenes en línea"*.

Normalmente, en las hojas de estilos y páginas HTML sólo se indica la URL de la imagen que debe descargar el navegador. En la técnica de las imágenes embebedidas no se indica la URL, sino que se incluyen directamente los bytes de la imagen, para que el navegador pueda mostrarla de forma inmediata.

Los datos de la imagen se incluyen mediante un *esquema* especial llamado `data:`, de la misma forma que las URL se indican mediante el esquema `http:`, las direcciones de correo electrónico mediante el esquema `mailto:`, etc. El esquema `data:` se define en el estándar RFC 2397 (http://www.ietf.org/rfc/rfc2397.txt) y su sintaxis es la siguiente:

```
data:[<mediatype>][;base64],<data>
```

El atributo `<mediatype>` corresponde al tipo de contenido de los bytes que se incluyen. Los tipos de contenido están estandarizados y los que utilizan habitualmente las imágenes son: `image/gif`, `image/jpeg` y `image/png`. Si no se indica el tipo de forma explícita, el navegador supone que es `text/plain`, es decir, texto normal y corriente.

El valor `base64` es opcional e indica que los datos de la imagen se han codificado según el formato `base64`. Si no se indica este valor, el navegador supone que los bytes de la imagen no están codificados.

A continuación se muestra el ejemplo de una imagen HTML (`<img>`) que no se indica mediante una URL sino que se incluyen sus bytes directamente en la página:

```
<!-- Imagen externa que el navegador debe descargar desde el servidor -->
<img alt="Icono de un libro" width="16" height="16" src="/imagenes/icono_libro.png" />

<!-- Imagen embebida que el navegador puede mostrar directamente porque ya dispone de
sus bytes -->
<img alt="Icono de un libro" width="16" height="16" src="
BORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8/9hAAAABGdBTUEAAK/INwWK6QAAABl0RVh0U29mdHd
hcmUAQWRvYmUgSW1hZ2VSZWFkeXHJZTwAAAHjSURBVDjLdZO/alVBEMZ/5+TemdHd0U29mdHdhcmUAQW
RvYmUgSW1hZ2VSZWFkeXHJZTwAAAHjSURBVDjLdZO/alVBEMZ/5+TemxAbFUUskqAoSOJNp4KC
4AsoPoGFIHY+gA+jiJXaKIiChbETtBYLUbSMRf6Aydndmfks9kRjvHdhGVh2fvN9uzONJK7fe7Ai6algA
3FZCAmQqEF/dnihpK1v7x7dPw0woF64Izg3Xl5s1n9uIe0lQYUFCtjc+sVuEqHBKfpVAXB1vLzQXFtdYP
HkGFUCoahVo1Y/fnie+bkBV27c5R8A0pHxyhKvPn5hY2MHRQAQeyokFGJze4cuZfav3gLNYDTg7Pklzpw
4ijtIQYRwFx6BhdjtCk+erU0CCPfg+/o2o3ZI13WUlLGo58YMg+GIY4dmCWkCAAgPzAspJW5ePFPlV3VI
4uHbz5S5IQfy/yooHngxzFser30iFcNcuAVGw3A0Ilt91IkAsyCXQg5QO0szHEIrogkiguwN2acCoJhjn
ZGKYx4Ujz5WOA2YD1BMU+BBSYVUvNpxkXuIuWgbsOxTHrG3UHIFWIhsgXtQQpTizNBS5jXZQkhkcywZqQ
QlAjdRwiml7wU5xWLaL1AvZa8WIjALzIRZ7YVWDW5CiIj48Z8F2pYLl1ZR0+AuzEX0UX035mxIkLq0dhD
w5vXL97fr5O3rfwQHJhPx4uuH57f2AL8BfPrVlrs6xwsAAAAASUVORK5CYII=" />
```

El atributo `src` de la imagen del ejemplo anterior utiliza el esquema `data:` para incluir en la página los bytes de la imagen. El valor `image/png` indica que los datos corresponden a una imagen en formato `PNG`. El valor `base64` indica que los datos incluidos están codificados según el formato base64.

Aunque el ejemplo anterior funciona correctamente, como todos los datos se incluyen en el propio código HTML, el navegador no puede hacer uso de la caché para reutilizarlos posteriormente. El resultado es que el número de peticiones HTTP se reduce drásticamente, pero aumenta significativamente el tamaño de todas las páginas HTML.

El siguiente paso consiste en utilizar las imágenes embebidas en las hojas de estilos CSS, de forma que se mantengan las ventajas del ejemplo anterior y se solucionen todas sus desventajas. Para embeber las imágenes en CSS se sigue la misma estrategia que en HTML, ya que sólo es necesario sustituir la URL por el esquema `data:`, tal y como muestra el siguiente ejemplo:

```
/* Imagen externa que el navegador debe descargar desde el servidor */
ul#menu li { background: #FFF no-repeat center center url("/imagenes/icono_libro.png");
}

/* Imagen embebida que el navegador puede mostrar directamente porque ya dispone de sus
bytes */
ul#menu li {
  background: #FFF no-repeat center center url("
  AAAANSUhEUgAAABAAAAAQCAYAAAAf8/9hAAAABGdBTUEAAK/INwWK6QAAABl0RVh0U29mdHdhcmUAQW
  RvYmUgSW1hZ2VSZWFkeXHJZTwAAAHjSURBVDjLdZO/alVBEMZ/5+TemxAbFUUskqAoSOJNp4KC4AsoP
  oGFIHY+gA+jiJXaKIiChbETtBYLUbSMRf6Aydndmfks9kRjvHdhGVh2fvN9uzONJK7fe7Ai6algA3FZ
  CAmQqEF/dnihpK1v7x7dPw0woF64Izg3Xl5s1n9uIe0lQYUFCtjc+sVuEqHBKfpVAXB1vLzQXFtdYPH
  kGFUCoahVo1Y/fnie+bkBV27c5R8A0pHxyhKvPn5hY2MHRQAQeyokFGJze4cuZfav3gLNYDTg7Pklzp
  w4ijtIQYRwFx6BhdjtCk+erU0CCPfg+/o2o3ZI13WUlLGo58YMg+GIY4dmCWkCAAgPzAspJW5ePFPlV
  3VI4uHbz5S5IQfy/yooHngxzFser30iFcNcuAVGw3A0Ilt91IkAsyCXQg5QO0szHEIrogkiguwN2acC
  oJhjnZGKYx4Ujz5WOA2YD1BMU+BBSYVUvNpxkXuIuWgbsOxTHrG3UHIFWIhsgXtQQpTizNBS5jXZQkh
  kcywZqQQlAjdRwiml7wU5xWLaL1AvZa8WIjALzIRZ7YVWDW5CiIj48Z8F2pYLl1ZR0+AuzEX0UX035m
  xIkLq0dhDw5vXL97fr5O3rfwQHJhPx4uuH57f2AL8BfPrVlrs6xwsAAAAASUVORK5CYII=");
}
```

La ventaja de utilizar las imágenes embebidas en CSS es que sólo aumenta el tamaño de las hojas de estilos y no el de todas las páginas HTML del sitio. Además, los navegadores guardan en su

cache las hojas de estilos completas, por lo que el aumento en su tamaño no penaliza en exceso el rendimiento global de la descarga de páginas.

El proceso de codificación de los bytes de las imágenes según el formato base64 es una tarea más propia de programadores web que de diseñadores web. Si conoces por ejemplo el lenguaje de programación PHP puedes utilizar la siguiente instrucción:

```
$bytesCodificados = base64_encode(file_get_contents("/ruta/hasta/la/imagen.png"));
```

Si no dispones de conocimientos de programación, puedes utilizar alguna de las herramientas online que codifican directamente los contenidos del archivo indicado:

- Base64 encoder/decoder (http://www.motobit.com/util/base64-decoder-encoder.asp)
- Binary File to Base64 Encoder / Translator (http://www.greywyvern.com/code/php/binary2base64)

Las principales ventajas de la técnica de las imágenes embebidas son las siguientes:

- Reduce drásticamente el número de peticiones HTTP, lo que mejora notablemente el rendimiento.
- Permite guardar una página HTML completa en un único archivo HTML (embebiendo todas sus imágenes en su hoja de estilos o en el propio código HTML).
- Mejora el rendimiento de las peticiones HTTPS en las que las imágenes no se guardan en la cache.

Por contra, sus desventajas son considerables:

- El esquema `data:` sólo funciona en los navegadores modernos que se preocupan de los estándares (Firefox, Safari y Opera). Internet Explorer 6 y 7 no son capaces de procesar el esquema `data:`. Internet Explorer 8 asegura que permitirá utilizar `data:`, pero solamente para embeber imágenes en CSS.
- El proceso completo es lento y poco flexible, ya que es necesario codificar las imágenes en base64 y recodificarlas cada vez que se quieren modificar.
- Las imágenes embebidas aumentan considerablemente el tamaño de la hoja de estilos CSS. Además, la codificación base64 también aumenta mucho el tamaño de los datos de la imagen respecto a sus bytes originales.
- Los navegadores limitan el tamaño máximo de los datos que se pueden embeber mediante `data:`. Algunos navegadores como Opera permiten unos 4.000 bytes de datos, mientras que el navegador Firefox permite hasta 100.000 bytes por cada `data:`.

6.2. Mapas de imagen

Los mapas de imagen se llevan utilizando desde los orígenes de HTML. Combinando las etiquetas `<map>` y `<area>` junto con el atributo `usemap` de la etiqueta `<img>` es posible definir diferentes zonas pinchables dentro de una misma imagen.

Hoy en día los mapas de imagen HTML han sido sustituidos por otras soluciones como Flash, que son más fáciles de utilizar y disponen de más posibilidades. No obstante, recientemente ha surgido un nuevo tipo de mapa de imagen creado sólo con CSS.

Estos mapas de imagen CSS no suelen utilizarse para definir zonas pinchables dentro de una imagen, sino que se emplean para mostrar información adicional y comentarios sobre las diferentes zonas de una imagen. El sitio de fotografía Flickr (http://www.flickr.com/) utiliza los mapas de imagen para mostrar notas y comentarios de los usuarios. Otros sitios web como Facebook (http://www.facebook.com/) utilizan los mapas de imagen para que los usuarios etiqueten las fotografías indicando el nombre de las personas que aparecen en cada una.

A continuación se explican los pasos necesarios para crear un mapa de imagen exclusivamente con CSS similar a los de Flickr y Facebook.

En primer lugar, selecciona la imagen en la que se van a mostrar las notas. En este ejemplo se utiliza una imagen de la fotógrafa visualpanic (http://www.flickr.com/photos/visualpanic/) que se puede utilizar libremente y que está disponible en Flickr (http://www.flickr.com/photos/visualpanic/233508614/) :

Figura 6.1. Imagen original en la que se va a mostrar el mapa de imagen

El funcionamiento del mapa de imagen terminado es el que muestra la siguiente secuencia de imágenes:

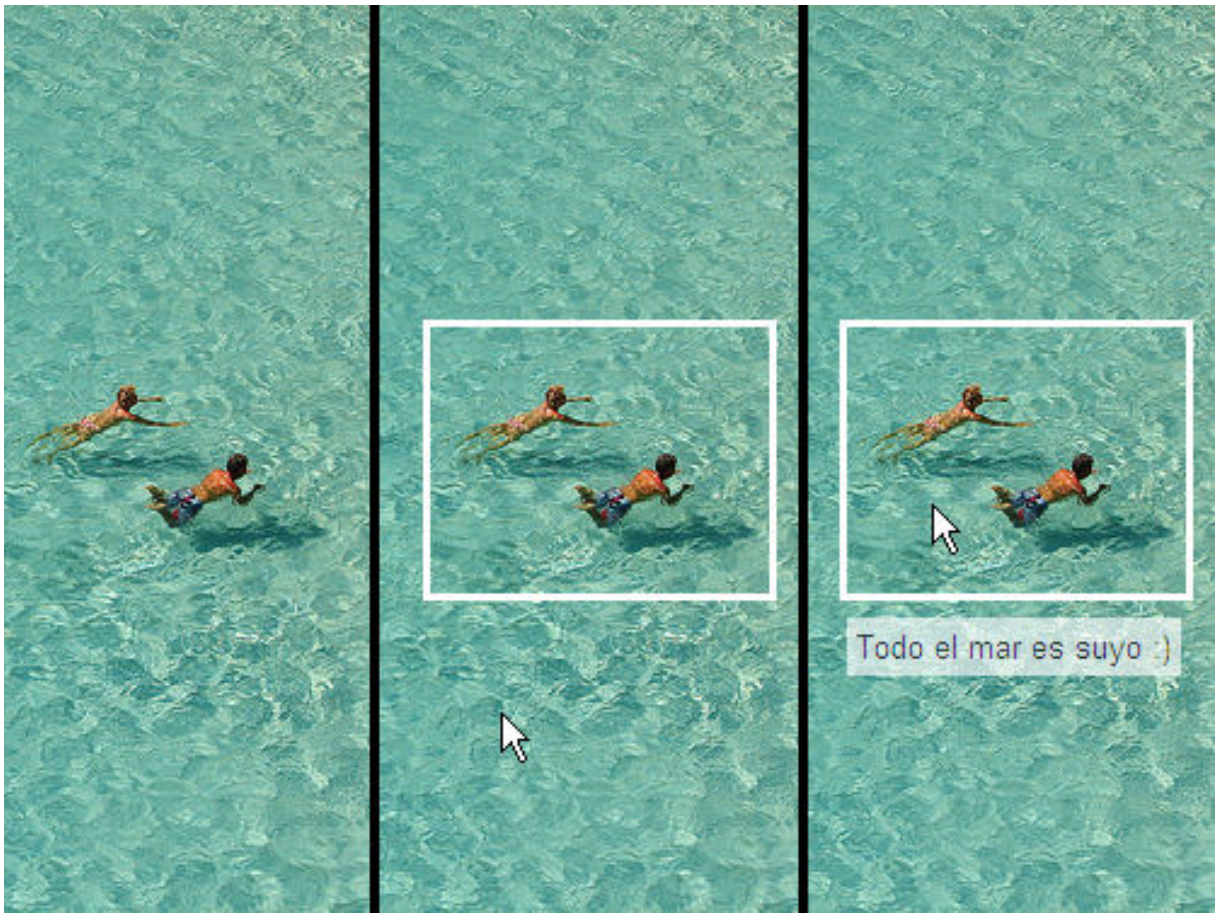


Figura 6.2. Funcionamiento del mapa de imagen creado con CSS

La imagen por defecto no muestra ninguna zona activa. Cuando el usuario pasa el ratón por encima de cualquier parte de la imagen, se muestra el recuadro de todas las zonas activas disponibles. Además, cuando el usuario pasa el ratón por encima de una zona activa, se muestra el comentario asociado.

El código HTML del mapa de imagen creado con CSS varía mucho en función de la solución utilizada. Aunque algunas soluciones crean varios `<div>` y tablas por cada nota/comentario, en este ejemplo se simplifica al máximo el código HTML y sólo se utiliza un elemento `<div>` y una lista `<ul>`:

```
<div class="mapa_imagen">
  <img src="imagenes/mar.jpg" />

  <ul class="notas">
    <li id="nota1"><p>Todo el mar es suyo :)</p></li>
    <li id="nota2"><p>Me encanta este color azul!</p></li>
    <li id="nota3"><p>Dan ganas de tirarse...</p></li>
  </ul>
</div>
```

El elemento `<div class="mapa_imagen">` encierra todos los elementos que forman el mapa de imagen (la imagen original y las notas/comentarios). Los comentarios se incluyen mediante una lista no ordenada `<ul>`, en la que cada elemento `<li>` es un comentario.

El elemento `<div>` y la lista `<ul>` deben utilizar el atributo `class` y no `id` porque en una misma página puede haber varios mapas de imagen. Por su parte, los elementos `<li>` de cada comentario deben utilizar atributos `id`, ya que cada comentario se muestra en una posición única y tiene unas dimensiones únicas de anchura y altura.

La clave de los mapas de imagen CSS consiste en posicionar cada `<li>` de forma absoluta respecto de la imagen y asignarles una anchura/altura adecuadas. Posteriormente, se emplea la pseudo-clase `:hover` para mostrar/ocultar elementos cuando el usuario pasa el ratón por encima.

1) Posicionar de forma absoluta cada nota:

```
div.mapa_imagen {
  position: relative
}
ul.notas li {
  position: absolute;
}
```

2) Aplicar los estilos básicos a las notas (borde blanco y sin adornos de lista):

```
ul.notas li {
  border: medium solid #FFF;
  list-style: none;
}
```

3) Ocultar por defecto las notas y mostrarlas cuando se pasa el ratón por encima de la imagen:

```
ul.notas li {
  display: none;
}
div.mapa_imagen:hover ul.notas li {
  display: block;
}
```

4) Aplicar los estilos básicos al texto de las notas y posicionarlo debajo de cada nota:

```
ul.notas li p {
  position: absolute;
  top: 100%;

  background: #FFF;
  opacity: 0.65;

  margin: 10px 0 0 0;
  padding: 0.3em;
}
```

5) Ocultar por defecto el texto de las notas y mostrarlo cuando se pasa el ratón por encima de la nota:

```
ul.notas li p {
  display: none;
}
ul.notas li:hover p {
```

```
  display: block;
}
```

6) Establecer la posición y dimensiones de cada nota:

```
ul.notas li#nota1 {
  width: 140px; height: 110px; top: 130px; left: 345px;
}
ul.notas li#nota2 {
  width: 30px; height: 200px; top: 10px; left: 10px;
}
ul.notas li#nota3 {
  width: 60px; height: 60px; top: 200px; left: 150px;
}
```

Aplicando las reglas CSS anteriores el mapa de imagen CSS ya funciona correctamente en los navegadores Firefox y Safari. Desafortunadamente, los navegadores Internet Explorer y Opera tienen errores que impiden que el ejemplo funcione correctamente. El problema reside en que los elementos `<li>` tienen un fondo transparente y tanto Internet Explorer como Opera tienen problemas con la pseudo-clase `:hover` sobre estos elementos.

La solución consiste en añadir un fondo (color o imagen) sobre los elementos `<li>`. Como lo único importante es añadir un fondo, independientemente de si el fondo es real o no, la siguiente regla CSS es suficiente:

```
ul.notas li {
  background: url("esta_imagen_no_existe");
}
```

El código CSS completo del mapa de imagen que funciona en todos los navegadores es el siguiente:

```
div.mapa_imagen {
  position: relative
}
ul.notas li {
  list-style: none;
  display: none;
  position: absolute;
  border: medium solid white;
  background: url("esta_imagen_no_existe");
}
div.mapa_imagen:hover ul.notas li {
  display: block;
}
ul.notas li p {
  margin: 10px 0 0 0;
  padding: .3em;
  display: none;
  background: #FFF;
  opacity: 0.65;
  position: absolute;
  top: 100%;
}
ul.notas li:hover p {
```

```
  display: block;
}

ul.notas li#nota1 {
  width: 140px; height: 110px; top: 130px; left: 345px;
}
ul.notas li#nota2 {
  width: 30px; height: 200px; top: 10px; left: 10px;
}
ul.notas li#nota3 {
  width: 60px; height: 60px; top: 200px; left: 150px;
}
```

El resultado final es el que muestra la siguiente imagen:

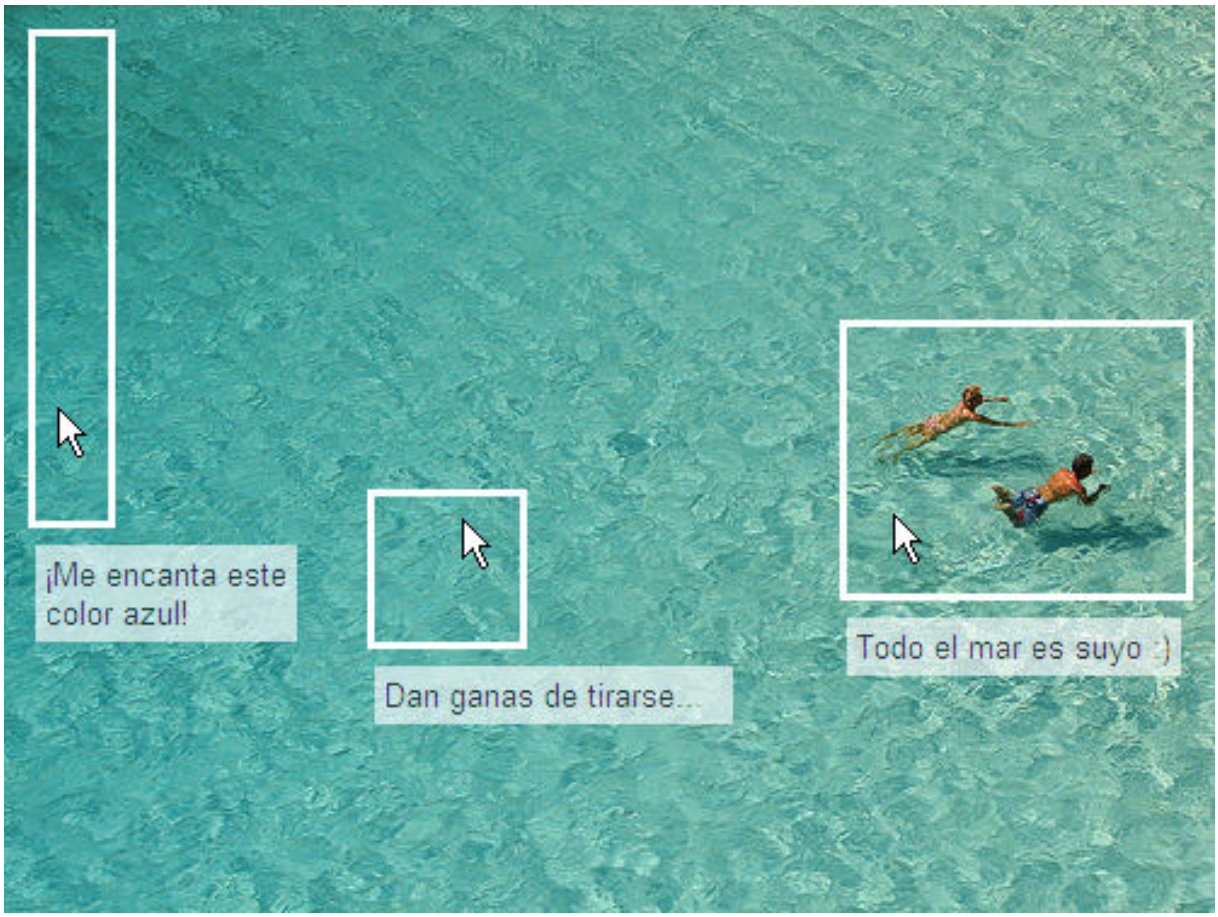


Figura 6.3. Mapa de imagen con todas sus zonas activadas

6.3. Variables en las hojas de estilos

La mayoría de diseñadores web profesionales creen que CSS carece de un mecanismo fundamental para facilitar su trabajo: las variables en las hojas de estilos. Cuando se define una hoja de estilos como la siguiente:

```
#cabecera {
  background-color: #369;
  color: #FFF;
}
```

```
#contenidos h1, #contenidos h2 {
  color: #369;
}
a {
  color: #557E29;
}
span {
  background-color: #557E29;
}
```

Los buenos diseñadores utilizan en sus trabajos un número muy reducido de colores que combinan entre sí para diseñar cada elemento de la página. De esta forma, en las hojas de estilos profesionales se repiten una y otra vez los mismos valores. No obstante, CSS no permite definir variables para almacenar los valores que se utilizan constantemente.

Si el diseñador quiere modificar el color verde `#557E29` de la hoja de estilos anterior, ¿cómo cambia todas las apariciones de ese color verde en la hoja de estilos? Modificar ese color de forma manual es un proceso tedioso y demasiado lento. Afortunadamente, los editores de texto disponen de herramientas para sustituir un valor por otro, aunque tampoco es una solución óptima.

Por otra parte, también es común diseñar sitios web idénticos o muy similares en los que sólo cambia su aspecto definido con CSS. En este caso, es habitual que un sitio web se diferencie de otro solamente en algunos pocos valores de su hoja de estilos CSS. La solución consiste nuevamente en modificar todos los valores de colores y tipos de letra para modificar el aspecto del sitio web.

Los problemas anteriores se podrían resolver fácilmente si CSS permitiera definir variables como en la siguiente hoja de estilos:

```
$colorAzul = #369;
$colorVerde = #557E29;

#cabecera {
  background-color: $colorAzul;
  color: #FFF;
}
#contenidos h1, #contenidos h2 {
  color: $colorAzul;
}
a {
  color: $colorVerde;
}
span {
  background-color: $colorVerde;
}
```

En la hoja de estilos anterior, el diseñador puede cambiar el tono de verde o de azul haciendo un solo cambio en la hoja de estilos y con la seguridad absoluta de que se modifican todos los elementos de la página y no se comete ningún error.

No obstante, si la hoja de estilos anterior se quiere reutilizar para otros diseños con colores completamente diferentes, el nombre de las variables anteriores no sería correcto. Por ello, lo más correcto sería crear variables cuyo nombre no incluya información sobre su aspecto:

```
$colorPrincipal  = #369;
$colorSecundario = #557E29;

#cabecera {
  background-color: $colorPrincipal;
  color: #FFF;
}
#contenidos h1, #contenidos h2 {
  color: $colorPrincipal;
}
a {
  color: $colorSecundario;
}
span {
  background-color: $colorSecundario;
}
```

Lamentablemente, CSS no permite utilizar variables en las hojas de estilos. Por este motivo se han desarrollado varias soluciones que hacen uso del servidor web y de los lenguajes de programación de servidor para incluir variables en las hojas de estilos CSS.

La solución más sencilla consiste en utilizar el mecanismo SSI (*Server Side Includes*) de los servidores web Apache y Microsoft IIS. A continuación se muestra el proceso completo para el servidor web Apache 2.2:

1) Modificar el archivo de configuración de Apache: dependiendo de tu instalación puede ser el archivo general de configuración `httpd.conf` o el archivo de configuración `.htaccess`

En cualquier caso, es necesario añadir las siguientes opciones de configuración:

```
Options +Includes
AddType text/css .scss
AddOutputFilter INCLUDES .scss
```

La configuración anterior le indica al servidor web que los archivos cuya extensión sea `.scss` son archivos CSS de servidor y que por tanto, se deben procesar antes de enviarlos al usuario.

Para que los cambios en la configuración sean efectivos, no olvides reiniciar el servidor web.

2) Crear la hoja de estilos y guardar el archivo con extensión `.scss`

3) Añadir las variables a la hoja de estilos utilizando la siguiente sintaxis:

Para definir la variable:

```
<!--#set var="nombre_de_la_variable" value="valor_de_la_variable" -->
```

Para utilizar la variable

```
<!--#echo var="nombre_de_la_variable" -->
```

Siguiendo con la hoja de estilos de los ejemplos anteriores, el resultado es el que se muestra a continuación:

```
<!--#set var="colorPrincipal" value="#369" -->
<!--#set var="colorSecundario" value="#557E29" -->

#cabecera {
  background-color: <!--#echo var="colorPrincipal" -->;
  color: #FFF;
}
#contenidos h1, #contenidos h2 {
  color: <!--#echo var="colorPrincipal" -->;
}
a {
  color: <!--#echo var="colorSecundario" -->;
}
span {
  background-color: <!--#echo var="colorSecundario" -->;
}
```

Si no se quiere utilizar otra extensión para los archivos CSS procesados por el servidor, es posible hacer uso de la directiva `XBitHack`, tal y como se explica en la documentación oficial de Apache 2.2 sobre SSI (http://httpd.apache.org/docs/2.2/howto/ssi.html) .

6.3.1. Lenguajes de programación de servidor

La solución alternativa al uso de las directivas SSI del servidor web consiste en emplear un lenguaje de programación de servidor. Aunque se trata de una técnica diferente, también se basa en procesar las hojas de estilos en el servidor antes de enviarlas al usuario.

Los detalles técnicos de la solución dependen del lenguaje de programación utilizado, pero el mecanismo que se utiliza es idéntico en todos los casos. A continuación se muestra un ejemplo que hace uso del lenguaje de programación PHP.

Para incluir variables en las hojas de estilos CSS, se crea un archivo con extensión `.php` y se utiliza la sintaxis que se muestra en el siguiente ejemplo:

```
<?php
header('content-type:text/css');

$nombre_variable1 = 'valor_variable1';
$nombre_variable2 = 'valor_variable2';

echo <<<FINCSS
selector {
  propiedad1: $nombre_variable1;
  propiedad2: $nombre_variable2;
}
FINCSS;
?>
```

Siguiendo con la hoja de estilos de los ejemplos anteriores, el archivo PHP completo es el siguiente:

```
<?php
header('content-type:text/css');

$colorPrincipal = '#369';
$colorSecundario = '#557E29';

echo <<<FINCSS
#cabecera {
  background-color: $colorPrincipal;
  color: #FFF;
}
#contenidos h1, #contenidos h2 {
  color: $colorPrincipal;
}
a {
  color: $colorSecundario;
}
span {
  background-color: $colorSecundario;
}
FINCSS;
?>
```

Por último, cuando enlaces la hoja de estilos generada con este método, no olvides utilizar la extensión `.php` en el nombre del archivo de la hoja de estilos:

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

6.3.2. CSS como lenguaje de programación

Las soluciones que permiten utilizar variables en las hojas de estilos han evolucionado de tal manera que alguna de ellas permite utilizar CSS como si fuera un lenguaje de programación. Una de estas soluciones es *CSS Cacheer* (http://www.shauninman.com/archive/2008/05/30/check_out_css_cacheer) , ideada por el diseñador Shaun Inman.

En primer lugar, *CSS Cacheer* permite utilizar variables en las hojas de estilos, aunque en esta solución se denominan "constantes":

```
@constants {
  nombre_de_la_constante: valor_de_la_constante;
}

selector {
  propiedad: const(nombre_de_la_constante);
}
```

CSS Cacheer también permite definir unos estilos base que se reutilizan en diferentes elementos:

```
@base(tipografia_basica) {
  font-family: Arial, sans-serif;
  line-height: 1.5;
}

p {
  based-on: base(tipografia_basica);
```

```
  font-size: 110%;
}

h1 {
  based-on: base(tipografia_basica);
  font-size: 200%;
}
```

CSS Cacheer es tan avanzado que permite incluso anidar selectores como se muestra en el siguiente ejemplo:

```
ul {
  propiedad1: valor1;

  li {
    propiedad2: valor2;
  }
}
```

Si se procesan las reglas CSS anteriores con *CSS Cacheer*, el resultado es el siguiente:

```
ul {
  propiedad1: valor1;
}

ul li {
  propiedad2: valor2;
}
```

6.4. Estilos alternativos

La mayoría de sitios web disponen de una o varias hojas de estilos CSS que se aplican automáticamente al cargar cada página en el navegador. En realidad, el estándar HTML/XHTML permite definir varias hojas de estilos CSS alternativas en una misma página.

De esta forma, el usuario puede seleccionar el estilo con el que se muestra la página entre una serie de estilos definidos por el diseñador web. En ocasiones los estilos alternativos se emplean por pura estética, por ejemplo para ofrecer *temas* y esquemas de colores diferentes en un sitio web. Sin embargo, el uso más adecuado de los estilos alternativos es la mejora de la accesibilidad ofreciendo hojas de estilos que faciliten el acceso a los contenidos para las personas discapacitadas.

Las hojas de estilos CSS que se enlazan en una página HTML mediante la etiqueta `<link>` pueden ser de tres tipos:

- Permanentes (*"persistent"*): las hojas de estilos que se aplican siempre.
- Preferentes (*"preferred"*): las hojas de estilos alternativas que se aplican por defecto.
- Alternativas (*"alternate"*): las hojas de estilos alternativas que el usuario puede seleccionar.

Las hojas de estilos permanentes se aplican siempre independientemente del resto de hojas de estilos definidas o de la hoja de estilo alternativa utilizada por el usuario. Para indicar que una

hoja de estilos CSS es permanente, se utiliza el atributo `rel="stylesheet"` y no se establece el atributo `title` en la etiqueta `<link>`:

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

En el ejemplo anterior, las dos hojas de estilos CSS se aplican siempre independientemente del medio y de otras posibles hojas de estilos definidas por la página.

Las hojas de estilos preferentes son las hojas de estilos alternativas que se aplican por defecto. Por lo tanto, si el usuario no selecciona explícitamente otra CSS alternativa, en la página también se aplican las hojas de estilos preferentes. Para indicar que una hoja de estilos CSS es preferente, se utiliza el atributo `rel="stylesheet"` y se establece el atributo `title` en la etiqueta `<link>`:

```
<link rel="stylesheet" title="Estilo alternativo" type="text/css" href="css/
estilos.css" />
```

La hoja de estilos enlazada en el ejemplo anterior se aplica en la página siempre que el usuario no seleccione otra hoja de estilos alternativa. El estándar HTML/XHTML permite crear grupos de hojas de estilos preferentes. Si dos o más hojas de estilos enlazadas tienen el mismo título, el navegador considera que forman parte de un grupo y las aplica de forma conjunta:

```
<link rel="stylesheet" title="Estilo alternativo" type="text/css" href="css/
estilos1.css" />
<link rel="stylesheet" title="Estilo alternativo" type="text/css" href="css/
estilos2.css" />
```

Las etiquetas `<link>` del ejemplo anterior enlazan dos hojas de estilos diferentes con el mismo valor en su atributo `title`, por lo que el navegador considera que forman un grupo. Como se trata de hojas de estilos preferentes (porque tienen un atributo `rel="stylesheet"` y un atributo `title` no vacío) el navegador aplica todas las hojas de estilos del grupo a no ser que el usuario seleccione explícitamente otra hoja de estilos alternativa.

El siguiente ejemplo incluye hojas de estilos permanentes y preferentes:

```
<link rel="stylesheet" type="text/css" href="css/estilos1.css" />
<link rel="stylesheet" type="text/css" href="css/estilos2.css" />
<link rel="stylesheet" title="Estilo alternativo" type="text/css" href="css/
estilos3.css" />
<link rel="stylesheet" title="Estilo alternativo" type="text/css" href="css/
estilos4.css" />
```

Las dos primeras hojas de estilos del ejemplo anterior son de tipo permanente, por lo que el navegador las aplica siempre. Las otras dos hojas de estilos son de tipo preferente, por lo que el navegador también las aplica a menos que el usuario seleccione otras hojas de estilo alternativas.

El último tipo de hojas de estilos son las alternativas, que sólo se aplican si el usuario las selecciona explícitamente. Para indicar que una hoja de estilos CSS es alternativa, se utiliza el atributo `rel="alternate stylesheet"` y se establece el atributo `title` en la etiqueta `<link>`:

```
<link rel="alternate stylesheet" title="Estilo alternativo" type="text/css" href="css/
estilos.css" />
```

La hoja de estilos del ejemplo anterior no se aplica en la página a menos que el usuario la seleccione entre todas las hojas de estilos alternativas. Esta característica no está disponible en todos los navegadores, por lo que los diseñadores no deben suponer que el usuario podrá utilizarla.

En los navegadores como Firefox y Opera, cuando una página dispone de varias hojas de estilos alternativas, el usuario puede acceder a todas ellas y seleccionar la que desee a través del menú `Ver > Estilo` o `Ver > Estilo de página`.

Si se considera una página HTML que enlaza las siete hojas de estilos siguientes:

```
<link rel="stylesheet" type="text/css" href="css/estilos1.css" />
<link rel="stylesheet" type="text/css" href="css/estilos2.css" />
<link rel="stylesheet" title="Estilo alternativo 1" type="text/css" href="css/
estilos3.css" />
<link rel="stylesheet" title="Estilo alternativo 1" type="text/css" href="css/
estilos4.css" />
<link rel="stylesheet" title="Estilo alternativo 2" type="text/css" href="css/
estilos5.css" />
<link rel="stylesheet" title="Estilo alternativo 2" type="text/css" href="css/
estilos6.css" />
<link rel="stylesheet" title="Estilo alternativo 3" type="text/css" href="css/
estilos7.css" />
```

Inicialmente, las hojas de estilos que se aplican en la página son:

- Las dos primeras hojas de estilos (`estilos1.css` y `estilos2.css`) por ser de tipo permanente y que por tanto, se aplican bajo cualquier circunstancia.
- Las dos siguientes hojas de estilos (`estilos3.css` y `estilos4.css`) por ser de tipo preferente y que por tanto, se aplican siempre que el usuario no seleccione explícitamente otro tipo de hoja de estilos alternativa.

Si el usuario utiliza un navegador avanzado (Firefox, Opera) y pulsa sobre el menú `Ver > Estilo`, se muestran las siguientes opciones:

- `Estilo alternativo 1`, seleccionado por defecto y que hace que se apliquen las hojas de estilos `estilos3.css` y `estilos4.css` y se dejen de aplicar el resto de hojas de estilos alternativas.
- `Estilo alternativo 2`, que hace que se apliquen las hojas de estilos `estilos5.css` y `estilos6.css` y se dejen de aplicar el resto de hojas de estilos alternativas, incluyendo las hojas de estilos preferentes `estilos3.css` y `estilos4.css`.
- `Estilo alternativo 3`, que hace que se aplique la hoja de estilos `estilos7.css` y se dejen de aplicar el resto de hojas de estilos alternativas, incluyendo las hojas de estilos preferentes `estilos3.css` y `estilos4.css`.

Por último, recuerda que independientemente del tipo de hoja de estilos enlazada, los navegadores también tienen en cuenta los medios (atributo `media`) al aplicar cada hoja de estilos en cada dispositivo con el que accede el usuario.

6.5. Comentarios condicionales, filtros y hacks

6.5.1. Comentarios condicionales

En ocasiones, cuando se diseña un sitio web, es preciso aplicar diferentes reglas y estilos en función del navegador. De esta forma, se pueden corregir los errores y limitaciones de un navegador sin afectar al resto de navegadores.

Microsoft introdujo en su navegador Internet Explorer 5 un mecanismo llamado *"comentarios condicionales"*, que todavía incluyen las versiones más recientes como Internet Explorer 8, y que permite aplicar diferentes estilos CSS según la versión del navegador.

La sintaxis de los comentarios condicionales se basa en la de los comentarios *normales* de HTML:

```
<!-- Comentario normal de HTML -->

<!--[if expresion]> Comentario condicional <![endif]-->
```

La sintaxis de los comentarios condicionales permite que su contenido se ignore en cualquier navegador que no sea de la familia Internet Explorer.

Las expresiones se crean combinando identificadores, operadores y valores. El único identificador definido es `IE`, que permite crear el comentario condicional más simple:

```
<!--[if IE]>
  Este navegador es cualquier versión de Internet Explorer
<![endif]-->
```

El operador más sencillo definido por los comentarios condicionales es el operador de negación (`!`), que se indica delante de una expresión para obtener el resultado contrario:

```
<!--[if !IE]>
  Este navegador es cualquiera salvo Internet Explorer
<![endif]-->
```

Si se quiere restringir el alcance del comentario condicional a una única versión de Internet Explorer, se puede indicar directamente el número de la versión:

```
<!--[if IE 5.5]>
  Este navegador es Internet Explorer 5.5
<![endif]-->

<!--[if IE 6]>
  Este navegador es Internet Explorer 6
<![endif]-->

<!--[if IE 8]>
  Este navegador es Internet Explorer 8
<![endif]-->
```

Además de las versiones específicas, también es posible restringir los comentarios condicionales a un grupo de versiones de Internet Explorer mediante los operadores *"menor que"* (`lt`), *"mayor que"* (`gt`), *"menor o igual que"* (`lte`), *"mayor o igual que"* (`gte`).

```
<!--[if lt IE 7]>
  Este navegador es cualquier versión anterior a Internet Explorer 7
<![endif]-->

<!--[if lte IE 6]>
  Este navegador es Internet Explorer 6 o cualquier versión anterior
<![endif]-->

<!--[if gt IE 7]>
  Este navegador es cualquier versión más reciente que Internet Explorer 7
<![endif]-->

<!--[if gte IE 8]>
  Este navegador es Internet Explorer 8 o cualquier versión más reciente
<![endif]-->
```

Por último, también se pueden utilizar otros operadores más complejos similares a los que se pueden encontrar en los lenguajes de programación. El operador AND (&) combina dos expresiones para crear una condición que sólo se cumple si se cumplen las dos expresiones. El operador OR (|) también combina dos expresiones y crea condiciones que se cumplen cuando al menos una de las dos expresiones se cumple. También se pueden utilizar paréntesis para crear expresiones avanzadas:

```
<!--[if !(IE 7)]>
  Este navegador es cualquier versión diferente a Internet Explorer 7
<![endif]-->

<!--[if (IE 7) | (IE 8)]>
  Este navegador es Internet Explorer 7 o Internet Explorer 8
<![endif]-->

<!--[if (gt IE 5) & !(IE 8)]>
  Este navegador es cualquier versión más reciente que Internet Explorer 5 pero que no
sea Internet Explorer 8
<![endif]-->

<!--[if (gte IE 5) & (lt IE 8)]>
  Este navegador es Internet Explorer 5 o cualquier versión más reciente que sea
anterior a Internet Explorer 8
<![endif]-->
```

Las principales ventajas de los comentarios condicionales son:

- Compatibilidad con cualquier navegador, ya que el resto de navegadores consideran que los comentarios condicionales son comentarios normales de HTML e ignoran su contenido.
- No requiere el uso de lenguajes de programación como JavaScript, aunque se puede combinar con este tipo de técnicas para detectar dinámicamente el tipo y versión de navegador.

Nota
Basándose en la idea de los comentarios condicionales, se ha desarrollado una técnica que permite crear hojas de estilos condicionales que aplican diferentes propiedades en función de la versión del navegador.

Esta técnica se denomina *Conditional CSS* (http://www.conditional-css.com/) y requiere procesar las hojas de estilos en el servidor antes de servirlas a los usuarios. Los lenguajes de programación de servidor soportados son PHP, C y C++.

6.5.2. Filtros y hacks

Los filtros y hacks es otra de las técnicas disponibles para aplicar diferentes estilos y reglas CSS en función del navegador que visualiza los contenidos. Al contrario que los comentarios condicionales, los filtros y hacks constituyen una forma poco elegante y técnica de solucionar este problema.

La idea de los filtros y hacks consiste en aprovechar los errores y carencias de los navegadores para ocultar los estilos CSS que no se deben aplicar en ese navegador. El siguiente ejemplo muestra uno de los hacks más antiguos y conocidos que aprovecha un error en las versiones obsoletas de Internet Explorer:

```
#elemento {
  width: 300px;
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 250px;
}

html > body #elemento {
  width: 300px;
}
```

Internet Explorer 5 y 5.5 no interpretan correctamente el modelo de cajas de CSS, por lo que no muestran los elementos con la misma anchura que el resto de navegadores. Si se quiere realizar un diseño con el mismo aspecto en cualquier navegador, es preciso utilizar propiedades diferentes para Internet Explorer 5/5.5 y el resto de navegadores.

Entre los muchos errores de Internet Explorer 5/5.5 se encuentra la imposibilidad de procesar correctamente la declaración `voice-family: "\"}\"";`. Por tanto, las reglas CSS anteriores se interpretan de la siguiente forma en cada navegador:

```
/* Internet Explorer 5/5.5 */
#elemento {
  width: 300px;
}

/* Internet Explorer 6 */
#elemento {
  width: 300px;
  width: 250px;
}

/* Resto de navegadores */
#elemento {
  width: 300px;
  width: 250px;
}
```

```
html > body #elemento {
  width: 250px;
}
```

El resultado final del hack anterior es que Internet Explorer 5/5.5 aplica un valor de `300px` a la propiedad `width` y el resto de navegadores aplican un valor de `250px`. Si se tienen en cuenta los márgenes, bordes y rellenos, el elemento se muestra con el mismo aspecto en cualquier navegador.

Algunos filtros y hacks son mucho más sencillos que el mostrado anteriormente. De hecho, existen tres filtros muy simples que se utilizan para aplicar diferentes estilos a cada versión de Internet Explorer. El primer filtro es `* html`, que aplica las reglas CSS a las versiones de Internet Explorer anteriores a la 7:

```
p {
  color: red;
}
* html p {
  color: blue;
}
```

Si se añade `* html` delante del selector de cualquier regla CSS, esa regla sólo se aplica a los navegadores de la familia Internet Explorer anteriores a la versión 7. De esta forma, en el ejemplo anterior los párrafos de la página se muestran de color azul en los navegadores Internet Explorer 5, 5.5 y 6, mientras que se muestran de color rojo en el resto de navegadores.

Otro de los hacks más utilizados es _ (guión bajo) que hace que una propiedad sólo se aplique a los navegadores de la familia Internet Explorer anteriores a la versión 7:

```
p {
  color: red;
  _color: blue;
}
```

La regla CSS anterior muestra los párrafos de la página de color rojo en cualquier navegador salvo en los navegadores Internet Explorer 5, 5.5 y 6, donde se muestran de color azul.

Por último, también existe el hack * (asterisco) que hace que una propiedad sólo se aplique en los navegadores de la familia Internet Explorer anteriores a la versión 8:

```
p {
  color: red;
  *color: blue;
}
```

En el ejemplo anterior, los párrafos de la página se muestran de color azul en cualquier navegador de la familia Internet Explorer cuya versión sea anterior a la 8 y se muestran de color rojo en cualquier otro navegador.

Se puede consultar la lista completa de filtros y hacks para cada navegador y cada versión en el sitio web http://centricle.com/ref/css/filters/

El uso de los filtros y hacks siempre debe considerarse como el último recurso para conseguir que los diseños tengan el mismo aspecto en cualquier navegador. Como los filtros y hacks se basan en aprovechar errores de los navegadores, su principal problema es que no garantizan su funcionamiento con las versiones más modernas de los navegadores. Además, los filtros y hacks complican el código CSS y lo hacen más difícil de leer y de mantener.

La mejor alternativa a los filtros y hacks es el uso de comentarios condicionales. Si no es posible el uso de comentarios condicionales, es preferible que el diseño de la página muestre ligeras diferencias visuales en cada navegador.

Afortunadamente, la importancia de los filtros y hacks es cada vez menor, ya que todos los navegadores modernos solucionan los graves errores de sus versiones anteriores e introducen menos errores nuevos.

6.6. Selector de navegador

Los selectores de navegador permiten aplicar reglas CSS a navegadores específicos. Aunque se trata de una idea similar a los comentarios condicionales, su gran ventaja es que funcionan en cualquier navegador y que no se limitan a seleccionar solamente el navegador.

El siguiente ejemplo muestra cómo aplicar diferentes reglas CSS en función del navegador utilizando los selectores de navegador:

```
.ie p  { ... }   /* estilos de los párrafos en Internet Explorer */
.ie6 p { ... }   /* estilos de los párrafos en Internet Explorer 6 */
.ff3 p { ... }   /* estilos de los párrafos en Firefox 3 */
```

Como CSS no incluye por defecto los selectores de navegador, se debe recurrir a soluciones basadas en el lenguaje de programación JavaScript. Una de estas soluciones ha sido creada por el diseñador Rafael Lima y se denomina *CSS Browser Selector* (http://rafael.adm.br/css_browser_selector/) .

La solución de *CSS Browser Selector* consiste en ejecutar un pequeño código JavaScript cuando se carga la página de forma que se añada una clase en el elemento `<html>` en función del navegador utilizado por el usuario.

El primer paso consiste en añadir el siguiente código JavaScript en la sección `<head>` de la página:

```
<head>
...
<script type="text/javascript">
/*
CSS Browser Selector v0.2.7
Rafael Lima (http://rafael.adm.br)
http://rafael.adm.br/css_browser_selector
License: http://creativecommons.org/licenses/by/2.5/
Contributors: http://rafael.adm.br/css_browser_selector#contributors
*/
var css_browser_selector = function() {
var ua=navigator.userAgent.toLowerCase(),
is=function(t){return ua.indexOf(t) != -1;},
```

```
h=document.getElementsByTagName('html')[0],
b=(!(/opera|webtv/i.test(ua))&&/msie (\d)/.test(ua))?('ie ie'+RegExp.$1):is('firefox/
2')?'gecko ff2':is('firefox/3')?'gecko ff3':is('gecko/')?'gecko':is('opera/9')?'opera
opera9':/opera (\d)/.test(ua)?'opera
opera'+RegExp.$1:is('konqueror')?'konqueror':is('applewebkit/')?'webkit
safari':is('mozilla/')?'gecko':'',
os=(is('x11')||is('linux'))?' linux':is('mac')?' mac':is('win')?' win':'';
var c=b+os+' js'; h.className += h.className?' '+c:c;
}();
</script>
...
</head>
```

Una vez añadido el código anterior, cada vez que un usuario accede a la página se añade una clase al elemento `<html>` que indica el navegador y sistema operativo utilizados:

```
/* navegador Firefox3, sistema operativo Windows, JavaScript habilitado */
<html class="gecko ff3 win js" ...>

/* navegador Internet Explorer, sistema operativo Windows, JavaScript habilitado */
<html class="ie ie8 win js" ...>

/* navegador Firefox 3, sistema operativo Linux, JavaScript habilitado */
<html class="gecko ff3 linux js" ...>
```

Las siguientes tablas muestran las clases que añade *CSS Browser Selector* en función del navegador y sistema operativo:

Clase	**Sistema Operativo**
`win`	Cualquier versión de Windows
`linux`	Cualquier versión y/o distribución de Linux
`mac`	Cualquier versión de Mac OS

Clase	**Navegador**
`ie`	Cualquier versión de Internet Explorer
`ie5`	Internet Explorer 5
`ie6`	Internet Explorer 6
`ie7`	Internet Explorer 7
`ie8`	Internet Explorer 8
`gecko`	Cualquier versión de Mozilla, Firefox y Camino
`ff2`	Firefox 2
`ff3`	Firefox 3
`opera`	Cualquier versión de Opera
`opera8`	Opera 8
`opera9`	Opera 9
`konqueror`	Cualquier versión de Konqueror

`webkit` `safari`	Cualquier versión de Safari y Shiira

Además, también se añade la clase `js` cuando el navegador tiene activado el soporte de JavaScript.

Como todos los valores anteriores se añaden como clase de `<html>`, en los navegadores que soportan correctamente los estándares es posible combinar varias clases para restringir aún más el alcance del selector:

```
/* estilos de los párrafos en Firefox 3 de Windows */
.ff3.win p { ... }
/* estilos de los párrafos en Firefox 3 de Linux */
.ff3.linux p { ... }
/* estilos de los párrafos en Firefox 3 de Mac OS */
.ff3.mac p { ... }
```

La versión más reciente de *CSS Browser Selector* se puede obtener a través del repositorio http://github.com/rafaelp/css_browser_selector/ donde también se encuentra una versión comprimida para mejorar ligeramente el rendimiento.