

CSS

avanzado

Capítulo 1. Técnicas imprescindibles	3
1.1. Propiedades shorthand	3
1.2. Limpiar floats	5
1.3. Elementos de la misma altura	8
1.4. Sombras	11
1.5. Transparencias	11
1.6. Sustitución de texto	13
1.7. Esquinas redondeadas	15
1.10. Rollovers y sprites	16
1.11. Texto	23
Capítulo 2. Buenas prácticas	28
2.1. Inicializar los estilos	28
2.2. Comprobar el diseño en varios navegadores	31
2.3. Mejora progresiva	32
2.4. Depuración	37
2.5. Hojas de estilos	42
2.6. Rendimiento	47
Capítulo 3. Selectores	48
3.1. Selector de hijos	48
3.2.1 Selector adyacente	49
3.3. Selector de atributos	51
3.4. Pseudo-clases	52
3.5. Pseudo-elementos	56
Capítulo 4. Propiedades avanzadas	58
4.1. Propiedad white-space	58
4.2. Propiedad display	60
4.3. Propiedad outline	66
Capítulo 5. Frameworks	70
5.1. El framework bootstrap	70
Capítulo 6. Técnicas avanzadas	71
6.1. Imágenes embebidas	71
6.2. Mapas de imagen	73
6.3. Variables en las hojas de estilos	¡Error! Marcador no definido.
6.2. Estilos alternativos	78
6.3. Comentarios condicionales, filtros y hacks	81
6.6. Selector de navegador	¡Error! Marcador no definido.

Capítulo 1. Técnicas imprescindibles

El estándar CSS 3 incluye un gran número de 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.
- *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.
- 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 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 .

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. 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:

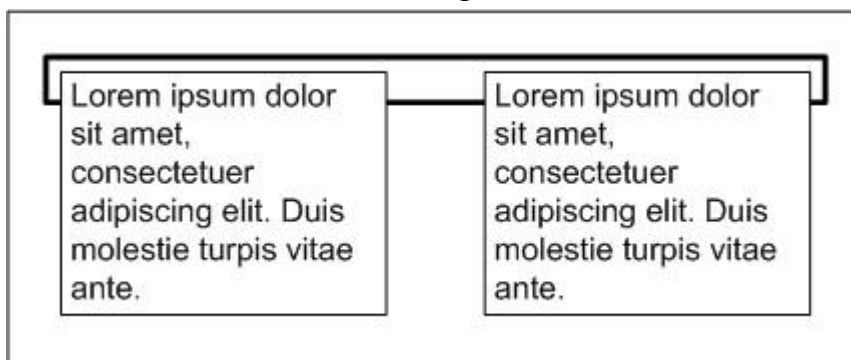


Figura 1.1. Los elementos posicionados de forma flotante se salen de su elemento contenedor

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

```
<div id="contenedor">
  <div id="izquierda">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis
  molestie turpis vitae ante.</div>
  <div id="derecha">Lorem ipsum dolor sit amet, consectetur 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>`, `
` 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, consectetur adipiscing elit. Duis
  molestie turpis vitae ante.</div>
  <div id="derecha">Lorem ipsum dolor sit amet, consectetur 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:

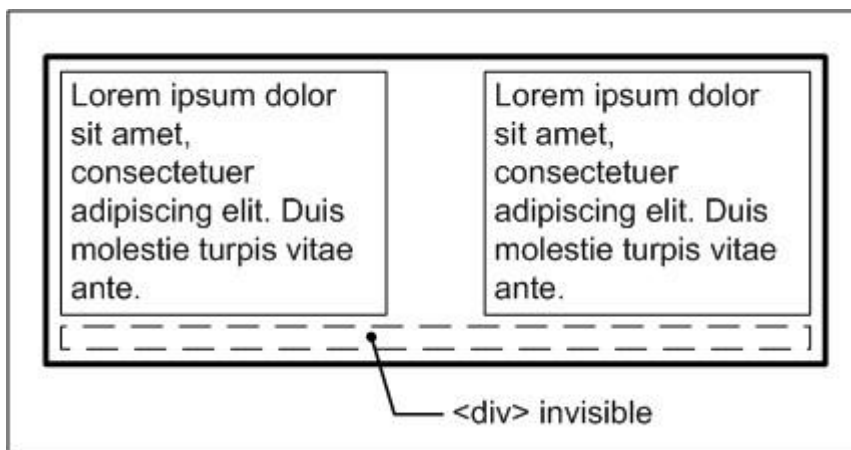


Figura 1.2. Solución tradicional al problema de los elementos posicionados de forma flotante

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%;
}
```

1.3. 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 consistía 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.

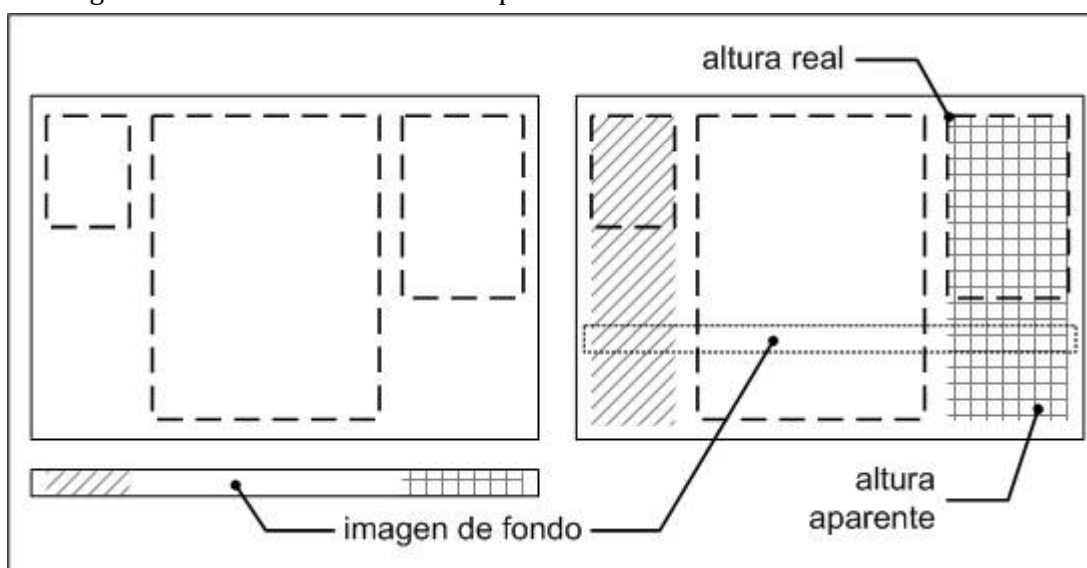


Figura 1.3. Las columnas parecen de la misma altura porque el elemento contenedor muestra una imagen de fondo repetida verticalmente

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.

1.4. Sombras

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

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;  
}
```

1.5. Transparencias

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.

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:

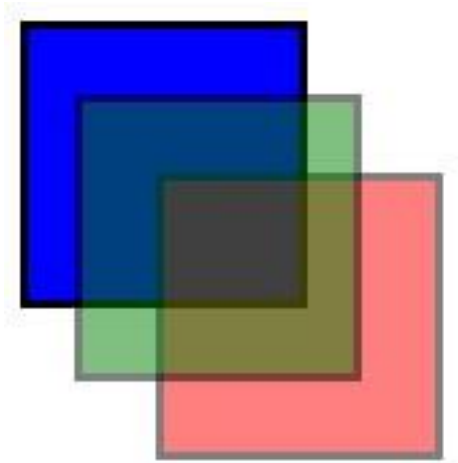


Figura 1.5. Creando elementos semitransparentes con la propiedad `opacity`

RGBA

En el `css3` podemos utilizar los códigos `rgba` donde **a** representa el valor `alpha` (como en *transparencia alpha*). De hecho el parámetro `alpha` representa la opacidad y es un número decimal de 0.0 (*completamente transparente*) a 1.0 (*totalmente opaco*).

La ventaja que ofrece `rgba` con respecto a `opacity` es que los elemento que estén contenidos no tendrán transparencias a diferencia de `opacity`.

1.6. Sustitución de texto

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-1/windows-fonts-by-version.html>).
- Sistemas operativos tipo Mac: Arial, Helvetica, Verdana, Monaco, Times.
- Sistemas 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.

1.6.1. 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.

1.6.2 Que son los Icon Fonts

Como sugiere el nombre los Icon Fonts son iconos utilizados mediante fuentes tipográficas, ya sean letras y números o caracteres Unicode. Un ejemplo en este caso puede ser muy ilustrativo.

Primero tenemos que definir la familia de fuentes a utilizar, los Icon Fontss en este caso. Para esto utilizamos la declaración `@font-face`

A continuación designamos las clases que utilizarán esta fuente. En este caso serán todas las clases que empiezan por "icon"

```
[class^="icon"] {font-family: 'miFuente';}
```

Hay varios métodos de utilizar los icon-fonts. Los podemos utilizar directamente en el HTML:

```
<span class="icon-lapiz">&#xe905;</span>
```

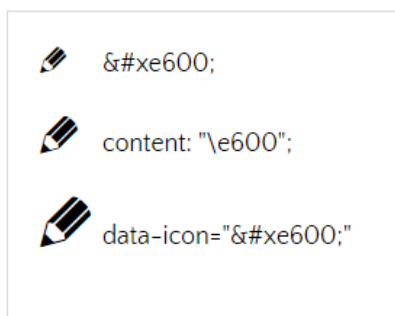
O en el CSS como el **content** del pseudo-elemento **:before**.

```
.icon-lapiz:before{content: "\e905;"}
```

O todavía mejor: utilizando el valor de un atributo **data-icon**

```
<span class="icon" data-icon="&#xe905;"></span>
```

```
.icon:before{ content:attr(data-icon);}
```



¿Por que utilizar Icon Fonts?

Utilizar iconos de fuentes tipográficas tiene muchas ventajas:

- Podemos redimensionarlos fácilmente, exactamente como cambiamos el tamaño de las letras: basta con cambiar el tamaño de fuente (`font-size: 24px;`)
- Podemos cambiar fácilmente el color de los iconos utilizando la propiedad `color` (`color:#abcdef;`)
- Podemos aplicarlos sombras con `text-shadow`, y generalmente, podemos hacer casi todo lo que podemos hacer con fuentes.

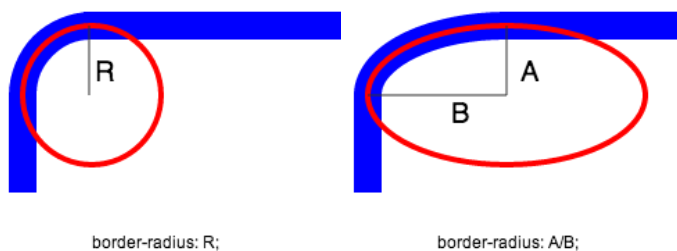
Probablemente las más conocidas Font Icons son las de Bootstrap que podemos descargar aquí: <http://fontawesome.github.io/Font-Awesome/>

1.7. Esquinas redondeadas

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.

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



Se podrían combinar las propiedades como en el ejemplo siguiente:



```
#combinado{height:6em;width:6em;background-color:#8AC007;border-top-left-radius: 3em 1em;border-top-right-radius: 1em 3em;border-bottom-right-radius: 3em 1em;border-bottom-left-radius:1em 3em;}
```

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:

Previsiones meteorológicas



Localidad 1: soleado, máx: 35°, mín: 23°



Localidad 2: nublado, máx: 25°, mín: 13°



Localidad 3: muy nublado, máx: 22°, mín: 10°



Localidad 4: tormentas, máx: 23°, mín: 11°

Figura 1.8. Aspecto de la previsión meteorológica mostrada con iconos

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

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

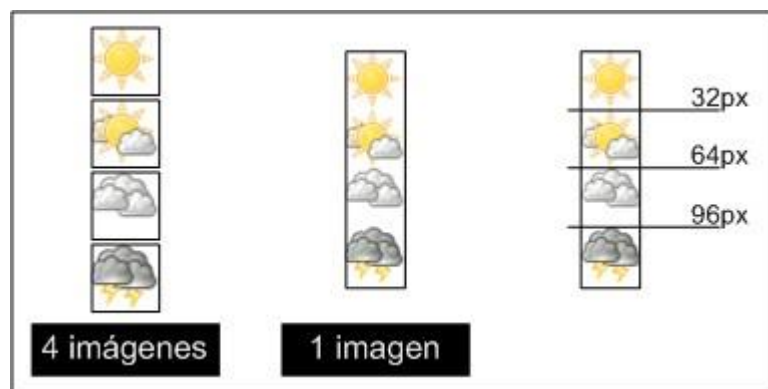


Figura 1.9. Creando un sprite de CSS a partir de varias imágenes individuales

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:

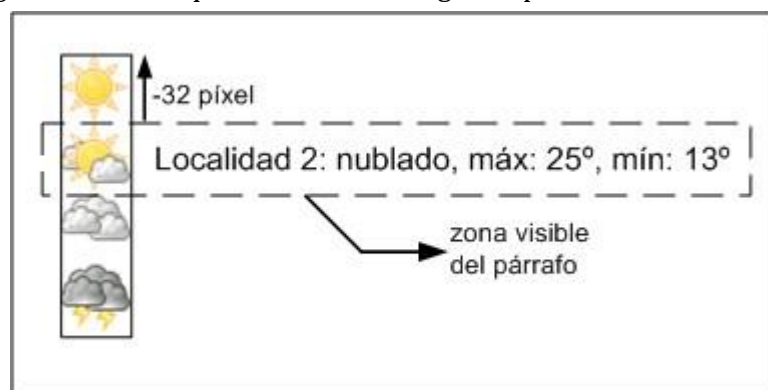


Figura 1.10. Funcionamiento de la técnica de los sprites CSS

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 píxeles, 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:

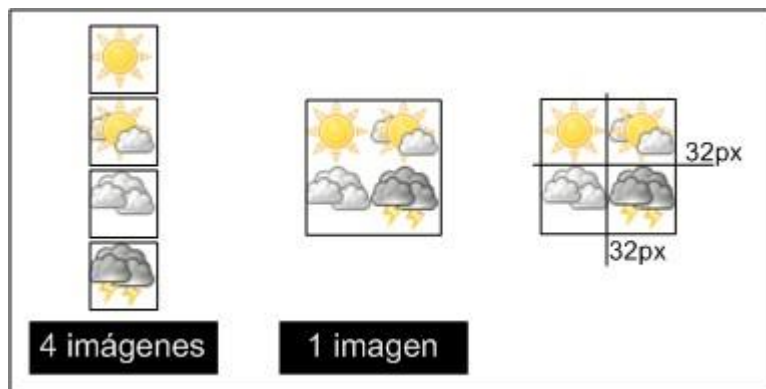


Figura 1.11. Sprite complejo que dispone las imágenes de forma vertical y horizontal

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:

Previsiones meteorológicas

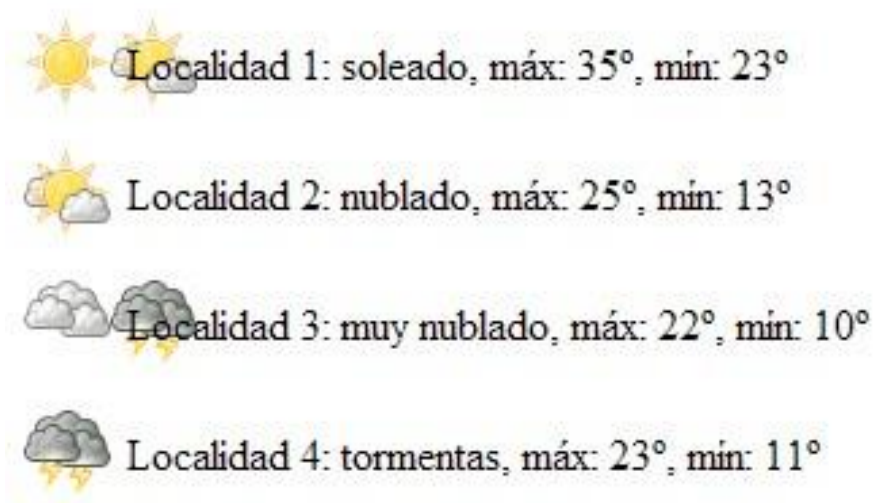


Figura 1.12. Errores producidos por utilizar un sprite complejo

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

Primero pondremos un SPAN en el párrafo:

```
<li><span id="localidad1"></span> Localidad 1: soleado, máx: 35°, mín: 23° </li>
```

En la declaración de css pondremos las siguientes propiedades:

```
#localidad1, #localidad2 , #localidad3 , #localidad4 {  
  
    display: inline-block;  
    height: 65px;  
    vertical-align: middle;  
    padding-left: 85px;  
    padding-top: 8px;  
    background-image: url("imagenes/el tiempo.png");  
    background-repeat: no-repeat;  
  
}
```

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:

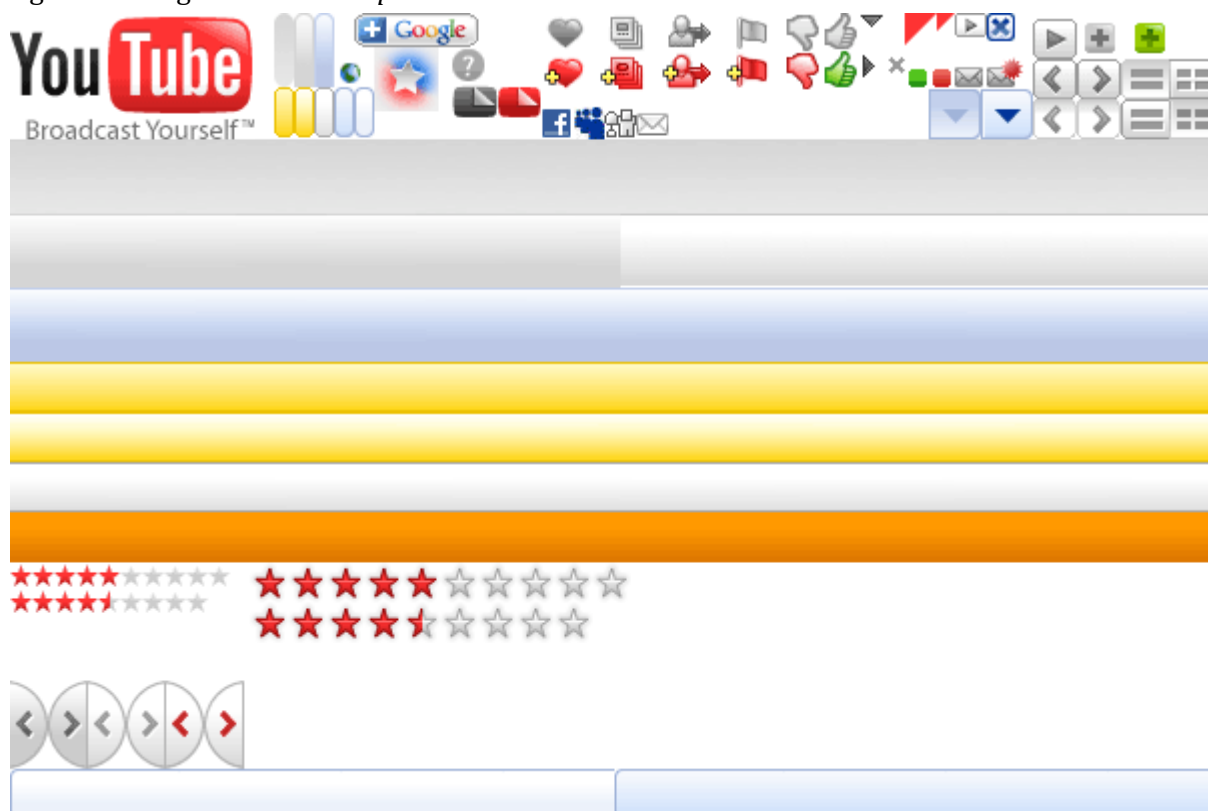


Figura 1.14. Sprite utilizado por el sitio web de YouTube

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

La propiedad llamada text-shadow se utiliza para mostrar textos con sombra.

```
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.

1.11.2.2. Efecto brillo

Si queremos crear un resplandor en vez de una sombra, tenemos que darle valor 0 a text-shadow en ambas distancias y aplicar un color claro.

```
text-shadow: 0px 0px 20px rgb(0,255,0);
```

Si queremos aumentar la intensidad del brillo, le añadimos más valores de sombra.

```
text-shadow: 0px 0px 20px rgb(0,255,0), 0px 0px 20px rgb(0,255,0), 0px 0px 20px  
rgb(0,255,0);
```



1.11.2.3 Textos en 3D

Veamos una aplicación práctica de text-shadow que dará como resultado un efecto distinto a una simple sombra paralela.

Sabemos que podemos aplicar diferentes sombras a un texto, y que podemos decidir que el difuminado de la sombra sea 0. Partiendo de esta base, podemos crear un efecto de pseudo 3D a cualquier texto.

Primero tenemos que aplicar el efecto sombra a nuestro texto. Creamos un texto blanco y una sombra a una distancia x e y de un color gris claro y con difuminado 0:

```
text-shadow: 1px 1px 0px rgb(230,230,230);
```



Ahora mismo tiene poco de 3D. Probemos a ir añadiéndole unas cuantas sombras más, aumentando en cada una 1 pixel las distancias x e y haciendo el gris más oscuro:

```
text-shadow:  
  1px 1px 0px rgb(230,230,230),  
  2px 2px 0px rgb(200,200,200),  
  3px 3px 0px rgb(180,180,180),  
  4px 4px 0px rgb(160,160,160);
```



Eso ya tiene algo más de aspecto 3D, pero aún le podemos dar un aspecto más realista. Aplicaremos una nueva sombra siguiendo el método anterior, con la diferencia que esta vez le daremos color negro. Por último añadimos una sombra mayor que todas las demás, de color gris claro y con difuminado:

```
text-shadow: 1px 1px 0px rgb(230,230,230),  
  2px 2px 0px rgb(200,200,200),  
  3px 3px 0px rgb(180,180,180),  
  4px 4px 0px rgb(160,160,160),  
  /*Añadimos*/ 5px 5px 0px rgb(0,0,0), 8px 8px 20px rgb(0,0,0);
```



1.11.2.4. Texto con relleno gradiente o degradado e imágenes

Combinando el texto con imágenes y degradados, 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.

Para ello utilizaremos las siguientes propiedades:

- **background-clip: text;** Con esta propiedad adaptaremos el background a la forma del texto.
- **text-fill-color: transparent;** Con esta propiedad ponemos el color del texto transparente para poder visualizar su background-image.

Utilizando estas propiedades podremos declarar lo siguiente:

```
h1 {  
  
    background-image:url("imagenes/sprite.png"); /* Imagen fondo */  
    -webkit-background-clip: text; /* Con esto adaptamos el degradado al texto */  
    -webkit-text-fill-color: transparent; /* Ocultamos el color del texto para no afectar el  
    degradado */  
    font-size:6em;  
  
}
```



Si utilizamos la propiedad **gradient** (que estudiaremos más adelante) podremos realizar un degradado en el fondo del texto:

```
h1 {  
  
    background-image:-webkit-gradient(linear,left top,left bottom,from(blue),to(black)); /*  
    Degradado */  
    -webkit-background-clip: text; /* Con esto adaptamos el degradado al texto */  
    -webkit-text-fill-color: transparent; /* Ocultamos el color del texto para no afectar el  
    degradado */  
    font-size:4em;  
  
}
```

Texto degradado

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, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut consectetur 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>
```

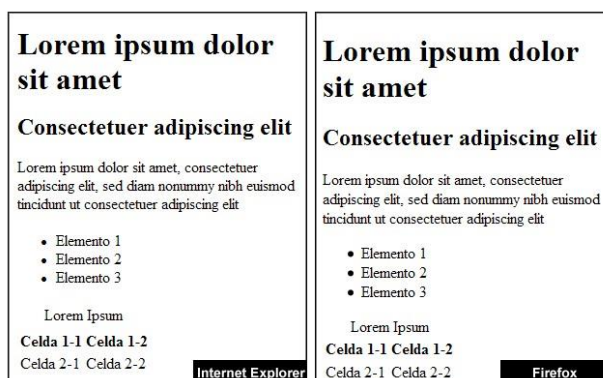


Figura 2.2. Visualización de una misma página en los navegadores Internet Explorer y Firefox

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/eric/tools/css/reset/>)

```
/* 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 , Firefox , Safari, Chrome y Opera .

En primer lugar, los diseñadores web profesionales disponen de todos los navegadores principales instalados en sus equipos de trabajo.

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.

Otras aplicaciones web que realizan la misma tarea:

- <http://crossbrowsertesting.com/>
- <https://browserling.com/>

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.

A continuación se muestra la mejora progresiva en la práctica mediante un ejemplo:

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>
  <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:

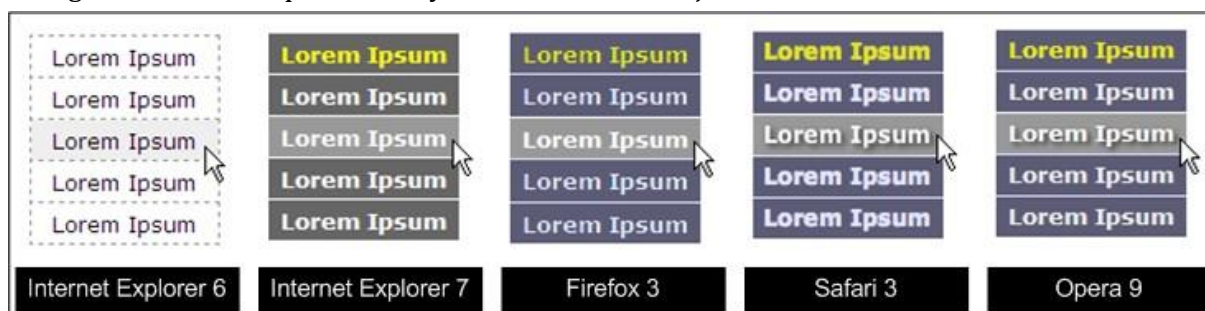


Figura 2.6. Aspecto del menú en los navegadores Internet Explorer 6 y 7, Firefox 3, Safari 3 y Opera 9

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:

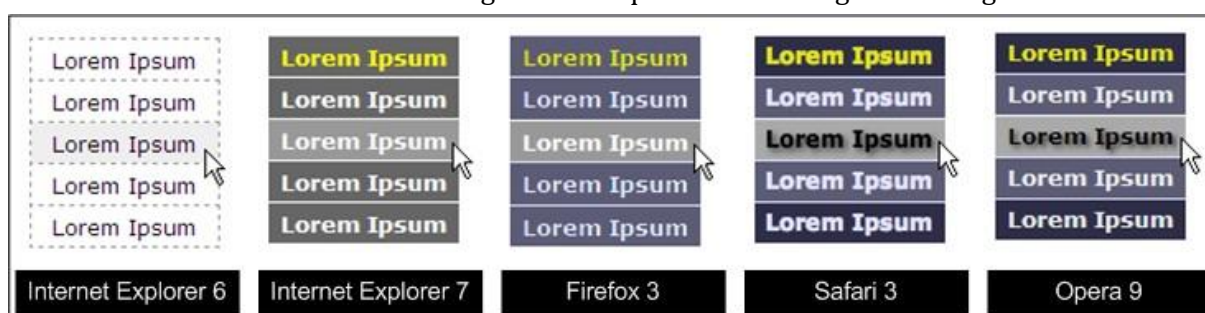


Figura 2.7. Aspecto del menú en los navegadores Internet Explorer 6 y 7, Firefox 3, Safari 3 y Opera 9

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.

2.4.1. Firebug

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:

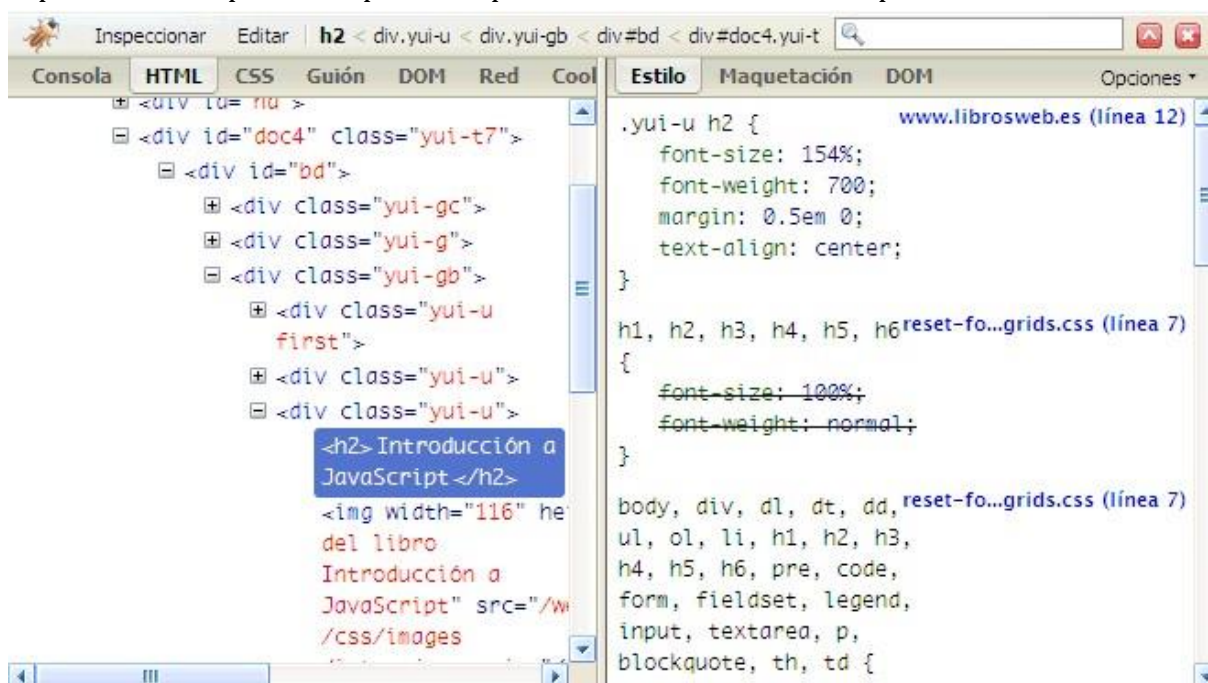


Figura 2.8. Panel HTML de Firebug

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:

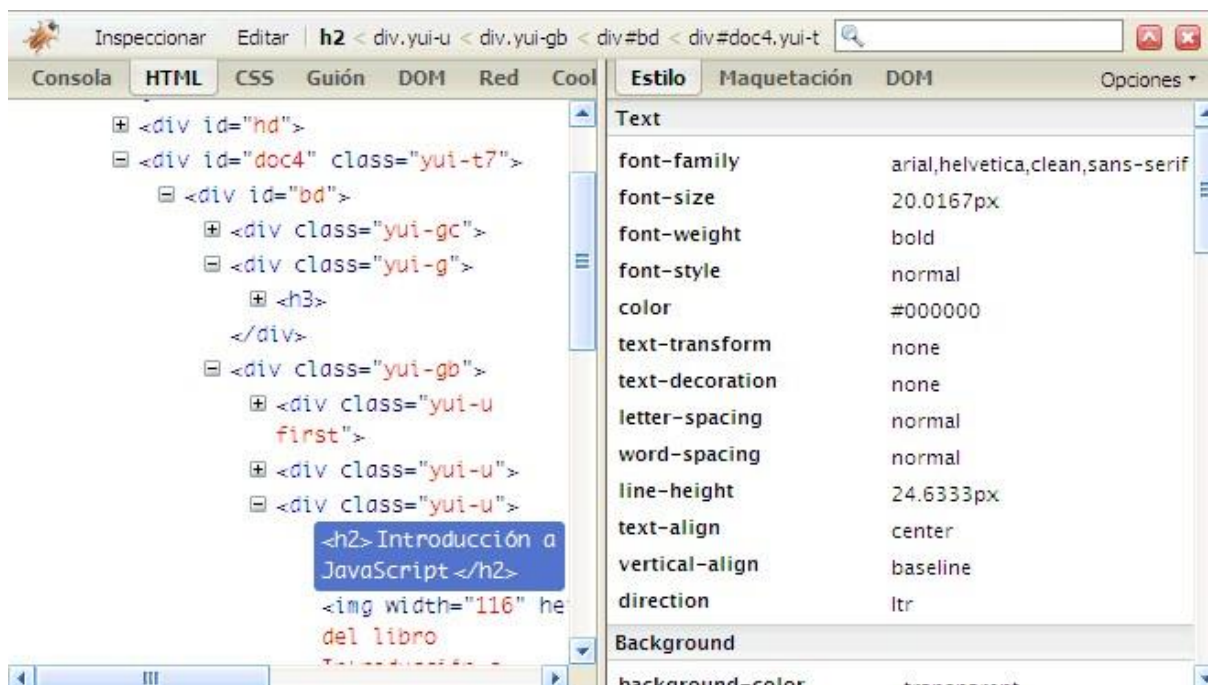


Figura 2.9. Panel HTML de Firebug con la opción Show Computed Style

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:

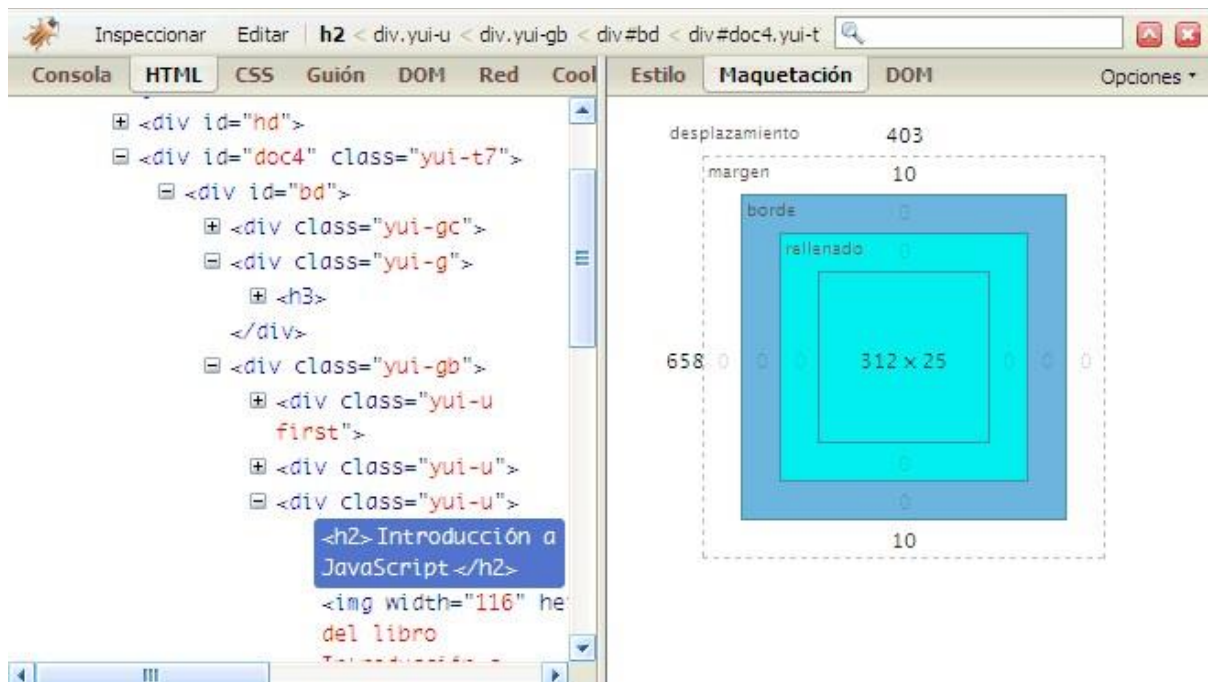


Figura 2.10. Pestaña Maquetación del panel HTML de Firebug

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:

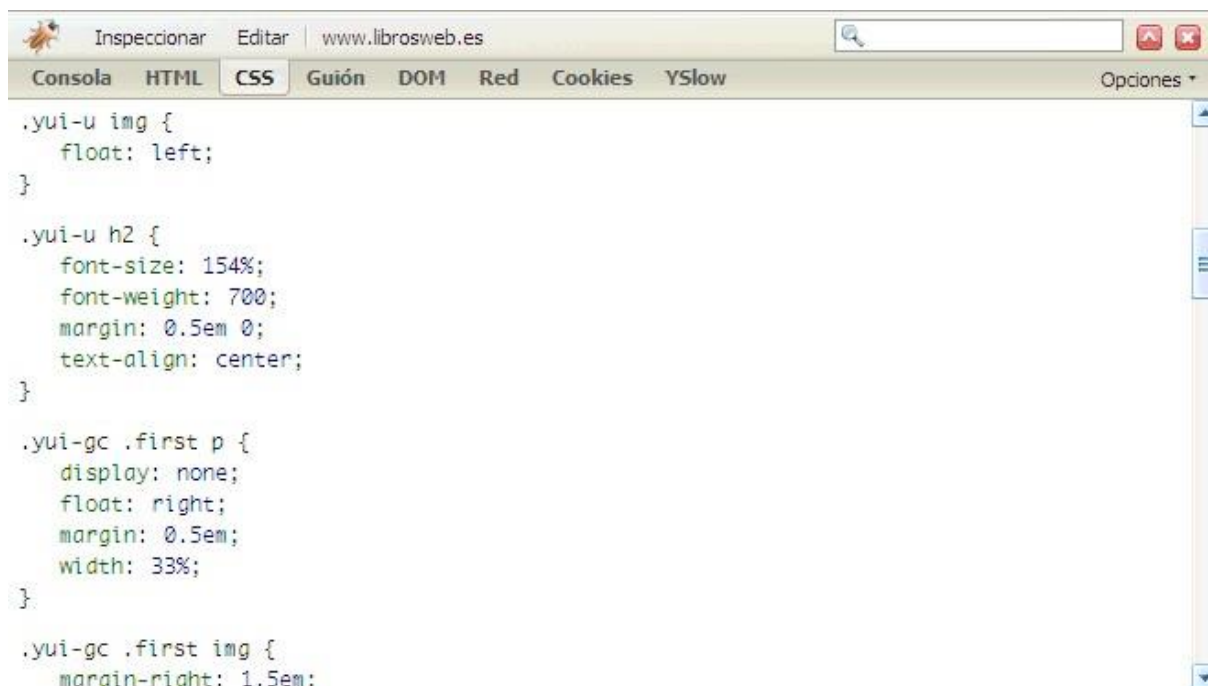
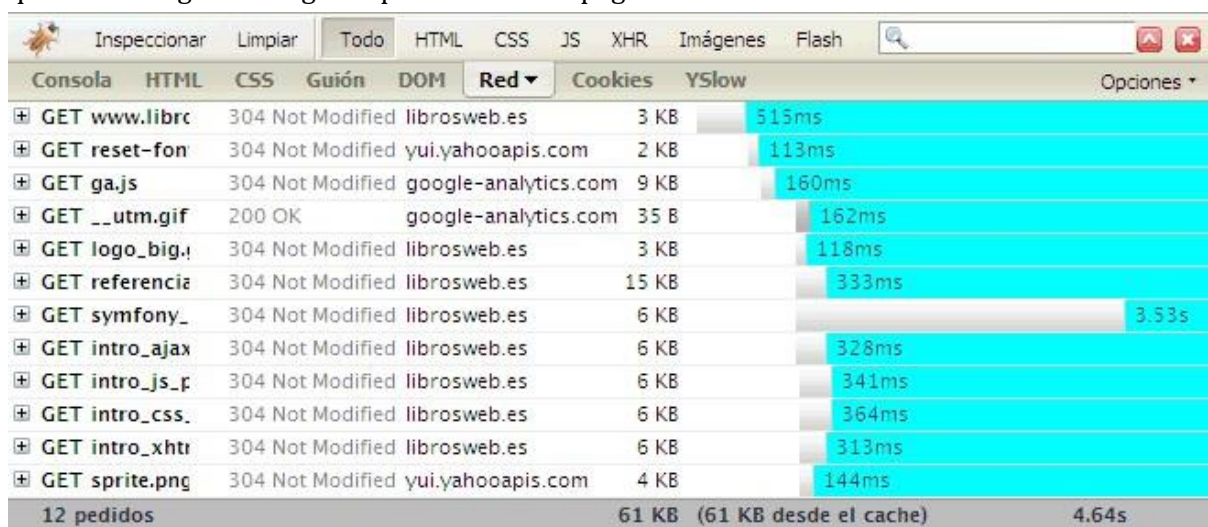


Figura 2.11. Panel CSS de Firebug

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:



Method	Status	Response	Size	Time
GET	304 Not Modified	librosweb.es	3 KB	515ms
GET	304 Not Modified	yui.yahooapis.com	2 KB	113ms
GET	304 Not Modified	google-analytics.com	9 KB	160ms
GET	200 OK	google-analytics.com	35 B	162ms
GET	304 Not Modified	librosweb.es	3 KB	118ms
GET	304 Not Modified	librosweb.es	15 KB	333ms
GET	304 Not Modified	librosweb.es	6 KB	3.53s
GET	304 Not Modified	librosweb.es	6 KB	328ms
GET	304 Not Modified	librosweb.es	6 KB	341ms
GET	304 Not Modified	librosweb.es	6 KB	364ms
GET	304 Not Modified	librosweb.es	6 KB	313ms
GET	304 Not Modified	yui.yahooapis.com	4 KB	144ms

12 pedidos 61 KB (61 KB desde el cache) 4.64s

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

• Chrome Developer Tools

Chrome Developer Tools te permite profundizar en las entrañas de una página web y revisar casi todo, desde su estructura hasta los recursos utilizados. En general puedes:

- Obtener una visión general de los estilos utilizados en una página y que estilo se aplica a cada elemento.
- Visualizar y modificar el código correspondiente a los elementos HTML
- Modificar un estilo y ver los cambios en el navegador en tiempo real
- Ejecutar código JavaScript en el contexto de la página
- Ver las peticiones HTTP realizadas por el navegador
- Identificar cuellos de botella que afecten al rendimiento
- Consultar métricas de rendimiento
- Investigar los recursos offline para averiguar que datos de la página se almacenan localmente



Explore and Master Chrome DevTools :

- **Herramienta del W3C para validación de hojas de estilo**

El servicio de validación de CSS es un software libre creado para ayudar a diseñadores y desarrolladores web a validar Hojas de Estilo en Cascada (CSS)

- <https://jigsaw.w3.org/css-validator/>

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 productividad 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 propiedades para facilitar su búsqueda y evitar duplicidades:

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

Según la [“Google HTML/CSS Style Guide”](#) el orden de las propiedades se recomienda alfabéticamente.

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 ``, `` y ``, 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.
- 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.

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 utilizando 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.

Capítulo 3. Selectores

Conocer y dominar todos los selectores de CSS es imprescindible para crear diseños web profesionales.

Utilizando solamente los cinco selectores básicos de CSS (universal, de tipo, descendente, de clase y de id) es posible diseñar cualquier página web. No obstante, los selectores avanzados de CSS 3 permiten simplificar las reglas CSS y también el código HTML.

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 `` cumple la condición del selector. Sin embargo, el segundo elemento `` 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.

3.2.1 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.

3.2.2 Selector general de elementos hermanos

Generaliza el selector adyacente. 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.

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.
- `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.

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 }

/* 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"] { ... }

```

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, consectetur adipiscing elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim. Praesent nulla
ante, <em>ultrices</em> id, porttitor ut, pulvinar quis, dui.</p>

```

El selector `p em:first-child` selecciona el primer elemento `` que sea hijo de un elemento `<p>`. Por tanto, en el ejemplo anterior sólo el primer `` 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, consectetur
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, consectetur 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, consectetur adipiscing
elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim.</p>
</div>
</body>

```

El selector `p:first-child` selecciona todos aquellos elementos `` 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 `` se ven de color rojo. El segundo párrafo de la página no es el primer hijo de ningún elemento, por lo que sus elementos `` 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 `` se ven de la misma forma que los del primer párrafo.

3.4.2 La pseudo-clase : `nth-child`

- `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.

Podemos seleccionar los hijos impares (`odd`) o pares (`even`) utilizando las palabras clave `odd` o `even`.

- `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.

Ej: Se podría utilizar para aplicar tramas a celdas vacías en una tabla:

```
Table td:empty{ background-image:url(images/pattern.png);}
```

- `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; }
```

`:not()`, 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" */
```

3.4.3. 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; }
```

3.4.4. 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  { ... }
```

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 { ... }`

3.5. 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**.

3.5.4 Pseudo-elemento **::seleccion**

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

```
h1::seleccion { background-color: yellow; }
```

3.5.5 El selector **:target**

Todos sabemos cómo enlazar a un elemento con un id especificado dentro de una página:

```
<p id="arriba">Arriba</p>  
.  
.  
.  
.  
<a href="#arriba">Ir arriba</a>
```

El elemento enlazado es un elemento de destino, o un elemento **"target"**. Podemos dar formato al elemento de destino utilizando el selector **:target** de CSS3.

```
:target {  
  
    border: 1px solid #d9d9d9;  
  
    background-color:#FFC;  
  
}
```

Capítulo 4. Propiedades avanzadas

4.1. Propiedad white-space

Definición	Establece el tratamiento de los espacios en blanco
Valores permitidos	Uno y sólo uno de los siguientes valores: <ul style="list-style-type: none"> ▪ 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, consectetur 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, consectetur 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, consectetur 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, consectetur 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 salvo el espacio en blanco que separa las palabras del texto.

, los dos párrafos se ven exactamente igual.

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,    consectetur 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,    consectetur 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 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	Respetar espacios en blanco	Respetar 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, consectetur 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, consectetur 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, consectetur 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, consectetur adipiscing elit. Vestibulum dictum. Class

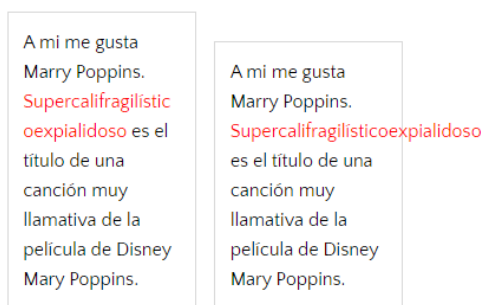
[white-space: pre-line] Lorem ipsum dolor sit amet, consectetur adipiscing elit

Vestibulum dictum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

4.1.2 Cambio de línea (word-break)

La propiedad word-break decide si las palabras tienen que romperse o no al final de línea. La propiedad word-break puede tomar dos valores:

word-break: *break-all* (rompe las palabras)
word-break: *keep-all* (no las rompe)



```
p.rompe_palabras{ word-break: break-all;;}
p.palabras_enteras{ word-break: keep-all;; }
```

4.2. Propiedad **display**

Definición	Establece el tipo de caja generada por un elemento		
Valores permitidos	Uno y sólo uno de los siguientes valores: <ul style="list-style-type: none"> ▪ inline ▪ run-in ▪ inline-table ▪ table-footer-group ▪ table-column ▪ none ▪ block ▪ inline-block ▪ table-row-group ▪ table-row ▪ table-cell ▪ inherit ▪ list-item ▪ table ▪ table-header-group ▪ table-column-group ▪ table-caption 		
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 infrutilizadas. 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, ya que establece el tipo de la caja que genera cada elemento.

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, consectetur 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, consectetur 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, consectetur 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, consectetur 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, consectetur 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, consectetur 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.

El ejemplo anterior se debe visualizar tal y como muestra la siguiente imagen:

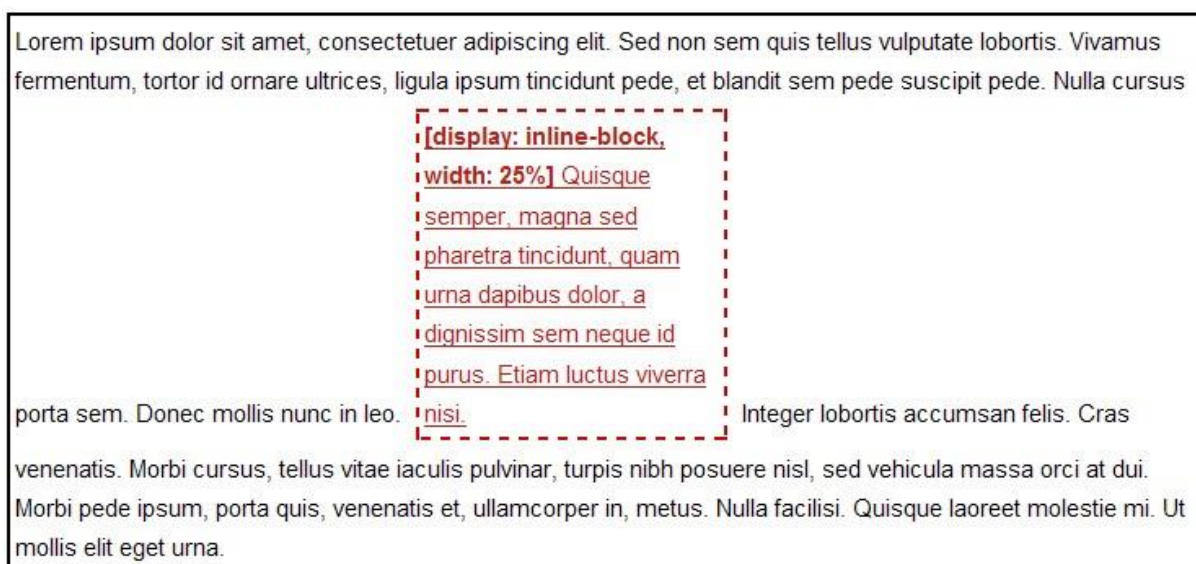


Figura 4.1. Ejemplo del valor inline-block de la propiedad display

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 ``). El siguiente ejemplo muestra tres párrafos que utilizan la propiedad `display: list-item` para simular que son una lista de elementos de tipo ``:

- Lorem ipsum dolor sit amet, consectetur 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,
consectetur 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 `` 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`.

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, consectetur 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>`.

[display: table-cell] Lorem ipsum dolor sit amet, consectetur 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.
---	---	--

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.

Por último tenemos el valor **FLEX** para la propiedad `display`. Flexbox representa un modelo básico de maquetación que supone la existencia de una caja padre llamada contenedor flexible o caja flex. Los elementos hijos situados dentro del contenedor flexible llevan el nombre de elementos o ítems flex, pero eso lo veremos en el siguiente anexo.

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: <ul style="list-style-type: none"> ▪ Color de perfil (<code>./outline-color.html</code>) ▪ Estilo de perfil (<code>./outline-style.html</code>) ▪ Anchura de perfil (<code>./outline-width.html</code>)
Valor inicial	Cada propiedad define su propio valor por defecto
Se aplica a	Todos los elementos
Válida en	medios visuales, medios interactivos (<code>./medios.html</code>)
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:

```
span { border:  
2px solid  
#369; }  
Ejemplo de  
texto largo que  
ocupa varias  
líneas
```

```
span { outline:  
2px solid #369;  
} Ejemplo de  
texto largo que  
ocupa 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 propiedad `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`.

`pseudo-clase: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.

```
}
```

Capítulo 5. Frameworks

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.

5.1. BOOTSTRAP.



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 embebidas 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 (``) 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 -->


<!-- Imagen embebida que el navegador puede mostrar directamente porque ya dispone
de sus bytes -->


```

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("data:image/png;base64,iVBORw0KGgo
AAAANSUHEUGAAABAAAAAQCAAAAAf8/9hAAABGdBTUEAAK/INwWK6QAAABl0RVh0U29mdHdhcmUAQ
W
RvYmUgSW1hZ2VSZWFkeXhJZTAAAHjSURBVDJldZ0/a1VBEMZ/5+TemxAbFUUskqAoSOJNp4KC4AsoP
oGFIHY+gA+jjXaKIiChbETtBYLUBSMRf6Aydndmfks9kRjvHdhGVh2fvN9uzONJK7fe7Ai6a1gA3FZ
CAmQqEF/dnihpK1v7x7dPw0woF64Izg3X15s1n9uIe0lQYUFCtjc+sVuEqHBKfpVAXB1vLzQXFtdYPH
kGFUCoahVo1Y/fnie+bkBV27c5R8A0pHxyhKvPn5hY2MHRQAQeyokFGJze4cuZfav3gLNyDTg7Pk1zpw
w4ijtIQYRwF6BhdjtCk+erU0CCPfg+/o2o3ZI13WU1LGo58YMg+GIY4dmCwkCAAgPzAspJW5ePFP1V
3VI4uHbz5S5IQfy/yooHngxzFser30iFcNcuAVGw3A0Ilt91IkAsyCXQg5Q00szHEIrogkiguWN2acC
oJhjnZGKYx4Ujz5W0A2YD1BMU+BBsYVUvNpxkXuIuWgbs0xThRG3UHIFWIhsgXtQQpTizNBS5jXZQkh
kcywZqQQlAjdrWiml7wU5xWLaL1AvZa8WIjALzIRZ7YVWDW5CiIj48Z8F2pYL11ZR0+AuzEX0UX035m
xIkLq0dhDw5vXL97fr503rFwQHJhPx4uuH57f2AL8BfPrVlrs6xwsAAAAASUVORK5CYII="); }

```

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 `` 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 fotografía 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 ``:

```
<div class="mapa_imagen">
  

  <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 ``, en la que cada elemento `` es un comentario.

El elemento `<div>` y la lista `` deben utilizar el atributo `class` y no `id` porque en una misma página puede haber varios mapas de imagen. Por su parte, los elementos `` 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 `` 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 `` 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 ``. 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. 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.

6.4. 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 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 servirlos a los usuarios. Los lenguajes de programación de servidor soportados son PHP, C y C++.

Capítulo 7. Transformación y Animación

7.1.Transform.

La especificación oficial y el estado actual de desarrollo del módulo Transforms en CSS 3 puede consultarse en <http://www.w3.org/TR/css3-transforms/>.

Al modificar el espacio de coordenadas, las transformaciones CSS permiten cambiar la posición del contenido afectado sin interrumpir el flujo normal del resto de cajas. Se llevan a cabo mediante un conjunto de propiedades CSS que permiten aplicar transformaciones lineales afines a los elementos HTML. Estas transformaciones incluyen la rotación, inclinación, escala y traslación tanto en el plano como en un espacio tridimensional.

Propiedades principales

Se utilizan dos **propiedades** principales para definir las transformaciones CSS: *transform* y *transform-origin*.

- **transform-origin**: especifica la posición de origen de la transformación. Por defecto se encuentra en el centro del elemento (como si definiésemos el punto en 50% 50%), pero se puede definir cualquier otro punto. Es utilizado por varias transformaciones, como rotación, escala o inclinación, que necesitan un punto inicial como parámetro.
- **transform**: especifica las transformaciones a aplicar al elemento. Se trata de una lista de funciones de transformación separadas por espacios, que se aplican una detrás de otra.

Las siguientes propiedades adicionales, añaden control adicional a las transformaciones, permitiendo incluso realizar **transformaciones 3D**:

- **perspective**: permite cambiar la perspectiva de los elementos y transmitir la sensación de encontrarse en un entorno en tres dimensiones.
- **perspective-origin**: especifica la posición de origen de la perspectiva.
- **transform-style**: permite a los elementos transformados en 3D y a sus descendientes también transformados en 3D, compartir un espacio 3D común.

TRANSFORM

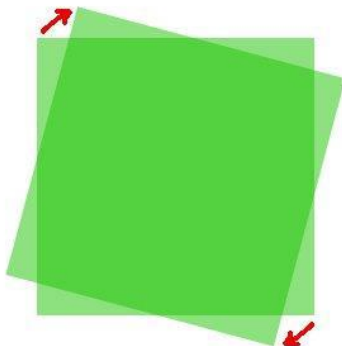
El atributo **transform** nos permite, como su propio nombre indica, transformar un elemento. Su sintaxis es la siguiente:

```
transform: tipo(cantidad);
```

El valor tipo puede tomar cuatro valores y cada uno de ellos realiza una función diferente:

- **Rotate:** Nos permite girar los elementos un número de grados. La sintaxis es:

```
transform: rotate(25deg);
```

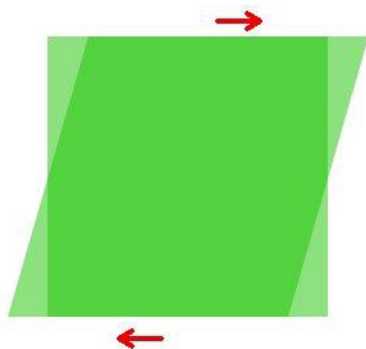


El valor que demos al giro se aplicará en sentido horario

- **Skew:** Podemos inclinar un elemento tanto en coordenadas X como Y. El valor se expresa en grados y la sintaxis es la siguiente:

```
/*transform: skew(gradosX, gradosY);*/
```

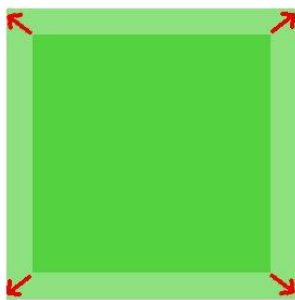
```
transform: skew(15deg, 3deg);
```



- **Scale:** Con este tipo podremos escalar nuestro elemento tanto en X como en Y en una cantidad expresada en tantos por uno:

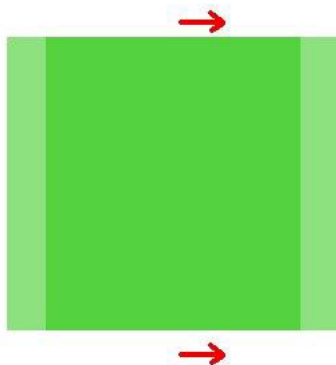
```
/*transform: scale(escalaX, escalaY);*/
```

```
transform: scale(1.5,0.6);*/
```



- **Translate:** Podemos desplazar el elemento tanto en X como en Y.

```
/*transform: translate(desplazamientoX, desplazamientoY);*/
transform:translate(12px, 19px);
```



NOTA: Se pueden aplicar diferentes transformaciones a un mismo elemento simplemente escribiéndolas de manera consecutiva:

```
transform: scale(1.6) skew(10deg) translate(5px) rotate(12deg);
```

7.1.2 Transform-origin.

Especifica la posición de origen de la transformación, puede tomar los parámetros bottom, right, top y left.

```
div { transform: rotate(90deg);
      transform-origin: bottom left;
}
```

```
<div>
```

```
    <iframe src="http://www.google.com/" width="500" height="600"></iframe>
</div>
```

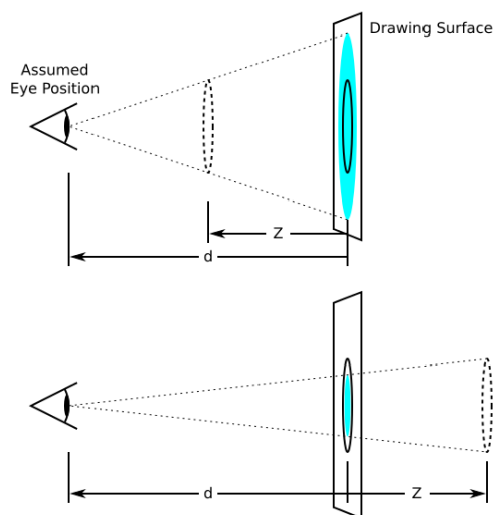
7.2 Propiedades 3D

La realización de transformaciones CSS en el espacio es algo más complejo. Hay que empezar configurando el espacio 3D, dándole un punto de vista. Después, hay que configurar cómo se comportarán los elementos 2D en ese espacio tridimensional.

7.2.1 La creación de un punto de vista (o perspectiva).

El primer elemento a configurar es la perspectiva. La perspectiva es lo que da la impresión de tres dimensiones. Cuánto más lejos del espectador se encuentran los elementos, más pequeños se nos muestran.

Cuán rápido estos elementos reducen su tamaño es definido por la propiedad `perspective`. Cuanto más pequeño es su valor, más profunda es la perspectiva.



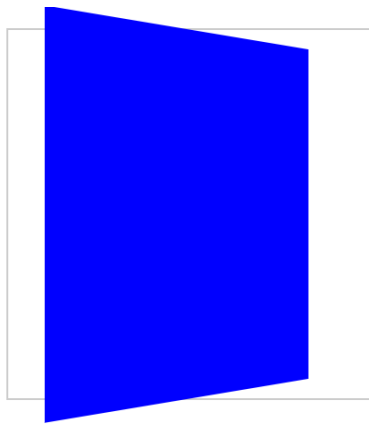
La perspectiva se podrá aplicar de dos maneras:

a) Mediante la propiedad **`transform: perspective (valor);`**

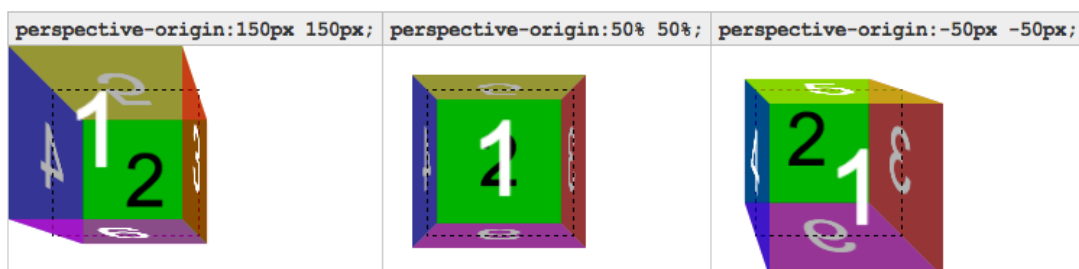
```
#red .box {  
  
  background-color: red;  
  
  transform: perspective( 600px ) rotateY( 45deg );  
  
}
```

b) Mediante la propiedad **perspective**:

```
#blue {  
  
  perspective: 600px;  
  
}  
  
#blue .box {  
  
  background-color: blue;  
  
  transform: rotateY( 45deg );  
  
}
```



El segundo elemento a configurar es la posición del espectador, con la propiedad `perspective-origin`. Por defecto, la perspectiva se centra en el espectador, lo cual no siempre es lo adecuado. Esta propiedad es equivalente a la propiedad `Transform-origin` en 2d. Una vez realizado esto, se puede trabajar sobre el elemento en el espacio 3D.

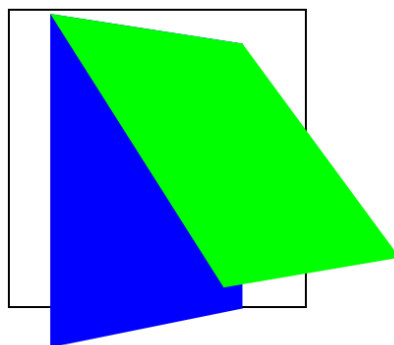


7.3.2 Transform-style

Esta es otra propiedad importante en el espacio 3D. Se necesitan dos valores: ***preserve-3D*** o ***flat***. Cuando el valor de estilo de transformación es *preserve-3d* entonces le dice al navegador que el elemento de los hijos también se debe colocar en el espacio 3D. Por otro lado, cuando el valor de esta propiedad es *flat*, indica que los hijos están presentes en el plano del propio elemento.

Ejemplo:

```
div {  
  
    height: 150px;  
  
    width: 150px;  
  
    margin: 10px auto 50px; }  
  
.contenedor {  
  
    border: 1px solid black;  
  
    -webkit-perspective: 500px; }  
  
.transformar {  
  
    background-color: blue;  
  
    -webkit-transform-style: preserve-3d;  
  
    -webkit-transform: rotateY(50deg); }  
  
.hijo {  
  
    -webkit-transform-origin: top left;  
  
    -webkit-transform: rotateX(40deg);  
  
    background-color: lime; }  
  
<div class="contenedor">  
  
    <div class="transformar">  
  
        <div class="hijo"></div>  
    </div>  
</div>
```



7.3.3 Funciones de transformación 3D

Funciones que se utilizan en la transformación 3D:

- **Translate3d** (<translation-value>, <translation-value>, <longitud>) : define una translación 3D.Lleva tres parámetros x, y, z valores. El valor Z especifica la translación en el eje Z.
- **TranslateZ** (<longitud>) : Funciona de manera similar a TranslateX () y TranslateY () . para el eje Z.
- **Scale3d** (<número>, <número>, <número>) : Esta función hace el escalamiento en las tres dimensiones. Se necesitan tres parámetros como sx, sy y sz. Cada valor define la escala en la dirección correspondiente.
- **ScaleZ** (<número>) : Al igual que el translate () función, igual scaleZ () define la escala en una sola dirección, es decir en la dirección Z. También tenemos scaleX ()y scaleY () funciones que también trabajan similar a scaleZ () , pero en sus respectivas direcciones.
- **Rotate3d** (<número>, <número>, <número>, <ángulo>) : Gira un elemento HTML en el ángulo especificado en el último parámetro de [tx, ty, tz] vector especificado por los primeros tres números.
- **RotateX** (<ángulo >) , **rotateY** (<angle>) y **RotateZ** (<ángulo >) tienen un único valor de ángulo para rotar en el eje correspondiente.

Nota: Rotar3D (1,0,0,30 grados) es igual a RotateX (30deg) , Rotar3D (0,1,0,30 grados) es igual a rotateY (30deg) y Rotar3D (0,0,1,30 grados) es igual a RotateZ (30deg).

7.4. Animaciones

7.4.1 Módulo Animations

Tradicionalmente cuando queríamos incluir cualquier tipo de animación en nuestras páginas webs, teníamos que recurrir a tecnologías como Flash o JavaScript. Si bien la potencia de los dos citados es descomunal en comparación, las nuevas funciones de animación de CSS3 abren una nueva puerta a nuestra imaginación, sin tener que depender de tecnologías extras.

7.4.1.1 Fotogramas claves de las animaciones

Un fotograma clave no es más que un punto destacado en el tiempo de nuestra animación. Cualquier animación consta al menos de dos fotogramas claves: el punto inicial y el punto final. Imaginad que nuestra animación es como una carretera:



El primer semáforo actuaría como fotograma clave inicial y el segundo como el final. Entre uno y otro se produciría nuestra animación, que no es más que el desplazamiento del coche hacia la derecha.

7.4.1.2 @keyframes

En CSS3 creamos animaciones completas mediante @keyframes, que son un conjunto de fotogramas clave. Su sintaxis es la siguiente:

```
@keyframes nombreAnimacion{
    puntoDelKeyframe{ atributosIniciales; }
    puntoDelKeyframe{ nuevosAtributos; }
    .....
    puntoDelKeyframe{ últimosAtributos; }
}
```

Veamos que tendríamos que hacer para desplazar nuestro coche a la derecha:

```
@keyframes animacionCoche{
    /*Indicamos que salimos de la posición 0*/

    from{
        left:0px;
    }
    /*Indicamos que al final la posición debe ser 350*/
    to{
        left:350px;
    }
}
```

Podemos crear animaciones más complejas estableciendo fotogramas claves intermedios mediante porcentajes:

```
@keyframes animacionCoche{
    from{
        left:0px;}
    /*Hasta el 65% de la reproducción solo queremos que se desplace 10 pixels*/
    65%{
        left:10px;}
    to{
        left:350px;}
}
```

7.4.1.3 Aplicando la animación a un elemento

Hemos creado nuestra animación. Para aplicarla sobre un elemento de nuestra página, deberemos acudir al mismo y agregarle el atributo de animación:

```
#coche{
    animation-name: animacionCoche;
    animation-duration: 3s;
    animation-iteration-count: 1;
    position:relative;
}
```

Hay tres nuevos atributos que no habíamos visto antes. Estos y otros tantos conforman los atributos destinados a la animación que describo a continuación:

- **Animation-name.** Indica a que animación corresponde nuestro elemento. Es posible definir más de una animación usando comas como separador.
- **Animation-duration.** Define la duración en segundos de nuestra animación.
- **Animation-iteration-count.** Define cuantas veces se reproduce la animación. Podemos darle el valor infinite para que se reproduzca hasta el fin de los tiempos.
- **Animation-direction.** Por defecto nuestra animación se reproducirá hacia delante, como es lógico. Sin embargo si le damos el valor alternate, al reproducirse completamente la animación esta volverá a reproducirse en sentido opuesto, es decir, como si se rebobinase.
- **Animation-delay.** Indica si se produce un retardo en el inicio de la animación.
- **Animation.** El shorthand de todos los anteriores. name duration timing-function delay iteration-count direction;
- **Animation-timing-function.** Define como progresa la animación entre los keyframes mediante ecuaciones matemáticas.

7.4.1.4 animation-timing-function

Es posible que la definición que acabas de leer sobre el atributo animation-timing-function te ha dejado pensando WTF. este atributo sirve para aplicar un efecto de suavizado a la animación.

Por defecto responde al valor ease, pero puede responder a diferentes valores:

- **Ease**
- **Linear**
- **Ease-in**
- **Ease-out**
- **Ease-in-out**

NOTA: Un buen diseñador o desarrollador debe tener los pies en la tierra y saber que las limitaciones de compatibilidad son un handicap brutal para el uso de estas tecnologías. El diseño, a diferencia del arte, debe cumplir una función práctica a parte de estética.