# Contenido



# Capítulo 1. Técnicas imprescindibles

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

En las próximas secciones se muestran las siguientes técnicas imprescindibles:

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

### 1.1. Propiedades shorthand

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

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

| **font** | **Tipografía** |
|---|---|
| **Valores** | `( (<font-style> \|\| <font-variant> \|\| <font-weight> )? <font-size>( /<line-height> )? <font-family> ) \| caption \| icon \| menu \| message-box \| small-caption \| status-bar \| inherit` |
| **Se aplica a** | Todos los elementos |
| **Valor inicial** | - |
| **Descripción** | Permite indicar de forma directa todas las propiedades de la tipografía de un texto |
| **margin** | **Margen** |
| **Valores** | `( <medida> \| <porcentaje> \| auto ) {1, 4} \| inherit` |
| **Se aplica a** | Todos los elementos salvo algunos casos especiales de elementos mostrados como tablas |
| **Valor inicial** | - |
| **Descripción** | Establece de forma directa todos los márgenes de un elemento |
| **padding** | **Relleno** |
| **Valores** | `( <medida> \| <porcentaje> ){1, 4} \| inherit` |
| **Se aplica a** | Todos los elementos excepto algunos elementos de tablas como grupos de cabeceras y grupos de pies de tabla |

| | |
|---|---|
| **padding** | **Relleno** |
| **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:

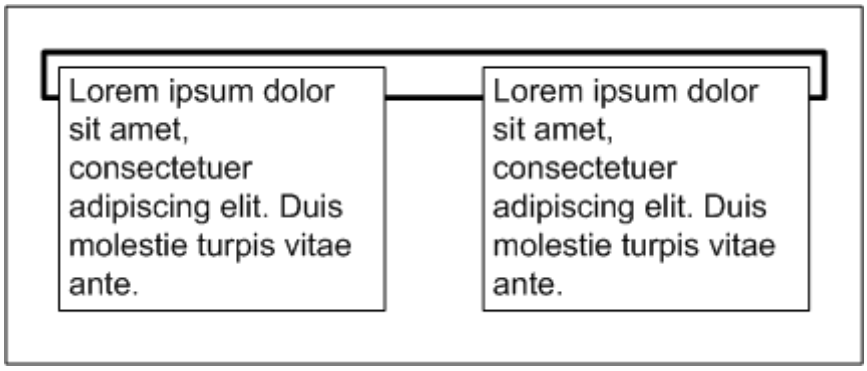



Figura 1.1. Los elementos posicionados de forma flotante se salen de su elemento contenedor

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

```
<div id="contenedor">
<div id="izquierda">Loremipsum dolor sitamet, consectetuer adipiscing
elit. Duis molestie turpis vitae ante.</div>
<div id="derecha">Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Molestie urpis vitae ante.</div>
</div>

#contenedor{
border: thick solid #000;
}

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

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

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

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

```
<divid="contenedor">
<divid="izquierda">Loremipsum dolor sitamet, consectetuer
adipiscing elit. Duismolestieturpis vitae ante.</div>
<divid="derecha">Lorem ipsum dolor sit amet, consectetuer
adipiscing elit. Nullabibendum mi non lacus.</div>
```

```
<divstyle="clear: both"></div>
</div>
```

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

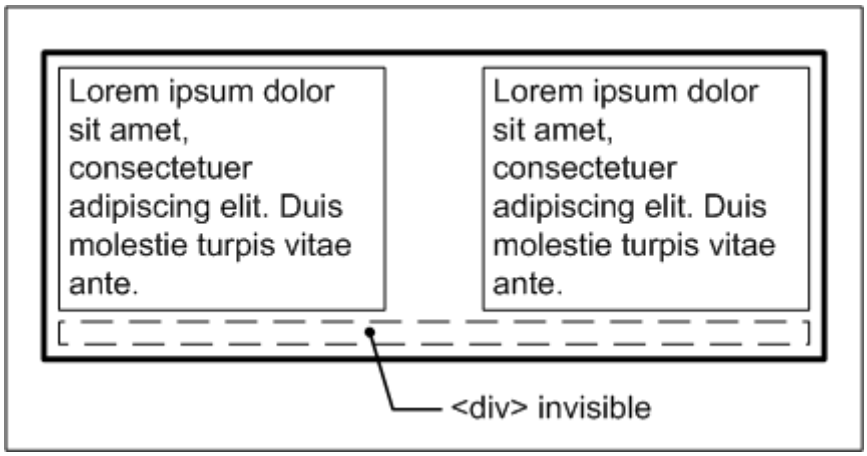



Figura 1.2. Solución tradicional al problema de los elementos posicionados de forma flotante

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 y se publicó por primera vez en el artículo *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.

**Ahora veamos el mismo trabajo pero con INLINE-BLOCK.**

pero sin el div clear.. jejeje.

```
<body>
<div class="content">
    <div class="cajasInline">Caja 1</div>
    <div class="cajasInline">Caja 2</div>
    <div class="cajasInline">Caja 3</div>
</div>
</body>
</html>
```

Se define una clase “cajasInline” para cada div dentro del “content”.

Luego creamos la clase de la siguiente manera:

```
.cajasInline{
    display: inline-block;
    width:200px;
    margin: 5px 8px;
    background-color: #F90;
    text-align:center;
    color:#fff;
}
```

utilizando la propiedad “**display**” con el valor “**inline-block**“, podemos realizar la misma acción que con el float:left.  Pero es este caso no es necesario definir un div class “**Clear**“.

El resultado es el siguiente:


No hay desbordamiento…

Como pueden notar el trabajo es menor, nos ahorramos una clase y div menos, esto en proyecto más grande nos va a ayudar a tener menos código.

### 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. Algunas técnicas son poco elegantes por lo que 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 *Equalheight 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.

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

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

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

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

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

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

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

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

### 1.4. Sombras

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

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

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

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

La sintaxis completa de la propiedad `box-shadow` es muy compleja y se define en el borrador de trabajo del módulo de fondos y bordes de CSS3. 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;
}
```

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

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

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

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

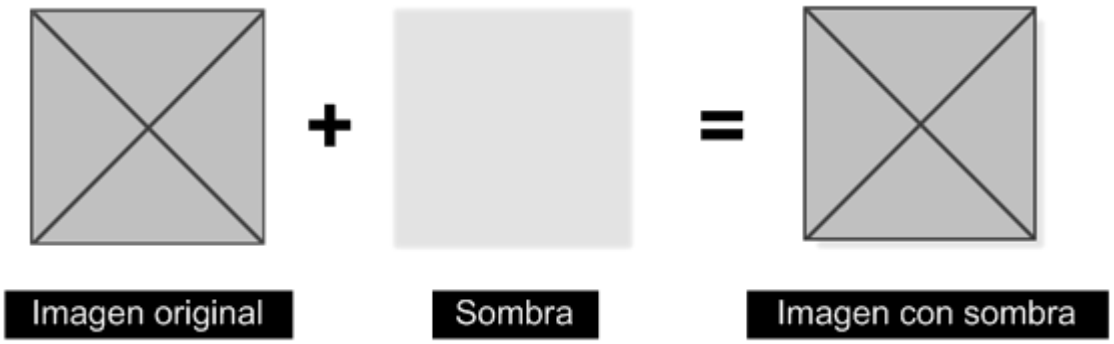



Figura 1.4. Aplicando una sombra a una imagen

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

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

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

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

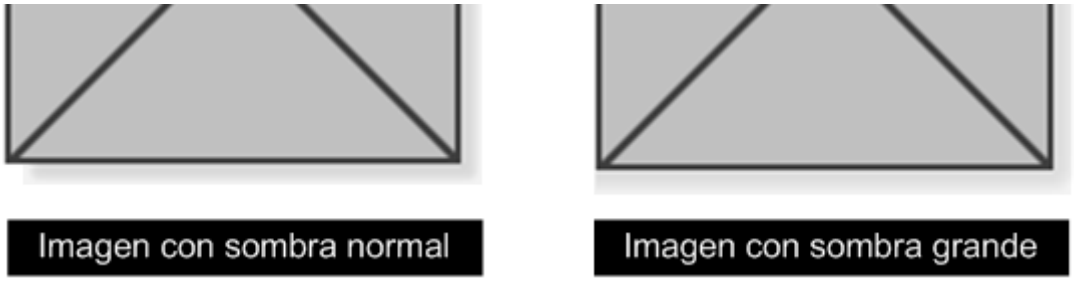



Figura 1.5. Las imágenes de sombra muy grande producen bordes más feos

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

## 1.5. Transparencias

El estándar de CSS 2.1 no incluye ninguna propiedad para controlar la opacidad de los elementos de la página. Sin embargo, la futura versión CSS 3 incluye una propiedad llamada `opacity`, definida en el módulo de colores de CSS 3 y que permite incluir transparencias en el diseño de la página.

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

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

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

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

#primero{
background-color: blue;
}

#segundo{
background-color: red;
}

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

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

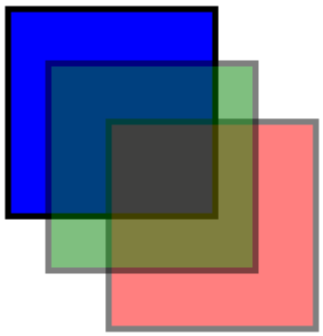


Figura 1.6. Creando elementos semitransparentes con la propiedad opacity

## 1.6. Uso de diferentes tipografías.

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:

- Sistemasoperativostipo 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.
- Sistemas operativos tipo Mac: `Arial`, `Helvetica`, `Verdana`, `Monaco`, `Times`.
- Sitemas operativos tipo Linux: incluyen decenas de fuentes, muchas de ellas versiones libres de las fuentes comerciales. También es posible instalar las fuentes más populares de Windows mediante el paquete Core Font.

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*.

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 Fontsfor @font-faceembedding*. Para ver el efecto de la directiva `@font-face`, debes acceder a esa página utilizando un navegador como Safari.

Es también muy recomendable www.**dafont**.com/es/

Y por supuesto https://www.google.com/fonts

## 1.7. Texto

### 1.7.1. 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 Techniquesfor Web Content AccessibilityGuidelines 1.0* 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 (*Techniquesfor WCAG 2.0*) 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`.

### 1.7.2. Efectos gráficos

#### *1.7.2.1. Texto con sombra*

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

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

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

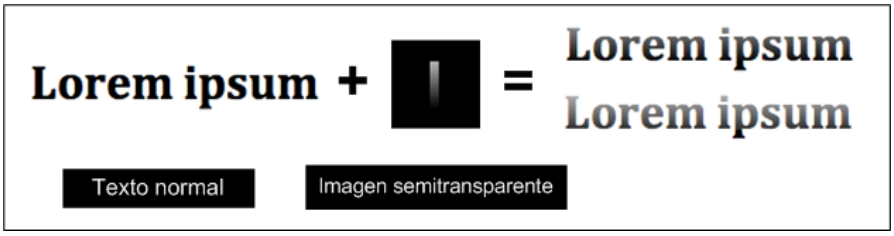
La sintaxis de la propiedad `text-shadow` obliga a indicar dos medidas y permite establecer de forma opcional una tercera medida y un color. Las dos medidas obligatorias son respectivamente el desplazamiento horizontal y vertical de la sombra respecto del texto. La tercera medida opcional indica lo nítido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

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

#### 1.7.2.2. Texto con relleno gradiente o degradado

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

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



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

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

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

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

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

```
h1 {
position: relative;
}

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

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

Utilizando este mismo truco pero con otras imágenes, se pueden conseguir efectos tan espectaculares como los que se pueden ver en los ejemplos del artículo *CSS Gradient Text*.

### 1.8. Sprites y Rollovers.

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: Whatthe 80/20 Rule TellsUsaboutReducing HTTP Requests* 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: ImageSlicing's Kiss of Death* y desde entonces se conoce con el nombre de *sprites CSS*.

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

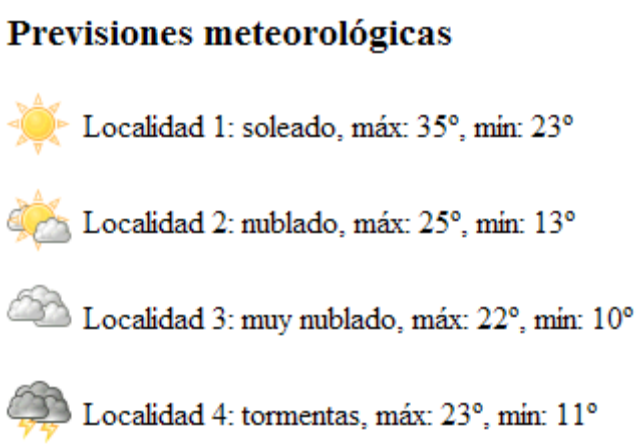



Figura 1.8. Aspecto de la previsión meteorológica mostrada con iconos

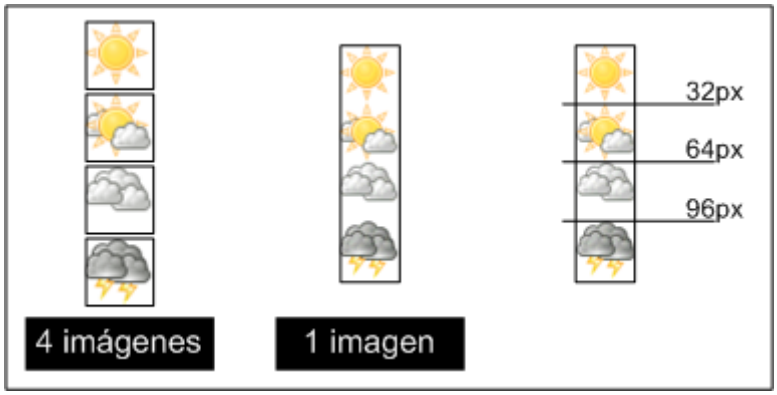
La solución tradicional para crear la página anterior consiste en utilizar cuatro elementos `<img>` en el código HTML y disponer de cuatro imágenes correspondientes a los cuatro iconos:

```
<h3>Previsiones meteorológicas</h3>
<p id="localidad1"><img src="imagenes/sol.png" /> Localidad 1:
soleado, máx: 35°, mín: 23°</p>
<p id="localidad2"><img src="imagenes/sol_nubes.png" /> Localidad 2:
nublado, máx: 25°, mín: 13°</p>
<p id="localidad3"><img src="imagenes/nubes.png" /> Localidad 3: muy
nublado, máx: 22°, mín: 10°</p>
<p id="localidad4"><img src="imagenes/tormentas.png" /> 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 *spritesCSS* 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 4=`128px`:




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: 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>
```

La clave de la técnica de los *spritesCSS* 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: 00;
}
#localidad2{
background-position: 0 -32px;
}
#localidad3{
background-position: 0 -64px;
}
#localidad4{
background-position: 0 -96px;
}
```

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

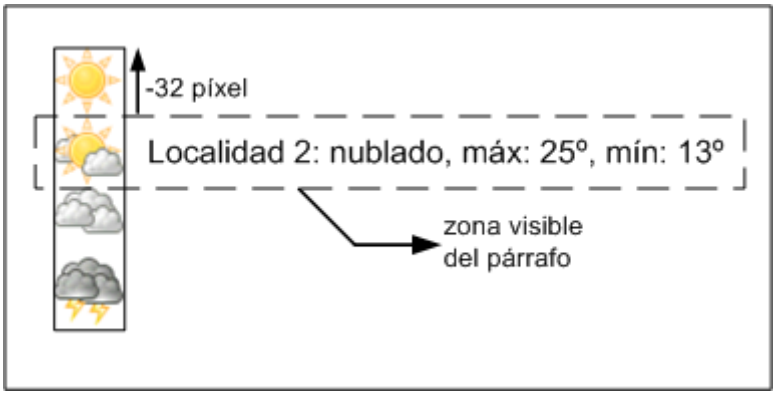



Figura 1.10. Funcionamiento de la técnica de los sprites CSS

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

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

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

La solución original utilizaba cuatro imágenes y realizaba cuatro peticiones al servidor. La solución basada en *spritesCSS* sólo realiza una conexión para descargar una sola imagen. Además, los iconos del proyecto Tango 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:

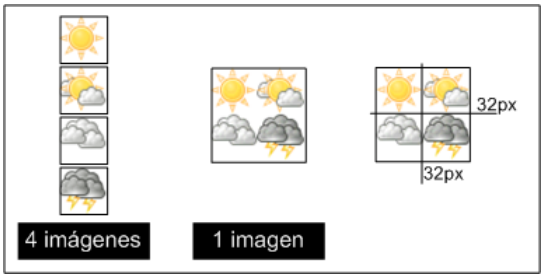


Figura 1.11. Sprite complejo que dispone las imágenes de forma vertical y horizontal

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: 00;
}
#localidad2{
background-position: -32px0;
}
#localidad3{
background-position: 0 -32px;
}
#localidad4{
background-position: -32px -32px;
}
```

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

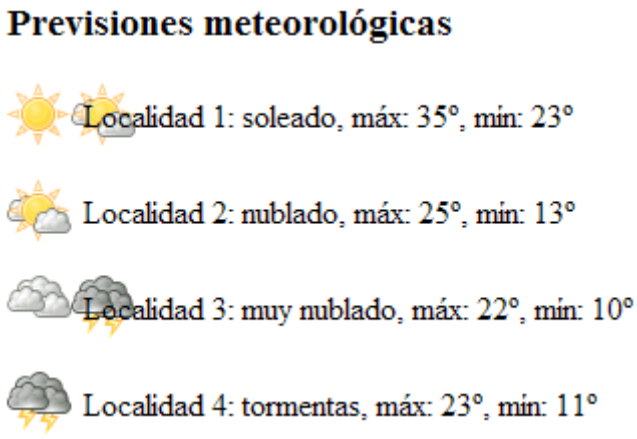



Figura 1.12. Errores producidos por utilizar un sprite complejo

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

```
<h3>Previsiones meteorológicas</h3>
<pid="localidad1"><imgsrc="imagenes/pixel.gif" /> Localidad 1:
soleado, máx: 35°, mín: 23°</p>
<pid="localidad2"><imgsrc="imagenes/pixel.gif" /> Localidad 2:
nublado, máx: 25°, mín: 13°</p>
<pid="localidad3"><imgsrc="imagenes/pixel.gif" /> Localidad 3: muy
nublado, máx: 22°, mín: 10°</p>
```

```
<pid="localidad4"><imgsrc="imagenes/pixel.gif" /> Localidad 4:
tormentas, máx: 23°, mín: 11°</p>
```

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

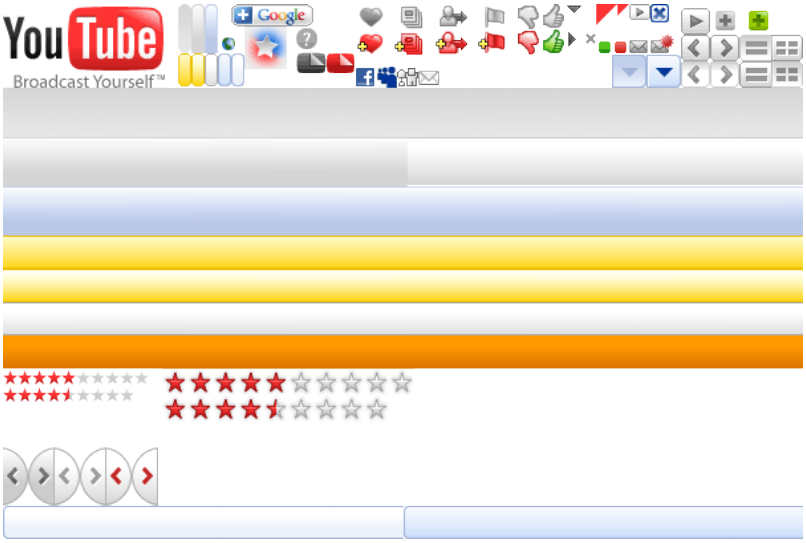
```
#localidad1img, #localidad2img, #localidad3img, #localidad4img{
height: 32px;
width: 32px;
background-image: url("imagenes/sprite2.png");
background-repeat: no-repeat;
vertical-align: middle;
}

#localidad1img{
background-position: 00;
}
#localidad2img{
background-position: -32px0;
}
#localidad3img{
background-position: 0 -32px;
}
#localidad4img{
background-position: -32px -32px;
}
```

Utilizar *spritesCSS* 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 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:




Sprite utilizado por el sitio web de YouTube

Los principales inconvenientes de los *spritesCSS* 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 SpriteGenerator* que generan el *sprite* a partir de un archivo comprimido en formato ZIP con todas las imágenes individuales.