

Manual de CSS, hojas de estilo

Tutorial completo y práctico sobre hojas de estilo en cascada (CSS). Aprende a utilizar esta tecnología que te ayudará a crear páginas más atractivas y precisas. El curso contiene la descripción, uso, sintaxis, y lista de atributos para crear estilos



Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

Miguel Angel Alvarez
Director de DesarrolloWeb.com y EscuelaIT
<http://www.desarrolloweb.com>
(27 capítulos)

Federico Elgarte
<http://www.cssboulevar.com.ar/>
(1 capítulo)

Fernando Campaña
Programación - Multimedia
<http://www.rakidwam.com.ar/>
(2 capítulos)

Serviweb
Diseño web Murcia
<http://www.serviweb.es/>
(1 capítulo)

Leonardo A. Correa
<http://www.webnova.com.ar>
(1 capítulo)

José Juan Corpas Martos
<http://www.recursosflash.es>
(2 capítulos)

Manu Gutierrez
<http://www.tufuncion.com>
(1 capítulo)

OldMith
Desarrollador Web. jQuery. Responsive Design. Wordpress. Friki por naturaleza.
<http://www.lacasadelmago.es>
(3 capítulos)

Daniel Martínez
Diseñador gráfico convertido a web
<http://www.coolvillage.es/>
(1 capítulo)

Parte 1:

Qué es CSS

Comenzamos presentando las Hojas de Estilo en Cascada y explicando de qué manera ayudan a los diseñadores de páginas web.

1.1.- Introducción a las CSS

Una breve introducción a CSS, para las personas que no saben nada sobre la tecnología el por qué de su creación y cómo ayuda a hacer páginas web mejores.

Entramos en materia con los antecedentes de CSS, las razones por las que se han desarrollado las **hojas de estilo en cascada**, y los objetivos que trata de cumplir.

El lenguaje HTML está limitado a la hora de aplicarle forma a un documento. Esto es así porque fué concebido para otros usos (científicos sobretodo), distinto a los actuales, mucho más amplios.

Para solucionar estos problemas los diseñadores han utilizado técnicas tales como la utilización de tablas imágenes transparentes para ajustarlas, utilización de etiquetas que no son estándares del HTML y otras. Estas "trampas" han causado a menudo problemas en las páginas a la hora de su visualización en distintas plataformas.

Además, los diseñadores se han visto frustrados por la dificultad con la que, aun utilizando estos trucos, se encontraban a la hora de maquetar las páginas, ya que muchos de ellos venían maquetando páginas sobre el papel, donde el control sobre la forma del documento es absoluto.

Finalmente, otro antecedente que ha hecho necesario el desarrollo de esta tecnología consiste en que las páginas web tienen mezclado en su código HTML el contenido del documento con las etiquetas necesarias para darle forma. Esto tiene sus inconvenientes ya que la lectura del código HTML se hace pesada y difícil a la hora de buscar errores o depurar las páginas. Aunque, desde el punto de vista de la riqueza de la información y la utilidad de las páginas a la hora de almacenar su contenido, es un gran problema que estos textos estén mezclados con etiquetas incrustadas para dar forma a estos: se degrada su utilidad.

En estas páginas de CSS pretendemos dar a conocer la tecnología con un enfoque práctico para que en pocos capítulos podáis usar las CSS de una manera depurada, reflejando toda nuestra experiencia en su uso. No pretendemos explorar todos los aspectos de la tecnología ya que para realizar esto necesitaríamos un la extensión de un libro entero.

Referencia: Este manual trata los aspectos más teóricos de las hojas en cascada. En DesarrolloWeb.com también podemos encontrar otro manual con unos [talleres prácticos de aplicación de las CSS](#).

A lo largo del [Manual de CSS](#) veremos diferentes estados de las Hojas de Estilo en Cascada, pues han ido evolucionando con el paso de los años. En este manual se estudiarán principalmente las especificaciones de CSS 1 y CSS 2 y dedicaremos un texto diferente al estudio de la más moderna especificación del lenguaje en estos momentos, su tercera especificación, en el [Manual de CSS 3](#).

Además, para las personas que lo deseen, hemos realizado diversos videotutoriales que serán especialmente interesantes para las personas que quieran aprender CSS de una manera práctica y visual. Está todo en el [Videotutorial de CSS](#).

Artículo por Miguel Angel Alvarez

1.2.- Características y ventajas de las CSS

Conoce las principales características del lenguaje, su potencia y sus posibilidades.

El modo de funcionamiento de las CSS consiste en definir, mediante una sintaxis especial, la forma de presentación que le aplicaremos a:

- Un web entero, de modo que se puede definir la forma de todo el web de una sola vez.
- Un documento HTML o página, se puede definir la forma, en un pequeño trozo de código en la cabecera, a toda la página.
- Una porción del documento, aplicando estilos visibles en un trozo de la página.
- Una etiqueta en concreto, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta. Esto es muy importante ya que ofrece potencia en nuestra programación. Podemos definir, por ejemplo, varios tipos de párrafos: en rojo, en azul, con márgenes, sin ellos...

La potencia de la tecnología salta a la vista. Pero no solo se queda aquí, ya que además esta sintaxis CSS permite aplicar al documento formato de modo mucho más exacto. Si antes el HTML se nos quedaba corto para maquetar las páginas y teníamos que utilizar trucos para conseguir nuestros efectos, ahora tenemos muchas más herramientas que nos permiten definir esta forma:

- Podemos definir la distancia entre líneas del documento.
- Se puede aplicar indentado a las primeras líneas del párrafo.
- Podemos colocar elementos en la página con mayor precisión, y sin lugar a errores.
- Y mucho más, como definir la visibilidad de los elementos, márgenes, subrayados, tachados...

Y seguimos mostrando ventajas, ya que si con el HTML tan sólo podíamos definir atributos en las páginas con píxeles y porcentajes, ahora podemos definir utilizando muchas más unidades como:

- Pixels (px) y porcentaje (%), como antes.
- Pulgadas (in)
- Puntos (pt)
- Centímetros (cm)

1.2.1.- Navegadores que lo soportan

Esta tecnología es bastante nueva, por lo que no todos los navegadores la soportan. En concreto, sólo los navegadores de Netscape versiones de la 4 en adelante y de Microsoft a partir de la versión 3 son capaces de comprender los estilos en sintaxis CSS. Además cabe destacar que no todos los navegadores implementan las mismas funciones de hojas de estilos, por ejemplo, Microsoft Internet Explorer 3 no soporta todo lo relativo a capas.

Esto quiere decir que debemos de usar esta tecnología con cuidado, ya que muchos usuarios no podrán ver los formatos

que apliquemos a las páginas con CSS. Así pues, utilizad las hojas de estilos cuando estas no vayan a suponer un problema.

Artículo por Miguel Angel Alvarez

1.3.- Arquitectura CSS: problemas

Proponemos contemplar CSS como si fuera un lenguaje de programación, analizando algunas de las inofensivas costumbres, que resultan no serlo tanto.

Para muchos desarrolladores web, ser bueno en CSS significa que puedes echar un vistazo visual a un contenido y reproducirlo perfectamente en código. No usas tablas, y te enorgulleces y usas cuantas más imágenes mejor. Si eres realmente bueno, usas las más avanzadas técnicas como “media queries”, transiciones y transformaciones. Es cierto que todo esto es usado por los buenos desarrolladores CSS, pero hay una parte completamente separada del CSS que raramente es mencionada cuando se trata de calificar las habilidades de uno mismo.



Normalmente no nos descuidamos de esa manera con otros lenguajes. Un desarrollador de Rails (RubyOnRails) no es considerado bueno solo porque su código respete la especificación. Es lo mínimo. Por supuesto que debe funcionar respetando la especificación; la pregunta es: ¿es el código legible? ¿Es fácil de cambiar o extender? ¿Es una parte desacoplada de otras partes de la aplicación? ¿Será escalable? Estas preguntas surgen de forma natural cuando estamos observando la base del código, y con CSS no debería ser diferente. Hoy en día las aplicaciones web son más grandes que nunca, y un pobre pensamiento sobre lo que significa la arquitectura CSS puede lisiar el desarrollo de un proyecto. Es hora de evaluar CSS de la misma forma con que evaluamos las otras partes de la aplicación. No puede ser algo añadido en el último momento o resuelto como meramente un problema del “diseñador”.

1.3.1.- Las Metas de la Arquitectura CSS

En la comunidad CSS es muy difícil lograr un consenso general sobre las mejores prácticas a seguir. Juzgando objetivamente los comentarios en [Hacker News](#) con el lanzamiento de [CSS Lint](#) queda demostrado que mucha gente está en desacuerdo incluso en los aspectos más básicos de CSS.

Así que, en vez de mostrarte un argumento propio sobre las mejores prácticas, creo que deberíamos empezar por definir nuestras metas. Si podemos ponernos de acuerdo en algunas metas, podemos empezar a desechar algunos conceptos de CSS, no porque rompan nuestras preconcebidas nociones sobre lo que es bueno, sino porque obstaculizan nuestro proceso de desarrollo.

Creo que las metas de la buena arquitectura CSS no deberían ser diferentes de todas las buenas metas del desarrollo de software. Quiero que mi CSS sea predecible, reutilizable, estable y escalable.

1.3.2.- Predecible

CSS predecible significa que tus reglas se comportan como tú esperarías. Cuando añades o actualizas una regla, no debería afectar a las partes de tu sitio web en las que no hay intención de que afectara. En los sitios pequeños eso raramente ocurre, pero en los sitios grandes con decenas o centenares de páginas, el que el CSS sea predecible es una obligación.

1.3.3.- Reutilizable

Las reglas CSS deberían ser abstractas y estar suficientemente desacopladas a la hora de construir rápidamente nuevos

componentes en partes ya establecidas sin tener que recodificar configuraciones sobre problemas que ya has solventado.

1.3.4.- Estable

Cuando nuevos componentes y capacidades necesitan ser añadidas, actualizadas o reiniciadas en tu sitio, hacer eso no debería requerir modificar demasiado el CSS existente. Añadir un componente X a la página no debería, por su mera presencia, romper el componente Y.

1.3.5.- Escalable

Cuando el sitio crece y se vuelve más complejo normalmente requiere mayor mantenimiento por parte de los desarrolladores. CSS escalable significa que puede ser fácilmente administrado por una persona o por un equipo de personas. También significa que la arquitectura CSS de tu sitio es fácilmente accesible sin requerir una enorme curva de aprendizaje. Solo porque seas el único desarrollador que toque el CSS hoy no significa que siempre vaya a ser así.

1.3.6.- Malas Prácticas Comunes

Antes de que echemos un vistazo a las maneras de alcanzar las metas de una buena arquitectura CSS en otro artículo especialmente dedicado a ello, pienso que puede ser útil ojear las prácticas comunes que están en medio del camino. Con frecuencia, solo a través de repetidos errores comenzamos a abrazar una ruta alternativa.

Los siguientes ejemplos son todos generalizaciones de código que he escrito, y, aunque el código es técnicamente válido, cada uno de esos trozos de código me ha llevado al desastre o, al menos, a un intenso dolor de cabeza. A pesar de mis mejores intenciones y de la promesa de que esa vez sería diferente, esos “snippets” me metieron en problemas.

1.3.7.- Modificando los Componentes Basándonos en Quiénes son Sus Padres

Cuando nos enfrentamos con esta situación todos los nuevos desarrolladores CSS (e incluso algunos experimentados) lo atendemos de la misma manera. Te imaginas un único elemento padre y creas una nueva regla para controlarlo.

```
.widget {
background: yellow;
border: 1px solid black;
color: black;
width: 50%;
}

#sidebar .widget {
width: 200px;
}

body.homepage .widget {
background: white;
}
```

Al principio podría parecer un código medianamente inofensivo, pero vamos a examinarlo basándonos en las metas que establecimos antes.

Primero, el *widget* del ejemplo no es predecible. Un desarrollador que haya hecho muchos de estos widgets esperará que tengan un cierto aspecto, esté colocado en el *sidebar* o en la página principal. Con el código actual, el aspecto del widget variará y no será siempre exactamente el mismo.

Tampoco es muy reutilizable o escalable. ¿Qué ocurre cuando deseemos que el aspecto que tiene en la homepage lo tenga también en otra página? Habrá que añadir más reglas.

Por último, no es demasiado estable porque si el widget es rediseñado habría que estar actualizándolo en muchos sitios del CSS, y como dijimos antes, es difícil que esta rara configuración que de momento funciona le parezca acertada al próximo que la mire.

Imagina que este tipo de código fuera hecho en cualquier otro lenguaje. Esencialmente estás haciendo una definición de

clase, y en otra parte del código estás alcanzando esa definición de clase cambiándola por un particular uso local. Esto directamente viola el principio abierto/cerrado de desarrollo software:

Las entidades Software (clases, módulos, funciones, etc.) deberían ser abiertas para extensión, pero cerradas para modificación.

1.3.8.- Selectores Demasiado Complicados

Ocasionalmente un artículo se mueve por Internet mostrando el poder de los selectores CSS y proclamando que puedes estilizar un sitio por completo sin usar clases ni IDs.

Aunque es técnicamente cierto, cuanto más desarrollo con CSS, más me alejo de los selectores complejos. Cuanto más se complica un selector, más se acopla al HTML. Aunque mantengas limpio y claro el código HTML, los selectores complejos ensucian tu CSS.

```
#main-nav ul li ul li div { }
#content article h1:first-child { }
#sidebar > div > h3 + p { }
```

Estos ejemplos que vemos tienen sentido lógico. El primero probablemente está dando estilo a un menú 'dropdown', el segundo indica que el contenido de cabecera principal del artículo tendría que mostrarse de forma distinta a la de los restantes elementos h1, y el último ejemplo está añadiendo un poco de espacio extra para el primer párrafo en las secciones del sidebar.

Si este HTML no fuera nunca a cambiar, se podrían argumentar sus méritos, pero...¿cómo de realista es asumir que el HTML nunca cambiará? Los selectores demasiado complicados pueden ser imponentes, pero raramente nos ayudan en nuestras metas para una buena arquitectura CSS.

Estos ejemplos no son reutilizables en absoluto. El selector está señalando a un punto muy concreto del maquetado, ¿cómo podría otro componente con distinta estructura HTML reutilizar ese estilo? Tomando el primer selector (el del menú 'dropdown') como ejemplo, ¿qué pasa si un 'dropdown' similar fuera necesitado en una página distinta que no estuviera dentro del elemento #main-nav? Tendrías que recrear el estilo por completo.

Estos selectores son también muy impredecibles si el HTML necesita cambiar. Imagina que un desarrollador quisiera pasar el div en el tercer ejemplo a una etiqueta 'section' de HTML5. La regla entera se rompería.

Finalmente, puesto que estos selectores solo trabajan cuando el HTML permanece constante, no son por definición ni escalables ni estables.

En muchas aplicaciones tienes que fabricar compensaciones y concesiones en cuanto a código. La fragilidad de los selectores complejos raramente es peor que el precio de poder decir con honestidad que tu HTML es "limpio".

1.3.9.- Nombres de Clase Demasiado Genéricos

Cuando creamos componentes de diseño reutilizables es muy común sumergir los sub-elementos del componente dentro del nombre de clase de este. Por ejemplo:

```
<div class="widget">
<h3 class="title">...</h3>
<div class="contents">
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
In condimentum justo et est dapibus sit amet euismod ligula ornare.
Vivamus elementum accumsan dignissim.
<button class="action">Click Me!</button>
</div>
</div>
.widget {}
.widget .title {}
.widget .contents {}
.widget .action {}
```

La idea es que los sub-elementos de clase .title, .contents, y .action puedan ser estilizados sin tener que preocuparse sobre cómo puedan desparramarse sobre otros elementos estilizados con las mismas clases. Eso es cierto, pero no

previene contra el estilizado de clases con aquellos mismos nombres que se desparrramen hacia el propio componente.

En un gran proyecto es muy probable que un nombre de clase como `.title` sea usado en otro contexto. Si eso ocurriera, el título del widget parecería de pronto muy difícil de relacionar concretamente con el estilo que se pretendía.

Los nombres de clase demasiado genéricos conducen a un CSS muy impredecible.

1.3.10.- Hacer que una Regla Haga Demasiado

A veces creas un contenido que necesita estar a 20 píxeles de distancia del límite superior y de la izquierda partiendo de la esquina de su correspondiente sección:

```
.widget {  
position: absolute;  
top: 20px;  
left: 20px;  
background-color: red;  
font-size: 1.5em;  
text-transform: uppercase;  
}
```

Después necesitas usar exactamente ese mismo contenido para ponerlo en un lugar diferente. El CSS de arriba no funciona porque no es reutilizable en diferentes contextos.

El problema está en que le estás pidiendo demasiado a tu selector. Estás definiendo el diseño así como la posición en la misma regla. El diseño es reutilizable pero la posición no. Y puesto que se usan juntas la regla entera está comprometida.

Aunque esto pueda parecer inofensivo al comienzo, con frecuencia conduce al 'copy/paste' por parte de los desarrolladores CSS menos eruditos. Si un nuevo equipo de miembros quiere poner un nuevo contenido como por ejemplo un `.infobox`, probablemente comiencen intentando usar esa clase. Pero si no funciona porque la posición de ese nuevo `infobox` es errónea, ¿qué es lo más probable que hagan? Mi experiencia me dice que la mayoría de nuevos desarrolladores no romperán la regla en reglas reutilizables. En vez de eso ellos simplemente copiarán y pegarán las líneas de código necesarias para su particular nuevo selector.

1.3.11.- La Causa

Todas estas malas prácticas comparten una similitud: no son una carga para el CSS.

Suena raro, ¿eh? Después de todo...¿no sería lo ideal que tuviéramos una hoja de estilos con poca carga? ¿No es lo que queremos?

La respuesta sencilla a esta pregunta es "sí", pero, como es habitual, las cosas no son tan sencillas. Separar el contenido de la presentación es una buena idea, pero solo porque tu CSS esté separado de tu HTML no quiere decir que tu contenido deba estar separado de tu presentación. Dicho de otra manera, poner barreras al código HTML no ayuda a cumplir la meta si tu CSS requiere un íntimo conocimiento de tu estructura HTML a la hora de trabajar.

Además, HTML es raramente solo contenido; es casi siempre estructura también. Y con frecuencia esa estructura consiste en elementos contenedores sin más propósito que permitir al CSS aislar un cierto grupo de elementos. Incluso sin clases cuyo propósito único sea el 'embellecimiento' es difícil separar la presentación estética que subyace en el propio código. ¿Es necesario mezclar por lo tanto la presentación estética con el contenido?

Yo creo que dado el actual estado del HTML y del CSS es necesario y razonable hacer que el HTML y el CSS trabajen juntos en una única capa. La capa de contenido puede ser abstraída vía plantilla o algo parecido.

1.3.12.- La Solución

Si tu HTML y tu CSS van a trabajar juntos para formar la capa de presentación de una aplicación web, necesitan hacerlo de forma que promueva todos los principios de una buena arquitectura CSS.

La mejor aproximación que he encontrado es que para cada CSS, su HTML sea tan pequeño como sea posible. El CSS

debería definir cómo un conjunto de elementos visuales deben mostrarse y (en orden de minimizar el acoplamiento con el HTML) esos elementos deberían aparecer tal y como son definidos aparezcan donde aparezcan en el HTML. Si un cierto contenido necesita ser mostrado diferente en un diferente escenario, debería ser nombrado de manera diferente y es la responsabilidad del HTML llamarlo así.

Como ejemplo, el CSS podría definir un componente botón vía clase `.button`. Si el HTML quiere que un elemento particular se asemeje a un botón, debería usar esa clase. Si hay una situación en la que el botón necesita mostrarse diferente (quizá más grande, con anchura al 100%), entonces el CSS necesita definir ese aspecto en una nueva clase, y el HTML puede incluir esa nueva clase para usar el nuevo aspecto.

El CSS define cómo se muestran tus componentes, y el HTML les asigna el aspecto visual. Cuanto menos necesite saber el CSS sobre la estructura HTML, mejor.

Un enorme beneficio de declarar exactamente lo que quieres en el HTML es permitir a otros desarrolladores echar un vistazo a la maquetación y saber exactamente qué elemento están buscando. La idea es intentarlo. Sin esta práctica es imposible saber si el aspecto de un elemento es intencional o accidental, y eso conlleva confusión para el equipo.

Una queja común al poner un montón de clases en el maquetado es el esfuerzo extra que requiere hacerlo. Una regla CSS simple podría dirigir miles de instancias de un particular componente. ¿Es realmente peor escribir esas clases miles de veces sólo para tenerlas explícitamente declaradas en el maquetado?

Aunque la idea está clara, puede despistar. Para usar un selector parental en CSS no tienes que preocuparte sobre las mil clases que imagino que ahora piensas que tendrías que escribir a mano, hay obviamente otras alternativas. Viendo los niveles de abstracción en "Rails" o en otros *frameworks* nos hace fijarnos en cómo podríamos declarar explícitamente el HTML sin tener que escribir las mismas clases una y otra vez.

Pero eso lo veremos ya en el siguiente artículo, en el que plantearemos la arquitectura CSS del lado de las soluciones.

[Philip Walton](#)

Artículo por [OldMith](#)

1.4.- Arquitectura CSS - Soluciones

Después de ver en el artículo "Arquitectura CSS - Problemas" los problemas de código y las consecuencias que podemos tener por tenerlos, es hora de darte algunos consejos para mejorar.

Aunque no es amplia, mi experiencia me ha enseñado que apegarse a estos principios te ayudará a lograr tus metas en una buena arquitectura CSS.



1.4.1.- Sé intencional

La mejor manera de asegurarte de que tus selectores no estilizan elementos que no quieres estilizar es no darles la oportunidad. Un selector como `#main-nav ul li ul li div` podría muy fácilmente aplicarse a otros elementos con un par de cambios. Un estilo como `.subnav`, por otro lado, tendrá la garantía de que no va a haber oportunidad de accidente aplicándose a un elemento indeseado. Aplicando clases directamente a los elementos que quieres estilizar es la mejor manera de mantener tu CSS predecible.

```
/* Granada */
#main-nav ul li ul { }

/* Rifle de Francotirador */
.subnav { }
```

Dados esos dos ejemplos, piensa que el primero es como una granada y el segundo como un rifle de francotirador. La granada podría hacer el trabajo bien, pero nunca sabes cuándo un civil inocente podría introducirse en el radio de la explosión.

1.4.2.- Separa tus asuntos

Ya he mencionado que una zona de contenidos bien organizada puede ayudar a aflojar el acoplamiento de la estructura HTML con el CSS. Por añadidura, tus componentes CSS deberían ser modulares por ellos mismos. Los componentes deberían saber cómo estilizar por ellos mismos y cómo hacer su trabajo bien, pero no deberían ser responsables de su distribución o posicionamiento, como tampoco deberían asumir cómo debieran ser colocados en relación con los elementos circundantes.

En general, los componentes CSS deberían definir el aspecto visual, no su distribución en la página, ni su posicionamiento. Ten cuidado cuando veas propiedades como "background", "color", y "font" junto a "position", "width", "height", y "margin".

La distribución y la posición deberían ir de la mano en una clase en un elemento contenedor separado (recuerda que separar efectivamente el contenido de la presentación con frecuencia es esencialmente separar el contenido de su contenedor).

1.4.3.- Personaliza el nombre de tus clases

Ya habíamos examinado por qué los selectores parentales no son 100% efectivos en la encapsulación y previniendo contaminación a través de los estilos. Una actitud mucho mejor es aplicar nombres de clases específicos a cada clase. Si un elemento pertenece a un componente visual, cada una de sus clases de sub-elementos debería usar como base para el nombre la base de nombre de clase del componente superior. Ejemplo:

```
/* Alto riesgo de contaminación a través del estilo */
.widget { }
.widget .title { }

/* Bajo riesgo de contaminación a través del estilo */
.widget { }
.widget-title { }
```

Especificar el nombre de clase hace que tus componentes se mantengan autocontenidos y modulares. Minimiza la probabilidad de que una clase ya existente entre en conflicto con la actual, y reduce la especificidad requerida al estilizar los elementos hijo.

1.4.4.- Extender componentes con clases modificadas

Cuando un componente ya existente necesita mostrarse visualmente ligeramente diferente en un cierto contexto, lo mejor es crear una clase modificada para extenderla:

```
/* Mal */
.widget { }
#sidebar .widget { }

/* Bien */
.widget { }
.widget-sidebar { }
```

Ya hemos visto los inconvenientes de modificar componentes basados en uno de sus elementos parentales, pero reitero: Una clase modificada puede ser usada donde sea. Modificar dependiendo de la localización hará que solo pueda usarse en ese sitio concreto. Las clases modificadas pueden ser reutilizadas todas las veces que necesites. Por último, las clases modificadas expresan la intención del desarrollador de ser claro en cuanto al código HTML. Las clases basadas en la localización, por otro lado, son completamente invisibles para el desarrollador que solo mira el HTML, incrementando mucho la probabilidad de que lo pase por alto.

1.4.5.- Organiza tu CSS en una estructura lógica

[Jonathan Snook](#), en su excelente libro [SMACSS](#), argumenta en favor de organizar tus reglas CSS en cuatro categorías separadas: base, distribución, módulos, y estado. La base consiste en resetear las reglas y en poner por defecto los elementos. La distribución es el posicionamiento genérico de los elementos, así como algún sistema de ayuda al respecto como por ejemplo los 'grids'. Los módulos son elementos visuales reutilizables y el estado se refiere a los distintos estilos que un elemento pueda tener activándose o desactivándose vía JavaScript.

En el sistema SMACSS los módulos (los cuales son equivalentes a lo que llamamos componentes) incluyen la vasta mayoría de todas las reglas CSS, y con frecuencia encontraré necesario romperlas para llevarlas a componentes más abstractos.

Los componentes son elementos visuales independientes. Los patrones, por otro lado, son bloques. No se soportan por sí mismos y raramente describen el aspecto visual, lo que se ve y se siente. En cambio son sencillos y repetibles patrones que pueden ser puestos juntos para formar un componente.

Veamos un ejemplo concreto, un componente que sea una caja modal de diálogo. La parte modal podría tener el sello del sitio como degradado en el fondo de la cabecera, y por ejemplo, una sombra caída alrededor, un botón en lo alto de la esquina derecha, y posicionamiento fijo, además de un centrado vertical y horizontal. Cada una de estas configuraciones podría ser usada repetidamente en cualquier sitio de la web, así que la idea sería no tener que ajustar el código cada vez que así fuera. Cada una de esas configuraciones son un patrón, y juntos componen el componente modal.

Normalmente no uso clases solo de patrones en el HTML a menos que tenga una buena razón. En cambio, uso un preprocesador para incluir los estilos de patrón en la definición del componente. Discutiré ésto detalladamente después.

1.4.6.- Usa clases para estilizar y solo para estilizar

Cualquiera que haya trabajado en un gran proyecto ha encontrado un elemento HTML con una clase cuyo propósito era completamente desconocido. Quiere quitarla, pero estás indeciso, no vayas a ser que tenga un propósito de cual no se te ha avisado. Como eso ocurre continuamente, con el paso del tiempo tu HTML se va rellenando con clases que en realidad no tienen propósito, y que los miembros del equipo tienen miedo de borrar.

El problema es que las clases normalmente tienen demasiadas responsabilidades en el desarrollo *front-end*. Estilizan elementos HTML, actúan como ganchos para el JavaScript *hooks*, se añaden al HTML como características de detección, se usan en tests automatizados, etc.

Es un problema. Cuando las clases son usadas en demasiadas partes de la aplicación, llegar a dar miedo quitarlas de tu HTML.

Sin embargo, con una convención establecida, este problema puede ser completamente evitado. Cuando ves una clase en el HTML, deberías ser capaz instantáneamente de saber su propósito. Mi recomendación es darle a todas las clases sin estilo un prefijo. Uso `.js-` para JavaScript y uso `.supports-` para las clases Modernizr. Todas las clases sin prefijo son para estilizar y solo para estilizar.

Esto hace que encontrar clases sin uso y eliminarlas del HTML sea tan fácil como buscarlas en la estructura de la hoja de estilos. Puedes incluso automatizar este proceso en JavaScript referenciando las clases del HTML con sus respectivas, mediante el objeto `document.styleSheets`. Las clases que no están en el `document.styleSheets` pueden ser eliminadas con seguridad.

En general, como una buena práctica el separar el contenido de la presentación, es también importante separar tu presentación de tu funcionalidad. Usando clases estilizadas como ganchos de Javascript profundiza en el acoplamiento entre CSS y JavaScript, de tal manera que hace imposible actualizar el enganche de ciertos elementos sin romper la funcionalidad.

1.4.7.- Nombra tus clases con una estructura lógica

En estos días, la mayoría de la gente escribe CSS usando guiones como separadores de palabras. Pero los guiones por sí solos no suelen ser normalmente suficientes para distinguir entre tipos de clases.

[Nicolas Gallagher](#) recientemente escribió sobre su [solución a este problema](#), la cual he adoptado (con ligeros cambios) con gran éxito. Para ilustrar la necesidad de una convención en cuanto a la nomenclatura consideremos lo siguiente:

```
/* Un componente */
.button-group { }

/* La modificación de un componente (modificando .button) */
.button-primary { }

/* Un sub-objeto del componente (independiente de .button) */
.button-icon { }

/* ¿Es ésto una clase de componente o de diseño? */
.header { }
```

Echando un vistazo a las clases de arriba es imposible saber qué tipo de regla aplicar. Esto no solo incrementa la confusión durante el desarrollo, también hace más difícil testar tu CSS y tu HTML de una manera automática. Una convención estructurada sobre la nomenclatura permitiría ojear un nombre de clase y saber exactamente qué relación tiene con otras clases y dónde debería aparecer en el HTML — fomentando una nomenclatura más sencilla y un testeo posible donde antes no lo era.

```
/* Reglas de patrón (usando marcadores Sass) */
%template-name
%template-name--modifier-name
%template-name__sub-object
%template-name__sub-object--modifier-name

/* Reglas de Componente */
.component-name
.component-name--modifier-name
.component-name__sub-object
.component-name__sub-object--modifier-name

/* Reglas de Diseño */
.l-layout-method
.grid

/* Reglas de Estado */
.is-state-type

/* Enganches No-Estilizados JavaScript */
.js-action-name
El primer ejemplo rehecho:
/* Un componente */
.button-group { }

/* Un componente modificado (modificando .button) */
.button--primary { }

/* Un sub-objeto de componen (independiente de .button) */
.button__icon { }

/* Una clase de diseño */
.l-header { }
```

1.4.8.- Herramientas

Mantener una efectiva y bien organizada arquitectura CSS puede ser muy difícil, especialmente en grandes equipos. Unas cuantas malas reglas aquí y allí pueden convertirse, cual bolas de nieve, en enredos inmanejables. Una vez que el CSS de tu aplicación ha entrado en la guerra del campo de la especificidad y en el triunfo de los "important", puede ser imposible para el siguiente poder analizarlo sin preferir en cambio comenzar de nuevo. La clave es evitar esos problemas desde el comienzo. Afortunadamente, hay herramientas que pueden controlar la arquitectura CSS de tu sitio de fácil manera.

1.4.9.- Preprocesadores

En los días que vivimos es imposible hablar sobre herramientas CSS sin mencionar los preprocesadores, y en este artículo no será diferente. Pero antes de que alabe su utilidad, os ofreceré algunas palabras de cautela.

Los preprocesadores te ayudan a escribir CSS más rápido, no mejor. En última instancia lo que se obtiene es CSS plano, y por lo tanto se aplican las mismas reglas que antes del preprocesador. Si un preprocesador te permite escribir tu CSS más rápido también te permite escribir tu CSS defectuoso más rápido, así que es importante comprender bien la arquitectura CSS antes de pensar que un procesador va a solventar tus problemas.

Muchas de las llamadas “capacidades” de los preprocesadores pueden actualmente ser muy perjudiciales para la arquitectura CSS. Las siguientes son algunas de las “capacidades” que yo evitaría a toda costa (y aunque estas ideas generales se aplican a todos los preprocesadores, estas guías deberían aplicarse específicamente en Sass):

- Nunca anides reglas por pura organización de código. Únicamente anida cuando el CSS procesado sea lo que quieres.
- Nunca uses un 'mixin' si no le vas a pasar un argumento. Los 'mixins' sin argumentos es mejor usarlos como 'templates', ya que pueden ser extendidos.
- Nunca uses @extend en un selector que no es una clase de hijo. No tiene sentido desde una perspectiva de diseño e hincha el CSS compilado.
- Nunca uses @extend para componentes UI en modificaciones de reglas de componente porque perderás la cadena de herencia.

Lo mejor de los preprocesadores son las funciones como [@extend](#) y [%placeholder](#). Ambas te permiten manejar la abstracción CSS fácilmente sin añadir hinchazón a tu CSS o sin añadir una enorme lista de clases base en tu HTML, que podría ser difícil de manejar.

@extend debería ser usado con precaución porque alguna vez querrás esas clases en tu HTML. Por ejemplo, cuando aprendes por primera vez sobre @extend podría ser tentador usarlo con todos tus clases modificadas de la siguiente manera:

```
.button {
/* Estilos de botón */
}

/* Mal */
.button--primary {
@extend .button;
/* Estilos de Modificación */
}
```

El problema de hacer eso es que pierdes la cadena de herencia con respecto al HTML. Ahora es muy difícil seleccionar todas las instancias del botón con JavaScript.

Por norma general nunca extendiendo componentes UI o ni nada que se le parezca. Eso es lo para lo que sirven los patrones ('templates') y es otra manera de distinguirlos de los componentes. Un patrón es algo a lo que nunca necesitarías apuntar desde la lógica de tu aplicación, y por consiguiente puede ser extendido con un preprocesador.

Aquí vemos ahora cómo se mostraría el ejemplo modal que vimos arriba:

```
.modal {
@extend %dialog;
@extend %drop-shadow;
@extend %statically-centered;
/* otros estilos */
}

.modal__close {
@extend %dialog__close;
/* otros estilos del botón */
}

.modal__header {
@extend %background-gradient;
/* otros estilos de la cabecera */
}
```

}

1.4.10.- CSS Lint

[Nicole Sullivan](#) y [Nicholas Zakas](#) crearon [CSS Lint](#) como una herramienta de calidad de código para ayudar a los desarrolladores a detectar malas prácticas en su CSS. Su sitio lo describe como:

CSS Lint señala los problemas de tu código CSS. Hace un chequeo básico de tu sintaxis, además de aplicar una serie de ajustes al código que parece mostrar signos de ineficiente o de problemática. Las reglas son todas opcionales, así que puedes editar u omitir sencillamente las que no te gusten.

Mientras que el set de reglas generales podría no ser perfecto para la mayoría de los proyectos, la mejor capacidad de CSS Lint es su habilidad para ser personalizado exactamente como tú quieres que sea. Eso significa que puedes elegir de entre todas las reglas que vienen por defecto y escoger las que quieras para poder escribirlas por tu cuenta.

Una herramienta como CSS Lint es esencial para cualquier gran proyecto a la hora de asegurar al menos una conformidad común en cuanto a consistencia y las convenciones de lo que se trata. Y como esboqué previamente, una de las grandes razones para asumir convenciones es que permiten a herramientas como CSS Lint identificar cualquier cosa que las rompa.

Basándonos en las convenciones que he propuesto a lo largo de estos dos artículos sobre Arquitectura CSS, vemos que llega a ser muy fácil escribir reglas que detecten las configuraciones que no queremos. Aquí aporto algunas sugerencias:

- No permitas los ID en tus selectores.
- No uses selectores de tipo no semántico (div, span) en ninguna regla.
- No uses más de dos combinadores en un selector.
- No permitas ningún nombre de clase que empiece por “js-”.
- Mantente alerta si frecuentemente usas diseño y posicionamiento usas en las reglas sin prefijos “!-” .
- Mantente alerta si una clase definida en un primer momento es luego redefinida como hijo de alguna otra clase.

Son obviamente solo sugerencias, pero son propuestas que te hacen pensar en cómo reforzar los estándares que quieres hacer cumplir en tus proyectos.

1.4.11.- HTML Inspector

Hace rato sugerí que sería fácil buscar las clases en tu HTML y enlazarlas con sus correspondientes en la hoja de estilos y dije que deberías tener cuidado si tu clase estaba definida en el HTML pero no en el CSS. He desarrollado una herramienta llamada HTML Inspector para hacer este proceso más sencillo.

HTML Inspector atraviesa tu HTML y (de forma parecida a CSS Lint) te permite escribir tus propias reglas que lanzarán errores y avisos cuando alguna convención se rompa. Actualmente uso estas normas:

- Mantente alerta si el mismo ID es usado más de una vez en una página.
- No uses ninguna clase que no haya sido mencionada en alguna hoja de estilos (como “js-”).
- Las clases modificadas no deberían ser usadas sin su clase de base.
- Las clases de sub-objetos no deberían ser usadas cuando ningún ancestro contiene a la clase base.
- Los elementos planos DIV o SPAN, sin clase adjuntada, no deberían ser usados en el HTML.

1.4.12.- Resumen

CSS no es solo diseño visual. No deseches las mejores prácticas de la programación solo porque estés escribiendo CSS. Conceptos como OOP, DRY, el principio abierto/cerrado, separación de asuntos, etc., se pueden aplicar a CSS.

Lo importante es que debes organizar el código, asegurándote de que juzgas tus métodos según lo que puedan ayudarte actualmente a desarrollar de una manera más sencilla y sostenible a largo plazo.

[Philip Walton](#)

Artículo por OldMith

Parte 2:

Distintas maneras de incluir estilos

Existen distintas formas de definir estilos en una página, a diversos niveles que van desde las más específicas, que permitirían definir estilos en un pequeño texto de una página, hasta las más generales, para definir estilos para toda una página o incluso un sitio web.

2.1.- Usos de las CSS I

Describimos las distintas aplicaciones de las Hojas de Estilo en cascada. En este capítulo veremos las más sencillas.

CSS sirve para definir el aspecto de las páginas web, eso ya debe haber quedado claro. No obstante, hay diferentes niveles a los que podemos aplicar los estilos y es algo que vamos a describir ahora.

Hemos denominado a este apartado los diferentes usos de las CSS y relata justamente eso, los distintos niveles a los que podemos usar las Hojas de Estilo, que van desde definir los estilos de manera específica, para un pequeño fragmento de una página, hasta los estilos para todo un sitio web completo. Todo esto pasando por diversos otros niveles que resultarán de mucha utilidad también en nuestro día a día como diseñadores.

Vamos por orden, describiendo los puntos desde el más específico al más general, de manera que también iremos aumentando la dificultad e importancia de los distintos usos. Hemos partido este capítulo en dos partes por su extensión y por haber varias formas distintas de aplicar estilos, aquí veremos las más sencillas y en el capítulo siguiente otras más complicadas pero más potentes.

Nota: CSS tiene una sintaxis propia. En este artículo ofreceremos diversos códigos de CSS, aunque no hemos explicado la sintaxis todavía.

A través de los próximos ejemplos veremos una pequeña introducción a la manera de escribir código CSS, pero lo explicaremos con detalle más adelante cuando tratemos la [sintaxis CSS](#).

2.1.1.- Pequeñas partes de la página

Para definir estilos en secciones reducidas de una página se utiliza la etiqueta ``. Con su atributo `style` indicamos en sintaxis CSS las características de estilos. Lo vemos con un ejemplo, pondremos un párrafo en el que determinadas palabras las vamos a visualizar en color verde.

```
<p>
Esto es un párrafo en varias palabras <SPAN style="color:green">en color verde</SPAN>. resulta muy fácil.
</p>
```

Que tiene como resultado:

Esto es un párrafo con varias palabras **en color verde**. resulta muy fácil.

2.1.2.- Estilo definido para una etiqueta

De este modo podemos hacer que toda una etiqueta muestre un estilo determinado. Por ejemplo, podemos definir un párrafo entero en color rojo y otro en color azul. Para ello utilizamos el atributo `style`, que es admitido por todas las etiquetas del HTML (siempre y cuando dispongamos de un navegador compatible con CSS).

```
<p style="color:#990000">
Esto es un párrafo de color rojo.
</p>
<p style="color:#000099">
Esto es un párrafo de color azul.
</p>
```

Que tiene como resultado:

Esto es un párrafo de color rojo.

Esto es un párrafo de color azul.

2.1.3.- Estilo definido en una parte de la página

Con la etiqueta `<DIV>` podemos definir secciones de una página y aplicarle estilos con el atributo `style`, es decir, podemos definir estilos de una vez a todo un bloque de la página.

```
<div style="color:#000099; font-weight:bold">
<h3>Estas etiquetas van en <i>azul y negrita</i></h3>
<p>
Seguimos dentro del DIV, luego permanecen los estilos
</p>
</div>
```

Que tiene como resultado:

Estas etiquetas van en azul y negrita

Seguimos dentro del DIV, luego permanecen los estilos

Hasta aquí hemos visto los usos de las CSS más específicos. Esta información continua en el próximo capítulo, en el que seguiremos viendo [otras formas más avanzadas de usar las CSS](#).

Nota: Para aprender de una manera visual y práctica los diferentes usos de las CSS recomendamos ver la [primera parte del videotutorial sobre las CSS](#).

Artículo por Miguel Angel Alvarez

2.2.- Usos de las CSS y II

Te describimos los usos más avanzados de las hojas de estilo en cascada. Para poder utilizarlas de la manera más potente.

2.2.1.- Estilo definido para toda una página

Podemos definir, en la cabecera del documento, estilos para que sean aplicados a toda la página. Es una manera muy cómoda de darle forma al documento y muy potente, ya que estos estilos serán seguidos en toda la página y nos ahorraremos así muchas etiquetas HTML que apliquen forma al documento. Además, si deseamos cambiar los estilos de la página lo haremos de una sola vez.

Este ejemplo es más complicado, puesto que se utiliza la sintaxis CSS de manera más avanzada. Pero no te preocupes puesto que con los ejemplos irás aprendiendo su uso y más tarde comentaremos la sintaxis en profundidad.

En el ejemplo vemos que se utiliza la etiqueta `<STYLE>` colocada en la cabecera de la página para definir los distintos estilos del documento.

A grandes rasgos, entre de `<STYLE>` y `</STYLE>`, se coloca el nombre de la etiqueta que queremos definir los estilos y entre llaves `{}` colocamos en sintaxis CSS las características de estilos.

```
<html>
<head>
<title>Ejemplo de estilos para toda una página</title>
<STYLE type="text/css">
<!--
H1 {text-decoration: underline; text-align:center}
P {font-Family:arial,verdana; color: white; background-color: black}
BODY {color:black;background-color: #cccccc; text-indent:1cm}
// -->
</STYLE>
</head>

<body>
<h1>Página con estilos</h1>
Bienvenidos...
<p>Siento ser tan hortera, pero esto es un ejemplo sin mucha importancia</p>
</body>
</html>
```

Como se puede apreciar en el código, hemos definido que la etiqueta `<H1>` se presentará

- Subrayado
- Centrada

También, por ejemplo, hemos definido que el cuerpo entero de la página (etiqueta `<BODY>`) se le apliquen los estilos siguientes:

- Color del texto negro
- Color del fondo grisáceo
- Margen lateral de 1 centímetro

Caber destacar que si aplicamos estilos a la etiqueta `<BODY>`, estos serán heredados por el resto de las etiquetas del documento. Esto es así siempre y cuando no se vuelvan a definir esos estilos en las siguientes etiquetas, en cuyo caso el estilo de la etiqueta más concreta será el que mande. Puede verse este detalle en la etiqueta `<P>`, que tiene definidos estilos que ya fueron definidos para `<BODY>`. Los estilos que se tienen en cuenta son los de la etiqueta `<P>`, que es más concreta.

Por último, ha de apreciarse los comentarios HTML que engloban toda la declaración de estilos: `<!--Declaración de estilos-->`. Estos comentarios se utilizan para que los navegadores antiguos, que no comprenden la sintaxis CSS, no incluyan ese texto en el cuerpo de la página. Si no se pusiera, los navegadores antiguos (por ejemplo Netscape 3) escribirían ese "feo código" en la página.

[Pulsa para ver el ejemplo anterior.](#)

Hemos preparado la misma página, pero con declaraciones de estilos distintas, para que comprobéis las diferencias en la forma del documento con sólo unos cambios en sus estilos. [Puedes verla pinchando aquí.](#)

2.2.2.- Estilo definido para todo un sitio web

Una de las características más potentes de la programación con hojas de estilos consiste en que, de una vez, podemos definir los estilos de todo un sitio web. Esto se consigue creando un archivo donde tan sólo colocamos las declaraciones de estilos de la página y enlazando todas las páginas del sitio con ese archivo. De este modo, todas las páginas comparten una misma declaración de estilos y, por tanto, si la cambiamos, cambiarán todas las páginas. Con las ventajas añadidas de que se ahorra en líneas de código HTML (lo que reduce el peso del documento) y se evita la molestia de definir una y otra vez los estilos con el HTML, tal como se comentó anteriormente.

Veamos ahora cómo el proceso para incluir estilos con un fichero externo.

1- Creamos el fichero con la declaración de estilos

Es un fichero de texto normal, que puede tener cualquier extensión, aunque le podemos asignar la extensión .css para aclararnos qué tipo de archivo es. El texto que debemos incluir debe ser escrito exclusivamente en sintaxis CSS, es decir, sería erróneo incluir código HTML en él: etiquetas y demás. Podemos ver un ejemplo a continuación.

```
P {
font-size : 12pt;
font-family : arial,Helvetica;
font-weight : normal;
}

H1 {
font-size : 36pt;
font-family : verdana,arial;
text-decoration : underline;
text-align : center;
background-color : Teal;
}

TD {
font-size : 10pt;
font-family : verdana,arial;
text-align : center;
background-color : 666666;
}

BODY {
background-color : #006600;
font-family : arial;
color : White;
}
```

2- Enlazamos la página web con la hoja de estilos

Para ello, vamos a colocar la etiqueta <LINK> con los atributos

- **rel="STYLESHEET"** indicando que el enlace es con una hoja de estilos
- **type="text/css"** porque el archivo es de texto, en sintaxis CSS
- **href="estilos.css"** indica el nombre del fichero fuente de los estilos

Veamos una página web entera que enlaza con la declaración de estilos anterior:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<link rel="STYLESHEET" type="text/css" href="estilos.css">
<title>P&acute;gina que lee estilos</title>
</head>
<body>
<h1>P&acute;gina que lee estilos</h1>
```

Esta página tiene en la cabecera la etiqueta necesaria para enlazar con la hoja de estilos. Es muy fácil.

```
<br>
<br>
<table width="300" cellspacing="2" cellpadding="2" border="0">
<tr>
  <td>Esto est&aacute; dentro de un TD, luego tiene estilo propio, declarado en el fichero externo</td>
</tr>
<tr>
  <td>La segunda fila del TD</td>
</tr>
</table>
</body>
</html>
```

[El resultado conseguido se puede ver aquí](#)

2.2.3.- Reglas de importancia en los estilos

Los estilos se heredan de una etiqueta a otra, como se indicó anteriormente. Por ejemplo, si tenemos declarado en el <BODY> unos estilos, por lo general, estas declaraciones también afectarán a etiquetas que estén dentro de esta etiqueta, o lo que es lo mismo, dentro de todo el cuerpo.

En muchas ocasiones más de una declaración de estilos afecta a la misma porción de la página. Siempre se tiene en cuenta la declaración más particular. Pero las declaraciones de estilos se pueden realizar de múltiples modos y con varias etiquetas, también entre estos modos hay una jerarquía de importancia para resolver conflictos entre varias declaraciones de estilos distintas para una misma porción de página. Se puede ver a continuación esta jerarquía, primero ponemos las formas de declaración más generales, y por tanto menos respetadas en caso de conflicto:

- Declaración de estilos con fichero externo. (Para todo un sitio web)
- Declaración de estilos para toda la página. (Con la etiqueta <STYLE> en la cabecera de la página)
- Definidos en una etiqueta en concreto. (Utilizando el atributo style en la etiqueta en cuestión)
- Declaración de estilo para una porción pequeña del documento. (Con la etiqueta)

Ya vimos cómo incluir estilos en la página, de todas las maneras posibles e hicimos un repaso con la lista anterior. Ahora estás en condiciones de empezar a usar las hojas de estilo en cascada para mejorar tus páginas y aumentar la productividad de tu trabajo. Pero estate atento a los siguientes capítulos donde aprenderás las lecciones que te faltan para dominar bien la materia: conocer la sintaxis, los distintos atributos de estilos y otras cosas que mejorarán tus páginas.

Artículo por [Miguel Angel Alvarez](#)

2.3.- Otra manera de definir estilos en un archivo externo

También podemos incluir estilos en un archivo externo con un código con la sintaxis @import url("estilo.css"). Se utiliza para definir estilos comunes cuando hay también definición de estilos específicos.

Veamos ahora otra manera de importar una declaración externa de estilos CSS: @import url("archivo_a_importar.css"), que se utiliza para importar unas declaraciones de estilos guardadas en la ruta que se indica entre paréntesis. (las comillas son opcionales, pero los paréntesis son obligatorios, por lo menos, en Explorer).

Se debe incluir en la declaración de estilos global a una página, es decir entre las etiquetas <style type="text/css"> y </style>, que se colocan en la cabecera del documento.

Es importante señalar que la sentencia de importación del archivo CSS se debe escribir en la primera línea de la declaración de estilos, algo parecido al código siguiente.

```
<style type="text/css">
@import url ("estilo.css");
```

```
body{  
  background-color:#ffffcc;  
}  
</style>
```

El funcionamiento es el mismo que si escribiésemos todo el fichero a importar dentro de las etiquetas de los estilos, con la salvedad de que, si redefinimos dentro del código HTML (entre las etiquetas </style>) estilos que habían quedado definidos en el archivo externo, los que se aplicarán serán los que hayamos redefinido.

Así, en el ejemplo anterior, aunque hubiésemos definido en estilo.css un color de fondo para la página, el color que prevalecería sería el definido a continuación de la importación: #ffffcc

La diferencia entre este tipo de importación del tipo y la que hemos visto anteriormente:

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

Es que @import url ("estilo.css") se suele utilizar cuando hay unas pautas básicas en el trabajo con los estilos (que se definen en un archivo a importar) y unos estilos específicos para cada página, que se definen a continuación, dentro del código HTML entre las etiquetas </style>, como es el caso del ejemplo visto anteriormente.

Artículo por Miguel Angel Alvarez

Parte 3:

lenguaje CSS

Distintos apartados que tienen que ver directamente con el lenguaje utilizado para definir los estilos en páginas web, el CSS. Veremos su sintaxis, los diferentes atributos o reglas de estilo que podemos aplicar a los elementos y cómo seleccionar conjuntos de elementos de la página para aplicarles estilo agrupados o por separado.

3.1.- Sintaxis y unidades CSS

En este capítulo explicamos la sintaxis CSS, de hojas de estilo en cascada, con especial atención a las unidades CSS.

Tal como se vió en los ejemplos de los artículos anteriores del [Manual de CSS](#), la sintaxis es bastante sencilla y repetitiva.

Básicamente se trata de colocar selectores de elementos (por ahora sólo hemos visto cómo seleccionar etiquetas, para asignarles estilos, pero más adelante conoceremos otros selectores), seguidos de los valores o atributos de estilo que queramos aplicar a dichos elementos.

A lo largo del manual aprenderemos más sobre la sintaxis de CSS, así como los distintos atributos disponibles para definir el formato o forma con la que se deben mostrar los contenidos. No obstante, quiero dar en este momento unas reglas básicas que debemos saber sobre la sintaxis de CSS, que nos servirán para entender mejor nuestros ejemplos e ir

avanzando en el conocimiento de las hojas de estilo en cascada.

Veamos entonces estas reglas básicas sobre la sintaxis CSS:

- Para definir un estilo se utilizan atributos como `font-size`, `text-decoration`... seguidos de dos puntos y el valor que le deseamos asignar. Podemos definir un estilo a base de definir muchos atributos separados por punto y coma.

Ejemplo:

font-size: 10pt; text-decoration: underline; color: black; (el último punto y coma de la lista de atributos es opcional)

- Para definir el estilo de una etiqueta se escribe la etiqueta seguida de la lista de atributos encerrados entre llaves.

Ejemplo:

H1{text-align: center; color:black}

- Los valores que se pueden asignar a los atributos de estilo se pueden ver en una tabla en el siguiente capítulo. Muchos estos valores son unidades de medida (**Unidades CSS**), por ejemplo, el valor del tamaño de un margen o el tamaño de la fuente. Las unidades de medida CSS se pueden clasificar en dos grupos, las relativas y las absolutas. Más la posibilidad de expresar valores en porcentaje.

Relativas: Se llaman así porque son unidades relativas al medio o soporte sobre el que se está viendo la página web, que dependiendo de cada usuario puede ser distinto, puesto que existen muchos dispositivos que pueden acceder a la web, como ordenadores o teléfonos móviles. En principio las unidades relativas son más aconsejables, porque se ajustarán mejor al medio con el que el usuario está accediendo a nuestra web. Son las siguientes:

Fuente actual:	em la unidad em es relativa a la fuente actual con la que se está trabajando por defecto en el sistema del usuario. Por ejemplo si un visitante tiene configurada la fuente por defecto en 12 puntos, 1em será igual a 12 puntos y 2em será igual a 24 puntos.
Altura de la letra "x":	ex 1ex será igual a la altura de la letra x, según la fuente actual del usuario. La altura de la letra x generalmente es la mitad de la de la fuente normal.
Píxeles:	px Un píxel es un punto en la pantalla del dispositivo. Dependiendo de la resolución de la pantalla, un píxel puede ser mayor o menor.

Absolutas: Las unidades absolutas son medidas fijas, que deberían verse igual en todos los dispositivos. Como los centímetros, que son una convención de medida internacional. Pese a que en principio pueden parecer más útiles, puesto que se verían en todos los sistemas igual, tienen el problema de adaptarse menos a las distintas particularidades de los dispositivos que pueden acceder a una web y restan accesibilidad a nuestra web. Puede que en tu ordenador 1 centímetro sea una medida razonable, pero en un móvil puede ser un espacio exageradamente grande, puesto que la pantalla es mucho menor. Se aconseja utilizar, por tanto, medidas relativas.

Puntos	pt Un punto es 1/72 pulgadas
Pulgadas	in

Centímetros	cm
Milímetros	mm
Picas	pc Una pica son 12 puntos.

Porcentaje: el porcentaje se utiliza para definir una unidad en función de la que esté definida en un momento dado. Imaginemos que estamos trabajando en 12pt y definimos una unidad como 150%. Esto sería igual al 150% de los 12pt actuales, que equivale a 18pt.

Porcentaje %
 Por ejemplo 120% es el 120 por cien de la unidad que estuviera anteriormente.

- Los colores se expresan con valores RGB, igual que los que conocemos para los [colores HTML](#). Con la salvedad que un color se puede especificar también con tres números hexadecimales, en lugar de 6, como era obligatorio en HTML. Por ejemplo #fff equivaldría a #ffffff, o #123 sería #112233. Además, los colores se pueden especificar también en valores RGB decimales, con la notación rgb(r,g,b), siendo los valores de r, g, b números entre 0 y 255, por ejemplo rgb(100,0,255). Otra notación posible es rgb(r%,g%,b%), siendo cada uno de los r%,g%, b% un valor entre 0 y 100, por ejemplo rgb(100%,50%,0%), que sería todo de rojo, la mitad de verde y cero de azul.
- Otro tipo de valores que se pueden utilizar en las hojas de estilo en cascada son las URL, que sirven para especificar rutas hacia elementos como imágenes a colocar en fondos de elementos. Las URL en CSS se especifican con la notación url(valor), siendo valor la URL a la que queremos dirigirnos, que puede ser absoluta o relativa. Si es relativa, el navegador la interpreta desde el documento CSS donde estamos, si es que es un archivo CSS, o desde el documento HTML donde estamos, si es que los estilos los estamos colocando directamente en el archivo HTML. a URL se puede indicar con comillas dobles, simples o sin comillas. Por ejemplo:

```
url(http://www.desarrolloweb.com/images/miimagen.gif)
url("../images/otraimagen.jpg")
```

Hasta aquí, he explicado todo lo que debes saber por ahora con respecto a la sintaxis CSS y las unidades de medida CSS disponibles. Ha sido todo un poco teórico, pero en el siguiente capítulo podrás encontrar una [lista de los atributos de las hojas de estilo en cascada](#), que te ayudarán a realizar ejercicios más prácticos. Si deseas además afianzar estos conocimientos de una manera más práctica, te recomendamos ver el [vídeo sobre CSS que habla de la sintaxis y unidades](#).

Artículo por Miguel Angel Alvarez

3.2.- Notación de colores CSS

Varias maneras, sintaxis o notaciones para definir colores con CSS de los elementos de la página.

Con CSS se puede especificar colores para cada elemento HTML de la página, incluso hay elementos que podrían admitir varios colores, como el color de fondo o el color del borde. Pero bueno, vamos a ver ahora es las distintas maneras de escribir un color en una declaración CSS.

Porque lo más habitual es que especifiquemos un color con su valor RGB, de una manera similar a como aprendimos a

[definir colores en HTML](#). Pero en CSS tenemos otras maneras de declarar colores que pueden interesarnos, como mínimo para poder entender el código CSS cuando lo veamos escrito.

3.2.1.- Notación hexadecimal RGB

Esta notación es la que ya conocemos. Se especifican los tres valores de color (rojo, verde y azul) con valores en hexadecimal entre 00 y FF.

```
background-color: #ff8800;
```

3.2.2.- Notación hexadecimal abreviada

Esta notación es muy parecida a la anterior, pero permite abreviar un poco la declaración del color, indicando sólo un número para cada valor rojo, verde y azul. Por ejemplo, para especificar el color de antes (#ff8800) podríamos haber escrito:

```
background-color: #f80;
```

3.2.3.- Nombre del color

También podemos definir un color por su nombre. Los nombres de colores son en inglés, los mismos que sirven para especificar colores con HTML.

```
color: red;  
border-color: Lime;
```

3.2.4.- Notación de color con porcentajes de RGB

Se puede definir un color por los distintos porcentajes de valores RGB. Si todos los valores están al 100% el color es blanco. Si todos están al 0% obtendríamos el negro y con combinaciones de distintos porcentajes de RGB obtendríamos cualquier matiz de color.

```
color: rgb(33%, 0%, 0%);
```

3.2.5.- Notación por valores decimales de RGB, de 0 a 255

De una manera similar a la notación por porcentajes de RGB se puede definir un color directamente con valores decimales en un rango desde 0 a 255.

```
color: rgb(200,255,0);
```

De entre todas estas notaciones podemos utilizar la que más nos interese o con la que nos sintamos más a gusto. Nosotros en nuestros ejemplos venimos utilizando la notación hexadecimal RGB por habernos acostumbrado a ella en HTML.

3.2.6.- Color transparente

Para finalizar, podemos comentar que también existe el color transparente, que no es ningún color, sino que especifica que el elemento debe tener el mismo color que el fondo donde está. Este valor, transparent, sustituye al color. Podemos indicarlo en principio sólo para fondos de elementos, es decir, para el atributo background-color.

```
background-color: transparent;
```

Artículo por *Miguel Angel Alvarez*

3.3.- Atributos de las hojas de estilo

Explicación y ejemplos de los distintos atributos de las CSS. Podrás encontrar la lista en una tabla para imprimirtela y trabajar fácilmente.

Tanto para practicar en tu aprendizaje como para trabajar con las CSS lo mejor es disponer de una tabla donde se vean los distintos atributos y valores de estilos que podemos aplicarle a las páginas web.

Aquí puedes ver la tabla de los atributos CSS más fundamentales para aplicar estilos a elementos básicos, que te vendrá perfectamente para comenzar con las CSS. Existen muchos otros atributos que aprenderás en capítulos sucesivos del [Manual de CSS](#). Recuerda además que si quieres ver cómo se aplican muchos de estos estilos en páginas web puedes ver el [vídeo tutorial sobre los atributos de las CSS](#).

Nombre del atributo	Posibles valores	Ejemplos
---------------------	------------------	----------

FUENTES - FONT

color	valor RGB o nombre de color	color: #009900; color: red;
--------------	-----------------------------	--------------------------------

Sirve para indicar el color del texto. Lo admiten casi todas las etiquetas de HTML. No todos los nombres de colores son admitidos en el estandar, es aconsejable entonces utilizar el valor RGB.

font-size	xx-small x-small small medium large x-large xx-large Unidades de CSS	font-size:12pt; font-size: x-large;
------------------	--	--

Sirve para indicar el tamaño de las fuentes de manera más rígida y con mayor exactitud.

font-family	serif sans-serif cursive fantasy monospace Todas las fuentes habituales	font-family: arial, helvetica; font-family: fantasy;
--------------------	--	---

Con este atributo indicamos la familia de tipografía del texto. Los primeros valores son genéricos, es decir, los exploradores las comprenden y utilizan las fuentes que el usuario tenga en su sistema. También se pueden definir con tipografías normales, como ocurría en html. Si el nombre de una fuente tiene espacios se utilizan comillas para que se entienda bien.

font-weight	normal bold bolder lighter 100 200 300 400 500 600 700 800 900	font-weight: bold; font-weight: 200;
--------------------	--	---

Sirve para definir la anchura de los caracteres, o dicho de otra manera, para poner negrillas con total libertad. Normal y 400 son el mismo valor, así como bold y 700.

font-style	normal italic oblique	font-style: normal; font-style: italic;
-------------------	---------------------------	--

Es el estilo de la fuente, que puede ser normal, itálica u oblicua. El estilo oblique es similar al italic.

PÁRRAFOS - TEXT

line-height	normal y unidades CSS	line-height: 12px; line-height: normal;
--------------------	-----------------------	--

El alto de una línea, y por tanto, el espaciado entre líneas. Es una de esas características que no se podían modificar utilizando

HTML.

text-decoration	none [underline overline line-through]	text-decoration: none; text-decoration: underline;
------------------------	--	---

Para establecer la decoración de un texto, es decir, si está subrayado, sobrerayado o tachado.

text-align	left right center justify	text-align: right; text-align: center;
-------------------	---------------------------------	---

Sirve para indicar la alineación del texto. Es interesante destacar que las hojas de estilo permiten el justificado de texto, aunque recuerda que no tiene por que funcionar en todos los sistemas.

text-indent	Unidades CSS	text-indent: 10px; text-indent: 2in;
--------------------	--------------	---

Un atributo que sirve para hacer sangrado o márgenes en las páginas. Muy útil y novedosa.

text-transform	capitalize uppercase lowercase none	text-transform: none; text-transform: capitalize;
-----------------------	---	--

Nos permite transformar el texto, haciendo que tenga la primera letra en mayúsculas de todas las palabras, todo en mayúsculas o minúsculas.

FONDO - BACKGROUND

Background-color	Un color, con su nombre o su valor RGB	background-color: green; background-color: #000055;
-------------------------	--	--

Sirve para indicar el color de fondo de un elemento de la página.

Background-image	El nombre de la imagen con su camino relativo o absoluto	background-image: url(mármol.gif) ; background-image: url(http://www.x.com/fondo.gif)
-------------------------	--	---

Colocamos con este atributo una imagen de fondo en cualquier elemento de la página, se puede ver una [página de ejemplo](#).

BOX - CAJA

Margin-left	Unidades CSS	margin-left: 1cm; margin-left: 0,5in;
--------------------	--------------	--

Indicamos con este atributo el tamaño del margen a la izquierda

Margin-right	Unidades CSS	margin-right: 5%; margin-right: 1in;
---------------------	--------------	---

Se utiliza para definir el tamaño del margen a la derecha

Margin-top	Unidades CSS	margin-top: 0px; margin-top: 10px;
-------------------	--------------	---------------------------------------

Indicamos con este atributo el tamaño del margen arriba de la página

Margin-bottom	Unidades CSS	margin-bottom: 0pt; margin-top: 1px;
----------------------	--------------	---

Con el se indica el tamaño del margen en la parte de abajo de la página

Padding-left	Unidades CSS	padding-left: 0.5in; padding-left: 1px;
---------------------	--------------	--

Indica el espacio insertado, por la izquierda, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas.

El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.

Padding-right	Unidades CSS	padding-right: 0.5cm; padding-right: 1pt;
----------------------	--------------	--

Indica el espacio insertado, en este caso por la derecha, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas.

El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.

Padding-top	Unidades CSS	padding-top: 10pt; padding-top: 5px;
--------------------	--------------	---

Indica el espacio insertado, por arriba, entre el borde del elemento-continente y el contenido de este.

Padding-bottom	Unidades CSS	padding-right: 0.5cm; padding-right: 1pt;
-----------------------	--------------	--

Indica el espacio insertado, en este caso por abajo, entre el borde del elemento-continente y el contenido de este.

Border-color	color RGB y nombre de color	border-color: red; border-color: #ffc000;
---------------------	-----------------------------	--

Para indicar el color del borde del elemento de la página al que se lo aplicamos. Se puede poner colores por separado con los atributos border-top-color, border-right-color, border-bottom-color, border-left-color.

Border-style	none dotted solid double groove ridge inset outset	border-style: solid; border-style: double;
---------------------	--	---

El estilo del borde, los valores significan: none=ningun borde, dotted=punteado (no parece funcionar), solid=solido, double=doble borde, y desde groove hasta outset son bordes con varios efectos 3D.

border-width	Unidades CSS	border-width: 10px; border-width: 0.5in;
---------------------	--------------	---

El tamaño del borde del elemento al que lo aplicamos.

Para ver otros ejemplos de Box [pulsar aquí](#)

float	none left right	float: right;
--------------	---------------------	---------------

Sirve para alinear un elemento a la izquierda o la derecha haciendo que el texto se agrupe alrededor de dicho elemento. Igual que el atributo align en imágenes en sus valores right y left.

clear

none | right | left

clear: right;

Si este elemento tiene a su altura imágenes u otros elementos alineados a la derecha o la izquierda, con el atributo `clear` hacemos que se coloque en un lugar donde ya no tenga esos elementos a el lado que indiquemos.

Para ver una página que utiliza `float` y `clear` [pulsar aquí](#)

La especificación de estilos CSS es muy amplia y seguro que se queda en el tintero algún atributo de estilo, pero creo que la inmensa mayoría están, y por supuesto la selección de los más importantes.

Actualizado: Efectivamente, los atributos que vemos en este texto han sido más bien pocos. Desde que se escribió este artículo han pasado años y se ha ido mejorando la especificación de CSS, con la evolución de Internet y del mundo del desarrollo de páginas web. De hecho, en estos momentos sería casi imposible concentrar en una página el listado completo de atributos con su explicación.

De todos modos, no te preocupes, porque a lo largo de este manual y de [otros manuales de CSS](#) y [talleres](#) que se han publicado en DesarrolloWeb.com aprenderás muchos otros atributos y sus diferentes valores.

Si deseas tener una hoja con todas las reglas de estilos para imprimir y tener a mano para tu referencia, te recomendamos acceder a algunas de las [hojas resúmenes](#) o de las [chuletas de CSS](#).

Artículo por [Miguel Angel Alvarez](#)

3.4.- Definición de estilos CSS Shorthand

Disminuye el peso de tus hojas de estilo utilizando la forma shorthand de especificación CSS, que no es más que una manera reducida de escribir las propiedades de estilos.

3.4.1.- Shorthand

Vamos a explicar cómo escribir de forma reducida nuestras reglas CSS para que nuestros archivos de estilo tengan menos peso y sean más entendibles a la hora de una actualización.

Según la W3C hay dos formas de escribir la misma regla de CSS: la estándar y la shorthand. Una es la larga y la otra es la reducida.

3.4.2.- Propiedad Font (fuente)

`font-style || font-variant || font-weight || font-size / line-height || familia de fuente`

Ejemplo:

```
P {font: italic normal bold 12px/14pt Verdana, Tahoma, Arial}
```

3.4.3.- Propiedad Background (fondo)

`background-color || background-image || background-repeat || background-attachment || background-position`

Ejemplo:

```
Body {background: #FFF url(..images/ejemplo.gif) no-repeat fixed center}
```

3.4.4.- Margin (Margen)

longitud | porcentaje | auto

Ejemplo:

```
Body {margin: 5px} /* todos los márgenes a 5px */ P {margin: 2px 4px} /* márgenes superior e inferior a 2px, márgenes izquierdo y derecho a 4px */ DIV {margin: 1px 2px 3px 4px} /* margen superior a 1px, right margin a 2px, bottom margin a 3px, left margin a 4px */
```

3.4.5.- Padding (Relleno)

longitud | porcentaje | auto

Ejemplo:

```
Body {padding: 2em 3em 4em 5em} /* Si definimos cuatro valores estamos aplicando el padding superior, derecho, inferior e izquierdo */ Body {padding: 2em 4em} /* Si definimos dos o tres valores, los valores faltantes se toman del lado opuesto: superior e inferior a 2em, derecho e izquierdo a 4em */ Body {padding: 5em} /* Si definimos un solo valor se aplican a todos los lados */
```

3.4.6.- Border (Borde)

border-width || border-style || color

Ejemplo:

```
H3 {border: thick dotted blue}
```



Artículo por Federico Elgarte

3.5.- Pseudo-element en CSS (pseudo elementos)

Vamos a conocer los pseudo elementos en CSS, hojas de estilo en cascada, que sirven, entre otras cosas, para definir estilos para la primera línea o letra de un texto.

Los pseudo-element (pseudo-elementos, si preferimos la traducción al castellano) sirven para aplicar estilos a partes más específicas dentro de una etiqueta. Es decir, para el ejemplo concreto de la etiqueta párrafo, con los pseudo elementos podemos definir el estilo para la primera letra del párrafo y para la primera línea. Es decir, con CSS podemos definir el estilo de una etiqueta, pero con los pseudo-elementos no nos limitamos a definir el estilo para todo el contenido de esa etiqueta, sino que indicamos el estilo para una parte restringida de esa etiqueta.

Existen diversos tipos de pseudo elementos, con distintas aplicaciones, para definir el estilo de diversas cosas, como veremos a continuación con ejemplos.

3.5.1.- Pseudo-element first-letter

Un efecto habitual de algunas publicaciones, por ejemplo las de cuentos para niños, es hacer más grande la primera letra de una página y decorarla de alguna manera. Con CSS podemos aplicar estilos específicos y hacer, por ejemplo, que esa primera letra sea más grande y tenga un color distinto del resto del párrafo.

Se utiliza de esta manera:

```
P:first-letter {
font-size: 200%;
color: #993333; font-weight: bold;
}
```

Así estamos asignando un tamaño de letra 200% más grande del propio del párrafo. También estamos cambiando el color de esa primera letra.

De entre todas las propiedades de estilos, sólo algunas se pueden aplicar a los pseudo-elementos first-letter. Son las siguientes, según la especificación del W3C: font properties, color properties, background properties, 'text-decoration', 'vertical-align' (sólo si 'float' está asignado a 'none'), 'text-transform', 'line-height', margin properties, padding properties, border properties, 'float', 'text-shadow' y 'clear'.

Se puede ver un [ejemplo de aplicación de un estilo con first-letter](#).

3.5.2.- Pseudo-element first-line

Como adelantaba, este pseudo-elemento, sirve para asignar un estilo propio a la primera línea del texto. Es posible que hayamos visto alguna revista o periódico que utilice este estilo para remarcar las primeras líneas del párrafo. Un ejemplo de su uso sería el siguiente:

```
P:first-line {
text-decoration: underline;
font-weight: bold;
}
```

Las propiedades de estilos que se pueden aplicar al pseudo-element first-line son las siguientes: font properties, color properties, background properties, 'word-spacing', 'letter-spacing', 'text-decoration', 'vertical-align', 'text-transform', 'line-height', 'text-shadow' y 'clear'.

Se puede ver un [ejemplo de aplicación de un estilo con first-line](#).

3.5.3.- Uso de clases con los pseudo-elementos

En determinadas ocasiones podemos necesitar crear una clase (class) de CSS a la que asignar los pseudo-elementos, de modo que estos no se apliquen a todas las etiquetas de la página. Por ejemplo, podemos desear que solamente se modifique el estilo de la primera línea del texto en algunos párrafos y no en todos los de la página.

Una clase se define con el nombre de la etiqueta seguido de un punto y el nombre de la clase. Si además deseamos definir un pseudo-elemento, deberíamos indicarlo a continuación, con esta sintaxis:

```
P.nombreclase:first-line {
font-weight: bold;
}
```

3.5.4.- Pseudo-elementos en CSS2

Aparte de first-line y first-letter, en las especificaciones de CSS 2 existen otros pseudo elementos que voy a nombrar para conocimiento de los lectores, aunque profundizaré en su uso. Se tratan de before y after y sirven para insertar "contenidos generados" antes y después del elemento al que asignemos estos pseudo-element.

Un ejemplo de ello es:

```
P.nota:before {
content: "Nota: "
}
```

Así se ha definido una clase de párrafo llamada "note" en la que se indica que antes de la propia nota se debe incluir el texto indicado, osea, "Nota: ".

Nota: Atención a la compatibilidad con CSS2, que, por lo menos para estos elementos, no está soportada en versiones 6 de Internet Explorer. Firefox, en cambio, sí que es compatible con estas características de CSS2.

Si queremos ver un ejemplo completo de uso de los pseudo elementos after y before podemos leer el siguiente artículo del taller de CSS, en el que mostramos una [técnica para conseguir las esquinas redondeadas en CSS 2](#).

Artículo por Miguel Angel Alvarez

3.6.- Trucos avanzados con CSS

Vamos a ver una serie de técnicas con hojas de estilo, imprescindibles para utilizar esta tecnología con toda su potencia.

Las hojas de estilos son un tema largo del que se han escrito libros enteros. Hasta este punto del [Manual de CSS](#) nos hemos dedicado a los temas más básicos. De momento vamos a hacer una parada en este artículo para explicar unas cuantas cosas interesantes que nos resultarán especialmente prácticas.

Además, para completar tus primeros conocimientos sobre CSS, te recomendamos ver el [vídeo sobre los selectores de CSS](#), que comenta algunas de las cosas de este capítulo y muchas otras que debes saber para manejar correctamente este lenguaje de estilo.

3.6.1.- Definir estilos utilizando clases

Las clases nos sirven para crear definiciones de estilos que se pueden utilizar repetidas veces.

Una clase se puede definir entre las etiquetas <STYLE> (en la cabecera del documento), o en un archivo externo a la página. Para definir las utilizamos la siguiente sintaxis, un punto seguido del nombre de la clase y entre llaves los atributos de estilos deseados. De esta manera:

```
.nombredelaclase {atributo: valor; atributo2: valor2; ...}
```

Una vez tenemos una clase, podemos utilizarla en cualquier etiqueta HTML. Para ello utilizaremos el atributo class, poniéndole como valor el nombre de la clase, de esta forma:

```
<ETIQUETA class="nombredelaclase">
```

Ejemplo de la utilización de clases

```
<html>
<head>
<title>Ejemplo de la utilización de clases</title>
<STYLE type="text/css">
.fondonegroletrasblancas {background-color:black;color:white;font-size:12;font-family:arial}
.letrasverdes {color:#009900}
</STYLE>
</head>

<body>
<h1 class=letrasverdes>Titulo 1</h1>
<h1 class=fondonegroletrasblancas>Titulo 2</h1>

<p class=letrasverdes>
Esto es un párrafo con estilo de letras verdes</p>
<p class=fondonegroletrasblancas>
Esto es un párrafo con estilo de fondo negro y las letras blancas. Es todo!</p>
</body>
```

</html>

[Ver el ejemplo anterior](#)

3.6.2.- Estilo en los enlaces

Una técnica muy habitual, que se puede realizar utilizando las hojas de estilo en cascada y no se podía en HTML, es la definición de estilos en los enlaces, quitándoles el subrayado o hacer enlaces en la misma página con distintos colores.

Para aplicar estilo a los enlaces debemos definirlos para los distintos tipos de enlaces que existen (visitados, activos...). Utilizaremos la siguiente sintaxis, en la declaración de estilos global de la página (<STYLE>) o del sitio (Definición en un archivo externo):

Enlaces normales

A:link {atributos}

Enlaces visitados

A:visited {atributos}

Enlaces activos (Los enlaces están activos en el preciso momento en que se pincha sobre ellos)

A:active {atributos}

Enlaces hover (Cuando el ratón está encima de ellos, solo funciona en ieplorer)

A:hover {atributos}

El atributo para definir enlaces sin subrayado es **text-decoration:none**, y para darles color es **color:tu_color**.

También podemos definir el estilo de cada enlace en la propia etiqueta <A>, con el atributo style. De esta manera podemos hacer que determinados enlaces de la página se vean de manera distinta

Ejemplo de estilos en enlaces

```
<html>
<head>
<title>Ejemplos de estilo en enlaces</title>
<STYLE type="text/css">
A:link {text-decoration:none;color:#0000cc;}
A:visited {text-decoration:none;color:#ffcc33;}
A:active {text-decoration:none;color:#ff0000;}
A:hover {text-decoration:underline;color:#999999;font-weight:bold;}
</STYLE>
</head>

<body>

<a href="http://dominioinexistente.nofunciona.com">Enlace normal</a>
<br>
<br>
<a href="enlaces.html">Enlace visitado</a>
Pulsar este enlace para verlo activo,
poner el rat&oacute;n por encima para que cambie.

</body>
</html>
```

[Ver el ejemplo anterior](#)

3.6.3.- URL como valor de un atributo:

Determinados atributos de estilos, como **background-image** necesitan una URL como valor, para indicarlas podemos definir tanto caminos relativos como absolutos. Así pues, podemos indicar la URL de la imagen de fondo de estas dos maneras:

background-image: url(fondo.gif) En caso de que la imagen esté en el mismo directorio que la página. **background-image: url(http://www.desarrolloweb.com/imagenes/fondo.gif)**

3.6.4.- Ocultar estilos en navegadores antiguos

En caso de definir dentro de la etiqueta <STYLE> unas declaraciones de estilos debemos asegurarnos que estas no se imprimirán en la página web con navegadores antiguos. Pensar en un navegador que no reconozca la etiqueta <STYLE>, pensará que corresponde con algo que no entiende y se olvidará de la etiqueta. Lo siguiente que encuentra es texto normal y hará que este se vea en la página, como con cualquier otro texto.

Para evitarlo debemos ocultar con comentarios HTML (<!-- esto es un comentario -->) todo lo que hay dentro de la etiqueta <STYLE>. Se puede ver un ejemplo de esto a continuación:

De este modo hemos terminado la primera parte del manual de CSS, que espero pueda ayudar a hacer páginas mejores y más rápidamente. Ahora el manual continua explicando [conceptos sobre capas](#) y [maquetación CSS](#), entre otros asuntos.

Quiero recordaros que siempre es útil ver como han hecho sus páginas otros programadores de Internet. Para ver una página definida enteramente con hojas de estilos podemos visitar [Web Site Album](#), que está incluso [maquetada con CSS](#). También podemos visitar la dirección www.guiarte.com, que está maquetada con tablas, pero todos los estilos se aplican con css.

Nota: Para ver otros manuales, artículos y enlaces a páginas que enseñan a utilizar las hojas de estilos visitar la [sección CSS a fondo](#).

En el siguiente capítulo de este manual pasamos página para contaros uno de los "nuevos elementos" que cobran una especial importancia desde la llegada de CSS, [las capas](#).

Artículo por Miguel Angel Alvarez

Parte 4:

Modelo de caja

En CSS se crea un nuevo modelo de caja que nos sirve para agrupar elementos en contenedores, a los que luego podremos aplicar estilos con CSS. Se trata de las capas, o cajas, que cobrarán una gran importancia a la hora de realizar tus diseños.

4.1.- Qué son las capas

Vemos las diferencias entre varias etiquetas para aplicar estilos y crear capas y una pequeña introducción a las capas.

Veamos una pequeña introducción a lo que son las capas, la etiqueta HTML <DIV> utilizada para construirla y los atributos CSS con los que podemos aplicar estilos.

Como ya hemos visto en nuestro [manual de CSS](#), sirve para aplicarle estilo a una pequeña parte de una página

HTML. Por ejemplo, con ella podríamos hacer que una parte de un párrafo se coloree en rojo. Con `` no es habitual englobar un trozo muy grande de texto, por ejemplo el que comprenda a varios párrafos.

Con `<DIV>` también podemos aplicar estilo a partes de la página HTML.

La diferencia entre `` y `<DIV>` es que con esta última sí que podemos aplicar estilo a una parte más amplia de la página, por ejemplo a tres párrafos. **Además que la etiqueta `<DIV>` tiene un uso adicional que es el de crear divisiones en la página a las que podremos aplicar una cantidad adicional de atributos para modificar sus comportamientos.** Por ejemplo, con el atributo `align` de HTML podemos alinear la división al centro, izquierda, derecha o justificado. Pero **su uso más destacado es el de convertir esa división en una capa.**

Una capa es una división, una parte de la página, **que tiene un comportamiento muy independiente** dentro de la ventana del navegador, ya que la podemos colocar en cualquier parte de la misma y la podremos mover por ella independientemente, por poner dos ejemplos. En el uso de capas se basan muchos de los efectos más comunes del DHTML.

Las etiquetas `<LAYER>` e `<ILAYER>` tienen como objetivo construir capas, pero estas no son compatibles más que con Netscape, por lo que es recomendable utilizar la etiqueta `<DIV>` para hacer capas preferentemente a las otras dos.

Los atributos que podemos aplicar a estas etiquetas, pero en concreto a las dos recomendadas `` y `<DIV>`, son principalmente de estilos CSS. Estos atributos nos permiten, como hemos podido ver en el manual de hojas de estilo en cascada de desarrolloweb, modificar de una manera muy exhaustiva la presentación de los contenidos en la página. Para aplicar estilos a estas etiquetas se utiliza el atributo de HTML `style`, de esta manera:

```
<SPAN style="text-decoration:underline;font-weight:bold">...</SPAN>
```

```
<DIV style="color:red;font-size:10px">...</DIV>
```

Como ya pudimos ver muchos ejemplos en el [manual de CSS](#), nos referimos a él para ampliar esta información. Pero no habíamos visto todavía una serie de atributos que nos sirven para posicionar la división en la página como una capa. Estos atributos se pueden aplicar a la etiqueta `<DIV>` que es la que servía para crear capas compatibles con todos los navegadores.

Los atributos para que la división sea una capa son varios y se pueden ver a continuación.

```
<div id="c1" style="position:absolute; left: 200px; top: 100px;">
Hola!
</div>
```

El primero, **position**, indica que se posicione de manera absoluta en la página y los segundos, **left** y **top**, son la distancia desde el borde izquierdo de la página y el borde superior.

Hay otros atributos especiales para capas como **width** y **height** para indicar la anchura y altura de la capa, **Z-index** que sirve para indicar qué capas se ven encima de qué otras, **clip** que sirve para recortar una capa y hacer que partes de ella no sean visibles, o **visibility** para definir si la capa es visible o no. Estos y otros atributos los veremos en el [siguiente capítulo, donde hablaremos del posicionamiento de capas](#).

Artículo por Miguel Angel Alvarez

4.2.- Atributos para capas

Veamos el posicionamiento de capas y otros atributos que podemos utilizar al definirlos.

Hemos visto en el capítulo anterior [qué son las capas y algunas pequeñas muestras sobre cómo crearlas y darle algún estilo](#). Ahora vamos a ver en detenimiento los atributos específicos para aplicar posicionamiento a una capa y otros estilos.

Antes que nada cabe decir que una capa puede tener cualquier atributo de estilos de los que hemos visto en el [manual de](#)

CSS. Así, el atributo color indica el color del texto de la capa, el atributo font-size indica el tamaño del texto y así con todos los atributos ya vistos.

Ahora bien, existen una serie de atributos que sirven para indicar la forma, el tamaño de las capas, la visibilidad, etc, que no hemos visto en capítulos anteriores y que veremos a continuación.

4.2.1.- Atributo position

Indica el tipo de posicionamiento de la capa. Puede tener dos valores, relative o absolute.

- relative indica que la posición de la capa es relativa a el lugar donde se estaba escribiendo en la página en el momento de escribir la capa con su etiqueta
- absolute indica que la posición de la capa se calcula con respecto al punto superior izquierdo de la página.

4.2.2.- Atributo top

Indica la distancia en vertical donde se colocará la capa. Si el atributo position es absolute, top indica la distancia del borde superior de la capa con respecto al borde superior de la página. Si el atributo position era relative, top indica la distancia desde donde se estaba escribiendo en ese momento en la página hasta el borde superior de la capa.

4.2.3.- Atributo left

Básicamente funciona igual que el atributo top, con la diferencia que el atributo left indica la distancia en horizontal a la que estará situada la capa.

4.2.4.- Atributo height

Sirve para indicar el tamaño de la capa en vertical, es decir, su altura.

4.2.5.- Atributo width

Indica la anchura de la capa

4.2.6.- Atributo visibility

Sirve para indicar si la capa se puede ver en la página o permanece oculta al usuario. Este atributo puede tener tres valores.

- visible sirve para indicar que la capa se podrá ver.
- hidden indicará que la capa está oculta.
- inherit es el valor por defecto, que quiere decir que hereda la visibilidad de la capa donde está metida la capa en cuestión. Si la capa no está metida dentro de ninguna otra se supone que está metida en la capa documento, que es toda la página y que siempre está visible.

4.2.7.- Atributo z-index

Sirve para indicar la posición sobre el eje z que tendrán las distintas capas de la página. Dicho de otra forma, con z-index podemos decir qué capas se verán encima o debajo de otras, en caso de que estén superpuestas. El atributo z-index toma valores numéricos y a mayor z-index, más arriba se verá la página.

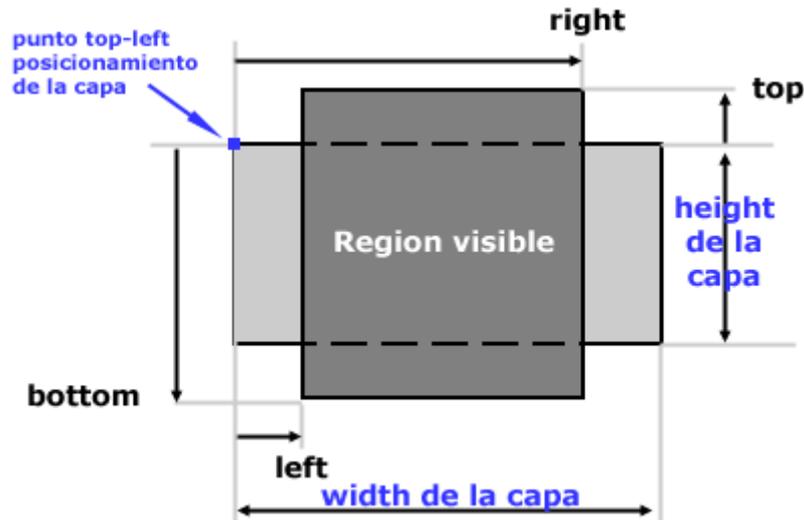
4.2.8.- Atributo clip

Es un atributo un poco difícil de explicar. En concreto sirve para recortar determinadas áreas de la capa y que no se puedan ver. Una capa que está visible se puede recortar para que se vea, pero que no se vea algún trozo de esta. El

clipping se indica por medio de 4 valores, con esta sintaxis.

```
rect (<top>, <right>, <bottom>, <left>)
```

Los valores <top>, <right>, <bottom> y <left> indican distancias que se pueden apreciar en este esquema.



Este es un ejemplo de capa que utiliza todos los atributos que hemos visto en este artículo y alguno más para aplicar estilo a la capa.

```
<div style="clip: rect(0,158,148,15); height: 250px; width: 170px; left: 10px; top: 220px; position: absolute; visibility: visible; z-index:10; font-size: 14pt; font-family: verdana; text-align: center; background-color: #bbbbbb">
```

Esta capa tiene un clipping, por eso se ve entrecortada.

```
<br>
```

```
<br>
```

Esto es una capa de prueba

```
</div>
```

Puede [verse el ejemplo en una página web](#), donde también podrá apreciarse el efecto conseguido al realizar el clipping.

Artículo por Miguel Angel Alvarez

4.3.- Problema con el posicionamiento absoluto de capas

A veces el posicionamiento absoluto de capas es demasiado rígido, pues si la definición de pantalla cambia de un usuario a otro las capas pueden quedar colocadas en lugares donde no deseamos. He aquí una solución.

He recibido una consulta en mi correo sobre colocación de capas de manera absoluta, pero en la que no nos importe la definición de la pantalla del usuario y otros ir y venir de los elementos HTML. Nuestro compañero expresó su duda de la siguiente manera:

Si trabajamos con position:absolute dando un left y un top funciona si tienes tu página alineada a la izquierda. Mi página está alineada en el centro, entonces lo que sucede es que dependiendo de la resolución de pantalla que tengas (ancho de 800px,1024px,etc) me baila toda la página y no cuadra nada.

Primero que todo, debemos saber que si trabajamos con el position relative las capas se colocan en el lugar donde

aparecen dentro del código HTML. De este modo, si colocamos una capa con position relative dentro de una celda de una tabla, dicha capa aparecería dentro de la celda donde la estamos colocando, independientemente del lugar donde se sitúe la celda al cambiar la definición de la pantalla.

El problema de esta solución es que la capa haría crecer la celda de la tabla donde queremos colocarla (al igual que cualquier otro elemento HTML que colocásemos dentro de la tabla) y es muy probable que nuestro diseño no nos permita este hecho. Seguramente ya habrías notado este problema y si no es así te invito a que crees la capa que intentas colocar con el atributo position a relative para ver si con eso tu problema ya está resuelto.

En casi todos los casos, la capa que intentamos colocar va a tener que tener el position absolute, porque con relative no arreglamos totalmente el problema. Entonces volvemos a el problema inicial, que era situar la capa con position absolute en el lugar exacto, independientemente de la definición de pantalla.

La solución final que propongo pasa por aplicar algún truquillo. De hecho, estuve hace unos días preguntándome sobre esa cuestión y al final encontré la solución, aunque no se me ocurrió a mi, sino que la extraje de [las librerías X-Library](#).

La idea es un poco compleja y para su puesta en marcha debemos realizar una serie de acciones que, sinceramente, considero excesivas para un problema inicialmente sencillo. Así pues, que no asuste lo que voy a soltar a continuación, que luego trataré de explicarlo un poco mejor.

Nuestro esquema de trabajo consistirá en una capa con posición relativa, que nos servirá de "ancla" y otra con la posición absoluta, donde colocaremos el contenido final a mostrar en la capa.

La capa relativa la colocaremos en el lugar aproximado donde queramos que aparezca la capa absoluta. La capa absoluta la posicionaremos, una vez cargada la página, en un lugar próximo a la capa relativa. Por supuesto, estas acciones las vamos a tener que realizar con Javascript, que es el lenguaje que nos permite actualizar las posiciones de las capas dinámicamente.

4.3.1.- Detenidamente

Decíamos que habría que colocar una capa relativa cercana al lugar donde tiene que aparecer la capa con position absolute. Insisto en que las capas relativas se colocan en el lugar donde las metemos dentro del código HTML, por lo que será fácil colocar la capa relativa en el lugar exacto y que este lugar sea válido para cualquier definición.

La segunda capa, la que tiene el contenido final, la pondremos inicialmente en una posición cualquiera y escondida, de manera que no se vea que está mal colocada. Una vez terminada de cargar la página, podremos acceder a la posición de la capa relativa, extrayendo sus valores top y left y colocándolos en los correspondientes top y left de la capa con posición absoluta. Una vez marcada la posición de la capa absoluta podemos volverla visible.

A la vista de la imagen siguiente, la capa con posición relativa la hemos colocado en el enlace. En realidad habría tres capas con posición relativa para poder posicionar otras tantas capas con posición absoluta. La parte que vemos sombreada de verde corresponde al espacio que abarcaría la capa relativa.

Su posición sería la que está marcada por el aspa roja que aparece en su esquina superior izquierda. Dicha posición depende del lugar donde aparezcan los enlaces en la página.

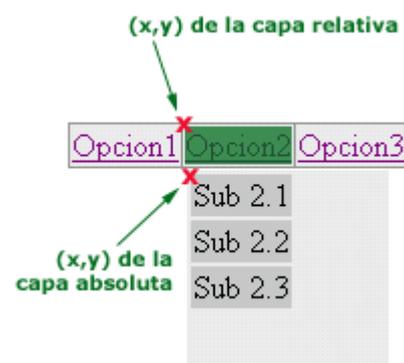
Luego, con Javascript deberíamos asignar la posición de la capa absoluta de una manera parecida a esta.

left de la capa absoluta = left de la capa relativa

top de la capa absoluta = top de la capa relativa + altura de la capa relativa

Podemos sumarle algún píxel más a la posición de la capa, si es que queremos moverla un poco abajo y a la derecha, tal como hemos visto en la imagen.

No pretendo en este artículo, muy a mi pesar y por falta de espacio y tiempo, explicar cómo se hacen esas operaciones de Javascript. Advierto que si no se conoce nada de Javascript va a ser imposible ponerse con una tarea tan tediosa como el manejo de capas. Si por el contrario, ya hemos tenido contacto con Javascript y DHTML anteriormente, no



debería ser un problema realizar esas acciones.

4.3.2.- Referencias Javascript

En DesarrolloWeb tenemos un par de manuales de Javascript, que sería necesario estudiar para empezar a introducirse en el lenguaje.

- [Programación en Javascript I](#)
- [Programación en Javascript II](#)

En el [Taller de Javascript](#) tenemos algún artículo sobre tratamiento dinámico de capas.

Pero sin duda, lo que mejor os va a venir para tratar con capas y su posicionamiento dinámico es utilizar algún framework Javascript, los cuales os ayudarán fácilmente a realizar cualquier tipo de operación DHTML con la certeza de que funcione bien en todos los navegadores. Os recomiendo las siguientes lecturas:

- [Manual de Mootools](#)
- [Manual de jQuery](#)

Artículo por Miguel Angel Alvarez

4.4.- El Modelo de Caja en CSS

Un aspecto especialmente relevante de CSS es el Modelo de Caja. Este artículo brinda una primera aproximación a su arquitectura y a las distintas posibilidades que ofrece.

Tarde o temprano, todo libro o taller práctico de CSS queda bajo la influencia del Modelo de Caja de CSS. Es un tema que, sin lugar a dudas, tenemos que dominar perfectamente y realmente no tiene ninguna dificultad. La práctica nos dará la soltura necesaria para que no tengamos ni que pensar cuando estemos diseñando o maquetando una web, pero para las personas que están empezando con CSS será muy interesante el prestar la debida atención.

En este artículo publicado en DesarrolloWeb.com daremos un repaso general a todos los integrantes o atributos que podemos utilizar para definir el modelo de caja en CSS, que hemos englobado dentro del [Manual de CSS](#).

4.4.1.- Una primera aproximación visual

Es uno de los elementos básicos de las Hojas de Estilo en Cascada y por lo tanto su correcta interpretación resulta fundamental a la hora de lograr los resultados deseados en un diseño, por más simple que éste resulte.

Para entrar en tema, vamos a construir un sencillo ejemplo utilizando un único elemento <div> al que aplicaremos un estilo. El resultado producido será analizado con la ayuda de una figura en la que hemos modelado el orden de apilado de los elementos del <div> en una disposición tridimensional que nos ayudará a comprenderlo.

Supongamos el siguiente código (lo mostramos fuera de su contexto, restringiéndonos a lo significativo para el ejemplo):

4.4.2.- El elemento <div>

```
<div>
<p>Este es el contenido de nuestra caja.</p>
</div>
```

4.4.3.- El estilo

```
div {
background-color: #be4061; /*color bordó para el fondo*/
background-image: url('cabeza.jpg');
border: 10px solid #e7a219; /*color naranja para el borde*/
margin: 10px;
padding: 20px;
}

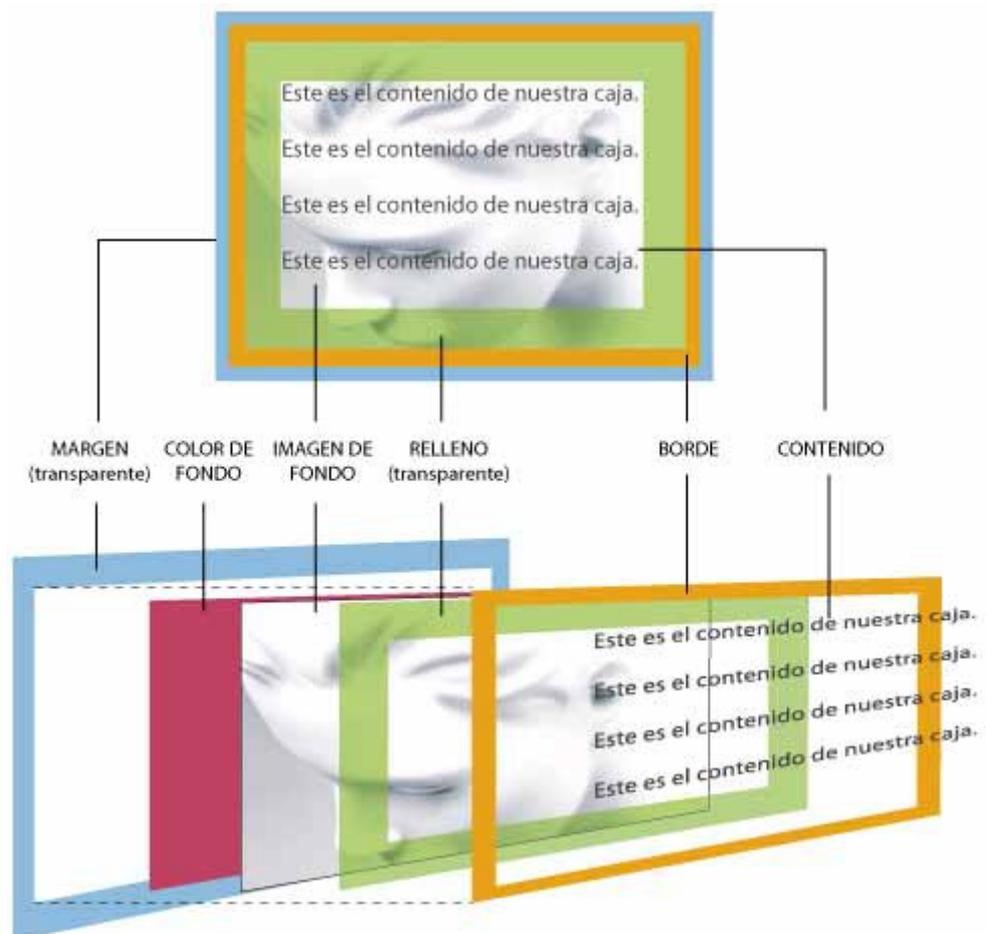
p {
margin: 0 0 20px 0; /*margen inferior de 20 px para el párrafo*/
padding: 0;
}
```

El código anterior generará una caja como la que muestra la figura siguiente, en la que adicionalmente se ha dado color a los elementos transparentes (margen y relleno) solo para hacerlos visibles.

Un detalle interesante que puede apreciarse en la representación tridimensional en que la capa superior del apilamiento no es el borde, como podría suponerse intuitivamente.

La capa situada encima de todas las demás es la de Contenido.

Aunque el caso específico sea materia de otro artículo, comentaremos que esta disposición fue utilizada por el diseñador Douglas Bowman de Stopdesign para el rediseño del sitio de Blogger, logrando las armoniosas líneas curvas de sus páginas mediante CSS, ubicando imágenes en la capa de Contenidos de modo que oculten el borde anguloso de las cajas.



4.4.4.- Áreas y propiedades

Cada caja en CSS posee, además de su área de Contenido, otras tres áreas opcionales:

- Área de Margen - **Margin**
- Área de Relleno - **Padding**
- Área de Borde - **Border**

Cada área, a su vez, puede dividirse en cuatro segmentos según su posición: izquierdo (left), derecho (right), superior (top) e inferior (bottom).

El tamaño de cada área o de sus segmentos está dado por el valor de las respectivas propiedades, definidas en forma global o discriminadas por segmento.

Por ejemplo, la siguiente sentencia asignará un margen homogéneo de 20 píxeles alrededor de la caja:

```
div { margin: 20px }
```

Si en cambio se desea asignar valores distintos a cada uno de los segmentos, pueden reflejarse los cuatro valores numéricos siguiendo el orden top - right - bottom - left.

El siguiente ejemplo asigna 10 píxeles arriba, 5 a la derecha, 20 abajo y nada a la izquierda:

```
div { margin: 10px 5px 20px 0 }
```

Pueden especificarse valores también con la siguiente notación, en la que ya no es necesario mantener el orden:

```
div {  
margin-top: 10px ;  
margin-right: 5px ;  
margin-bottom: 20px ;  
}
```

En cualquier caso puede obviarse el valor 0 ya que es el valor que toman las propiedades por defecto.

La lista completa de propiedades es la siguiente:

Propiedades del Margen

"margin-top", "margin-right", "margin-bottom", "margin-left" y "margin"

Propiedades del Relleno

"padding-top", "padding-right", "padding-bottom", "padding-left" y "padding"

Propiedades del Borde

1) Ancho (width)

"border-top-width", "border-right-width", "border-bottom-width", "border-left-width" y "border-width". Pueden ser valores específicos o los valores "thin" (fino), "medium" (medio) y "thick" (grueso)

2) Color (color)

"border-top-color", "border-right-color", "border-bottom-color", "border-left-color" y "border-color"

3) Estilo (style)

"border-top-style", "border-right-style", "border-bottom-style", "border-left-style" and "border-style". Toma una serie de posibles valores, tales como: none, hidden, dotted, dashed, solid, double, groove, ridge, inset y outset. Es una propiedad algo conflictiva ya que no todos los navegadores interpretan sus valores de la misma manera.

Como corolario de esta aproximación al modelo de caja resta analizar qué es lo que se verá en cada área cuando la página se muestre en un navegador:

- En el área de Contenido y en la de Relleno se verá aquello que se determine en la propiedad "background" del elemento (un color o una imagen, según el orden de apilado).
- En el área de Borde se verá aquello que se determine en las propiedades del Borde (ancho, color y estilo).
- El área de Margen es siempre transparente.

4.4.5.- El Secreto para dominar el modelo de caja en CSS

Hay un solo secreto para comprender cabalmente cada una de las propiedades y su utilización: probar, probar y probar. Al principio podrá parecerse complicado, pero con la práctica podremos trabajar con el modelo de caja sin tener que pensar, y así conseguir diseños de webs y maquetación CSS de calidad con poco esfuerzo y atendiendo a los estándares de desarrollo web.

Artículo por [Fernando Campaña](#)

4.5.- Problemas del Modelo de Caja en Internet Explorer 5

Si no se toman ciertos recaudos nuestros diseños se verán distinto en distintos navegadores y aún peor: en distintas versiones del mismo navegador. Este artículo analiza esos comportamientos anómalos y brinda algunos trucos para que nuestros diseños

4.5.1.- Una primera aproximación visual

Partiremos de la base de que el lector posee las nociones básicas sobre el Modelo de Caja.

Básicamente, la 'caja' en CSS puede ser asimilada a una tabla con una sola celda.

De forma obviamente rectangular, esa caja puede tener bordes (border), márgenes transparentes por fuera de los bordes (margin) y relleno transparente por dentro de los bordes (padding). Cualquiera de estos atributos puede ser asignado para cada uno de los cuatro lados de la caja separadamente.

El contenido de la caja se ubicará dentro del área de relleno.

En general, podemos hablar de dos tipos de diseño:

- Los diseños 'líquidos' no asignan un valor específico al ancho (width) y alto (height) de la caja, por lo que resulta que esas dimensiones surgirán del contenedor de la caja y de la extensión del contenido. Concretamente, el ancho de esa caja será todo el permitido por la estructura que la contenga (utilizará todo el ancho disponible) y el alto será el necesario para mostrar todo el contenido (crecerá hacia abajo todo lo necesario).
- Los diseños 'estáticos' surgen cuando se decide asignar a la caja un ancho específico mediante la asignación de un valor concreto a la propiedad width.

Los problemas comienzan cuando trabajamos con diseños estáticos. Nuestra caja tendrá un ancho dado, pero no todos los navegadores interpretarán ese ancho de la misma manera.

Para el W3C el ancho de una caja se mide desde el límite interno del relleno izquierdo (left-padding) hasta el límite interno del relleno derecho (right-padding). En caso de no existir relleno se toman los límites internos izquierdo y derecho del borde (border-right y border-left).

Cuando Microsoft lanzó su Internet Explorer 5 para Windows (IE5/win) no respetó este estándar, interpretando la propiedad width como el ancho comprendido entre los límites exteriores del borde (border-left y border-right). Para verlo con más claridad supongamos un ejemplo sencillo: una caja de 100 píxeles de ancho, 10 de relleno, 5 de borde y 10 de margen, todos ellos uniformes.

El elemento <div>

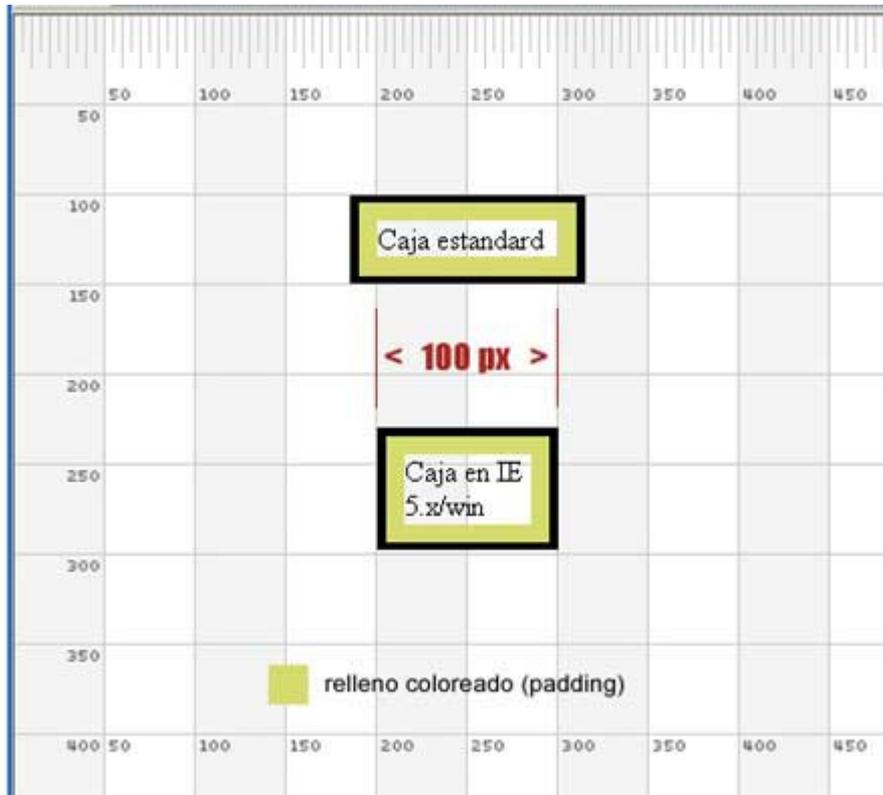
```
<body>
<div>Aquí el contenido de la caja</div>
</body>
```

Aplicando CSS ...

```
div {
width: 100px;
```

```
padding: 10px;
border: 5px solid black;
margin: 10px;
}
```

En la imagen siguiente (en la que se ha aplicado un color al padding únicamente con fin ilustrativo) puede verse una representación de la diferente interpretación entre el IE5/win y los demás navegadores y anticipar las desastrosas consecuencias para el diseño final.



Tal como se aprecia, la caja se ve más pequeña cuando se la visualiza con IE5/win.

La versión del navegador para Mac (IE5/mac) no tiene ese inconveniente e interpreta la caja según el estándar W3C. Afortunadamente, Microsoft remedió este problema en el Explorer 6, pero hay millones de usuarios que todavía utilizan IE5.x/win!

4.5.2.- La solución en general

La solución para este tipo de problemas consiste en utilizar trucos que aprovechan limitaciones o errores de los navegadores en la interpretación de CSS.

Hay varios de estos recursos, denominados en general Box Model Hacks, que basan su acción en suministrarle al navegador conflictivo un ancho de caja tal que su representación resulte la esperada, pero sin que este valor afecte a los otros navegadores.

En el caso de nuestro ejemplo, para igualar las áreas de contenido, deberíamos decirle al IE5.x/win (y únicamente al IE5.x/win) que el ancho es un valor tal que incluya relleno y borde:

borde izquierdo + relleno izquierdo + contenido + relleno derecho + borde derecho

5 px + 10 px + 100 px + 10 px + 5 px
 o sea ...

width: 130 px;

De este modo, la caja se vería idéntica en todos los navegadores.

El problema con muchas de las soluciones (hacks) propuestas es que de un modo u otro terminaban afectando a otros navegadores menos populares. Por lo tanto, aún cuando la cosa mejoraba sustancialmente en términos de audiencia, faltaba dar todavía con un recurso que no generara efectos indeseados.

4.5.3.- La mejor solución: Tan hack

La solución fue propuesta por Edwardson Tan y reúne las inusuales características de ser simple y perfectamente eficaz.

Volviendo a nuestro ejemplo:

```
<body>
<div>Aquí el contenido de la caja</div>
</body>
```

y aplicando CSS con algunos cambios ...

```
div {
width: 100px;
padding: 10px;
border: 5px solid black;
margin: 10px;
}
/* Aquí está el truco, el Tan Hack */
* html div {
width: 130px;
width: 100px;
}
```

4.5.4.- Analicemos un poco el código del truco

Observamos un asterisco (*) al comienzo. El asterisco en CSS es conocido como selector universal y alude a todos los elementos que están contenidos dentro de otro.

Luego encontramos html, el elemento raíz de toda página, y luego nuestro div.

La regla * html div { } se aplicará a todo elemento div contenido en un elemento html que a su vez esté contenido en otro. Suena raro, no?

Para cualquier navegador distinto del Explorer la regla será interpretada como errónea y por lo tanto ignorada, ya que no existe ningún elemento que contenga a html, que acabamos de decir que es el elemento raíz.

El Explorer (en todas las versiones) parece creer en la existencia de un misterioso elemento que engloba a html (obviamente se trata de un defecto de estos navegadores), por lo que la regla le resultará válida.

Hasta aquí logramos que únicamente los IE resulten afectados y redefinan el ancho a 130px. Solo resta lograr que esa redefinición del ancho (en el ejemplo a 130px) sea válida únicamente para el IE5.x/win, ya que habíamos dicho que tanto el IE5/mac como el IE6 interpretaban correctamente el modelo de caja y no necesitan ningún truco para respetar nuestro diseño.

Para eso utilizamos la siguiente línea, de apariencia un poco extraña:

```
width: 100px;
```

Tanto el IE5/mac como el IE6 son capaces de ignorar la barra invertida (escape) dentro del nombre de una propiedad, siempre que se cumplan determinadas condiciones que detallamos más adelante.

Los dos navegadores corregirán width transformándolo en width y volverán a modificar el ancho a 100px, su valor correcto.

El IE5.x/win no puede manejar esa barra invertida dentro del nombre y por lo tanto considera a esa línea como errónea y la ignora. El resultado es que el ancho, para IE5.x/win y sólo para él, permanece en 130px.

Es exactamente lo que queríamos!

Un último comentario acerca de la barra invertida dentro del nombre: hay ciertas limitaciones en cuanto a su posición.

- Debe estar dentro del nombre

- No debe estar justo antes de cualquiera de las primeras seis letras del alfabeto (a, b, c, d, e o f) ya que se interpretaría como un carácter hexadecimal y echaría a perder el truco.

Artículo por *Fernando Campaña*

4.6.- Por qué diseñar con CSS

Mostramos algunas ventajas al maquetar con CSS respecto a otras formas de hacerlo.

Las tablas existen y existieron desde el comienzo en HTML, pero no se crearon para diseñar un sitio, sino para la presentación de datos tabulares. La utilización del `border=0` y las imágenes transparentes hicieron posible crear una rejilla que permitió a los diseñadores organizar textos e imágenes, establecer tamaños y ubicar objetos. Pero ésto es sencillamente incorrecto. Las tablas no se crearon para maquetar y no deben utilizarse para eso, porque de esta forma se mezclan presentación y contenido.

La solución es clara: CSS+HTML/XHTML. Afortunadamente, cada vez son más las empresas que deciden dejar atrás las tediosas tablas y evolucionar desarrollando sus sitios respetando los estándares establecidos por la W3C (organización internacional que desde hace unos 12 años se dedica al desarrollo de pautas y estándares web), lo que facilita la accesibilidad y la correcta visualización de las páginas en los navegadores que respeten dichos estándares.

Algunas de las ventajas de la maquetación con CSS:

- Separación de forma y contenido. Generalmente CSS y HTML se encuentran en archivos separados, lo que facilita el trabajo en equipo porque diseñador y programador pueden trabajar independientemente. Por otro lado, permite el acceso a distintos navegadores y dispositivos.
- Tráfico en el servidor. Las páginas pueden reducir su tamaño entre un 40% y un 60%, y los navegadores guardan la hoja de estilos en la caché, ésto reduce los costos de envío de información.
- Tiempos de carga. Por la gran reducción en el peso de las páginas, mejora la experiencia del usuario, que valora de un sitio el menor tiempo en la descarga.
- Precisión. La utilización de CSS permite un control mucho mayor sobre el diseño, especificando exactamente la ubicación y tamaño de los elementos en la página. También se pueden emplear medidas relativas o variables para que la pantalla o la caja contenedora se acomode a su contenido.
- Mantenimiento. Reduce notablemente el tiempo de mantenimiento cuando es necesario introducir un cambio porque se modifica un solo archivo, el de la presentación, sin tener que tocar las páginas que contienen la estructura con el contenido.
- Diseño unificado y flexibilidad. Es posible cambiar completa o parcialmente el aspecto de un sitio con sólo modificar la hoja de estilos. Por otro lado, el tener el estilo de una web en un solo archivo permite mantener la misma apariencia en todas las páginas.
- Posicionamiento. Las páginas diseñadas con CSS tienen un código más limpio porque no llevan diseño, sólo contenido. Esto es semánticamente más correcto y la página aparecerá mejor posicionada en los buscadores. Google navega obviando el diseño.

Recomiendo un sitio simpático y didáctico sobre el tema: <http://www.hotdesign.com/seibold/spanish/>

Artículo por *Serviweb*

Parte 5:

Posicionamiento

CSS y maquetación

A partir de la implantación de CSS el estándar se volvió la herramienta indicada para la maquetación de las páginas web. En los próximos artículos hablaremos sobre el posicionamiento de elementos en la página y sobre la maquetación de los contenidos atendiendo a cualquier distribución que el diseñador necesite.

5.1.- Maquetación CSS

Una introducción sobre cómo maquetar una página utilizando capas, que ilustra el funcionamiento y la potencia de las CSS con ejemplos.

En este artículo vamos a conocer la maquetación de páginas utilizando Hojas de estilos en cascada (CSS). Veremos cómo realizar este tipo de maquetación, junto con algunas ventajas e inconvenientes. Para muchos será todavía un campo por explorar. Aunque no vamos a entrar en grandes detalles, vamos a intentar dar a conocer la maquetación con CSS para que cubrir la posible laguna por parte del lector. En capítulos sucesivos ampliaremos la información y ofreceremos tutoriales más prácticos.

Como se ha podido aprender en el [Manual de CSS](#), las hojas de estilo en cascada ayudan a separar el contenido de la forma, es decir, los elementos que componen una página de la forma con la que se muestran. Además, CSS ayuda en gran medida a la definición de estilos en la página, ya que permite ajustar de una manera mucho más precisa cualquier aspecto de cualquier elemento de la página.

La maquetación con CSS lleva la utilización de las hojas de estilo a su grado máximo, de manera que cualquier definición del aspecto de la página se realiza en la declaración CSS que enlazamos con el documento HTML. Para definir la situación de los elementos en la página se utilizan las capas, a las que se aplica un posicionamiento a través también de las hojas de estilo.

Para crear las capas se utilizan etiquetas <DIV>, en las que se introducen los elementos que queremos que aparezcan en la página. Los elementos dentro de los <DIV> también se pueden anidar, para heredar las propiedades y posicionamiento de las capas padre.

En la maquetación por capas se definen únicamente etiquetas <DIV> y las tablas sólo se utilizan para mostrar información tabulada, es decir, mostrada en filas y columnas. Cabe señalar que en la maquetación tradicional se utilizan las tablas para ajustar la posición de los elementos en la página. Seguro que muchos de los lectores están muy familiarizados con el uso de tablas para maquetar una web, pues se trata de una técnica bastante sencilla, aunque bastante poco práctica y que complica un tanto el código de las páginas web resultantes.

5.1.1.- Ventajas e inconvenientes de la maquetación CSS

La maquetación por capas, también llamada maquetación CSS, es una forma de crear webs más evolucionada y que mejora en mucho a la maquetación tradicional (que antes se hacía por tablas), aunque también tiene sus inconvenientes.

Veamos primero algunas de las **ventajas** de la maquetación CSS:

- **La separación del contenido de la página y del estilo o aspecto con el que se deben mostrar.** Tener en cuenta que, cuanto más separemos estos dos elementos, más sencillo será el mantenimiento de las páginas y el procesamiento de la información. Con ello también podremos obtener páginas más limpias y claras.
- **Ahorro en la transferencia.** Si todos los estilos y posiciones de los elementos se introducen en un documento externo, liberaremos el código de la página y ocupará mucho menos. Como la declaración de estilos se almacena en la caché del navegador, sólo se transfiere en la primera página que se visita del sitio, con lo que la segunda y posteriores páginas que se soliciten se cargarán mucho más rápido.
- **Facilidad para alterar el aspecto de la página sin tocar el código HTML.** Como toda la información de los estilos y el posicionamiento de las capas se encuentra en un mismo archivo, si deseamos cambiar cualquier elemento de la página -ya sea su posición o su aspecto-, sólo tenemos que actualizar la hoja de estilos y los cambios se verán automáticamente en todo el web.

Como decimos, también hay algunas **desventajas**:

Actualizado: La verdad es que en la actualidad casi no podemos decir que existan desventajas en la maquetación CSS, salvo la dificultad que puedan encontrar los desarrolladores para aprender una cosa nueva. Actualmente todos los navegadores la soportan y la interpretan de una manera muy parecida, lo que facilita bastante la labor de desarrollo.

- **Compatibilidad con navegadores antiguos.** Se necesita que el visitante disponga de un navegador ~~bastante~~ **avanzado** actual. La mayoría de los visitantes disponen de navegadores que soportan características avanzadas de las CSS, pero todavía hay mucha gente que no ha actualizado sus equipos o que navega en sistemas de sólo texto. Los navegadores que no soportan hojas de estilos, por lo menos leerán el código de la página y lo mostrarán sin ningún posicionamiento. Ello puede resultar fastidioso, pero por lo menos visualizarán todos los datos de la página, aunque descolocados y sin estilo.
Actualizado: Es conveniente señalar que en la actualidad ya todos los navegadores soportan maquetación CSS, por lo que este inconveniente podemos descartarlo prácticamente.
- **Diferencias entre navegadores.** Dependiendo del navegador también cambian las etiquetas de estilos soportadas, por lo que las páginas puede que no se vean exactamente igual en unos clientes que en otros. También, al igual que ocurre con HTML, hay atributos no estándar o que tienen valores por defecto diferentes. Cuando se empieza con la maquetación en CSS, puede resultar un tema bastante complicado y crearnos bastantes dolores de cabeza, no obstante, se trata de, poco a poco, ir aprendiendo todos los atributos y los navegadores donde se visualizan o no.
Actualizado: Las diferencias entre navegadores en la actualidad se han minimizado bastante y lo cierto es que con unas pocas técnicas podremos diseñar páginas que se vean exactamente igual en cualquier navegador.
- **Dificultad.** Sin duda, si estamos acostumbrados al HTML, pasar a CSS resulta más complicado y requiere un estudio más profundo. Sin embargo, este paso nos brindará un mayor control de los elementos de la página y ampliará nuestras fronteras a la hora de maquetar.

5.1.2.- Ejemplo CssZenGarden, ilustra las posibilidades de la maquetación CSS

Existe un sitio muy interesante e ilustrador que nos puede ayudar a conocer rápidamente la potencia de las CSS y hacernos una idea de lo que puede significar su uso. Es un sitio donde se muestra un contenido y un diseño bastante logrado. Además, dispone de varios enlaces para poder ver el mismo sitio, con el mismo contenido, pero con distinto aspecto. De ese modo podemos ver cómo se puede llegar a alterar el diseño de una página con tan solo cambiar la hoja de estilos.

La URL del sitio es <http://www.csszengarden.com>. Es muy interesante que seleccionéis otros diseños para ver el cambio radical que puede tener las páginas web con distintas hojas de estilos.

Nosotros hemos explorado un poco las capacidades de las CSS y hemos realizado un ejemplo de diseño de CssZenGarden por nuestra cuenta. Podemos [verlo en nuestro propio servidor en este enlace](#).

5.1.3.- Por donde continuar para aprender a maquetar con CSS

En DesarrolloWeb.com hemos publicado una compilación de artículos sobre el tema en el [Manual de Maquetación CSS](#). En el manual podremos encontrar algunas [notas interesantes e introductorias para comenzar a maquetar con CSS](#), pero además diversos artículos prácticos que nos ayudarán a aprender con casos reales.

Prestar especial atención al taller de maquetación comienza con [Maquetar una página con CSS](#) y los artículos siguientes donde se continúa ese ejemplo. Además, para los desarrolladores más avanzados que deseen disponer de unas herramientas que ayuden a la maquetación más rápida, recomendamos estudiar los artículos del [framework css 960 Grid System](#).

Artículo por Miguel Angel Alvarez

5.2.- Formas de aplicar estilos en maquetación CSS

Repaso a los métodos por los que se pueden aplicar estilos a las páginas web mediante CSS.

Vamos a ver otra vez distintos modos de aplicar estilos a las páginas. Es un tema que ya vimos en el [manual de CSS](#), pero merece la pena refrescar conceptos y ampliar la información que se ofreció en su día.

5.2.1.- Aplicación de estilo a etiquetas

Se puede asignar el estilo a una etiqueta concreta de HTML. Para ello, en la declaración de estilos escribimos la etiqueta y entre llaves, los atributos de estilo que deseemos.

```
body {
    background-color: #f0f0f0;
    color: #333366;
}
```

Podemos aplicar el mismo estilo en un conjunto de etiquetas. Para ello, indicamos las etiquetas seguidas por comas y luego, entre llaves, los atributos que queramos definir.

```
h1, p{
    color: red;
}
```

En este caso se define que los encabezados de nivel 1 y los párrafos, tengan letra roja.

5.2.2.- Definición de clases

Podemos utilizar una clase si deseamos crear un estilo específico, para luego aplicarlo a distintos elementos de la página. Las clases en la declaración de estilos se declaran con un punto antes del nombre de la clase.

```
.miclase{
    color: blue;
}
```

Para asignar el estilo definido por una clase en un elemento HTML, simplemente se añade el atributo class a la etiqueta que queremos aplicar dicha clase. El atributo class se asigna al nombre de la clase a aplicar. Por ejemplo:

```
<p class="miclase">este párrafo tiene el estilo definido en la clase "miclase".</p>
```

El párrafo anterior se presentaría con color azul. La definición de clases y su utilización es sencilla, pero veamos un ejemplo más detallado:

Para la siguiente declaración de estilos:

```
body, td, p{
    background-color: #000000;
    color: #ffffff;
}

.inverso{
    background-color: #ffffff;
    color: #000000;
}
```

Se ha definido un fondo negro y color del texto blanco para el cuerpo de la página, así como las celdas y los párrafos. Luego se ha declarado una clase, de nombre "inverso", con los colores al revés, es decir, fondo blanco y texto negro.

```
<body>
<p>Hola esto es un parrafo normal</p>
<p class="inverso">Párrafo con los colores invertidos</p>
<table>
<tr>
    <td class="inverso">INVERSO</td>
    <td>NORMAL</td>
</tr>
</table>
</body>
```

Esta página tiene, generalmente, el fondo negro y el texto blanco. El primer párrafo, que es un párrafo normal, sigue esa definición general de estilos, pero el segundo párrafo, al que se ha aplicado la clase "inverso", tiene el fondo blanco y el texto en negro. Por lo que respecta a la tabla, en su primera celda se ha asignado la clase "inverso", por lo que se verá con fondo blanco y color de texto en negro. Mientras que la segunda celda, que no tiene asignada ninguna clase, se presentará como se definió en la regla general.

Para conocer los resultados obtenidos en el anterior ejemplo podemos [verlo en una página aparte](#).

5.2.3.- Estilos que sólo se utilizan una vez

También podemos tener un estilo específico para un único elemento, que no va a repetirse en ningún otro caso. Para ello tenemos los estilos asignados por identificador. Los identificadores se definen en HTML utilizando el atributo id en la etiqueta que deseamos identificar. El valor del atributo id será el que definamos nosotros.

```
<div id="capal">
```

En la hoja de estilos, para definir el aspecto de ese elemento con id único, se escribe el carácter almohadilla, seguido del identificador indicado en la etiqueta y entre llaves los atributos css que deseemos.

```
#capal{
    font-size: 12pt;
    font-family: arial;
}
```

En este caso se ha asignado fuente de tamaño 12 puntos y cuerpo arial.

Como se puede concluir en la lectura de estas líneas, generalmente se prefiere utilizar estilos definidos en clases a los definidos con identificadores, a no ser que estemos seguros que ese estilo no se va a repetir en todo el documento.

Referencia: En nuestro [taller de CSS](#) hemos publicado varios artículos para mostrar el [proceso de maquetación de una página en CSS](#).

Artículo por *Miguel Angel Alvarez*

5.3.- Flujo HTML y atributos CSS

El flujo HTML es el modo en el que se van colocando los componentes de la página, a partir de cómo aparecen en el código HTML y los atributos CSS de los elementos.

Merece la pena detenerse a explicar lo que es el flujo HTML, pues es un concepto sencillo y básico para poder entender muchos asuntos acerca del posicionamiento web y en concreto el posicionamiento con CSS.

El flujo de la página es algo así como el flujo de escritura de elementos dentro del lienzo que nos presenta el navegador. Sabemos que las páginas web son codificadas en HTML y los elementos aparecen en el código en una posición dada. El navegador, en el momento que interpreta el código HTML de la página, va colocando en la página los elementos (definidos por medio de etiquetas HTML) según los va encontrando en el mismo código.

Por ejemplo, pensemos en una página que tiene un titular con H1, luego varios párrafos y alguna imagen. Pues si lo primero que aparece en el código HTML es el encabezamiento H1, pues ese encabezado aparecerá en la página también en primer lugar. Luego se colocarán los párrafos y si la imagen aparecía en el código por último, en la página también aparecerá al final. Es decir, los elementos aparecen colocados tal como estén ordenados en el código. A esto se le llama el flujo HTML, la colocación de los elementos en el lugar que corresponda según su aparición en el código.

Esto en general ocurre con cualquiera de los elementos de la página. Sin embargo, hay algunos atributos HTML que pueden marcar distintas propiedades en el flujo, como que una imagen se alinee a la derecha, con `align="right"`, con el texto del párrafo que pueda haber a continuación rodeando la imagen. Pero con HTML, si por ejemplo, una imagen va antes que un párrafo, nunca vamos a poder intercambiar sus posiciones y colocar la imagen detrás del párrafo que le sigue en el código.

Esto no ocurre de igual manera cuando trabajamos con CSS, puesto que existen diversos atributos que pueden cambiar radicalmente la forma en la que se muestran en la página, por ejemplo el atributo `position` que puede definir valores como `absolute`, que rompe el flujo de la página, o mejor dicho, saca del flujo de la página al elemento que se le asigna.

5.3.1.- Comportamientos inline y block y cómo afectan al flujo de la página

Cuando tratamos con etiquetas, existen dos modos principales de comportamiento. Etiquetas como una imagen, o una negrita, que funcionan en línea ("`inline`"), es decir, que se colocan en la línea donde se está escribiendo y donde los elementos siguientes, siempre que también sean "`inline`" se posicionan todo seguido en la misma línea. Tenemos por otra parte los elementos que funcionan como bloque ("`block`") que implican saltos de línea antes y después del elemento. Por ejemplo, los párrafos o encabezamientos son elementos con comportamiento predeterminado tipo "`block`".

Dos etiquetas muy utilizadas en la [maquetación CSS](#) son DIV y SPAN. Una de las diferencias principales es que DIV funciona con comportamiento "`block`" y SPAN funciona como "`inline`". En realidad este es el comportamiento por defecto, puesto que nosotros con CSS en cualquier momento podemos cambiarlo por medio del atributo `display`. Por ejemplo:

```
<div style="display: inline;">
Este elemento funcionará en línea
</div>
```

O bien:

```
<span style="display: block;">
Este span ahora funciona como bloque
</span>
```

Realmente ambas posibilidades funcionan dentro del flujo HTML normal, así que, tanto los elementos `display inline` como `display block`, se encuentran dentro del flujo HTML estándar, la única diferencia es que los bloques se escriben en líneas independientes, es decir, con saltos de línea antes y después del elemento, así como una cantidad de margen arriba y abajo que depende del tipo de elemento de que se trate.

5.3.2.- Atributo CSS Float y el flujo

Otro atributo que afecta al fluir de los elementos en la página es el atributo float de CSS, que se utiliza bastante en la maquetación web. Este atributo podemos utilizarlo sobre elementos de la página de tipo "block" y lo que hace es convertirlos, en "flotantes" que es un comportamiento parecido a lo que sería el mencionado anteriormente "inline". Con float podemos indicar tanto left como right y conseguiremos que los elementos se posicionen a la izquierda o a la derecha, con el contenido que se coloque a continuación rodeando al elemento flotante. La diferencia es que los elementos continúan siendo tipo "block" y aceptan atributos como el margen (atributo CSS margin), para indicar que haya un espacio en blanco a los lados y arriba y abajo del elemento.

Por ejemplo, los elementos de las listas (etiqueta LI) son por defecto de tipo "block", por eso aparecen siempre uno abajo de otro, en líneas consecutivas. Pero nosotros podríamos cambiar ese comportamiento con:

```
li{
    float: right;
}
```

Así, una lista como esta:

```
<ul>
<li>Elemento1</li>
<li>Elemento2</li>
<li>Elemento3</li>
</ul>
```

Veríamos como el primer elemento aparece a la derecha del todo y los otros elementos van colocándose en la misma línea en el siguiente espacio libre que haya. Así, el segundo elemento se colocaría en la misma línea, todo a la derecha que se puede, conforme al espacio que se tenga en el contenedor donde estén colocados.

5.3.3.- Flujo y el atributo position

El atributo position de CSS sí que es capaz de cambiar radicalmente el flujo de los elementos de la página. Este atributo, que explicaremos con detalle más adelante en otros artículos de DesarrolloWeb.com, por defecto tiene el valor "static", que indica que el elemento forma parte del flujo HTML normal de la página.

Sin embargo, con el atributo CSS position, podemos indicar otros valores que hacen que los elementos salgan del flujo HTML y se posicionen en lugares fijos, que no tienen que ver con la posición en la que aparezcan en el código HTML. Por ejemplo:

```
<div style="position: absolute; top: 10px; left: 100px;">
Este elemento tiene posicionamiento absoluto
</div>
```

Hace que ese elemento quede fuera del flujo de elementos en la página y entonces aparecería en el lugar que se indica con los atributos top y left (top indica la distancia desde la parte de arriba y left la distancia desde el borde izquierdo). Los otros elementos que formen parte del flujo de la página no quedan afectados por los elementos con posicionamiento absoluto.

Otro valor para el atributo position que hace que los elementos queden posicionados fuera del fluir normal de elementos en la página es "fixed", cuyo comportamiento veremos más adelante en otros artículos. Recomendamos seguir la lectura, para las personas que quieran profundizar en este tema, a partir del artículo [Posicionamiento CSS](#).

Artículo por Miguel Angel Alvarez

5.4.- Posicionamiento CSS

Las hojas de estilo en cascada incorporan múltiples formas para posicionar elementos en la página, lo que comúnmente se conoce como posicionamiento CSS.

En el Manual de CSS de DesarrolloWeb.com hemos tratado en varios puntos el posicionamiento CSS, con las distintas técnicas para crear y colocar partes de la página de manera absolutamente precisa. En todos los casos explicamos diversos detalles desde distintos enfoques que sin duda dan un visión particular sobre el posicionamiento de elementos en páginas web. Por ejemplo, ya hemos hablado sobre lo que son [las capas](#), y también los [atributos para su posicionamiento](#), así como otros asuntos como [la maquetación CSS](#) o el [atributo overflow](#).

No obstante, quedaba pendiente ofrecer unas explicaciones generales y con detalle sobre el posicionamiento CSS, que puedan dar a los lectores una idea global sobre este interesante asunto. Sin duda es un tema que merece la pena estudiar y también practicar. En este artículo vamos a ofrecer diversos conocimientos teóricos y a la vez estamos preparando un vídeo en el que mostraremos por la práctica las distintas opciones para posicionamiento web.

5.4.1.- Atributos para posicionamiento CSS

Existen numerosos atributos para posicionar con CSS cualquier elemento de la página. Además, a medida que van siendo presentadas nuevas versiones de CSS, estos atributos y sus posibles valores van aumentando. En CSS 2 contamos con diversos atributos que veremos a continuación.

Atributo position:

Este atributo es, digamos, el principal para definir el tipo de posicionamiento de un elemento. Merece la pena verlo por separado y en detalle. Más adelante lo trataremos en el artículo [Tipos de posicionamiento con el atributo position](#), pero adelantamos que va a permitir varios valores para establecer cómo se posicionará el elemento en la página y si formará parte del [flujo normal de HTML](#). Sus valores posibles son absolute, fixed, relative, static e inherit.

Atributos top, left, right, bottom:

Sirven para indicar la posición de un elemento, cuando éste tiene los valores de position "absolute", "relative" o "fixed" (en otros valores del atributo position estos atributos son ignorados). El atributo top indica la distancia desde el borde superior de la página y left desde el borde de la izquierda. También se puede indicar opcionalmente la posición con bottom, que es la distancia desde abajo y right, que es la distancia desde la derecha.

Atributos float y clear:

Float sirve para establecer que un elemento tiene que "flotar", colocándose los valores "right" o "left" para que floten a izquierda o derecha. Por si sirve de aclaración, que los elementos floten es algo así como lo que pasa cuando definimos el atributo HTML align="right" o align="left" en las imágenes o tablas. Con el atributo clear hacemos que el elemento se coloque en el primer área libre que tenga al lugar donde se indique. Por ejemplo el valor de clear "right" hace que el elemento se coloque en el primer lugar donde no tenga ningún elemento flotando a la derecha. El valor de clear "both" hace que el elemento se coloque donde no tenga elementos flotando, tanto a la derecha como a la izquierda.

Atributo clip:

Establece un área de recorte de la porción visible de un elemento. Este área de recorte se establece con varios valores, como se detalla en el artículo [atributos para capas](#).

Atributo display:

Especifica el tipo de caja que debe tener un elemento, que puede ser de diversas formas. Este atributo también tiene bastante utilización y entre los valores más corrientes podríamos destacar: "none", que hace que esa caja o elemento no aparezca en la página ni se reserve espacio para ella. "block", que sirve para que la caja sea un bloque y se muestre en una línea o líneas independientes de otros elementos de la página. "inline", que indica que esa caja tiene que mostrarse en la misma línea que otros elementos escritos antes o después.

Atributo overflow:

Este atributo sirve para decir qué es lo que pasa con los elementos que no caben en una caja debido a las dimensiones de la misma y del contenido que tenga. Se explica con detalle en el artículo [Overflow en CSS](#).

Atributo visibility:

Atributo para definir la visibilidad de un elemento. Con este atributo podemos decir que ciertos elementos de la página sean visibles o invisibles, pero atención, aunque un elemento sea invisible, continúa ocupando espacio en la página. Si queremos que no sea invisible y no se le reserve espacio en la página, hay que utilizar el atributo display con el valor "none". Los valores más corrientes de visibility son: "visible", que hace que el elemento se vea (valor por defecto) y "hidden", que hace que el elemento sea invisible, aunque continúe ocupando espacio.

Atributo z-index:

Este atributo tiene como valor cualquier número entero. Sirve para indicar qué capa se tiene que ver por encima o por debajo de otra u otras, en caso que varias capas estén superpuestas. A mayores valores de z-index, la capa se coloca más al frente, tapando otras capas que tengan valores menores de z-index.

Este ha sido un repaso general a los distintos atributos de hojas de estilo que están implicados en lo que se conoce como posicionamiento en CSS. Para la referencia de los interesados, recomendamos la lectura de los artículos mencionados al principio del [Manual de CSS](#), en especial el artículo sobre [atributos para capas](#).

En el [siguiente artículo](#) veremos distintos casos de uso del atributo position, que es clave para entender el posicionamiento CSS.

Artículo por [Miguel Angel Alvarez](#)

5.5.- Tipos de posicionamiento con el atributo position de CSS

Explicamos los distintos valores que puede tener el atributo position de CSS, con detalle.

En lo que se conoce como posicionamiento CSS, el atributo de hojas de estilo en cascada que más importancia tiene es el position. Vamos a dedicar todo el presente artículo a explicar los distintos valores de position, para explicarlos y proponer unos ejemplos que esperamos acaben de aclarar los posibles valores que puede tomar.

Recordemos que en el artículo anterior de este [manual de CSS](#) ya se introdujo el concepto de [posicionamiento CSS](#) y se vio un listado de los atributos existentes hasta CSS 2 para realizar posicionar elementos en la página. Pasemos entonces al tema en cuestión, viendo las posibilidades que nos ofrece este lenguaje. Además, a lo largo de este artículo de DesarrolloWeb.com, vamos a mencionar repetidas veces un concepto que también se explicó anteriormente: [el flujo del HTML normal](#).

5.5.1.- position: static

Es el valor predeterminado del atributo y el posicionamiento normal de los elementos en la página. Quiere decir que los elementos se colocarán según el flujo normal del HTML, es decir, según estén escritos en el propio código HTML. Por decirlo de otra manera, static no provoca ningún posicionamiento especial de los elementos y por tanto, los atributos top, left, right y bottom no se tendrán en cuenta.

Podemos ver un ejemplo de posicionamiento static:

```
<div style="position: static; background-color: #ff9; padding: 10px; width: 300px;">Esto es una capa con posicionamiento estático</div>
<div style="position: static; background-color: #f9f; padding: 10px; width: 500px;">posicionamiento static, predeterminado.</div>
<h1>CSS</h1>
<div style="background-color: #9ff; padding: 10px; width: 400px;">Posicionamiento static, aunque en este caso no se indicó el atributo position static, pues no hace falta.</div>
```

[Puede verse en una página aparte.](#)

5.5.2.- position: absolute

El valor absolute en el atributo position permite posicionar elementos de manera absoluta, esto es de manera definida por valores de los atributos top, left, bottom y right, que indican la distancia con respecto a un punto. Las capas o elementos con posicionamiento absoluto quedan aparte del flujo normal del HTML, quiere decir esto que no se afectan por el lugar donde aparezcan en el código HTML y tampoco afectan ellas a otros elementos que sí que formen parte del flujo normal del HTML.

Los valores top, left, bottom y right se expresan con [unidades CSS](#) y son una distancia con respecto al primer elemento contenedor que tenga un valor de position distinto de static. Si todos los contenedores donde esté la capa posicionada

con position absolute (todos sus padres hasta llegar a BODY) son static, simplemente se posiciona con respecto al lado superior de la página, para el caso de top, el inferior para bottom, del lado izquierdo para left o el derecho, en el caso de utilizar right.

Veamos el siguiente código HTML en el que hemos preparado varias capas con position absolute, pero con distintas características:

```
<div style="position: absolute; width: 300px; height: 140px; top: 100px; left: 30px; background-color: #ff8800; color: #fff; padding: 15px; z-index: 2;">
Esta capa tiene posicionamiento absoluto.
<br>
<br>
Me permite especificar top y left para colocarla con respecto a la esquina superior izquierda.
</div>

<div style="position: absolute; width: 820px; height: 30px; padding: 10px; background-color: #ddf; top: 150px; left: 10px; z-index: 1;">Posicionamiento absoluto con z-index menor (la capa aparece por debajo de otras que se superponen con z-index mayor.</div>

<div style="position: absolute; width: 100px; height: 20px; padding: 10px; background-color: #ddf; bottom: 10px; right: 10px;">Posicionamiento absoluto con atributos bottom y right</div>

<h1>Posicionamiento CSS</h1>
```

Podemos [ver el ejemplo en una página aparte](#).

La primera capa (llamamos así a los elementos DIV que tienen posicionamiento CSS), tiene como todas las del ejemplo, posicionamiento absoluto. Los atributos top: 100px y left: 30px quieren decir que se posiciona a 100 píxeles de la parte superior de la página y a 30 píxeles de la izquierda. En este caso las distancias top y left para ubicar la capa con position absolute son relativas a la esquina superior izquierda del área disponible del navegador, pues esta capa no está dentro de ninguna otra con posicionamiento distinto de static. Cabe llamar la atención en esta primera capa también sobre el atributo z-index: 2, que servirá para indicarle al navegador la posición de la capa, en la tercera dimensión, con respecto a otras que se puedan superponer, para que sepa cuál tiene que estar debajo y cuál arriba.

La segunda capa podemos ver que tiene un z-index: 1. Eso quiere decir, que en caso se posicione en el mismo lugar se ocultará por la capa primera, que tiene un z-index mayor.

En la tercera capa hemos probado el posicionamiento utilizando los atributos bottom y right, así que la estamos posicionando con respecto a la esquina inferior derecha.

Veamos un segundo ejemplo donde vamos a colocar una capa con posicionamiento absoluto y dentro varias capas también posicionadas con absolute.

```
<div style="position: absolute; top: 100px; left: 200px; background-color: #ff9966; width: 400px; height: 100px;">
<div style="position: absolute; top: 10px; left: 10px;">
Uno
</div>
<div style="position: absolute; top: 10px; left: 100px;">
Dos
</div>
<div style="position: absolute; top: 10px; left: 200px;">
Tres
</div>
</div>
```

En este caso la primera capa, que no está dentro de ninguna otra, se posiciona con top y left con respecto a la esquina superior izquierda del espacio disponible en el navegador para el cuerpo de la página. Las capas anidadas están también con position: absolute, pero al estar dentro de otra capa que tiene posicionamiento distinto de static, sus valores top y left son relativos a la esquina superior izquierda de la capa que las contiene.

Podemos [ver el ejemplo en marcha en una página aparte](#).

5.5.3.- position: relative

El valor relative en el atributo position indica que la capa sí forma parte del flujo normal de elementos de la página, por lo que su posición dependerá del lugar donde esté en el código y el flujo HTML. Además, las capas con posicionamiento relative, admiten los valores top y left para definir la distancia a la que se colocan con respecto al punto donde esté en ese momento el flujo normal del HTML. Como afectan al mencionado flujo del HTML, los elementos colocados después de las capas relative, tendrán en cuenta sus dimensiones para continuar el flujo y saber dónde colocarse. Sin embargo, no se tendrá en cuenta los top y left configurados.

Veamos un ejemplo que quizás aclare las cosas.

```
<h1>Hola</h1>
<div style="background-color: #606; color:#ffc; padding:10px; text-align: center; width:
300px;">Hola esto es una prueba</div>
<div style="position: relative; width: 300px; padding: 10px; background-color: #066; color:#ffc;
top:100px; left: 30px;">capa de posicionamiento relative<br>Se tiene en cuenta esta capa para
posicionar las siguientes.</div>
<h2>hola de nuevo!</h2>
```

Podemos [ver la página en marcha](#).

Las etiquetas H1 y H2 respetan el flujo HTML y también tenemos un elemento DIV donde no se ha especificado nada en position, luego es static y por tanto también es afectada por el flujo. Hay una capa relative, en el segundo elemento DIV, que también se posiciona con respecto al flujo normal. Como tiene un top y left, aparece un poco desplazada del lugar que le tocaría con respecto al flujo.

El último H2 que aparece se coloca teniendo en cuenta al flujo y tiene en cuenta la capa relative, por eso deja un espacio en blanco arriba, pero no atiende a la posición real de ésta, que se marcó con los atributos top y left.

5.5.4.- position: fixed

Este atributo sirve para posicionar una capa con posicionamiento absoluto, pero su posición final será siempre fija, es decir, aunque se desplace el documento con las barras de desplazamiento del navegador, siempre aparecerá en la misma posición.

El lugar donde se "anclará" la capa siempre es relativo al cuerpo (el espacio disponible del navegador para la página). Si utilizamos top y left, estaremos marcando su posición con respecto a la esquina superior izquierda y si utilizamos bottom y right su posición será relativa a la esquina inferior derecha.

Veamos un ejemplo.

```
<div style="position: fixed; width: 300px; height: 140px; top: 100px; left: 30px; background-color:
#ff8800; color: #fff; padding: 15px;z-index: 1;">
Esta capa tiene posicionamiento fixed.
<br>
<br>
Me permite especificar top y left para colocarla con respecto a la esquina superior izquierda.
</div>

<div style="position: fixed; width: 700px; height: 30px; padding: 10px; background-color: #d0f; top:
150px; left: 10px; z-index: 2;">Posicionamiento fixed</div>
<h1>Hola</h1>
<div style="position: fixed; width: 100px; height: 30px; padding: 10px; background-color: #0df;
bottom: 10px; right: 10px; z-index: 4;">Posicionamiento fixed</div>
<br>
<br>
<br>
<br>
Pongo texto para que se vea!!
<br>
<br>
<br>
Esto hace desplazamiento, con tanto br
<br>
<br>
...
```


Se puede [ver la página en marcha con este código](#).

Se puede ver que hay varias capas con position: fixed y un montón de BR para que la página pueda tener un desplazamiento. Si vemos la página en marcha y hacemos scroll hacia abajo con la barra de desplazamiento, veremos que las capas fixed siempre mantienen la misma posición.

Nota: El valor fixed en el atributo position funciona en todos los navegadores, pero en el caso de Internet Explorer sólo funciona en la versión 7 y superiores. Además, para que funcione en Explorer tiene que declararse un DOCTYPE!. Servirían varios tipos de DOCTYPE!, sin embargo debería declararse con el formato completo. Algo así como:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

O por poner otro ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

5.5.5.- position: inherit

El valor inherit indica que el valor de position tiene que heredarse del elemento padre. No funciona en Explorer, al menos hasta la versión 8. Tiene en verdad poca utilidad y además, como no funciona en el navegador más utilizado en la actualidad, tiene aun menos sentido usarlo. Por ello, no ponemos ejemplos.

5.5.6.- Conclusión sobre el atributo position de CSS

Esperemos que con las anteriores explicaciones y ejemplos se hayan podido entender bien las distintas posibilidades del atributo position, que es sin duda clave para el posicionamiento CSS. Lo más común para la maquetación web es utilizar el posicionamiento static, pero el posicionamiento absoluto, junto con el posicionamiento fixed, e incluso relative, pueden ser muy útiles para diseños más complejos, donde se requiera una mayor precisión en la colocación de los distintos elementos o las capas.

Además, para hacer efectos Javascript y DHTML en general, se utilizan frecuentemente posicionamientos absolutos. Son muy útiles porque permite que los elementos dinámicos no formen parte del flujo normal del HTML, y por tanto, podemos situarlos en cualquier lugar el área disponible del navegador, e incluso moverlos dinámicamente al cambiar sus propiedades top y left mediante scripts en el lado del cliente. Todos estos comportamientos dinámicos quedan ya fuera de la temática de este texto, aunque los explicamos con detalle en distintos manuales de la sección [Javascript a Fondo](#), de DesarrolloWeb.com.

Artículo por Miguel Angel Alvarez

5.6.- EL atributo Overflow de CSS

Explicamos esta propiedad interesante contemplada en la especificación CSS 2.

En este artículo del [Manual de CSS](#) de DesarrolloWeb.com vamos a explicar una propiedad interesante de CSS, contemplada en la especificación CSS 2, Overflow. Es un atributo de esos que nos servirán para maquetar las capas de una web de una manera más versátil y detallada.

Overflow sirve en el modelado de cajas para indicar al navegador qué es lo que debe hacer con el contenido que no cabe dentro de una capa, según las dimensiones que se le han asignado.

Como sabemos, a las capas (elementos DIV) podemos asignarles un tamaño, en anchura y altura. Pero muchas veces el contenido que colocamos en la capa sobrepasa el espacio que hemos destinado a ella. Entonces lo que suele ocurrir es que la capa crece lo suficiente para que el contenido colocado dentro quepa. Habitualmente las capas crecen en altura, por lo que a más contenido más tamaño tendrá en altura. Este es un comportamiento que podemos alterar con el uso del atributo overflow.

Dicho de otro modo, overflow permite que se recorte el contenido de una capa, para mostrar únicamente el contenido que quepa, según sus dimensiones. Para acceder al contenido que no se muestra, porque no cabe en la capa, se puede configurar overflow para que aparezcan unas barras de desplazamiento.

Así pues, pasemos directamente a ver cuáles son los atributos posibles con el atributo overflow:

- **visible:** Este valor indica que se debe mostrar todo el contenido de la capa, aunque no quepa en tamaño con la que la hemos configurado. En Internet Explorer ocurre que la capa crece en tamaño lo suficiente para que quepa todo el contenido que hemos colocado dentro. En Firefox ocurre que la capa tiene el tamaño marcado, pero el contenido se sigue viendo, aunque fuera del espacio donde de la capa, pudiendo superponerse a un texto o imagen que hubiera debajo. El contenido no se recorta en caso alguno, es decir, siempre estará visible.
- **hidden:** Este valor indica que los contenidos que, por el tamaño de la capa, no quepan en la misma, se deben recortar. Por ello, la capa tendrá siempre el tamaño configurado, pero los contenidos en ocasiones podrán no verse por completo.
- **scroll:** Este valor indica que la capa debe tener el tamaño que se haya configurado inicialmente y que además se deben mostrar unas barras de desplazamiento, para mover el contenido de la capa dentro del espacio de la misma. Las barras de desplazamiento siempre salen, se requieran o no.
- **auto:** Con este valor también se respetarán las dimensiones asignadas a una caja. El contenido será recortado, pero aparecerán las barras de desplazamiento para moverlo. Sin embargo, en este caso las barras de desplazamiento podrán salir o no, depende de si son necesarias o no para ver todo el contenido de la capa.

Así pues, el atributo overflow nos permitirá tener un mayor control sobre los espacios que destinamos a cada caja de nuestro diseño. Es muy utilizado para mostrar textos largos, que se desean integrar dentro de otro texto o una interfaz donde no tenemos espacio disponible para colocarlos o no deseamos que crezcan más de la cuenta. Por ejemplo para mostrar código fuente dentro del texto de un artículo, como sigue:

```
<html>
<head>
  <title>Título</title>
</head>

<body>

Cuerpo...

</body>
</html>
```

Este ejemplo, habremos podido apreciar la barra de desplazamiento vertical, se obtiene con un atributo overflow: auto;. El código utilizado es como sigue:

```
<div style="overflow: auto; width: 300px; height: 100px; background-color:#ededed; border: 1px solid #990000;">
CONTENIDO...
</div>
```

Ahora veamos otro ejemplo, en el que simplemente se recorta el texto que no cabe en la capa. Hemos indicado overflow: hidden, por lo que el texto que sobra no se va a visualizar.

Esta capa tiene un contenido mayor del que cabe en el espacio que he asignado con el atributo width y height. Como le he puesto overflow: hidden, lo que ocurrirá es que parte del texto que estoy colocando no se va a ver...

En este caso vemos como el texto aparece recortado, porque no cabe en el espacio asignado de la capa. El código sería como el que sigue:

```
<div style="overflow: hidden; width: 200px; height: 50px; border: 1px solid #990000;">  
CONTENIDO...  
</div>
```

Aquí se pueden [ver varios ejemplos de uso de overflow](#).

Artículo por Miguel Angel Alvarez

Parte 6:

Trucos y consejos avanzados sobre CSS

Algunos trucos y consejos fundamentales que debemos conocer si deseamos realizar un uso de CSS medianamente avanzado.

6.1.- Trucos CSS para no enloquecer

Algunas soluciones a los problemas típicos que te puedes encontrar con CSS.

Tranquilo! Todavía no tires el monitor contra la pared!! te aseguro que con un poco de investigación y algunos consejos podrás encontrar la solución a tu problema.

Aquí encontrarás los principales trucos CSS para hacer frente a los típicos problemas que se enfrentan los diseñadores web cuando maquetan con CSS. Podrán existir discrepancias entre los lectores, pero aclaro que estas son técnicas que a mi personalmente me han dado resultado, después de muchas pruebas e intentos aprendí esto...

6.1.1.- Usa un contenedor global para todas las cajas (cuando las cosas se disparan)

De esta forma estas prefijando globalmente el orden de todas las demás cajas. En referencia a este contenedor ordena el resto de las cosas interiores. Es como si haces una cerca o valla para que nada es escape. Obviamente estamos hablando de sitios fijos'no elásticos.

A veces es bueno usar un contenedor hasta el cuerpo del sitio, luego dejar el pié afuera.

Ejemplo para un contenedor de 900px centrado:

```
#contenedor {
margin-top: 0px;
margin-right: auto;
margin-bottom: 0px;
margin-left: auto;
width: 900px;
}
```

6.1.2.- Que flote a la izquierda (cuando las cajas se superponen)

Esta es una muy buena forma de evitar incompatibilidades entre navegadores. El uso de hacks de CSS se debía en gran parte porque se trabajaba centrando las cajas. Si por ejemplo precisas poner tres cajas de 300px en un contenedor de 900px puedes hacer lo siguiente.

Ejemplo:

```
#caja {
float: left;
width: 300px;
}
```

6.1.3.- Calcular bien los paddings o rellenos (cuando las cajas se van abajo)

Casi todos los dolores de cabeza y maldiciones hechas sobre el CSS se deben al mal uso o a la mala interpretación que se hace del padding. Pero es más simple de lo que parece.

¿Para que sirven los paddings o rellenos? Bueno, lo que hace es generar un relleno de determinada medida para dar por ejemplo como un margen a los elementos, pero lo hace sobre el ancho en píxeles que esté prefijado. Por ejemplo: si tenemos una caja de 300px y le aplicamos un padding de 10px en la izquierda, ahora tendremos una caja de 310px. Esto hará desbordar al resto de las cajas y las desplazarán para abajo. Ahí es cuando el diseñador principiante se vuelve loco. El tema es que si hay una diferencia de hasta un 1px se producirán estos desbordes, sino fijate cuando le incluyes bordes a tu caja, se producirán diferencias.

Lo que se debe hacer es simple, calcular bien y recordar cada ajuste que se haga de los rellenos. Ahora tendremos que hacer una caja de 290px con paddings de 10px a la izquierda.

Ejemplo:

```
#caja {
float: left;
width: 290px;
padding-left: 10px;
background-color: #FFE6DD;
}
```

6.1.4.- El pié de página con ancho fijo (cuando el pié de página enloquece):

Para entender mejor como funciona el uso de cajas en CSS se puede pensar en un grupo de objetos de diferentes formas que luchan por adaptarse y ocupar el espacio que se ha prefijado. Sucede muchas veces que los pié de página son los más problemas traen cuando se maqueta un sitio. O se va para arriba, se alinea a la izquierda, o se desborda, etc.

Muchos resolvíamos este tema prefijando valores fijos a las alturas de cajas, pero no tiene sentido. Lo que se debe hacer es de nuevo establecer un valor de ancho fijo. De esta forma el pié se va a hacer su lugar del resto e irá a parar donde tiene que ir.

Ejemplo:

```
#pie {
width: 900px;
background-color: #666666;
}
```

No todo es 1+1=2 en CSS (cuando los anchos no cierran)

Un problema común en CSS es pensar que todos los anchos entre cajas cierran perfectamente. A veces es necesario jugar con los valores de los contenedores, a veces contrario a la lógica hay que añadir algunos px a los contenedores.

6.1.5.- Otros trucos rápidos

Trucos sencillos, de los que no hace falta explicar mucho pero que son muy prácticos y te harán más fácil el trabajo y evitarán posibles errores.

- Usa colores diferentes para distinguir las cajas
- Pon una palabra descriptiva en cada caja
- Comenta el código fuente y señala los finales de los contenedores grandes
- No mezquines espacios entre los divs
- No seas un fundamentalista y no quieras escribir tu CSS con dos o tres líneas. Si no quieres errores escribe lo necesario.
- Cuidado con el tamaño de las imágenes que insertas, estas cambian el ancho de los contenedores.
- Elige bien los nombres de cada div y trata de ser ordenado en el código.
- Si vas a trabajar con varias cajas, trata de agruparlas de a grupo, esto es muy importante. Por ejemplo un contenedor que agrupe tres o cuatro cajas.

6.1.6.- Conclusión

Todas estos párrafos son simplemente algunas sugerencias o comentarios de lo que me ha dado resultado a mi. Existen otras muchas ataduras de este tipo, si tienes alguna no dudes en comentarlas en este mismo artículo.

Que pasa cuando no puedes resolver un problema con CSS o similar? A mi me ha dado resultado levantarme un rato, hacer cualquier otra cosa y luego volver e intentar de nuevo.

Dejar de renegar y no enloquecer con CSS dependerá de la cantidad de tiempo, trabajo y esfuerzo que le metas a tu trabajo. No lo dudes.

Artículo por [Leonardo A. Correa](#)

6.2.- Lo que nadie te contó de la propiedad z-index

Analizamos la propiedad CSS z-index para intentar descifrar su correcto uso más allá del habitual conocimiento que tenemos de ella, normalmente consistente en saber que las capas con mayor valor de z-index se colocan delante de las de valor menor.

El problema con z-index es que muy poca gente comprende cómo funciona realmente. No es complicado, pero si nunca te has tomado la molestia de leer su especificación, hay ciertamente algunos aspectos cruciales que has pasado por alto.



¿No me crees? Bien, veamos si puedes solventar este problema:

6.2.1.- El problema

En el siguiente HTML tienes tres elementos `<div>`, y cada `<div>` contiene un único elemento ``. Cada `` tiene un color de fondo — rojo, verde y azul respectivamente. Cada `` está también posicionado absolutamente cerca de la posición superior izquierda del documento, escasamente solapando a los otros elementos `` así que puedes ver cuáles están apilados en frente de cada cual. El primer `` tiene un valor z-index de 1, mientras que los otros dos no tienen ningún valor z-index fijado.

Aquí tenemos el HTML y CSS básicos para entenderlo. También he incluido una *demo* visual (vía [Codepen](#)) con el código que se muestra a continuación:

```

<div>
<span class="rojo">Rojo</span>
</div>
<div>
<span class="verde">Verde</span>
</div>
<div>
<span class="azul">Azul</span>
</div>
.rojo, .verde, .azul {
position: absolute;
}
.rojo {
background: red;
z-index: 1;
}
.verde {
background: green;
}
.azul {
background: blue;
}
    
```

Puede verse el [ejemplo completo aquí](#).



Aquí está el desafío: intenta ver si puedes hacer que el rojo se apile detrás del azul y del verde sin romper ninguna de las siguiente reglas:

- No alteres el HTML de ninguna manera.
- No añadas o cambies la propiedad z-index de ningún elemento.
- No añadas o cambies la propiedad de posicionamiento de ningún elemento.
- Para que puedas hacerte una idea clara, cliquea en "edit" en el enlace de Codepen que hay arriba y juega un momento con el código. Si tienes éxito, debería quedarte algo como lo que sigue. [Prueba aquí](#).

Advertencia: No cliques en la pestaña de CSS o te dará inmediatamente la respuesta.



6.2.2.- La solución

La solución es añadir un valor de opacidad menor que 1 en el primer <div> (el padre del rojo). Aquí está el CSS que se añade al Codepen:

```

div:first-child {
opacity: .99;
}
    
```

Si te estás rascando la cabeza ahora mismo en estado de shock y te muestras estupefacto por el hecho de que esa opacidad haya causado efecto en cómo los elementos se apilan los unos respecto a los otros, bienvenido al club. Me pasó lo mismo cuando tropecé la primera vez con el mismo problema.

Tranquilo, el resto del artículo dejará las cosas un poco más claras.

6.2.3.- Ordenar el apilamiento

Z-index parece sencillo: elementos con un z-index mayor son apilados delante de los elementos con un menor z-index, ¿no? Bien, de hecho, **no**. Esto es parte del problema del z-index. Parece sencillo, así que los desarrolladores no dedican

tiempo a leer sus reglas.

Cada elemento en un documento HTML puede estar delante o detrás de otro elemento en el documento. Esto se conoce como el orden de apilamiento. Las reglas que determinan este orden están bonita y claramente definidas en la especificación, pero como he constatado, no son completamente entendidas por la mayoría de desarrolladores.

Cuando el z-index y las propiedades de posición no están declaradas, las reglas son muy simples: básicamente, el orden de apilamiento es el mismo que el orden de aparición en el HTML. (DE ACUERDO, es de hecho un [poco más complicado](#) que eso, pero si no estás usando márgenes negativos para solapar elementos en la misma línea de posición, probablemente no te encontrarás con elementos problemáticos).

Cuando introduces la propiedad de posición en la mezcla, cualesquiera elementos posicionados (y sus hijos) se muestran delante de elementos no posicionados (decimos que un elemento está “posicionado” cuando tiene un valor de posición distinto de “static”, como “relative”, “absolute”, etc.)

Finalmente, cuando el z-index se involucra también, las cosas se complican un poco. Al principio es natural asumir que los elementos con valores z-index mayores están delante de los elementos con valores z-index menores, y que cualquier elemento con z-index está en frente de los elementos que no tienen z-index, pero no es tan simple. Lo primero de todo, z-index solo funciona con elementos posicionados. Si intentas asignarle un z-index a un elemento sin posición especificada, no hará nada. En segundo lugar, los valores z-index pueden crear contextos de apilamiento, y ahora de pronto lo que parecía sencillo es bastante más complicado.

6.2.4.- Contextos de apilamiento

Grupos de elementos con un padre común que se mueven hacia adelante o hacia atrás juntos en el orden de apilamiento, se integran en lo que es conocido como contexto de apilamiento. Una completa comprensión de los contextos de apilamiento es la llave para realmente captar cómo funciona el z-index y el orden de apilamiento.

Cada contexto de apilamiento tiene un elemento HTML como su elemento raíz. Cuando un nuevo contexto de apilamiento se forma en un elemento, ése contexto confina todos sus elementos hijos en un lugar particular del orden. Eso significa que si un elemento es contenido en un contexto de apilamiento en lo más bajo del orden, no hay manera de que aparezca delante de otro elemento en un contexto de apilamiento que está en un nivel superior en el orden, ¡incluso con un z-index de mil millones!

Los nuevos contextos de apilamiento pueden ser formados en un elemento de tres maneras:

- Cuando un elemento es el elemento raíz de un documento (el elemento <html>).
- Cuando un elemento tiene un valor de posición que no sea “static” y valor de z-index distinto a “auto”.
- Cuando un elemento tiene un valor de opacidad menor que uno.

La primera y la segunda manera de formar contextos de apilamiento tienen mucho sentido y son generalmente bien comprendidos por los desarrolladores web (incluso si no conocen el propio concepto de contexto de apilamiento).

La tercera manera (opacidad) es apenas mencionada fuera de los documentos de especificación de la W3C.

6.2.5.- Determinando una posición de elemento en el orden de apilamiento

Actualmente, determinar el orden de apilamiento global para todos los elementos de una página (incluyendo bordes, fondos, capas de texto, etc.) es extremadamente complicado y se aleja del propósito de este artículo (de nuevo, hago referencia a la [especificación](#)). Pero para nuestro propósito, una comprensión básica del orden puede ayudar a desarrollar un CSS predecible. Así que vamos a dividir el orden en individuales contextos de apilamiento:

Orden de apilamiento dentro del mismo contexto de apilamiento

Aquí están las reglas básicas de orden de apilamiento dentro de un contexto simple (desde el fondo hacia delante):

1. El contexto de apilamiento del elemento raíz.
2. Elementos posicionados (y sus hijos) con valores z-index negativos (los valores más altos son apilados delante de valores menores; elementos con el mismo valor son apilados de acuerdo a su aparición en el HTML).
3. Elementos no posicionados (ordenados por aparición en el HTML).

4. Elementos posicionados (y sus hijos) con un valor z-index de auto (ordenados por aparición en el HTML).
5. Elementos posicionados (y sus hijos) con valores z-index positivos (los valores mayores son apilados delante de valores menores; elementos con el mismo valor son apilados de acuerdo a su aparición en el HTML).

Nota: elementos posicionados con z-indexes negativos se ordenan primero dentro de un contexto de apilamiento, lo cual significa que aparecen detrás de otros elementos. A causa de esto, llega a ser posible para un elemento aparecer detrás de su propio padre, lo cual normalmente no es posible. Esto funcionará solamente si el elemento parental está en el mismo contexto de apilamiento y no es el elemento raíz de ese contexto de apilamiento. Un gran ejemplo de esto son los [drop-shadows con CSS y sin imágenes](#) de Nicolas Gallagher.

6.2.6.- Orden de apilamiento global

Con una firme comprensión de cómo funcionan los contextos de apilamiento, y de cómo funciona el orden de apilamiento dentro de un contexto de apilamiento, imaginar dónde un elemento particular podrá aparecer en el orden de apilamiento global se convierte en una tarea más sencilla.

La clave es evitar la confusión generada cuando se forman nuevos contextos de apilamiento. Si pones un z-index con un valor de mil millones en un elemento y no se mueve hacia delante en el orden de apilamiento, echa un vistazo a su ancestro y mira si alguno de sus padres está formando contextos de apilamiento. Si lo hacen, tu z-index es inútil y no hace nada bueno.

6.2.7.- En conclusión

Volvamos al problema original, donde recreé la estructura HTML añadiendo comentarios dentro de cada etiqueta indicado su lugar en el orden de apilamiento.

```
<div><!-- 1 -->
<span class="rojo"><!-- 6 --></span>
</div>
<div><!-- 2 -->
<span class="verde"><!-- 4 --><span>
</div>
<div><!-- 3 -->
<span class="azul"><!-- 5 --></span>
</div>
```

Cuando añadimos la regla de opacidad al primer <div>, el orden de apilamiento cambia así:

```
<div><!-- 1 -->
<span class="rojo"><!-- 1.1 --></span>
</div>
<div><!-- 2 -->
<span class="verde"><!-- 4 --><span>
</div>
<div><!-- 3 -->
<span class="azul"><!-- 5 --></span>
</div>
```

"span.rojo" solía ser 6, pero cambió a 1.1. Se muestra que un nuevo contexto de apilamiento ha sido creado, y "span.rojo" es ahora el primer elemento dentro de ese nuevo contexto.

Espero que ahora esté un poco más claro por qué la caja roja se movió detrás de las otras cajas. El ejemplo original contenía solo dos contextos de apilamiento, el raíz y el formado por "span.rojo". Cuando hemos añadido opacidad al elemento padre de "span.rojo", hemos formado un tercer contexto de apilamiento, y, como resultado, el valor z-index de "span.rojo" solo se aplicaba dentro de ese nuevo contexto. Como el primer <div> (el único al que le aplicamos opacidad) y sus elementos hermanos no tienen asignados ni posición ni valor de z-index, su orden de apilamiento es determinado por su orden de aparición en el HTML, lo cual significa que el primer <div>, y todos los elementos contenidos dentro de su contexto de apilamiento son renderizados detrás del segundo y el tercer <div>.

6.2.8.- Recursos adicionales (en inglés)

- [Elaborate description of Stacking Contexts](#)
- [The stacking context](#)
- [The Z-Index CSS Property: A Comprehensive Look](#)

Philip Walton

Artículo por OldMith

6.3.- Utilizar porcentajes para tamaños de texto con CSS

El tamaño del texto se puede cambiar por medio de CSS, para agrandarlo o disminuirlo, por medio de porcentajes, de modo que el nuevo tamaño sea relativo al tamaño actual.

El porcentaje es otra de las medidas o unidades que podemos utilizar en los atributos de hojas de estilo en cascada (CSS) para definir un tamaño. En este artículo veremos ejemplos acerca de modificar los tamaños de los textos por medio de porcentajes, para conseguir nuevos tamaños que sean relativos a los que se están utilizando.

Por ejemplo, podríamos definir un estilo para escribir con un texto el doble de grande del que se esté trabajando:

```
<span style="font-size:200%">Este texto es el doble de grande</span>
```

Esto quiere decir que el texto será el doble de grande, 2 por las unidades de texto que estemos trabajando. Por ejemplo, si estamos trabajando con tamaños de texto de 10pt, el texto dentro del anterior span sería 20pt. El del siguiente código ejemplifica este caso concreto:

```
<span style="font-size:10pt;">Hola amigos <span style="font-size:200%">Este texto es el doble de grande</span> </span>
```

Lo mismo se puede hacer, pero para definir un texto menor, asignando porcentajes por debajo del 100%. Por ejemplo, si quisiéramos hacer un texto de la mitad del tamaño utilizaríamos la siguiente etiqueta:

```
<span style="font-size:50%">Este texto es la mitad del anterior</span>
```

Si estuviéramos trabajando con un tamaño de texto de 16pt, con la anterior etiqueta se escribiría con tamaño 8pt. El código sería el siguiente:

```
<span style="font-size:16pt;">Hola amigos  
<span style="font-size:50%">Este texto es la mitad del anterior</span>  
</span>
```

Ahora vamos a definir un par de clases para un texto mayor y menor, que podríamos utilizar para aumentar y reducir el texto respectivamente.

```
<style type="text/css">  
  .mayor {font-size:150%}  
  .menor {font-size:75%}  
</style>
```

Este código indica que la clase mayor es un texto el 150%, es decir, la mitad más que el anterior, y la clase menor un texto del 75%, es decir tres cuartas partes del anterior. Podríamos utilizar estas clases con un código como este:

Este es un texto normal y este es mayor, este vuelve a ser normal y este es menor

Los distintos ejemplos de este artículo podemos [verlos en una página aparte](#).

Artículo por Miguel Angel Alvarez

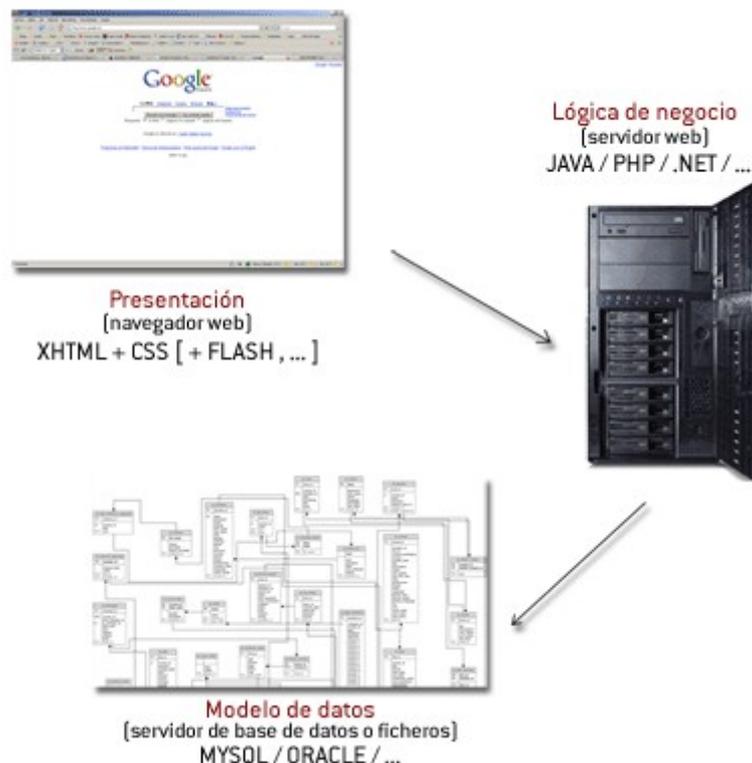
6.4.- CSS semánticas. Un nuevo enfoque

Existen infinidad de sitios en Internet que ofrecen miles de propuestas diferentes a la hora del uso de hojas de estilo CSS en la maquetación de portales web, como sabemos aplicables a múltiples tecnologías: XHTML, Flash, etc...

Lo que es difícil de encontrar es aquella solución que se adapte perfectamente a tus desarrollos, a tu entorno concreto. Quizá más difícil aun es adecuar algún enfoque similar al tuyo. Esta tarea es proclive a múltiples errores, que irán saliendo conforme se vaya utilizando y que llegado el momento, podemos comprobar que nos hemos equivocado de base, lo que exige una reestructuración desde el inicio.

Este es el contexto en el que nos encontramos actualmente. Tras una fuerte apuesta por la reestructuración y organización de CSS basadas en su semántica de uso se ve que si es quizá uno de los enfoques más acertados, deja bastantes puntos abiertos que es necesario concretar. Esa es la tarea que nos proponemos aquí.

Para los no iniciados, comentar que el enfoque semántico se basa en la idea de que la manera de estructurar la información relativa a la **capa de presentación** de nuestros proyectos web debe de seguir el criterio de qué es y el contexto donde se usa cada elemento.



El entorno web tiene una característica fundamental que pocos otros tienen y es la capacidad y potencialidad de uso en múltiples tipos de dispositivos, lo cual nos abre aun más el abanico de puntos que debemos controlar a la hora de crear nuestras hojas de estilos, a la vez que multiplica la casuística y potenciales errores que es necesario controlar.



PC
XHTML + CSS [+ FLASH , ...]



Dispositivos móviles
(Teléfono, PDA, ...)
XHTML/WML
+ CSS [+ FLASH LITE, ...]



Impresoras



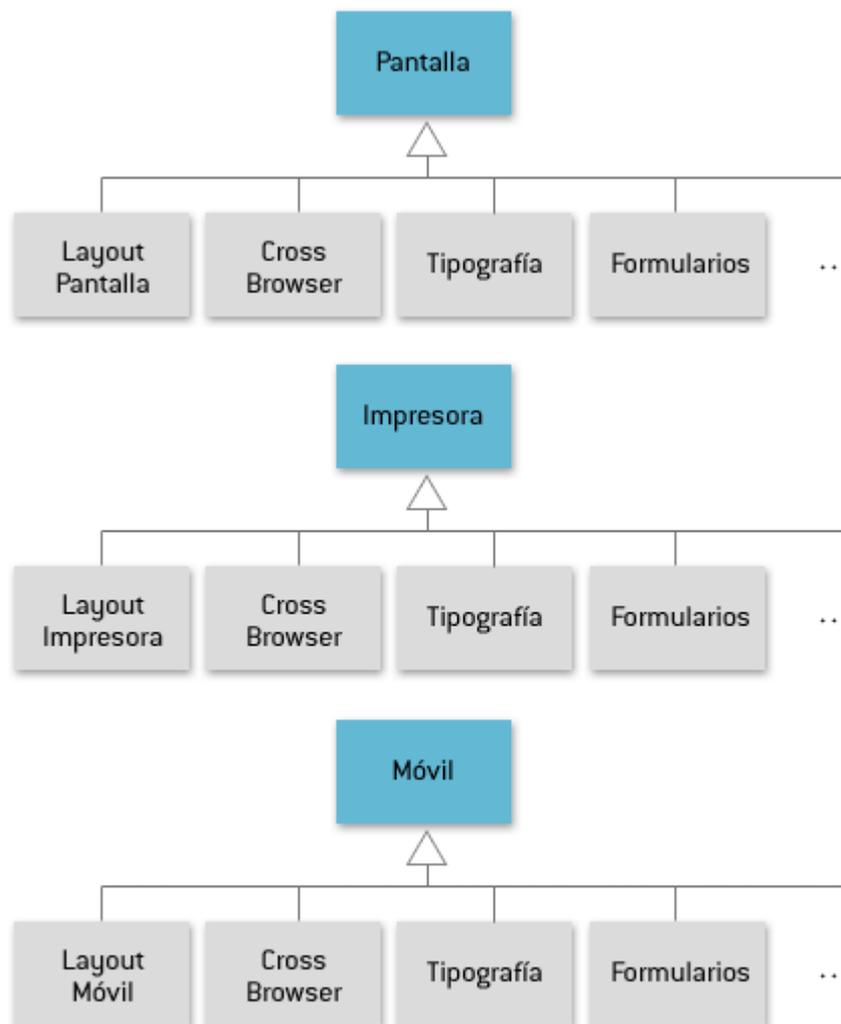
Dispositivos para accesibilidad
(Braille, alto contraste, etc...)

Aquí trataremos de dar una posible solución que se adecue a los principios de CSS semántica y que siga las pautas de accesibilidad y de facilidad de extensión a múltiples dispositivos.

6.4.1.- Puntos a tener en cuenta

Estructura jerárquica de las CSS

Nuestras hojas de estilo seguirán una estructura jerárquica, cuyo elemento principal será el que incluya, para cada tipo de dispositivo, las hojas de estilo correspondientes.



Explicación de cada una de las hojas de estilo

- **Pantalla.css, Impresora.css y Movil.css.** Estas son CSS específicas para cada tipo de dispositivo. Su misión en la parte superior de la jerarquía es la importación de las CSS hijas (`@import url(¿css?)`). No incluyen estilos concretos.
- **LayoutPantalla.css, LayoutImpresora.css y LayoutMovil.css.** Incluyen la información relativa a las diversas capas que forman el layout de la página, es decir, información de maquetación de las distintas zonas del portal (Banner, menús, contenidos, créditos, ..).
- **CrossBrowser.css.** Aglutina trucos, fixes y demás elementos para hacer que las páginas se vean igual en todos los navegadores (Internet Explorer, Mozilla, Firefox, Opera, etc.)
- **Tipografía.css.** Cualquier elemento relacionado con la forma en que se presenta la tipografía de la página queda recogida aquí. Comienza con una medida relativa de 62.5% en la etiqueta body, que se establece como medida equivalente a 1em. El resto de tamaños vendrán supeditados a este (mayores: 1.2em, 1.5em, etc., y

menores: 0.8em, 0.5em, etc.)

- **Formularios.css.** Definición de los estilos aplicables a todas las etiquetas relacionadas con los formularios genéricos.

6.4.2.- Documentos asociados a este artículo

- [CSS Semánticas.](#) Incluye los distintos CSS descritos.
- [Página ejemplo en XHTML con una maquetación tipo basada en capas](#)

Artículo por José Juan Corpas Martos

6.5.- CSS semánticas. Un nuevo enfoque (II)

Este artículo es la continuación de CSS semánticas. Un nuevo enfoque, y trata de dar una serie de consejos o recomendaciones a la hora del uso de hojas de estilo y escritura de código XHTML.

Toda la información está basada en el post citado, así como los ficheros relacionados.

6.5.1.- Recomendaciones de uso y escritura de código

Usar las etiquetas HTML estrictamente para lo que se han concebido

Si se consigue crear un código XHTML lo más limpio y claro posible se verá reducido considerablemente el número de estilos propios que tendremos que crear y evitaremos la mayoría de los errores e inconsistencias comunes.

- Maquetar en base a capas (div), no usar tablas (table).
- Párrafos: (p)
- Encabezados: (h1, h2, .h6)
- Tablas: (table, thead/tr/th, tbody/tr/td)
- Listas: (ul/li, ol/li)
- Menús: basados en listas con el atributo display:inline

Uso de medidas relativas en los tamaños de las fuentes

Las medidas relativas son aquellas que no establecen un tamaño fijo en píxeles o puntos para un elemento. En concreto son el porcentaje (%) y el em (1em equivale a 100%, 1.2em a 120%, 0.7em 70%, ..).

Es muy importante este punto por varios aspectos:

- No todas las personas tienen el monitor a la misma resolución y nuestra tipografía le puede resultar muy pequeña o grande según el caso.
- Al establecer la medida de tipografía de manera relativa en la etiqueta body, se tomará el tamaño relativo al navegador y dispositivo que interprete la página. Esto es especialmente en dispositivos con pequeñas pantallas (móviles, pda, etc.).

Maquetación de layout fijo basado en capas

Esta es una antigua polémica, el maquetado de tamaño fijo de ancho o el maquetado líquido o de tamaño variable.

Los estudios de usabilidad web aconsejan el uso de tamaños fijos adecuados a la resolución más usada por los usuarios de internet, intentando que el porcentaje discriminado sea el menor posible. La explicación de este aspecto se justifica por el constante crecimiento en las ventas de monitores de gran tamaño (17 pulgadas o más) y resolución (1024×768 en adelante).

La expansión de las páginas sobre todos en resoluciones de monitores panorámicos dificulta enormemente la legibilidad de los párrafos de texto.

Crear estilos o redefinirlos solo en el caso estrictamente necesario

Si creamos nuestras páginas haciendo un uso adecuado de las etiquetas XHTML y seguimos las pautas descritas en este documento, se reducirá drásticamente la necesidad de creación de estilos innecesarios y redundantes, lo que conlleva enormes beneficios sobre todos relativos a la facilidad de mantenimiento y reducción de errores y comportamientos extraños.

Artículo por José Juan Corpas Martos

6.6.- 10 errores comunes en los css

Esta es una recopilación de errores comunes en las hojas de estilo. Es bastante provechoso hacer una lista con estos y otros errores comunes.

6.6.1.- 1. Uso innecesario del valor 0

El código siguiente no necesita la unidad especificada si el valor es cero.

```
padding:0px 0px 5px 0px;
```

En su lugar puede ser escrito de esta manera:

```
padding:0 0 5px 0;
```

De la misma manera es igual para otros estilos. Ej.:

```
margin:0;
```

No malgastes espacios agregando unidades tales como px, pt, em, etc, cuando el valor es cero. La única razón de hacer esto es si necesitas cambiar estos valores más tarde. Si no declarar estas unidades no tiene sentido. Los pixeles cero son iguales que los puntos cero.

Sin embargo, line-height puede no tener unidad. Por eso es válido lo siguiente:

```
line-height:1;
```

De cualquier manera puedes utilizar una unidad en concreto como em si lo deseas.

6.6.2.- 2. Los colores en formato hexadecimal necesitan una almohadilla

Esto está mal:

```
color: ea6bc2;
```

Debe ser:

```
color: #ea6bc2;
```

O esto otro:

```
color: rgb (234.107.194);
```

6.6.3.- 3. Valores duplicados en los códigos de colores

No escribir el código de esta manera:

```
color: #ffffff;  
background-color:#000000;  
border:1px solid #ee66aa;
```

Los valores duplicados pueden ser omitidos. Escribiendo los códigos de esta manera:

```
color:#fff;
background-color:#000;
border:1px solid #e6a;
```

¡Por supuesto esto no debes hacerlo con códigos como este!

```
color: #fe69b2;
```

6.6.4.- 4.Evitar repeticiones de código innecesaria

Evita usar varias líneas cuando lo puedes conseguir con una sola. Por ejemplo, al fijar los bordes, algunas veces se debe hacer por separado pero en casos como el siguiente no es necesario:

```
border-top:1px solid #00f;
border-right:1px solid #00f;
border-bottom:1px solid #00f;
border-left:1px solid #00f;
```

Podríamos resumirlo en una única línea esta:

```
border:1px solid #00f;
```

6.6.5.- 5.La duplicación es necesario con los estilos en cascada

En los estilos en cascada es aceptable repetir el mismo código para un elemento elemento dos veces, si significa evitar la repetición mencionada en el punto arriba. Por ejemplo, digámos que tengamos un elemento donde solamente es diferente el "border" izquierda. En vez de poner cada "border" escrito usando cuatro líneas, uso sólo dos:

```
border:1px solid #00f;
border-left:1px solid #f00;
```

En este caso primero definimos todos los "borders" con el mismo color pero más tarde para ahorrarnos dos líneas de código redefinimos el "border" izquierda a otro color, de esta manera hemos ahorrado dos líneas de código.

El ejemplo malgastando espacio quedaría así:

```
border-top:1px solid #00f;
border-right:1px solid #00f;
border-bottom:1px solid #00f;
border-left:1px solid #f00;
```

Obviamente supuestamente este ahorro de carga supone un retraso en la carga de la página pues estamos definiendo el "border" izquierda dos veces, pero la carga de este proceso es insignificante.

6.6.6.- 6.Los estilos inválidos no hacen nada

Un ejemplo es suficiente para explicar este error:

```
padding:auto
```

Este estilo solo puede ser aplicado a width y height pero no a padding.

6.6.7.- 7.Código Específico para cada navegador

Obviamente este tipo de código solo funcionará en el navegador al que va destinado , pero es hay que pensar si es rentable puesto que solo algunos usuarios podrán apreciar esos cambio.

6.6.8.- 8.Espacio perdido

No estoy seguro del porqué pero muchos diseñadores estan empeñados en desaprovechar el espacio en su código, usando un montón de innecesarios saltos de línea. Recuerda que eso sólo lo verás tu y estas haciendo un uso excesivo de ancho de banda. Tambien tu código será más facil de leer puesto que tendrá menos "boquetes".

Por supuesto es sabio dejar un cierto espacio para mantenerlo legible, aunque a algunos les encanta condensar todo, no

dejando ningún espacio.

6.6.9.- 9.Especificar los colores sin usar palabras

Definir los colores usando las palabras que lo definen no es una buena idea puesto que estaríamos confiando en el navegador para que el interprete que color y código debe aplicar. Las tonalidades para un mismo nombre de color cambian mucho de un navegador a otro.

Es una buena práctica especificar siempre el color por su código hexadecimal.

Ej.: utilizar "#fff" en lugar de blanco.

6.6.10.- 10.Agrupar estilos idénticos

Es común ver los estilos escritos una y otra vez con el mismo código, aún cuando el estilo es igual.

Sería conveniente agruparlos y así optimizaríamos espacio:

```
h1, p, #footer, .intro {  
font-family:Arial,Helvetica,sans-serif;  
}
```

También nos hará mucho más fácil la tarea de actualizar el código.

Artículo por [Manu Gutierrez](#)

6.7.- Reglas de estilo CSS de usuario y de autor

Veamos qué son las reglas de estilo de usuario, configuradas opcionalmente por cada usuario en su navegador, y las reglas de estilo de autor, que define el desarrollador de cada web.

Los navegadores modernos implementan dos tipos de reglas de estilo para una página web, las de usuario y las de autor. Son conjuntos de reglas de estilo que afectan a los elementos de la página, es decir, a cualquier documento HTML que se vea en el navegador.

Las **reglas de estilo de usuario** (user stylesheet rules) las define cada persona en su navegador, a modo de configuración global, para todas las páginas que visita.

Las **reglas de estilo de autor** (author stylesheet rules) son las que definen los autores de las páginas, es decir, los diseñadores o desarrolladores de cada una de las páginas que visitamos. Por decirlo de otra forma, las hojas de estilo de autor son las CSS que conocemos y que hemos aprendido a crear en el [Manual de CSS](#).

Las reglas de estilo de usuario son menos importantes para el navegador, de cara al orden de precedencia o prioridad. Es decir, en caso que en las reglas de estilo de usuario y de autor se defina una misma propiedad de estilos CSS, la que se tiene en cuenta es la regla de estilo de autor, o sea, lo que haya configurado el diseñador de la página. Sin embargo, esto se puede cambiar puntualmente si el usuario lo desea.

Como sabemos, cualquier elemento de la página tiene unas reglas de formato predeterminadas, que definen los navegadores. La hoja de estilo de usuario nos permite cambiar esas reglas de estilo predefinidas, de modo que si una página web no declara un estilo determinado para un elemento, se tengan en cuenta las reglas de estilo definidas por el usuario. Por ejemplo, generalmente las tablas no tienen borde, o al menos es la configuración predeterminada en los navegadores que conozco. Con las reglas de estilo de usuario yo podría definir que todas las tablas en el navegador a partir de ahora tuvieran un borde de 1 píxel y de color rojo. En ese caso, si el autor no define cómo debe ser el borde de una tabla en sus CSS, la tabla siempre tendría borde de 1 píxel y color rojo.

Las reglas de estilo de usuario son sobreescritas por las de autor, pero ello no quita importancia a las reglas de usuario, para determinado tipo de usuarios. Imaginemos que por cualquier razón deseamos que las fuentes de las páginas que visitamos sean de un tamaño mayor y de una familia tipográfica determinada, para poder leerse mejor. Entonces

podríamos escribir unas reglas de estilo de usuario como estas:

```
body{
  font-size: 16pt;
  font-family: verdana, arial;
}
```

A partir de ahora, el cuerpo de la página de cualquier web, a no ser que el desarrollador haya definido otra cosa en su elemento body o en cualquiera de los otros elementos de la página, tendrá esas características.

Como se puede apreciar, las reglas de estilo de usuario tienen la misma sintaxis, atributos y valores que utilizamos también en las hojas de estilo normales.

6.7.1.- Dónde se colocan las reglas de estilo de usuario

Cada navegador se configura de una manera distinta, luego las reglas de usuario, que aplicamos a cada cliente web que las soporte, se tienen que indicar de manera distinta dependiendo de los navegadores que estemos utilizando.

En principio, se trata de un archivo de texto que debe contener el código CSS que queremos que se utilice de manera predeterminada al ver una página web. Podremos alterar el estilo de cualquier elemento, igual que lo hacemos con las hojas de estilo de autor, con la diferencia que esos estilos se aplicarán a todas las páginas que visitemos.

En el navegador Firefox se pueden definir las reglas de estilo de usuario en un archivo llamado "userContent.css" que se encuentra en la carpeta "chrome", del perfil de usuario que estemos utilizando. El directorio donde están los perfiles de Firefox depende del sistema operativo que estemos trabajando, en el caso de Windows Vista, y para mi usuario en particular, está en:

```
C:\Users\Miguel\AppData\Roaming\Mozilla\Firefox\Profiles\j6bik046.default\
```

Así que simplemente podrás crear el archivo con los estilos que quieras, que deberías colocarlo en una ruta como esta.

```
C:\Users\Miguel\AppData\Roaming\Mozilla\Firefox\Profiles\j6bik046.default\chrome\userContent.css
```

El directorio "j6bik046.default" es el perfil de mi usuario concreto, que tendrá un nombre seguramente diferente en el tuyo, así como "Miguel", en el principio de la ruta (C:\Users\Miguel\...), que es mi nombre de usuario en el Windows.

En el directorio "chrome" verás probablemente unos archivos llamados "userContent-example.css", que pueden servirte de guía para crear tus propias reglas de estilo de usuario.

6.7.2.- Alterar la precedencia para que las reglas de usuario dominen sobre las de autor

Como hemos dicho, en caso que una regla de estilo de usuario se defina también como regla de estilo de autor, se tiene en cuenta lo que se haya definido por el autor o diseñador de la web. Pero esto podemos cambiarlo en las reglas de estilo que queramos.

Imaginemos el caso del usuario que decidió que quería ver siempre las fuentes con un tamaño mayor, para leer mejor el contenido de las webs en su ordenador. Esta persona definió en su archivo de reglas de estilo de usuario un tamaño de letra mayor para determinados elementos de la página. Pero si un desarrollador luego ha definido un tamaño de letra distinto, el tamaño definido por el usuario se pierde y con ello quizás ahora no pueda leer la web tan cómodamente.

Podemos utilizar entonces la directriz de CSS !important, que cuando se coloca en las reglas de estilo de usuario, hace que siempre se tenga en cuenta lo que se haya definido allí.

Así pues, esta persona puede obligar a que en el cuerpo de la página siempre se utilice el tamaño de fuente que había determinado, de la siguiente manera.

```
body{
  font-size: 16pt !important;
  font-family: verdana, arial;
}
```

Si vemos el anterior código de ejemplo, al atributo font-size le hemos aplicado la declaración !important, luego siempre

se tendrá en cuenta antes que los estilos declarados en las reglas de estilo de autor y por tanto, aparecerán todos los textos del cuerpo de la página con tamaño de 16pt. Ahora bien, se había definido una tipografía como Verdana, Arial, pero no era !important, luego sólo se utilizará esta regla si el diseñador no llegó a especificar la familia tipográfica con sus CSS para el cuerpo de la página.

Artículo por Miguel Angel Alvarez

6.8.- Declaración !important en CSS

En CSS podemos declarar reglas de estilos como !important para que tomen precedencia sobre otras reglas de estilos que se puedan encontrar en una página web.

Vamos a ver en este artículo de DesarrolloWeb.com, englobado dentro del [Manual de CSS](#), una declaración un tanto especial que podemos utilizar al definir reglas de estilos para una página web. Se trata de !important, una palabra que hará que determinadas propiedades tomen mayor importancia y, por tanto, se tengan más en cuenta que otras que puedan sobrescribirlas.

Otra cosa que veremos de paso es cómo el uso de !important nos proporcionará una sencilla y válida técnica para poder definir reglas de estilos distintas para navegadores antiguos, como Internet Explorer 6.

Además, important! se puede utilizar en las hojas de estilo de usuario, para que cada persona pueda definir para su propio navegador si lo desea, un estilo CSS por defecto que se tenga en cuenta en todas las web que visitemos. Este caso ya se explicó en el artículo [Reglas de estilo CSS de usuario y de autor](#).

6.8.1.- Uso y efecto de la declaración !important

Para utilizar !important en una regla de estilo, siempre se coloca en la parte del valor del atributo, antes del punto y coma ";". Por ejemplo

```
body{
font-family: verdana, arial !important;
}
```

El efecto es que siempre se aplicará el estilo definido como !important, aunque luego se pueda sobrescribir con otro estilo más tarde en la misma declaración o en otra distinta. Veamos este ejemplo:

```
td{
font-size: 16pt !important;
font-size: 8px;
}
```

Tenemos una declaración de estilos para los elementos TD, donde definimos dos veces el atributo font-size. En condiciones normales, se tendría en cuenta el valor definido en segundo lugar, porque lo sobrescribe. Sin embargo, que el primer font-size está definido como !important, en realidad lo que ocurrirá es que se tenga en cuenta finalmente y el tamaño de letra por tanto será 16pt.

Este efecto lo podemos aplicar también a distintos tipos de [selectores de CSS](#). De modo que podremos encontrarnos que para un elemento se indique un estilo y luego para una clase (class de CSS) se aplique otro y se tenga en cuenta el definido como !important. Veamos este ejemplo de CSS:

```
td{
font-family: verdana, arial !important;
}
td.micelda{
font-family: monospace;
}
```

Que aplicado sobre el siguiente HTML:

```
<table>
```

```
<tr>
<td class="micelda">Hola</td>
<td>23232</td>
</tr>
</table>
```

Darí­a como resultado, en condiciones normales, que la primera celda, de clase "micelda", tuviese la fuente font-family: monospace y la segunda celda, que no tiene ningún class, tuviera el estilo font-family: verdana, arial. Sin embargo, como el font-family definido en primer caso tiene la declaración !important, la fuente será siempre verdana, arial, para las dos celdas.

6.8.2.- Usar !important para definir estilos diferentes en navegadores antiguos

La declaración !important no la entienden todos los navegadores, por tanto, algunos simplemente la ignorarán y otros no. El caso más representativo, por ser un navegador que todavía se utiliza habitualmente por los internautas, sería Internet Explorer 6.

Así pues, utilizando !important podemos conseguir definir estilos diferentes para Internet Explorer 6 y para la mayoría de los otros navegadores que pueden visitar nuestra web. Esto lo podemos conseguir de la siguiente manera.

```
div{
  background-image: url(fondo-semitransparente.png) !important;
  background-image: url(fondo.gif);
}
```

Como Internet Explorer 6 ignora la directriz !important, ocurrirá que tendrá en cuenta el segundo valor de background-image, ya que está repetido y por tanto sobrescribe al primero. Por ello, en este caso IE6 mostrará como fondo el archivo llamado "fondo.gif".

Los otros navegadores, como entienden !important, mostrarán el estilo que había definido anteriormente y por tanto utilizarán como fondo el archivo "fondo-semitransparente.png".

Nota: Dicho sea de paso, como IE6 tiene problemas al mostrar fondos semitransparentes (con [canal alpha en el PNG](#)) esta sería una posible técnica para conseguir que en Explorer 6 se utilice un fondo de imagen distinto (por ejemplo en .gif) que el que se utiliza en otros navegadores que no tienen problema con el .png.

Artículo por Miguel Angel Alvarez

6.9.- Regla @media de CSS

Explicamos la regla @media permite definir estilos CSS específicos para distintos medios en los que se pueda mostrar una página web.

Una página web se puede mostrar en diferentes dispositivos. Esto es algo que ya conocemos y que, a medida que pasan los años y avanzan las tecnologías, se hace cada vez más patente. Por tanto, podemos dejar de pensar que nuestra web se va a acceder simplemente desde un ordenador y para ayudarnos a que se vea bien en cualquier dispositivo, podemos definir estilos distintos para cada tipo de medio con la regla @media.

Incluso en el hipotético caso que descartéis que vuestra web pueda tener usuarios venidos de teléfonos móviles o televisiones (por poner un par de ejemplos), puede que necesitéis especificar estilos específicos para cuando la página web se está imprimiendo en papel. Todo ello son tipos de medios distintos, que podemos gestionar desde la llegada de CSS 2.

6.9.1.- Sintaxis y usos de la regla @media de CSS 2

La regla @media permite especificar estilos para distintos tipos de medios en la misma hoja de estilos. En ella podemos

informar sobre los tipos de medio sobre los que queremos definir estilos CSS. Por ejemplo, podríamos escribir estilos para tipos de medios como la impresión o estilos para el medio pantalla del ordenador.

Para definir un estilo para un tipo de medio, o medios, específicos podemos escribir la regla `@media` seguida de los tipos de medios sobre los que queremos aplicar los estilos, separados por comas.

Así definiríamos estilos que funcionarían sólo en la impresión en papel:

```
@media print {
  table{
    width: 90%;
    border: 2px solid #ff000;
  }
  .miclase{
    display: none;
  }
}
```

Como decíamos, podemos indicar estilos CSS para varios medios a la vez:

```
@media tv, handheld {
  body{
    font-size: 0.5 em;
  }
}
```

Además, si lo deseamos, podemos especificar estilos para todos los medios, con el tipo de medio "all".

```
@media all {
  div.imprimir {
    display: hidden;
  }
}
```

Nota: Aparte de la regla `@media` que estamos explicando existe una manera de especificar estilos definidos en archivos externos, que sólo se apliquen para determinados tipos de medios. Esto se hace con la directiva "media" que se aplica en la etiqueta LINK para enlazar con una hoja de estilos externa, en el atributo "media".

```
<link media="print" href="css_solo_para_impresion.css" rel="stylesheet" type="text/css">
```

Para más información, por favor, accede al artículo [CSS para imprimir páginas web](#).

6.9.2.- Tipos de medios en CSS 2

Ahora podemos ver un listado de los tipos de medios que se definen en las especificaciones del lenguaje CSS 2.

- All: Cualquier tipo de medio.
- Braille: medio relacionado con dispositivos táctiles braile.
- Embossed: Para impresoras braile.
- Handheld: para dispositivos de bolsillo o de mano que normalmente tienen una pantalla pequeña.
- Print: medio específico para cuando se imprimen documentos en la impresora. Desde la vista previa para imprimir que tienen los navegadores, generalmente en el menú de Archivo, también podemos ver el resultado de la página para impresión que utiliza los estilos CSS del tipo de media "print".
- Projection: tipo de medio que se aplica para las presentaciones que se muestran con proyector.
- Screen: medio que se utiliza para pantallas grandes, generalmente las pantallas de los ordenadores personales.
- Speech: medio para sintetizadores de voz.
- Tty: tipo de medio que se utiliza en dispositivos que tienen un tamaño fijo de carácter, como un teletipo, terminal, consola de comandos etc. En este tipo de media no se puede usar la unidad de medida de píxeles (px) porque todo lo que se puede mostrar es a nivel de carácter.
- Tv: para cuando se accede a una web desde un dispositivo de televisión.

Nota: Al escribir los tipos de medios es indiferente si lo hacemos con mayúsculas o minúsculas.

Estos tipos de medios son los que eran válidos con las especificaciones de CSS 2. Es obvio que con el paso del tiempo se crearán otros tipos de medios que se irán incorporando al lenguaje. Si se utiliza un tipo de medio que no existe o que no es reconocido, el sistema simplemente lo ignora. Por ejemplo:

```
@media tv, nevera{
  p{
    background-color: #ccc;
  }
}
```

Esta declaración de estilos sólo se aplica en las televisiones y en los monitores acoplados en las neveras de la cocina. Como en estos momentos no existe el tipo de medio "nevera", pues simplemente se ignora y en la práctica ese estilo sólo servirá para cuando se muestre la página en un televisor.

Artículo por [Miguel Angel Alvarez](#)

6.10.- Depurar CSS

Una lista de los errores más comunes en el código CSS y cómo averiguar cuál de esos errores se está produciendo en tu código, para depurar tus CSS.

El lenguaje para especificar estilos en páginas web, CSS, a veces resulta poco práctico en fase de depuración. Muchas veces es difícil saber qué errores hemos cometido y cómo solucionarlos, pero con ayuda de una serie de consejos y buenas prácticas, podremos minimizar el tiempo que perdemos en la búsqueda de los problemas o errores cometidos cuando algo no se ve exactamente como nosotros queríamos.

En este artículo no pretendemos dar una receta, paso por paso, para hallar inequívocamente los errores CSS de un código dado, porque pueden ser variados y a veces esquivos, sino ofrecer una serie de consejos o prácticas que nos ayudarán a encontrar un problema. De hecho, no siempre una única técnica te ayudará a encontrar el problema exacto en cualquier código CSS, por lo que nos conviene ser expertos y tener a mano varios recursos para no volvernos locos cuando algo falla.

Quizás merezca la pena comentar que en DesarrolloWeb.com se publicó hace tiempo un tutorial sobre cómo hacer tu código CSS más correcto y resumido, así como no cometer algunos errores de sintaxis básicos. Todo ello lo puedes encontrar en el artículo [10 errores comunes en tu código CSS](#). Este texto es interesante para ayudarte a no cometer errores, pero no cubre muchos problemas básicos que podemos encontrar cuando probamos los estilos que hemos definido. Es decir, te ayudará mientras estás codificando, para ser más correcto, pero no mucho cuando has cometido el error y quieres saber dónde está y cuál es el problema, que es lo que pretendemos mostrar en este momento.

Así que vamos ya con esas técnicas, herramientas, consejos y pruebas que puedes hacer para saber qué es lo que falló en tu CSS.

6.10.1.- Firebug tu mejor amigo para depurar CSS

La herramienta de debug en Firefox, Firebug, es bien conocida es por cualquier desarrollador con un poco de experiencia y amor a su tiempo productivo. Supongo que a estas alturas ya debes conocer cuáles son las ventajas y las ayudas que te ofrece Firebug, pero si no es así, te conviene ir instalando esta extensión de Firefox.

En artículos anteriores ya hemos hablado repetidas veces de Firebug, puedes hacer un búsqueda con el buscador interno de DesarrolloWeb.com por esa palabra o ir directamente a la [presentación de Firebug](#).

Firebug tiene, entre otras cosas, un inspector de elementos que nos permite seleccionar cualquier cosa en la página y ver sus atributos CSS y otros detalles. Es ideal porque te ayudará a encontrar errores en tus CSS, como los siguientes:

- Errores de sintaxis
- Errores en rutas de imágenes y otros recursos
- Errores en tu HTML

Los errores de sintaxis (principalmente los que se producen por los típicos errores humanos) son muy fáciles de identificar con Firebug pues a la hora de inspeccionar elementos te permite ver en un panel los estilos CSS que se están aplicando a esos elementos. Si está faltando algún estilo de entre aquellos que habías definido es que Firefox no lo había entendido y por eso no lo había procesado. Entonces quiere decir que o bien has escrito mal el nombre del atributo CSS o bien su valor, o bien ha faltado un ":" o un ";". Revisa la sintaxis en la declaración de estilos que esté faltando en Firebug.

Otra de las cosas que se averiguan fácilmente con Firebug es si las rutas a las imágenes están correctas. Por ejemplo, puede que hayamos escrito mal una ruta para una imagen de fondo. Entonces Firebug mostrará la ruta que hayamos escrito en el atributo CSS, pero al poner el ratón encima de esa ruta no nos mostrará la imagen que hay en ese archivo. (Si la ruta estuviera bien, nos mostraría una miniatura de la imagen que estamos invocando).

Muchas veces los problemas con CSS pueden venir de haber escrito mal el HTML y en ello Firebug también nos ayudará, al mostrarnos también el código HTML que se ha entendido y que se está procesando en el navegador.

Las razones por las que usar Firebug no acaban. Por poner dos ejemplos más, podríamos señalar que tiene una consola avanzada que nos muestra errores Javascript, pero que también la podemos configurar para que nos muestre los errores de sintaxis CSS que podamos estar cometiendo. Además con Firebug podemos estar atentos a los estilos que cada elemento hereda de otros elementos en los que está contenido. Y es que los problemas muchas veces no son de los estilos CSS de un elemento, sino de los que está heredando de otros elementos padre.

6.10.2.- Otros productos similares a Firebug para navegadores distintos de Firefox

Como hemos dicho, Firebug en principio está disponible como extensión de Firefox, pero podemos encontrar productos similares para otros navegadores que nos vendrán bien por dos motivos. Primero por si nosotros preferimos desarrollar con otro navegador, pero sobre todo para poder hacer inspección de elementos (para ver el código HTML que está entendiendo el cliente web los atributos CSS que tienen los elementos de la página) en otros navegadores, cuando algo funciona como esperábamos en Firefox pero no en otros browsers.

[IE Developer Toolbar:](#)

Un complemento para Internet Explorer 7, ya que Internet Explorer 8 incluye herramientas para desarrolladores en su instalación básica que pueden ayudarnos a inspeccionar elementos.

[DebugBar para Explorer:](#)

Otro complemento muy interesante para Explorer que tiene múltiples opciones para debug e inspección de código de páginas web.

[Firebug para Google Chrome:](#)

Chrome, el navegador de Google, incluye herramientas similares a las que nos aporta Firebug en todas sus instalaciones. Sin embargo, también existe la herramienta Firebug en versión compatible con Chrome.

[Opera Dragonfly:](#)

Un kit de herramientas para desarrolladores que funcionan dentro del navegador Opera.

6.10.3.- Validar tu hoja de estilos con un validador CSS

Validar la hoja de estilos CSS ha sido según mi experiencia clave para encontrar un error en situaciones confusas. Hay ocasiones que no se encuentra fácilmente el error con Firebug o que el error se muestra en unos navegadores y otros no.

Por ello, siempre es una buena idea validar tu código CSS. Si tu código CSS tiene algún problema el validador te alertará sobre ello y podrás corregirlo. Quizás ese error no significaba un problema en un navegador en concreto y sí en otro.

Existen varios validadores CSS, pero quizás el que más nos interese es el que comentamos en el artículo [Herramienta del W3C para validación de hojas de estilo](#)

En diferentes navegadores existen atajos para acceder a los validadores, a través de opciones del propio navegador o a través de extensiones. Por ejemplo, en Opera, podemos acceder al validador a través de la tecla rápida ALT + CTRL + Mayúsculas + U.

6.10.4.- Aislar y reducir el código HTML

En el caso de que seamos incapaces de detectar el problema de nuestro código CSS, a pesar de utilizar un inspector de elementos como el que nos ofrece algún complemento como Firebug, podemos probar algo más laborioso. Se trata de aislar el código fuente que te está dando el problema, o reducirlo todo lo posible para simplificarlo, de modo se cambie el contexto donde se produce el error, para ver qué si eso nos da una pista.

Imagina que tienes una columna que ocupa más anchura de la que debería. Podrías hacer lo siguiente. Vas quitando el código HTML de cada uno de los elementos que hay en esa columna y viendo si ocurre algún cambio. Si quitando un elemento, como una tabla, una división con DIV, un formulario, un banner o cualquier otra cosa, todo vuelve a funcionar, puedes entender que algo con ese elemento está provocando el problema.

Como se puede imaginar, esta técnica es sólo un poco de prueba y ensayo, pero siempre procurando quitar cosas en tu código, haciéndolo cada vez más simple y por consiguiente más sencillo para localizar el problema. Claro que esta técnica representa más trabajo y es sólo un recurso para cuando estás bastante desesperado, porque teóricamente con Firebug sería más fácil encontrar el fallo. De hecho, esta era una de las técnicas más utilizadas para encontrar problemas cuando no existían herramientas como Firebug.

6.10.5.- Utilizar un DOCTYPE

Este no es un truco para hacer debug de CSS, pero es una práctica que ayudará mucho a que nuestras tareas de depuración sean mucho más sencillas y no nos desesperemos intentando encontrar la solución a problemas tontos. Sobre todo, es un consejo importante para que tengamos menos errores de compatibilidad de nuestra página entre distintos navegadores.

El DOCTYPE es una cadena de texto que se debe incluir al principio del código HTML, para decirle qué versión de HTML o XHTML estamos utilizando. Distintos DOCTYPEs pueden hacer que un mismo código HTML o CSS se vea de distinta manera, pero no es ese el problema. Lo importante es que con un DOCTYPE declarado todos los navegadores atenderán a ese tipo de documento e interpretarán el código HTML + CSS de una manera más similar.

Mi recomendación es indicar un DOCTYPE al comenzar tu trabajo e ir desarrollando tu sitio web poco a poco con ese DOCTYPE activo desde el principio. No importa mucho qué DOCTYPE uses, pero sí importa que al menos uses uno de ellos. Así tendrás muchas más probabilidades que el trabajo que estás realizando se vea igual en todos los navegadores. Es mejor que colocar el DOCTYPE desde el principio y no al final, puesto que en el momento que lo hagamos puede haber algunos elementos de la página o estilos CSS que cambien su manera de mostrarse.

Algunos DOCTYPE con los que podemos trabajar, con su explicación sobre qué significan, podemos verlos en el [artículo Doctype HTML](#).

Artículo por Miguel Angel Alvarez

Parte 7:

Por dónde continuar

Ahora que ya sabes bastante sobre CSS, quizás quieras abrir nuevos horizontes y para ello te damos algunas referencias para continuar aprendiendo.

7.1.- Listado de distintos frameworks CSS

Un listado de los frameworks CSS que hay en el mercado para ayudar en la maquetación de páginas web con CSS.

En este artículo vamos a enumerar y comentar algunas cosas sobre frameworks CSS, si es que se les puede llamar frameworks, puesto que ese concepto se usa muchas veces para sistemas que facilitan la programación de aplicaciones y en el caso de los CSS, no es programación, sino que se utilizan para obtener ayudas en la maquetación de webs.

Como sabemos, en el momento actual las páginas se maquetan con CSS. Con HTML especificamos los contenidos y con CSS la forma o disposición con la que deben presentarse al usuario en los navegadores. CSS por tanto es un lenguaje que sirve para especificar el estilo de las páginas, pero muchas veces hacemos cosas repetitivas, como divisiones de página en columnas, cajas de determinados tipos, etc. Pues los Frameworks CSS nos ayudan a realizar esas tareas de maquetación básicas, que muchas veces tenemos que implementar repetidas veces en diversos sitios, para generar las estructuras de elementos de la página.

Los frameworks CSS disponen una serie de clases (de hojas de estilo) ya creadas con las que ayudan a posicionar elementos en la página y crear estructuras de maquetación, más o menos versátiles. Así, en el desarrollo de páginas nuevas, o en el rediseño de páginas antiguas, podemos ayudarnos de frameworks CSS para disponer de una rejilla donde posicionar los distintos componentes de nuestro diseño. Con ello nos ahorraremos el tiempo de tener que crear de nuevo decenas de clases que estamos aburridos de implementar para crear maquetaciones a 2, 3 ó 4 columnas, con divisiones de cabecera, cuerpo y pie, etc.

En DesarrolloWeb.com hemos analizado un framework CSS y realizado un manual para explicar sus características y funcionamiento, que recomendamos leer para obtener más explicaciones. Se titula [Maquetación CSS con 960 Grid System](#), en el que encontraréis además diversos vídeos muy ilustradores para conocer de primera mano el proceso de diseño de una web utilizando uno de estos frameworks CSS.

En este artículo pretendo simplemente enumerar este y otros frameworks CSS, en un listado que pretende dar una idea de las posibilidades que nos ofrece en este momento el mercado.

7.1.1.- 960 Grid System

Es, tal vez, el más utilizado de los frameworks CSS, cuyas páginas se construyen en anchuras de 960 píxeles (de ahí su nombre). Ofrece dos posibilidades de maquetación de páginas, con una rejilla de 12 ó 16 columnas. Nosotros escogimos

este framework CSS para explicarlo a los lectores de DesarrolloWeb.com, justamente por ser tan popular. En nuestro trabajo con este sistema hemos podido comprobar que es muy versátil y sobre todo, sencillo de utilizar.

[Página web de 960 Grid System](#)
[Manual de 960 Grid System](#)

7.1.2.- Simple

Este framework CSS lo presenta un desarrollador chileno, con lo que todos los ejemplos y documentación que puedas encontrar está en español. El creador lo ha realizado para poder simplificar las cosas, no sólo en el desarrollo de las páginas, sino también en el aprendizaje. Lo destacamos en segundo lugar por ser un producto en castellano, que siempre se agradece tener herramientas para trabajar en nuestro propio idioma.

[Página web de Simple](#)
[Artículo en DesarrolloWeb.com sobre Simple](#)

7.1.3.- Blueprint

Es un framework CSS que pretende reducir el tiempo de desarrollo de las páginas web. Ofrece una estructura sólida en la que fundar tu trabajo de diseño, por medio de la típica rejilla. Pero no se limita simplemente a eso, sino que ofrece otra serie de clases muy útiles para estilizar componentes típicos, como formularios, botones, pestañas, tipografías o para que tus páginas web se puedan imprimir de manera óptima en papel.

[Página web de Blueprint](#)

7.1.4.- YUI Grids CSS

El framework CSS de Yahoo! Es un código CSS que permite maquetar páginas con distintas anchuras (750px, 950px, y 974px) y hacer todas las cosas típicas que se pueden desear en una página. Tiene 6 plantillas predefinidas y la posibilidad de crear más de 1.000 combinaciones de maquetación, en regiones de 2, 3 y 4 columnas. Forma parte de la Yahoo! User Interface Library, lo que da una garantía adicional, por venir de tan renombrado buscador.

[Página web de YUI Grid CSS](#)

7.1.5.- Tripoli

Tripoli no es un framework CSS y según remarcan los creadores, ello significa que no te obliga a desarrollar tu página de una manera determinada. En contra, lo que ofrece es un código CSS que resetea los estilos predefinidos de los navegadores y los redefine, consiguiendo una base estable sobre la que realizar un sitio y que se vea igual en cualquier cliente web. Puede ser interesante porque intenta no caer en los problemas que tienen los frameworks CSS.

[Página web de Tripoli](#)

7.1.6.- Boilerplate

Este framework me ha parecido interesante porque está creado por uno de los desarrolladores iniciales de Blueprint, que ha fundado el nuevo proyecto para poner en marcha sus ideas particulares. Como él dice, este framework está pensado para simplificar las cosas y ser ligero, aportando todo lo básico para poder maquetar una web, pero sin las complejidades que tiene Blueprint y con convenciones de nombres que dan más sentido y significado a lo que estamos codificando.

[Pagina web de Boilerplate](#)

7.1.7.- BlueTrip

Según sus creadores, BlueTrip es una combinación de lo mejor de distintos frameworks CSS de los que hemos hablado

ya. Su nombre viene de la unión de BLUEprint - TRIPoli, haciendo referencia a esa unión de ideas de los mejores frameworks, entre los que también se han inspirado en nuestro framework preferido en estos momentos, 960 grid, por su sencillez.

[Página web de BlueTrip](#)

7.1.8.- Otros Framework CSS

Pongo otros frameworks CSS que he encontrado y que no he investigado tanto las posibilidades que ofrecen, aunque también pueden ser interesantes, sobre todo puede que den enfoques diferentes que puedan ser útiles en ciertas ocasiones.

[Elements](#)

[ESWAT](#)

[Content with style](#)

[My CSS Framework](#)

[Hartija](#)

[Malo](#) (Framework CSS ultra pequeño)

[emastic](#)

Como referencias sobre frameworks CSS seguramente tendremos más que suficiente. Pero llegado a este punto y antes de decidirte por qué framework usar o si te interesa o no utilizar uno de ellos, te recomendamos leer el artículo sobre las [ventajas e inconvenientes del uso de frameworks CSS](#).

Artículo por *Miguel Angel Alvarez*

7.2.- Frameworks CSS: Ventajas e inconvenientes

Ventajas e inconvenientes del uso de Frameworks CSS, Discutimos acerca de la conveniencia o no de usar frameworks CSS.

La conveniencia de uso, o no, de los frameworks siempre es un tema polémico. No ocurre sólo con los frameworks CSS, sino también con los frameworks de otros lenguajes o tecnologías, aunque quizás en el caso de las librerías CSS todavía se acentúa más la discusión.

Las personas a favor del uso de frameworks siempre podrán decir que aceleran los procesos de desarrollo y que ello es suficiente razón para usarlos, pero conviene saber que también cuáles son las desventajas y decidir si nos interesa o no usarlos también en función de ellas.

Conviene aclarar antes de nada qué nos ofrece un framework CSS, pues la mayoría de las ventajas están directamente relacionadas con las áreas donde éstos actúan.

7.2.1.- Componentes habituales de un framework CSS

Cada framework CSS es distinto, pero la mayoría proveen de código fuente -hojas de estilo en cascada- para resolver los mismo asuntos:

- Creación de una rejilla, para que el desarrollador pueda colocar en ella los distintos elementos que forman parte de la web. Esta rejilla, que divide los espacios en distintas columnas, ayuda a posicionar con CSS los componentes de una página de una manera precisa y versátil.
- Definiciones de estilos de tipografía para los elementos HTML, de modo que no tengamos que definirlos nosotros para cada proyecto.
- Solución de casos de incompatibilidad entre navegadores, para que un mismo código se vea igual en todos los clientes web más habituales

- Creación de clases CSS estándares, que puedan ayudarnos a estilizar componentes avanzados de interfaces de usuario.

7.2.2.- Ventajas de los frameworks CSS

Ahora que ya sabemos dónde inciden las librerías CSS, vamos a listar una serie de ventajas posibles de su utilización.

Maquetación acelerada y código más limpio:

Con un framework CSS podemos maquetar de una manera mucho más rápida una página, gracias a la rejilla que nos ofrece. Pero además utilizarlo nos ayudará a tener un código más limpio y más estructurado.

Solución a los problemas CSS comunes:

Casi todos los frameworks están realizados bajo la experiencia de trabajo con CSS en muchas páginas web. Por ello siempre ofrecen soluciones a problemas comunes de los desarrolladores.

Compatibilidad entre navegadores:

Los navegadores a veces interpretan de manera distinta un mismo código fuente. Esto es algo que a menudo se acentúa en Internet Explorer y que los frameworks CSS solucionan de alguna u otra manera.

Aprender trucos y prácticas habituales:

Al leer el código fuente de los frameworks CSS podremos aprender las prácticas de otros desarrolladores y en concreto para los frameworks CSS es muy sencillo examinar su código fuente.

Desarrollar conforme a un mismo patrón:

Cuando desarrollamos con un framework tendremos un procedimiento determinado para resolver las necesidades comunes. Esto quiere decir que la manera de actuar en diferentes proyectos puede ser muy similar y por ello a la larga te costará menos esfuerzo de mantenerlos y podrás recordar mejor cuáles son los criterios que utilizaste para desarrollarlos.

Ayuda al trabajo en grupo:

Dado que trabajar con un framework CSS nos obliga a desarrollar con un patrón determinado, en proyectos en grupo, las personas que integren el equipo de trabajo podrán tener más claras cuáles fueron las decisiones que se tomaron a la hora de maquetar con un breve vistazo en el código.

En definitiva, las ventajas más importantes de usar un framework CSS es que agilizará el proceso de desarrollo y nos ayudará bastante a la hora de hacer una web que se vea perfecta en cualquier navegador. Pero dependiendo de nuestro contexto de trabajo podemos encontrar otras ventajas interesantes.

7.2.3.- Desventajas del uso de frameworks CSS

Hasta ahora, todo lo que hemos comentado de los frameworks CSS es muy bueno, pero existen algunas desventajas que también debemos conocer, para valorar su conveniencia o no y minimizar en la medida de lo posible los aspectos negativos.

Para mi, existe una desventaja principal bastante importante y otra serie de desventajas menos determinantes. Quizás la desventaja principal es suficiente como para desistir en el uso de frameworks CSS y para entenderla tenemos que conocer antes una de los motivos por los que se creó CSS.

Como quizás sepamos, CSS es un lenguaje para definir estilos en páginas web, que se creó con la intención de separar el contenido de la presentación. Con HTML especificamos el contenido de una página y con CSS especificamos, por separado, la presentación. Pues bien, la mayoría de los frameworks CSS se cargan esta ventaja del lenguaje, o al menos la limitan. Esto es porque, cuando queremos usar la rejilla para posicionar los elementos, muchas veces estamos utilizando código HTML con clases (Class de CSS) que especifican la posición que van a tener esos elementos. Eso hace que estemos volviendo a mezclar contenido con presentación y quiebra por tanto algunas de las ventajas que habíamos adquirido al trabajar con CSS.

Asimilada para mi esta desventaja principal, veamos un completo listado de los inconvenientes del uso de Frameworks CSS:

Mezclas de nuevo el contenido y la presentación:

Cuando diseñas una web con CSS podrías cambiar su aspecto radicalmente con sólo cambiar la hoja de estilos, incluso la manera como los elementos se posicionan en la página. Pero cuando utilizas un framework estás colocando clases (de CSS) que indican dónde se van a posicionar los elementos en una rejilla y, si quieres cambiar la posición de esos elementos más adelante te verás obligado a cambiar el código HTML de la página y colocar otras clases CSS, que permitan situarlos en otros puntos de esa rejilla.

Tendrás código CSS que no utilices nunca:

El framework CSS contiene diversos códigos CSS útiles en casos generales, pero lo cierto es que cuando diseñas una página web una buena parte de ese código no lo vas a usar nunca. Es decir, al utilizar un framework estamos cargando innecesariamente el peso en bytes de nuestro código CSS. Esto lo podemos arreglar "a mano" limpiando el CSS del framework, eliminando código que no lleguemos a utilizar en ninguna página del sitio.

Curva de aprendizaje más lenta:

Aunque aprender a usar un framework CSS no es nada complicado, tendrás que conocer sus particularidades para sacarle provecho. Tendrás también que aprender a diseñar de una manera determinada, para usar la rejilla con la que se posicionan los elementos. Todo esto, insisto, no es difícil, pero lleva su tiempo. Tanto es así que, si vas a diseñar una sola web, quizás tardes más tiempo en aprender a trabajar con el framework CSS que lo que tardarías en diseñarla sin utilizarlo. En definitiva, sólo sacarás provecho al framework cuando lo conozcas, es decir en el segundo, o siguientes diseños que realices con él.

No estás aprendiendo a valerte por ti mismo:

Muchos de los problemas de diseño típicos ya vienen resueltos en un framework y esa situación, a pesar de ser una ventaja, puede derivar en que al final no sepas solucionar esos problemas básicos por ti mismo. Como ya estaba resuelto, no tuviste la ocasión de resolver el problema y por tanto no aprendiste con esa experiencia. Del mismo modo, probablemente aprenderás a resolver tus necesidades aplicando clases CSS del framework, pero realmente puede que no sepas qué estilos CSS estás aplicando y por qué. En definitiva, si antes de empezar a trabajar con frameworks no tienes una experiencia y conocimiento suficiente del lenguaje CSS y sus usos, puede que aprender a diseñar con un framework represente una desventaja para aprender bien CSS.

Otro detalle sobre este mismo punto es que cuando quieras cambiar de Framework, tendrás que aprender de nuevo la forma de trabajar y con toda probabilidad cambiar el código HTML de tu página para ajustarlo a los nuevos nombres de clases y estilos CSS.

7.2.4.- Conclusión: ¿Usar o no frameworks CSS?

La decisión final sobre usar o no un framework debe correr a cargo de cada desarrollador. Un framework CSS no es nada del otro mundo, sino una serie de estilos CSS útiles para muchas personas, pero no por eso deben ser útiles para ti. Incluso, puede que a lo largo de tu experiencia ya hayas creado una serie de clases CSS y estilos básicos que ya incluyes en todos tus proyectos. Por decirlo de alguna manera, puede que ya estés usando tu propio "framework" rudimentario y no lo sepas.

En mi opinión, su conveniencia o no también depende de cómo utilices el framework. Si ya conoces CSS y te interesas un poco sobre cómo funcionan las cosas, puedes minimizar las desventajas que tienen e incluso pueden venirte bien para aprender. Si los usas sin conocer realmente CSS y sin importarte lo que tienen dentro y qué estás aplicando realmente cuando invocas las clases CSS puede que te resulten complicados y a la larga tampoco te beneficien mucho en tu línea de aprendizaje como desarrollador web.

Referencia: Si quieres aprender algún framework CSS te recomendamos estos manuales de DesarrolloWeb.com, sobre dos de los frameworks más populares:

- 1.- [Blueprint](#)
- 2.- [960 Grid System](#)

Artículo por Miguel Angel Alvarez

7.3.- Introducción a CSS 3

CSS 3 trae grandes novedades para el diseño de webs y algunos navegadores comienzan a implementar CSS 3.

Desde que CSS comenzó han pasado muchos años y ya vamos por la especificación de CSS3, que incorpora una serie de novedades que vamos a tratar de resumir en este artículo.

7.3.1.- Qué es CSS

Si no sabes lo que es CSS probablemente te interesaría comenzar leyendo nuestro [manual de CSS](#) o la sección de [CSS a fondo](#). No obstante, cabría decir que CSS es un lenguaje para definir el estilo o la apariencia de las páginas web, escritas con HTML o de los documentos XML. CSS se creó para separar el contenido de la forma, a la vez que permite a los diseñadores mantener un control mucho más preciso sobre la apariencia de las páginas.

7.3.2.- Con CSS 3, más control sobre la forma

El objetivo inicial de CSS, separar el contenido de la forma, se cumplió ya con las primeras especificaciones del lenguaje. Sin embargo, el objetivo de ofrecer un control total a los diseñadores sobre los elementos de la página ha sido más difícil de cubrir. Las especificaciones anteriores del lenguaje tenían muchas utilidades para aplicar estilos a las webs, pero los desarrolladores aun continúan usando trucos diversos para conseguir efectos tan comunes o tan deseados como los bordes redondeados o el sombreado de elementos en la página.

CSS 1 ya significó un avance considerable a la hora de diseñar páginas web, aportando mucho mayor control de los elementos de la página. Pero como todavía quedaron muchas otras cosas que los diseñadores deseaban hacer, pero que CSS no permitía especificar, éstos debían hacer uso de trucos para el diseño. Lo peor de esos trucos es que muchas veces implica alterar el contenido de la página para incorporar nuevas etiquetas HTML que permitan aplicar estilos de una manera más elaborada. Dada la necesidad de cambiar el contenido, para alterar al diseño y hacer cosas que CSS no permitía, se estaba dando al traste con alguno de los objetivos para los que CSS fue creado, que era el separar por completo el contenido de la forma.

CSS 2 incorporó algunas novedades interesantes, que hoy ya utilizamos habitualmente, pero CSS 3 todavía avanza un poco más en la dirección, de aportar más control sobre los elementos de la página.

Así pues, la novedad más importante que aporta CSS 3, de cara a los desarrolladores de webs, consiste en la incorporación de nuevos mecanismos para mantener un mayor control sobre el estilo con el que se muestran los elementos de las páginas, sin tener que recurrir a trucos o hacks, que a menudo complicaban el código de las web.

7.3.3.- Propiedades nuevas en CSS 3

He aquí una lista de las principales propiedades que son novedad en CSS3.

Bordes

- [border-color](#)
- [border-image](#)
- [border-radius](#)
- [box-shadow](#)

Fondos

- [background-origin](#)
- background-clip

- background-size
- [hacer capas con múltiples imágenes de fondo](#)

Color

- colores HSL
- colores HSLA
- [colores RGBA](#)
- Opacidad

Texto

- [text-shadow](#)
- text-overflow
- [Rotura de palabras largas](#)
- [Web Fonts](#)

Interfaz

- box-sizing
- resize
- outline
- nav-top, nav-right, nav-bottom, nav-left

Selectores

- Selectores por atributos

Modelo de caja básico

- [overflow-x, overflow-y](#)

Degradados CSS3

- [Degradados lineales](#)
- [Degradados radiales](#)
- [Degradados lineales de repetición](#)
- [Degradados radiales de repetición](#)

Otros

- media queries
- [creación de múltiples columnas de texto](#)
- propiedades orientadas a discurso o lectura automática de páginas web
- [animaciones CSS3](#)

Parte de este listado de nuevas propiedades de CSS 3 lo he sacado de: <http://www.css3.info/preview/>. Es un sitio en inglés, pero puede estar bien visitar para ir conociendo más detalles sobre CSS 3. Sin embargo, en ese sitio faltaban algunas cosas como los degradados o las animaciones, por lo menos cuando lo visitamos, por lo que hemos completado para la realización de este índice.

En futuros artículos ofreceremos algunas claves y explicaciones sobre varias de estas propiedades, al menos las más interesantes, así como ejemplos que sirvan para ir conociendo esta nueva especificación de CSS. Todo ello lo iremos colocando en el [Manual de CSS 3](#).

Artículo por Miguel Angel Alvarez

7.4.- Unidades CSS viewportwidth y viewportheight

Nuevas unidades de CSS3: viewportwidth y viewportheight. Cómo podemos hacer un útil uso de ellas al aplicarlas a tamaños de las tipografías.

En este artículo vamos a hablaros de dos unidades nuevas de CSS, que tienen especial utilidad en el diseño adaptable (conocido como "responsive") y que vienen a solucionar algunas necesidades a la hora de definir tamaños en dispositivos, sobre todo útiles para las fuentes (o tipografías). Se trata de las unidades CSS3 "viewportwidth" y "viewportheight", que son relativas a la anchura y altura del dispositivo, respectivamente.



Estas unidades son una opción interesante para definir alturas y anchuras en dispositivos. Las unidades son abreviadas con las siglas "vw" y "vh". Su medida es equivalente a un centésimo de la anchura o altura del dispositivo donde se está visualizando la web. Por ejemplo, si deseamos asignarle a un elemento una altura igual al tamaño completo del dispositivo, le aplicaremos el valor 100vp.

Por expresarlo mediante porcentajes, 1vw = 1% de la anchura del dispositivo. Por su parte, 1vh = 1% de la altura del dispositivo.

Nota: Para entender mejor estas medidas, si es que no lo sabes ya, te interesa [saber lo que es el "Viewport"](#). En resumen, viewport es igual a las dimensiones de la pantalla de tu móvil o tableta, mientras que en ordenadores de escritorio serían las dimensiones de la ventana del navegador.

La pega del viewportheight, bastante importante, es que solo funciona en dispositivos móviles y no en ordenadores de escritorio. Pero aun así merece la pena conocer y aplicar cuando realmente le podamos sacar partido, siempre en teléfonos y tabletas, porque en *desktop* tiene unos comportamientos un poco erráticos.

7.4.1.- Uso de viewportheight para definir alturas de una manera cómoda

Las alturas en CSS siempre son un quebradero de cabeza cuando se quiere usar medidas relativas, con unidades como "%" de CSS. Seguramente que si tenemos algo de experiencia sobre maquetación web se sabe de lo que estamos hablando. Para tener las alturas "controladas" con unidades relativas de CSS tenemos que darle dimensiones de altura a todos los contenedores. O sea, definir las dimensiones de height de todos los elementos de la jerarquía del documento hasta llegar al elemento que realmente queramos definir su altura con un valor porcentual. La opción para alturas es el viewportheight

El correspondiente "compañero" de Viewportheight para las anchuras, viewportwidth, no es tan crucial para definir las dimensiones width, debido a que los comportamientos de unidades CSS para las anchuras no nos dan ningún problema. Pero no conviene perderlo de vista, especialmente pensando en las tipografías.

7.4.2.- Medidas "viewport" para tipografías

Estas medidas, tanto viewportheight como viewportwidth, tienen un comportamiento especial que merece la pena mencionar. ¡Se puede usar en la tipografía! Esto quiere decir que podemos asignarle medidas en viewportwidth a un

texto y éste siempre te medirá lo mismo en todas las pantallas, relativamente a las dimensiones de la pantalla que tengamos. Para que nos enteremos, si definimos el tamaño de una fuente con `viewportwidth`, y si en un dispositivo vemos que ocupa un 50% de la anchura de su pantalla, podemos tener certeza de que, en todos los dispositivos ocupará más o menos ese mismo porcentaje de anchura de la pantalla, aunque estemos trabajando con una resolución diferente.

Lo asombroso de todo esto es que el tamaño de la letra es "responsive", o en castellano "adaptable", sin necesidad de utilizar ningún *script* para conseguirlo o de otras técnicas como *mediaqueries* que se podrían aplicar con CSS para intentar conseguir ese efecto de fluidez.

Os animamos a probar estas medidas de CSS para sacar vuestras conclusiones. Su uso no difiere de cualquier otro que puedas estar trabajando en tus hojas de estilo.

```
body{
  fontsize: 2.5vw
}
h1{
  fontsize: 4.8vh
}
```

Es una maravilla que podemos usar y al menos en dispositivos nos dará la tranquilidad de que esas fuentes se van a ver igual de grandes, relativamente al tamaño de la pantalla donde se esté mostrando la página.

Nota: Esta información la hemos extractado del evento transmitido en directo [Unidades de medida CSS #designIO](#), realizado en DesarrolloWeb.com con la participación de Daniel Martínez @Wakkos y Daniel Ruiz @MrViSiOn2, que nos ofrecieron mucha información sobre unidades de medida en hojas de estilo en cascada y con la mente en el diseño adaptable. Si te interesan las medidas CSS más adecuadas para diseños adaptables te recomendamos leer también el artículo [Unidades de medida CSS para fuentes Responsive Web Design](#).

Artículo por Daniel Martínez