

---

# HTML5 y CSS3

---

PID\_00253029

Àlex Bartrolí Muñoz

Jordi Collell Puig

Anna Ferry Mestres



Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia de Reconocimiento-NoComercial-SinObraDerivada (BY-NC-ND) v.3.0 España de Creative Commons. Podéis copiarlos, distribuirlos y transmitirlos públicamente siempre que citéis el autor y la fuente (FUOC. Fundació per a la Universitat Oberta de Catalunya), no hagáis de ellos un uso comercial y ni obra derivada. La licencia completa se puede consultar en <http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es>

# Índice

<b>1. Introducción.....</b>	<b>5</b>
<b>2. HTML5.....</b>	<b>6</b>
2.1. ¿Qué es HTML5? .....	6
2.2. El largo camino entre HTML4 y HTML5 .....	6
2.3. Novedades en HTML5 .....	7
2.4. Diferencias entre HTML4 y HTML5 .....	8
<b>3. CSS3.....</b>	<b>9</b>
3.1. Introducción al CSS .....	9
3.2. Breve historia de CSS .....	9
3.3. Familias de navegadores .....	10
3.4. Beneficios del uso del CSS3 .....	11
3.5. Novedades CSS3 .....	12
3.5.1. Nuevos selectores .....	12
3.5.2. Selectores de atributos .....	12
3.5.3. Combinadores .....	14
3.5.4. Seudoclases .....	15
3.5.5. Colores y opacidad .....	16
3.5.6. Opacidad ( <i>opacity</i> ) .....	17
3.5.7. Nuevas propiedades .....	18
3.5.8. Sombras ( <i>box-shadow</i> , <i>text-shadow</i> ) .....	19
3.5.9. Múltiples imágenes de fondo .....	20
3.5.10. Filetes ( <i>borders</i> ) con imágenes .....	21
3.5.11. Columnas de texto .....	22
3.5.12. WebFonts .....	23
3.5.13. Media queries .....	24
3.5.14. Transiciones CSS .....	24
3.5.15. Flexbox, el nuevo modelo de cajas flexibles .....	26
3.6. Ejercicios .....	27
<b>4. Preprocesadores CSS.....</b>	<b>29</b>
4.1. Ventajas de utilizar preprocesadores .....	29
4.2. Desventajas de los preprocesadores .....	30
4.3. Variables .....	30
4.4. Anidación .....	30
4.5. Funciones y mixins .....	31
4.6. Modularizar el código .....	31
4.7. Errores comunes .....	32
4.8. Cómo escoger un preprocesador .....	33
4.9. SASS .....	33
4.9.1. Instalar SASS .....	33

4.9.2.	Grunt y SASS .....	34
4.9.3.	SASS / SCSS .....	35
4.9.4.	Propiedades .....	37
4.9.5.	Selectores multilínea .....	37
4.9.6.	Importaciones .....	37
4.10.	LESS .....	38
4.10.1.	Instalación .....	38
4.10.2.	Variables .....	39
4.10.3.	Mixins .....	40
4.10.4.	Anidación de selectores .....	41
4.10.5.	Funciones y operaciones .....	42
4.10.6.	Compilar archivos LESS .....	43
4.11.	Stylus .....	43
4.11.1.	Compilar archivos Stylus .....	44
<b>5.</b>	<b>Bootstrap.....</b>	<b>46</b>
5.1.	¿Qué es Bootstrap? .....	46
5.2.	Descarga e instalación de Bootstrap .....	46
5.3.	Descarga y enlace de Bootstrap .....	47
5.4.	Diseño responsive en rejilla .....	48
5.5.	El elemento container .....	51
5.6.	Configuración inicial .....	52
5.7.	Otros elementos Bootstrap .....	53

## 1. Introducción

Cuando Tim Berners Lee inventó la World Wide Web, el lenguaje HTML solo se usaba para añadir estructura al texto. Los creadores podían marcar sus textos como títulos o párrafos usando las etiquetas HTML `<h1>` y `<p>` respectivamente.

A medida que la WWW fue ganando popularidad, los diseñadores empezaron a buscar posibilidades para añadir formato a los documentos en línea, así que los fabricantes de los navegadores (entonces Netscape y Microsoft) inventaron nuevas etiquetas HTML, entre las cuales se encontraban, por ejemplo, `<font>`, que se diferenciaba de las etiquetas originales HTML en que definían el formato y no la estructura.

Al mismo tiempo se dio el caso de que las etiquetas estructurales originales, especialmente `<table>`, se usaban cada vez más de manera incorrecta para dar formato a las páginas en vez de para añadir estructura al texto.

CSS se inventó para poner remedio a esta situación, proporcionando alternativas de formato a los diseñadores web soportadas por todos los navegadores, y a la vez separando la presentación de los documentos de su contenido, con la consecuente mejora en el mantenimiento de los sitios web.

En resumen, HTML se utiliza para estructurar el contenido y CSS para formatear el contenido estructurado previamente y ambos son la base de cualquier sitio web actual.

En este módulo veremos cómo han cambiado HTML5 y CSS3 respecto a la versión anterior y nos adentraremos en el mundo de los preprocesadores CSS, para ayudarnos a escribir hojas de estilos de una manera más sencilla, escalable y mantenible.

## 2. HTML5

En este apartado se recogen los fundamentos de HTML5, indicando el camino seguido para llegar de la cuarta a la quinta especificación del lenguaje HTML, las novedades aportadas por el HTML5 y las diferencias existentes entre la versión 4 y 5 de HTML.

Más que entrar en detalles técnicos, nos centraremos en aspectos más conceptuales para entender los pilares que sustentan el HTML5, utilizando una serie de ejercicios para poner en práctica los conceptos técnicos.

### 2.1. ¿Qué es HTML5?

HTML5 (HyperText Markup Language, versión 5) es la quinta revisión del lenguaje básico de la World Wide Web, HTML.

HTML5 especifica dos variantes de sintaxis para HTML:

- un «clásico» HTML (texto/html), conocida como HTML5 y
- una variante XHTML conocida como XHTML5, que es declarada como XML (XHTML) (application/xhtml+xml).

Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo, y actualmente es el estándar en la web.

### 2.2. El largo camino entre HTML4 y HTML5

La especificación de HTML 4.01 se completó en 1999. Cuando el HTML4 estaba a punto de cerrarse, el W3C concluyó que, en lo referente a los lenguajes de marcado, el futuro de la web no era HTML, sino XML y XHTML. Consecuentemente, el W3C se concentró en la especificación de XHTML1.0, acabada a principios del 2000.

XHTML1.0 es exactamente igual que HTML4.01, pero usando las reglas de sintaxis de marcado de XML. De cerca lo siguió el XHTML2.0, que añadía un montón de novedades y características. El problema del XHTML2.0 es que no era compatible con el marcado ya existente en la web y no reflejaba lo que los desarrolladores web estaban haciendo realmente. Por este motivo, en el 2004, un grupo de desarrolladores e implementadores (incluyendo representantes de Opera, Mozilla y, algo más tarde, Apple) se reunieron y formaron un grupo para realizar las especificaciones. Este grupo se denominó WHATWG y tenía como objetivo escribir una mejor especificación de marcado HTML, capaz de soportar la creación de la nueva generación de aplicaciones web, sin –y esto

es fundamental– romper la compatibilidad con el HTML existente. El resultado fue la especificación Web Applications 1.0, que documentaba los comportamientos y características existentes de los navegadores, así como nuevas características para la web, API y nuevas reglas de análisis del DOM.

En 2007, el W3C retomó el trabajo en HTML con un nuevo Grupo de Trabajo. Una de las primeras decisiones de este grupo fue adoptar la especificación Web Applications 1.0 y denominarla HTML5. Algunas de las características de HTML5 que garantizaron su buena acogida son:

- La compatibilidad con la web ya existente, que evita tener que aprender algún lenguaje nuevo.
- Añade nuevas y potentes características de HTML que antes solo estaban disponibles en la web utilizando tecnologías como plugin de Flash y/o JavaScript.
- Es más apto para escribir aplicaciones dinámicas que las versiones anteriores de HTML.
- Tiene un algoritmo de interpretación de documentos HTML claramente definido, de forma que todos los navegadores que lo implementen serán capaces de crear un mismo modelo DOM a partir de un mismo documento marcado.

### **2.3. Novedades en HTML5**

En cualquier caso, está claro que HTML5 ha sido un gran avance para el desarrollo de aplicaciones web y multimedia, y que HTML5, utilizado como término descriptivo de un conjunto de tecnologías, está a la altura de la web 2.0.

Entre las novedades de HTML5, son muchos los aspectos tratados que tienen más que ver con aplicaciones que con páginas web: el nuevo elemento <canvas>, los web sockets, las cachés de aplicación, el almacenamiento de datos en el cliente, etc.

Por otro lado, en el lenguaje de marcado, destacan ampliaciones del lenguaje y novedades como las nuevas características para los formularios, elementos nativos para audio y vídeo, nuevos elementos semánticos para definir la estructura de una página,...

Os recomendamos la lectura de este artículo sobre la explicación de las novedades y ventajas de HTML5 y los artículos que en él se referencian.

## 2.4. Diferencias entre HTML4 y HTML5

Antes de finalizar con las lecturas propuestas cabe decir que, al ser HTML5 una continuación del HTML4, quien conoce bien el HTML4 ya conoce una gran parte de HTML5.

Para aquellos que ya conozcan HTML4 y quieran profundizar en las diferencias existentes entre los dos, recomendamos leer el artículo del W3C que especifica las diferencias entre ambas versiones de HTML.

La otra referencia básica es la especificación de HTML5.



## 3. CSS3

### 3.1. Introducción al CSS

El CSS es un lenguaje de estilos creado para definir la presentación de documentos electrónicos HTML y XHTML. Nace de la necesidad de diseñar la información de tal manera que podemos separar el contenido de la presentación, y así, un mismo documento se puede visualizar correctamente en una infinidad de dispositivos diferentes, ya sea por pantalla (de medidas diferentes), impresora, lectores de voz o tabletas Braille.

Cuando se crea una página web, primero utilizamos el lenguaje HTML/XHTML para marcar los contenidos, es decir, para designar qué función tiene cada uno de los elementos dentro de la página: párrafo, titular, texto destacado, tabla, lista, pie, cabecera, etc.

Una vez tenemos la estructura, utilizamos el lenguaje CSS para definir el aspecto de cada elemento: tamaño, color y tipo de letra del texto, colores de fondo, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

La especificación del CSS la mantiene el World Wide Web Consortium (W3C).

### 3.2. Breve historia de CSS

Las diferentes revisiones de las hojas de estilo tienen el origen en la primera especificación publicada por el W3C en diciembre de 1996 y que pretendía unificar la sintaxis y la forma de definir una hoja de estilos por los diferentes lenguajes derivados del SGML.

Las principales características que se pueden definir mediante esta primera especificación son las siguientes:

- 1) Propiedades del tipo de letra así como el estilo de este.
- 2) Colores de los textos y de los fondos.
- 3) Atributos del texto tales como espacio entre caracteres, palabras y líneas.
- 4) Alineación de tablas, bloques de texto, imágenes, párrafos.

5) Márgenes externos e internos, filetes y posición de la mayoría de los elementos.

6) Definición única de elementos mediante ids y agrupamiento de atributos mediante *classes*.

En mayo de 1998 se publicó la segunda recomendación oficial, popularmente conocida como la especificación CSS2. Como características fundamentales que añade la revisión, estaba la posibilidad de definir posiciones de forma absoluta, relativa y fija, así como la profundidad de los elementos (z-index) cuando existen superposiciones, además de soporte para formatos de voz y textos bidireccionales. De esta, aparece la revisión 2.1, que corrige algunos errores encontrados en CSS2, elimina funcionalidades poco soportadas o inoperables en los navegadores y añade alguna nueva especificación.

La tercera revisión empezó en 2005 y a día de hoy sigue publicando actualizaciones.

A diferencia de las otras especificaciones, esta versión se ha dividido en módulos, de forma que disponemos de diferentes temas que pueden crecer y evolucionar en paralelo y no como un gran bloque monolítico con muchísimas revisiones. Hay más de 50 módulos, cada uno con un estatus diferente, y son adoptados por los fabricantes de software a diferente velocidad. Por ejemplo, «Selectores», «Espacios de nombres» y «Color» se convirtieron en recomendaciones oficiales del W3C en 2011, mientras que «Fondos y colores», «Consultas de medios» o «Diseños multicolumna» se encuentran en fase de «candidatos» y considerados razonablemente como estables.

### 3.3. Familias de navegadores

Los fabricantes de navegadores han pasado a ser los implementadores de las especificaciones indicadas por el W3C. A grandes rasgos todos implementan la especificación, pero tienen formas diferentes de interpretar y dibujar (*render*) el contenido con la hoja de estilos.

La parte del navegador que se encarga de interpretar el código HTML y CSS para mostrar las páginas se denomina motor, y para maquetar es más relevante la versión del motor que utiliza que la del navegador. Agrupamos los diferentes tipos de motores en cuatro familias de navegadores, que son las siguientes:

1) Familia Trident, basados en el motor de dibujo de Internet Explorer.

2) Familia Gecko, que utilizan el motor de dibujo Gecko, desarrollado por Mozilla. El más popular es Firefox, pero existen otras implementaciones en diferentes dispositivos.

3) Familia WebKit. Basados en el motor de dibujo WebKit, desarrollado por el navegador Konqueror y mejorado por el Safari de Apple. También tenemos el motor de dibujo de las dos plataformas de dispositivos móviles más extendidos, iOS (iPhone/iPad) y Android.

4) Familia Blink. Opera y Google Chrome utilizan el motor Blink, una bifurcación del proyecto WebKit.

Tened en cuenta que solo hablamos de la tecnología de dibujo, es decir, la parte del software que lee el documento SGML (*standard generalized markup language*), aplica los estilos que encuentra en las hojas de estilo y lo dibuja en la pantalla o en cualquier otro dispositivo de salida. Además de esto, un navegador también contiene un intérprete de JavaScript, que nos permite ejecutar secuencias de comandos e interactuar con el DOM (*document object model*), el conjunto de SGML con los estilos aplicados.

### 3.4. Beneficios del uso del CSS3

#### 1) Reducción del tiempo de desarrollo y mantenimiento

La llegada de las nuevas propiedades y métodos de CSS3 ha supuesto un cambio muy positivo en cuanto al tiempo de desarrollo, y es que nos permite ahorrarnos la realización de tareas que podían llegar a ser muy aburridas: por ejemplo, a la hora de hacer fondos con esquinas redondeadas, sombras o degradados, antes se tenían que hacer con imágenes, lo que nos obligaba a trabajar conjuntamente con un editor de gráficos. Esto también nos obligaba a poner mucho más código en la página (imágenes y divs), así que ahora hay menos código que renderizar y una maquetación más limpia y fácil de mantener.

#### 2) Incrementar el rendimiento de las páginas

Menos etiquetas HTML indican menos código a la hora de descargarse del servidor y menos código a la hora de interpretar y dibujar en el navegador. Esto supone dos ahorros: uno de ancho de banda y el otro de rendimiento del ordenador. Además, como ya hemos comentado, muchas de las técnicas de CSS3 nos ahorran imágenes, que todavía mejora más el rendimiento.

#### 3) Menos código JavaScript

En versiones anteriores necesitábamos utilizar JavaScript para funcionalidades de diseño que ahora se pueden resolver fácilmente con CSS3.

#### 4) La mejora progresiva

Uno de los elementos clave a la hora de emplear CSS es utilizar una técnica de desarrollo llamada mejora progresiva, que consiste en empezar por generar un código genérico que funcione en todos los navegadores para, poco a poco, ir introduciendo mejoras para navegadores más modernos. Esto es posible porque los intérpretes de CSS de los navegadores, si no conocen una propiedad, la ignoran.

Empleando esta técnica logramos un control total óptimo del aspecto, puesto que a mejores prestaciones del navegador, mejor visualización.

#### 5) Diseño adaptativo

Una de las características de CSS3 que permite maximizar la experiencia de usuario en los dispositivos móviles es usar las media queries. Estas nos permiten añadir estilos o reglas específicas según la medida de pantalla, la dirección del dispositivo o la densidad de píxeles.

Las media queries nos permiten crear una experiencia mejorada según las características de los dispositivos, puesto que los usuarios de dispositivos móviles esperan que los sitios web a los que acceden ofrezcan la misma experiencia que los ordenadores.

### 3.5. Novedades CSS3

#### 3.5.1. Nuevos selectores

Los selectores CSS son la herramienta más potente del lenguaje de estilos, puesto que nos permiten seleccionar diferentes elementos del contenido HTML en función de la etiqueta o de sus atributos, sin tener que hacer uso de su clase, su ID o JavaScript.

A continuación veremos las nuevas incorporaciones más interesantes, pero encontraréis la lista completa de selectores (de todas las especificaciones CSS) en la página de w3schools: [https://www.w3schools.com/cssref/css\\_selectors.asp](https://www.w3schools.com/cssref/css_selectors.asp)

#### 3.5.2. Selectores de atributos

[atrib]

Selecciona elementos que tienen el atributo «atrib».

[atrib=valor]

Selecciona elementos con un atributo «atrib» y cuyo valor sea exactamente «valor».

#### **[atrib~=valor]**

Selecciona elementos que contienen un atributo «atrib» con una lista de palabras separadas por espacios, una de las cuales sea «valor».

#### **[atrib|=valor]**

Selecciona los elementos que tienen el atributo «atrib» y el valor es una serie de palabras separadas con guiones, pero que empieza con «valor». Este tipo de selector solo es útil para los atributos de tipo lang que indican el idioma del contenido del elemento.

#### **[atrib^=valor]**

Selecciona elementos con un atributo «atrib» que empieza por «valor».

#### **[atrib\$=valor]**

Selecciona elementos con un atributo «atrib» que acaba con «valor».

#### **[atrib\*=valor]**

Selecciona elementos que contienen un atributo «atrib» que contiene «valor».

Los tres últimos selectores (atrib^=valor, atrib\$=valor y atrib\*=valor) son las nuevas incorporaciones en la versión CSS3.

Ejemplo:

```
/* Todos los span con el atributo "lang" en negrita */
span [lang] {font-weight: bold;}

/* Todos los span en portugués en verde */
span [lang="pt"] {color: green;}

/* Todos los span en inglés americano en azul */
span [lang ~="en-us"] {color: blue;}

/* Cualquier campo en chino en rojo, selecciona chino simplificado (zh-CN) y tradicional (zh-TW) */
span [lang |= "zh"] {color: red;}

/* Todos los enlaces internos con el fondo dorado */
a[href ^="#"] {background-color: gold}
```

```
/* Todos los enlaces en los que la URL finaliza en ".cn" en rojo */
a[href $=".cn"] {color: red;}

/* Todos los enlaces en los que la URL contiene "ejemplo" con el fondo gris */
a[href *="ejemplo"] {background-color: #CCCCCC;}
```

### 3.5.3. Combinadores

Los combinadores permiten aplicar una selección por contexto entre varios selectores simples. En la especificación CSS2.1 teníamos los siguientes:

- **Combinador descendente E F:** Selecciona un elemento F que descende de E. Los descendientes de E son sus elementos hijos, nietos, etc. El elemento combinador es el espacio entre E y F.
- **Combinador hijo E>F:** Selecciona un elemento F que es hijo de E, pero no selecciona los descendientes con más profundidad (nietos, biznietos, etc.). El carácter > es el elemento combinador.
- **Combinador hermanos adyacentes E + F:** Cuando dos elementos E y F comparten el mismo padre, se selecciona F si es inmediatamente precedido por E. El carácter + es el elemento combinador.

A CSS3 se incorpora el selector de hermanos general (siblings):

**Combinador hermano (sibling) E~F:** Una etiqueta hermana es la que existe en el mismo nivel o que tiene un padre en común y que está a continuación de la referenciada. El carácter ~ es el elemento combinador.

En el ejemplo siguiente podemos decir que son etiquetas hermanas los p y los h2, puesto que todos tienen el mismo padre (div#uno). Pero solo se seleccionarán los elementos p posteriores a la definición del h2.

```
<style>
    h2~p { color:green; }
</style>

<div id="uno">
<p>Este elemento no será visible porque es hermano pero anterior</p>
<h2>This is a test</h2>
<p>Este p es hermano de h2</p>
<p>Este otro p también es hermano de h2</p>
</div>

<div id="dos">
<p>Este p no se pintará</p>
```

```
</div>
```

Pintará de verde todos los p del div id=uno, puesto que son hermanos del h2. Tienen de padre en común el #uno, mientras que el p del div id=dos no se pintará, puesto que no tiene ningún hermano h2.

### 3.5.4. Seudoclases

Las pseudoclases son uno de los añadidos más extendidos en uso de CSS3. En la versión 2.1 teníamos la pseudoclase :first-child, pero se han sumado las siguientes:

#### Seudoclases de hijos:

- **:nth-child(n)** selecciona elementos basándose en la posición de los hijos. Puede utilizar números, expresiones y las palabras odd (impar) y even (par). De este modo podemos hacer el típico zebraado en las tablas de forma muy sencilla.
- **:nth-last-child(n)** sigue la misma idea que el anterior, pero selecciona a partir del último elemento.
- **:last-child** selecciona el último elemento de una lista (es lo mismo que :nth-child(1)).
- **:only-child** es equivalente a :first-child:last-child, o bien a :nth-child(1):nth-last-child(1), seleccionando un elemento que es hijo único (es primero y último hijo).

#### Seudoclases de tipos

Por otro lado, las pseudoclases de tipos tienen el mismo esquema de operación que las de hijos, pero se seleccionan hermanos con el mismo tipo de elemento, mientras que con la de hijos se seleccionan todos los hijos.

- **:nth-of-type(n)**
- **:nth-last-of-type(n)**
- **:first-of-type** que es igual a :nth-of-type(1)
- **:last-of-type** que es igual a :nth-last-of-type(1)
- **:only-of-type** que es igual a :first-of-type:last-of-type y también es igual a :nth-of-type(1):nth-last-of-type(1)

**Otras:**

- **:checked** selecciona los elementos de tipo input que estén seleccionados
- **:disabled** selecciona los elementos de tipo input que estén deshabilitados
- **:enabled** selecciona los elementos de tipo input que estén habilitados
- **:empty** selecciona elementos que están vacíos, es decir que no tienen hijos
- **:not** selecciona elementos que no cumplen la declaración especificada
- **:required** selecciona elementos de tipo input que tengan el atributo «required» especificado

### 3.5.5. Colores y opacidad

#### RGBA

La posibilidad de especificar colores en modo RGB ya existía, pero en la versión 3 se ve ampliada con posibilidad de añadir un canal alpha sobre el color que especifica la opacidad del elemento. De este modo podemos crear expresiones CSS del tipo:

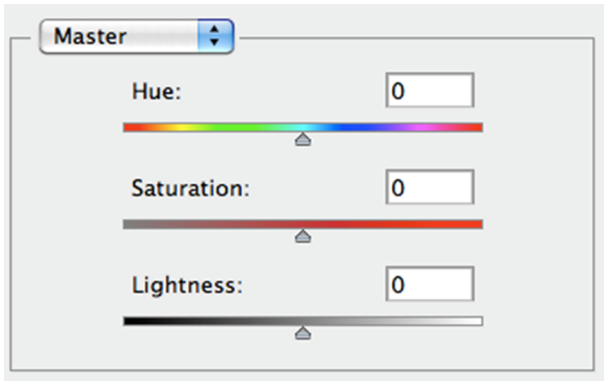
```
p { color: rgba(0,0,0,0.5) }
```

Crearé un color negro con una transparencia del 50 %. Para ver el efecto habrá que tener una imagen por debajo, o si nuestro color de fondo es blanco, nos aparecerá color gris.

#### Colores HSL

El modelo de color HSL (*hue, saturation, lightness*) define el color a partir de los tres parámetros tono, saturación y luz. Así, para una tonalidad de color dada, podemos alterar la saturación a partir del segundo parámetro y la cantidad de luz en el tercero.





Por ejemplo:



Será definida por los colores:

```
background:hsl(320, 100%, 10%);  
background:hsl(320, 80%, 30%);  
background:hsl(320, 60%, 50%);  
background:hsl(320, 40%, 70%);  
background:hsl(320, 20%, 90%);
```

También podemos añadir el componente de alpha si definimos como: `hsla(320, 80 %, 30 %, 0.7)`

### 3.5.6. Opacidad (*opacity*)

Como su nombre indica, la opacidad nos permite definir el nivel de transparencia por un selector. A diferencia del modelo RGB, la transparencia se aplica al objeto y a todos sus hijos y/o contenidos. Así, por ejemplo:

```
  
  
  
  

```

Los anteriores estilos en línea de las imágenes darían los siguientes resultados:



Esta propiedad es compatible con todos los navegadores modernos, pero en versiones de Internet Explorer anteriores a 9, para realizar el mismo efecto tenemos que utilizar esta alternativa:

```
filter: alpha(opacity = 50);
```

### 3.5.7. Nuevas propiedades

#### Esquinas redondeadas (*border-radius*)

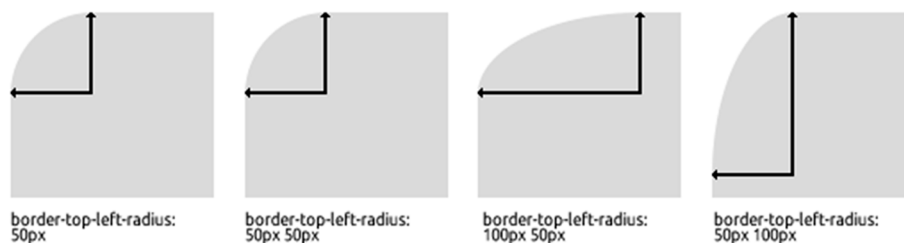
Permite redondear las esquinas de un elemento (div, span, button...) dado un radio, sin la necesidad de utilizar una imagen de fondo.

Podemos manipular las esquinas de una caja de manera uniforme utilizando la propiedad, *border-radius: 15px*; de este modo, tendríamos:

```
.bordered {
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
  border-radius: 5px;
}
```

Fijaos en la presencia de las propiedades con prefijos *-moz* y *-webkit*, que son específicas de los navegadores basados en Gecko y Webkit.

También podemos definir las propiedades para cada una de las esquinas utilizando las propiedades *border-bottom-left-radius*, *border-bottom-right-radius*, *border-top-left-radius*, *border-top-right-radius*. Estas propiedades admiten dos parámetros para definir el radio de curvatura de la esquina. Si definimos solo uno, conseguimos esquinas simétricas, mientras que si definimos los dos, estamos tratando independientemente el parámetro *x* del *y*, como se puede apreciar en el gráfico siguiente:



Por otro lado, la propiedad *border-radius* también puede ser empleada para definir de una vez las cuatro esquinas de la caja:

```
border-radius: 5px 10px 5px 10px / 10px 5px 10px 5px;
border-radius: 5px;
border-radius: 5px / 10px;
```

En la primera definición, el primer grupo de 4 medidas definen las propiedades horizontales de los cuatro radios, mientras que el segundo grupo utilizado separado por el carácter / define los radios verticales. En el segundo ejemplo se definen todos con un radio de 5 px. En el tercero se define un radio horizontal de 5 px y uno vertical de 10 px para las 4 esquinas.

Todas estas propiedades están disponibles en los navegadores basados en Webkit, Gecko, Opera, y las últimas versiones de Internet Explorer.

### 3.5.8. Sombras (**box-shadow**, **text-shadow**)

En la versión 2 de la especificación se introdujo la posibilidad de generar sombras directamente desde código CSS. Posteriormente, en la versión 2.1 se quitó y se ha vuelto a introducir en la versión 3.

Las propiedades para generar sombras en una caja son las siguientes:

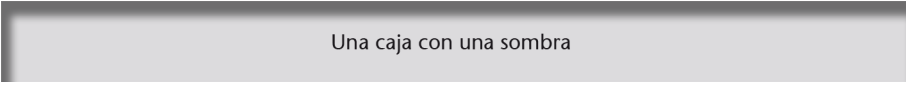
```
-moz-box-shadow: 10px 10px 5px #888;  
-webkit-box-shadow: 10px 10px 5px #888;  
box-shadow: 10px 10px 5px #888;
```

El código anterior generará:



Una caja con una sombra

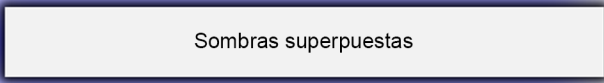
La propiedad box-shadow admite una lista de 5 elementos que definen (en este orden): desplazamiento horizontal, desplazamiento vertical, desenfoque, extensión y color. Opcionalmente se puede añadir la palabra clave inset, que permite hacer que la sombra sea interna en lugar de externa:



Una caja con una sombra

Podemos tener más de una sombra por elemento, es decir, que se pueden superponer como efecto de papel cebolla. Por ejemplo:

```
box-shadow: 0 0 10px 5px black, -20px 0px 50px blue;
```



Sombras superpuestas

Generará dos sombras, una negra y una azul. Podemos encadenar diferentes sombras simplemente separando la lista por comas.

Por último, la propiedad `text-shadow` se comporta del mismo modo y nos permite generar sombras en los textos. El primer parámetro es la posición de la sombra horizontal, el segundo la posición vertical y el tercero, que es opcional, el radio de difuminación. Igual que con `box-shadow`, `text-shadow` también permite sumar varias sombras separadas con comas.

Podemos generar una sombra sencilla y plana:

```
text-shadow: 2px 2px #aaa;
```

**Texto con sombra**

Una sombra difuminada:

```
text-shadow: 2px 2px 5px #666;
```

**Texto con sombra**

Sombras superpuestas:

```
text-shadow: 2px 2px 5px #666;
```

**Texto con sombra**

### 3.5.9. Múltiples imágenes de fondo

Otra característica interesante de CSS3 es la posibilidad de incluir varias imágenes de fondo en un mismo elemento. Por ejemplo:

```
#example1 {  
  width: 500px;  
  height: 250px;  
  background-image: url(fondo1.png), url(fondo2.png);  
  background-position: center bottom, left top;  
  background-repeat: no-repeat;  
}
```

Fijaos en que definimos dos imágenes de fondo: fondo1.png y fondo2.png, y también dos posiciones diferentes para cada una. Una centrada en la parte inferior y la otra en la parte superior izquierda.

### 3.5.10. Filetes (*borders*) con imágenes

A pesar de que no es muy habitual, también podemos definir imágenes para los filetes de nuestros bloques de la siguiente manera:

```
background-image: url(fondo1.png) 25% repeat;
```

El porcentaje es relativo a la parte que queremos emplear y finalmente, si queremos que se repita como un patrón o queremos que solo se muestre una sola vez. Veamos un ejemplo. Dada una imagen:



Queremos utilizarla como fondo utilizando el siguiente código:

```
#uno {  
  padding:30px;  
  border-width:10px 10px 10px 10px;  
  border-image: url('fondo.png') 10 100 10 10 repeat stretch;  
  background-color:#fff;  
}
```

Obtenemos el siguiente resultado:

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Primero definimos un ancho de filete (10 píxeles) con la propiedad `border-width`, después definimos la imagen que utilizaremos y las proporciones en píxeles (también admite porcentajes) de imagen por cada una de las esquinas. Finalmente, tenemos la propiedad (`stretch/repeat`) por si queremos que repita o se ajuste.

### 3.5.11. Columnas de texto

Otra de las novedades es la posibilidad de trabajar con columnas de texto. Con la anchura actual de las pantallas ha sido necesario, puesto que un ancho demasiado grande en los párrafos de texto afecta a la legibilidad de este. Las propiedades a emplear son:

```
-webkit-column-count: 3;  
-moz-column-width: 200px;  
-webkit-column-width:200px;  
-moz-column-gap: 20px;  
-webkit-column-gap: 20px;
```

El código anterior nos generará una estructura de 3 columnas siempre y cuando puedan cumplir la medida de ancho de columna preferido fijada con `-column-width` (si no fijáramos ninguna, emplearía la parte proporcional del área disponible). Con la propiedad `*-column-gap: 20px`, definimos el margen que queremos entre columnas. Finalmente, con `column-rule` podemos definir un filete que divida las columnas.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet,

consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet,

consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

### 3.5.12. WebFonts

Otra de las posibilidades que resultan muy atractivas a la hora de emplear los CSS3 y que han facilitado mucho la vida de los maquettadores web es la capacidad de adjuntar tipografías a un documento HTML. Esto se consigue con la etiqueta `@font-face`, que como el ejemplo siguiente, nos permite definir una tipografía.

```
@font-face {  
    font-family: Gentium;  
    src: url(gentium.otf);  
}
```

Esta etiqueta nos permite adjuntar al documento una tipografía, pero los problemas vienen en los diferentes formatos que cada navegador y versión entiende del archivo de tipografías, así que el resultado para una correcta visualización en todos los navegadores se complica de esta manera:

```
@font-face {  
    font-family: 'UbuntuRegular';  
    src: url('webfont.eot');  
    src: url('webfont.eot?#iefix') formato('embedded-opentype'),  
        url('webfont.woff') formato('woff'),  
        url('webfont.ttf') formato('truetype'),  
        url('webfont.svg#UbuntuRegular') formato('svg');  
    font-weight: normal;  
    font-style: normal;  
}
```

De este modo conseguimos que en todos los navegadores se visualice la fuente correctamente. Hay herramientas que nos ayudan a generar todas las versiones de la fuente, como por ejemplo el webfont generator de Font Squirrel. Este servicio, dado un archivo ttf u OpenType, nos generará las versiones que hemos de subir al servidor y también la definición CSS para emplear en nuestro documento.

Viendo el lío existente y la necesidad de los diseñadores de trabajar con tipografías personalizadas, se han creado servicios en línea que permiten adjuntar tipografías a nuestros documentos sin tenernos que preocupar ni de licencias ni de conversiones de archivos.

El más popular actualmente, sin ningún tipo de duda, es el servicio de Google Fonts, una gran colección de tipografías OpenSource que podemos adjuntar a nuestros documentos.

También está la alternativa de Adobe Typekit, un servicio de pago que permite hacer uso de tipografías comerciales.

Ambos servicios se encargan de proporcionar las diferentes versiones de la tipografía para cada versión del navegador facilitándonos un pequeño código CSS que nos dará acceso al uso de la tipografía.

### 3.5.13. Media queries

Desde la versión 2.1 de CSS existe la posibilidad de definir estilos en función del uso de la hoja: uno para pantalla (*screen*), otro para impresión (*print*) y otro para lectores de voz (*voice*). A partir de la especificación 3 esto va un paso más allá y nos permite definir también estilos específicos para las diferentes medidas de la pantalla. Por ejemplo, la definición siguiente:

```
@media screen and (max-width: 600px) {  
    .class { background: #ccc; }  
}
```

Define estilos específicos para navegadores con un ancho de pantalla más pequeño que 600 píxeles. También lo podríamos hacer directamente con una hoja de estilos separada empleando la etiqueta:

```
<link rel="stylesheet" media="screen and (min-width: 600px)" href="small.css" />
```

En este caso estaríamos refiriéndonos a tamaños de pantalla mayores de 600 píxeles.

Hay dos propiedades diferentes que podemos emplear: *min-width* hace referencia al área visible en estos momentos en el dispositivo, mientras que con *device-width* hace referencia a la resolución del dispositivo.

De este modo podemos crear la versión móvil de una web simplemente utilizando CSS.

### 3.5.14. Transiciones CSS

Las transiciones CSS son pequeños cambios en propiedades de la hoja de estilos, llamados por eventos generados por interacciones del usuario, como por ejemplo, cuando el ratón pasa por encima de algo (*:hover*) o bien un campo de formulario se encuentra activo. En una transición, estos cambios a las propiedades se producen de forma progresiva durante un intervalo de tiempo.

Veamos el siguiente ejemplo:

```
<a href="#" class="boto">Botón por transición</a>
```



Encima de estas líneas está el enlace al documento HTML sobre el que tendrá lugar la transición, y a continuación vemos el código CSS:

```
a.boto {
    text-decoration:none;
    color:#fff;
    padding: 5px 10px;
    background: #f76c6c;
    -webkit-transition-property:background;
    -webkit-transition-duration: 0.5s;
    -webkit-transition-timing-function:ease;
}
a.boto:hover {
    background: #d22828;
}
```

En este caso tenemos una transición que se aplica al fondo del botón y tendrá lugar al pasar el cursor por encima (hover). Del código anterior tenemos que aclarar una serie de cosas:

- La transición no se tiene que poner dentro del evento (en este caso el selector hover) sino en el elemento que tiene que cambiar.
- Lo que definimos en la transición es la propiedad que queremos animar, en este caso el background.
- Podemos definir la duración y también la interpolación de tiempo. En este caso utiliza una función de aceleración. Disponemos de las siguientes funciones: ease, linear, ease-in, ease-out, ease-in-out y cubic-bezier.
- También podemos definir un retraso en la ejecución de la animación con la propiedad -webkit-transition-delay: 0.5s.
- Se puede escribir de manera compacta en una sola línea:

```
-webkit-transition: background 0.3s ease 0.5s;
```

Se pueden definir transiciones de más de una propiedad separándolas con una ( , ):

```
-webkit-transition: background .3s ease, color 0.2s linear;
```

También tenemos la opción de aplicar la transición de forma que todas las propiedades cambien con la instrucción *all*:

```
transition: all 3s ease, color 0.2s linear;
```

Aquí podremos encontrar una lista de las propiedades que pueden ser animadas (transiciones): <http://www.w3.org/tr/css3-transitions/#properties-from-css->.

### 3.5.15. Flexbox, el nuevo modelo de cajas flexibles

Además de los modelos de posicionamiento absoluto/relativo y de cajas flotantes, aparece un nuevo modelo de estructuración de los elementos: el modelo de cajas flexibles.

Este modelo permite colocar los elementos de una página para que se comporten de forma predecible cuando el diseño de la página deba acomodarse a diferentes medidas de pantalla y diferentes dispositivos. En muchos casos el modelo «caja flexible» produce una mejora sobre el modelo «bloque» porque no utiliza la propiedad *float* ni hace que los márgenes del contenedor flexible interfieran con los márgenes de sus contenidos.

Lo que caracteriza un diseño flexible es su habilidad para alterar la anchura y altura de sus elementos para ajustarse lo mejor posible al espacio disponible en cualquier dispositivo. Un contenedor flexible expande sus elementos para llenar el espacio libre o los comprime para evitar que sobrepasen el área prevista.

Veamos un ejemplo sencillo partiendo de un contenedor con 7 elementos:

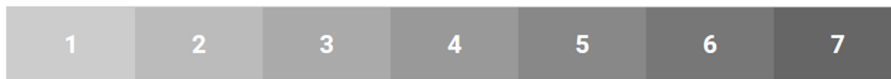
```
<div class="contenedor">
  <div class="element">1</div>
  <div class="element">2</div>
  <div class="element">3</div>
  <div class="element">4</div>
  <div class="element">5</div>
  <div class="element">6</div>
  <div class="element">7</div>
</div>
```

Aparte de estilos de diseño definidos para visualizar el ejemplo, colocamos de inicio una medida de 25 % de anchura para los elementos en relación con el contenedor padre. Para empezar a utilizar flexbox, añadir al contenedor la propiedad *display: flex*.

```
.contenedor{
  display: flex;
}

.elemento{
  width: 25%;
}
```

El resultado es:



Al no haber definido todavía ningún comportamiento de dirección y tamaño de los elementos, se ignora el valor de 25 % que se ha indicado en la anchura y los elementos se adaptan a su padre ocupando el 100 % del ancho entre la suma de todos. Este es el comportamiento «flexible» que indica su nombre.

También tiene propiedades para manipular las direcciones y comportamientos de los ítems. Mediante *flex-direction* podemos modificar la dirección del eje principal del contenedor para que se oriente en horizontal (por defecto) o en vertical. Tiene estos posibles valores:

- **row** - Establece la dirección del eje principal en horizontal
- **row-reverse** - Establece la dirección del eje principal en horizontal (invertido)
- **column** - Establece la dirección del eje principal en vertical
- **column-reverse** - Establece la dirección del eje principal en vertical (invertido)

Así, si al contenedor del ejemplo anterior le añadimos la propiedad *flex-direction: row-reverse*, obtendremos lo siguiente:



Flexbox es un modelo tan completo que necesitaríamos todo un módulo para verlo en profundidad, así que después de ver un par de pinceladas podéis encontrar una documentación más extensa en la web de W3schools, CSS3 Flexbox<sup>1</sup>, en la documentación de Mozilla MDN, CSS3 Flexbox<sup>2</sup> o bien jugar con las diferentes opciones que ofrece flexbox en las siguientes webs:

- <https://demo.agektmr.com/flexbox/>
- <http://the-echoplex.net/flexyboxes/>

### 3.6. Ejercicios

1. Hacer unos botones CSS con efectos de degradado interno y sombra exterior y animación en el hover.

<sup>(1)</sup>[https://www.w3schools.com/css/css3\\_flexbox.asp](https://www.w3schools.com/css/css3_flexbox.asp)

<sup>(2)</sup>[https://developer.mozilla.org/es/docs/web/css/css\\_flexible\\_box\\_layout/usando\\_las\\_cajas\\_flexibles\\_css](https://developer.mozilla.org/es/docs/web/css/css_flexible_box_layout/usando_las_cajas_flexibles_css)

2. Maquetar un pequeño layout usando media queries, webfonts, esquinas redondas borders en imágenes.

3. Maquetar un pequeño layout usando flexbox, media queries y webfonts.

### **Bibliografía**

#### **Adobe Typekit**

<https://typekit.com/>

#### **Blink (Wikipedia)**

<https://es.wikipedia.org/wiki/Blink>

#### **CSS Flexbox (W3schools)**

[https://www.w3schools.com/css/css3\\_flexbox.asp](https://www.w3schools.com/css/css3_flexbox.asp)

#### **CSS Selectors (W3schools)**

[https://www.w3schools.com/cssref/css\\_selectors.asp](https://www.w3schools.com/cssref/css_selectors.asp)

#### **CSS3 transition properties (W3)**

<http://www.w3.org/TR/css3-transitions/#properties-from-css->

#### **Google Chrome (Wikipedia)**

[https://es.wikipedia.org/wiki/Google\\_Chrome](https://es.wikipedia.org/wiki/Google_Chrome)

#### **Google Fonts**

<https://fonts.google.com/>

#### **Media queri**

<http://www.mediaqueri.es>

#### **Usando las cajas flexibles (MDN docs)**

[https://developer.mozilla.org/es/docs/Web/CSS/CSS\\_Flexible\\_Box\\_Layout/Usando\\_las\\_cajas\\_flexibles\\_CSS](https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Usando_las_cajas_flexibles_CSS)

#### **Webfont Generator (Font Squirrel)**

<https://www.fontsquirrel.com/tools/webfont-generator>

#### **WebKit (Wikipedia)**

<https://es.wikipedia.org/wiki/WebKit>

## 4. Preprocesadores CSS

La aparición del lenguaje CSS en 1996 significó una importante mejora en el proceso de estilizar la estructura HTML de un sitio web, pero tiene la limitación de que se trata de uno que otorga estilos mayormente estáticos (es decir, que no varían después de la carga de la página), a no ser que se complemente con JavaScript. Además, las hojas de estilo cada vez son más grandes, más complejas y más difíciles de mantener.

A raíz de esto aparecieron los preprocesadores de CSS: unas aplicaciones que permiten escribir hojas de estilo en metalenguajes con más características que el CSS, pero que compilan a CSS convencional para que las puedan interpretar los navegadores. Podríamos decir que tenemos un lenguaje de programación que genera CSS.

Con los preprocesadores se extienden las funcionalidades de un CSS tradicional, permitiéndonos tener variables, nidificaciones, herencias, funciones, *mixins*, reutilizar código, tener más flexibilidad en el momento del desarrollo, etc. El objetivo de estos preprocesadores es tener un código más sencillo de mantener y editar.

Actualmente hay muchos preprocesadores, pero los más populares son SASS, LESS y Stylus (en este orden).

### 4.1. Ventajas de utilizar preprocesadores

- **Reducción del tiempo de desarrollo**, puesto que se puede escribir menos código y de manera más limpia.
- **Reutilización**. Por ejemplo, si varias clases tienen que usar un color, podemos almacenarlo en una variable, evitando escribir varias veces el mismo valor en formatos incómodos como RGB o hexadecimal.
- **Características dinámicas**. Utilización de hojas de estilo CSS con comportamientos dinámicos (variables, clases, operaciones, sentencias, métodos y funciones). Por ejemplo, se puede asignar al ancho de un bloque un valor dinámico, expresado a través de una operación aritmética que pueda incluir variables. De este modo, el ancho no tendrá un valor fijo en píxeles ni uno relativo a las dimensiones de la pantalla, sino que dependerá del factor que se ha determinado.
- **Código más organizado**. Se puede hacer un uso mejorado de la propiedad `@import` de CSS, lo que permite separar el código en varios archivos para

mejorar la organización y que al compilar a CSS se genere un solo archivo minificado, que será el que descargará el usuario final.

- **Proyectos más fáciles de mantener.** Al poder separar el código en múltiples archivos, resulta más fácil mantener los proyectos, puesto que se pueden organizar de forma modular.

## 4.2. Desventajas de los preprocesadores

- **Código complejo.** Si el lenguaje no se escribe correctamente, el código compilado puede ser repetitivo y tener una gran cantidad de líneas de código innecesarias, aumentando así el peso del archivo CSS que se descarga el usuario final.
- **Malas prácticas.** Utilizar un metalenguaje de CSS sin conocer a fondo el verdadero lenguaje puede malacostumbrar a los desarrolladores inexpertos a depender de herramientas de terceros.
- **Tienen que aprender a usarse.** A pesar de que la curva de aprendizaje no es demasiado pronunciada, el hecho de tener que aprender una nueva tecnología desmotiva a algunos desarrolladores, especialmente a los maquettadores de la «vieja escuela».

## 4.3. Variables

Una de las principales características de los preprocesadores es que nos permiten tener variables. Gracias a las variables podemos almacenar valores y reutilizarlos en cualquier parte del código. Nos ahorrarán mucho trabajo cuando tengamos que editar un valor que se repite a lo largo de nuestro código.

## 4.4. Anidación

Con CSS, nos encontramos con que muchas veces tenemos que definir estilo para clases e ids del mismo objeto, de forma que toca definir cada estilo por separado:

```
ul {}  
ul li {}  
ul li a {}
```

Con los preprocesadores de CSS podemos crear anidaciones, de tal manera que un código como el anterior queda de la siguiente manera:

```
ul {  
  li {
```

```
a {  
    ...  
}  
}  
}
```

Así pues, con la anidación de los selectores podemos escribir un conjunto de reglas de manera limpia y ordenada. Esta forma de escribir nuestras reglas hace el código mucho más legible y que haya que escribir menos líneas de código.

#### 4.5. Funciones y mixins

Los mixins son conjuntos de propiedades agrupados bajo un mismo nombre. Gracias a las funciones y los mixins podemos evitar escribir código duplicado, a pesar de que también los podemos usar para evitar escribir continuamente los prefijos propietarios de cada navegador. En un proyecto con mucho código nos ayudará mucho a evitar la repetición.

Por ejemplo, si queremos que dos elementos tengan un fondo de color gris, un margen de 30 píxeles y texto blanco, podemos englobar estas características bajo el nombre «MiMixin» e incluirlo en la declaración de los dos elementos. Así se genera un código más comprensible y menos repetitivo.

También podemos aprovechar las funciones para crear parrillas, triángulos u otras posibilidades con CSS que necesitamos repetir en nuestro código.

#### 4.6. Modularizar el código

Muchos preprocesadores ofrecen la característica de importar archivos. En lugar de escribir todo el código en un solo CSS o tener muchos archivos CSS independientes, tendremos varios archivos preprocesables, que al final acabarán unificados en un solo fichero con el CSS resultante.

Gracias a esto podremos modularizar nuestros archivos de una forma lógica y sencilla de mantener, teniendo diferentes módulos para cada elemento de la web. Por ejemplo, podemos tener un archivo en el cual se definan los estilos de los botones, otro para los formularios, otro para el sistema de parrilla, etc. E igualmente podríamos dividirlo por páginas creando un archivo para la página inicial, otro para el bloque, otro para la sección de contacto, y así sucesivamente.

Al hacer nuestro código de esta forma, encontrar lo que queremos modificar resulta mucho más sencillo. Si se quiere cambiar el color de los botones, se va al archivo de los botones y se modifica. No perdemos tiempo en buscar dónde está el código de los botones en un archivo inmenso.

## 4.7. Errores comunes

Para utilizar correctamente un preprocesador, es muy importante conocer antes cómo funciona CSS. Como se ha visto, los preprocesadores son herramientas con muchas ventajas y que pueden resultar muy útiles, pero si no se tiene cuidado se puede obtener un CSS poco óptimo. A continuación veremos uno de los ejemplos más comunes.

Una de las características de la cual abusan mucho algunos desarrolladores es la anidación, que permite anidar selectores y ahorrarnos escribirlos enteros de nuevo. Por ejemplo:

```
div {  
  p {  
    ...  
  }  
}
```

Da como resultado en CSS:

```
div {}  
div p {}
```

Si abusamos de la anidación obtendremos selectores largos y lentos, que además incrementan el peso del archivo. Hacer esto:

```
html {  
  body {  
    div {  
      p {  
        span {...}  
      }  
    }  
  }  
}
```

Se exportaría a CSS así:

```
html {}  
html body {}  
html body div {}  
html body div p {}  
html body div p span {}
```

Si seguimos anidando, cada vez habrá selectores más grandes, que pueden ser un problema si tenemos un proyecto grande con mucho código.



Por estos detalles es conveniente conocer bien CSS. Un preprocesador no hace que seamos mejores desarrolladores si no sabemos distinguir si el CSS resultante que produce es bueno.

#### 4.8. Cómo escoger un preprocesador

Se dice que LESS es el más simple y más sencillo de aprender, mientras que SASS y Stylus tienen funciones más avanzadas. Por otro lado, LESS y SASS son los que tienen una mayor comunidad, lo que significa que siempre habrá documentación actualizada por la propia comunidad.

Lo más recomendable es probar los tres y ver con cuál nos sentimos más cómodos. Hay quien utiliza Stylus por su sintaxis más ágil, otros prefieren mantener la sintaxis clásica de CSS y no necesitan funciones muy avanzadas y optan por LESS.

Al final, como no hay unas diferencias muy grandes entre ellos, una vez se ha visto cómo funciona uno, no es muy complicado adaptarse a otro. A continuación veremos unas pinceladas de ellos.

#### 4.9. SASS

Syntactically Awesome Style Sheets<sup>3</sup> (literalmente «Hojas de estilo sintácticamente impresionantes») es una extensión de CSS que permite utilizar variables, reglas anidadas, mixins, importaciones en línea y mucho más, todas con una sintaxis completamente compatible con CSS.

<sup>(3)</sup>Página oficial de SASS: <http://sass-lang.com/>.

##### 4.9.1. Instalar SASS

Hay dos maneras de instalar SASS: mediante alguna aplicación o bien mediante la línea de comandos en la consola.

No entraremos a detallar el funcionamiento de cada uno de los programas con que se puede utilizar SASS, sino que nos centraremos en su utilización básica con línea de comandos, puesto que será similar a las instalaciones de los otros preprocesadores que veremos.

Para obtener el listado completo y actualizado de aplicaciones disponibles y más información sobre la instalación de SASS con línea de comandos, diríjase a la sección de instalación de la página oficial de SASS: <http://sass-lang.com/install>.

Antes de poder instalar SASS por línea de comandos se debe tener instalado Ruby<sup>4</sup> en el equipo. Los Macs ya tienen Ruby instalado por defecto, pero en caso de usar Windows o Linux, se tendrá que instalar Ruby antes<sup>5</sup>. Después, desde la consola de comandos hay que escribir la instrucción:

```
gem install sass
```

<sup>(4)</sup>Ruby es el lenguaje de programación interpretado, reflexivo y orientado a objetos con el que se ha creado SASS.

<sup>(5)</sup>Podéis encontrar aquí el instalador de Ruby: <https://rubyinstaller.org/>.

Una vez instalado, se puede ejecutar de la siguiente manera:

```
sass entrada.scss salida.css
```

Donde «entrada.scss» es el fichero de origen donde se ha escrito el código SASS, y «salida.css» es el fichero CSS compilado. De este modo la compilación se hace una sola vez, pero también se le puede decir que vigile los cambios sobre el fichero de entrada, de forma que cada vez que detecte que se ha actualizado el fichero, se compila de nuevo en el archivo de salida:

```
sass --watch entrada.scss:salida.css
```

O podemos vigilar una carpeta entera:

```
sass --watch ruta/carpetasass
```

La línea de órdenes de arriba verá todos los ficheros .scss / .sass en ruta / directorio y cada vez que uno de los ficheros de este directorio se cambie, SASS actualizará los ficheros correspondientes o creará uno si no hay ninguno.

Si necesitamos SASS para generar los ficheros en un directorio específico, lo podemos hacer de este modo:

```
sass --watch ruta/carpetasass:ruta/carpetacss
```

También podemos ver un fichero específico en vez del directorio con esta línea de órdenes:

```
sass --watch ruta/carpetasass:ruta/carpetacss
```

#### 4.9.2. Grunt y SASS

Muchos desarrolladores usan herramientas para trabajar más ágilmente y una de las más populares es Grunt. Se utiliza para realizar de manera automática tareas frecuentemente utilizadas, como por ejemplo minificación, compilación, pruebas unitarias, etc.

SASS puede integrarse con Grunt para que este se encargue de vigilar si ha habido cambios para actualizar el fichero CSS de salida. Para hacer esto hay que utilizar la librería *grunt-contrib-sass* y configurar el fichero *gruntfile.js* de forma similar a esto:

```
modulo.exports = function(grunt) {  
  grunt.initConfig({  
    sass: {  
      dist: {  
        files: {  
          'css/estilos.css' : 'scss/estilos.scss'  
        }  
      }  
    },  
    watch: {  
      css: {  
        files: 'scss/*.scss',  
        tasks: ['sass']  
      }  
    }  
  });  
  grunt.loadNpmTasks('grunt-contrib-sass');  
  grunt.loadNpmTasks('grunt-contrib-watch');  
  grunt.registerTask('default', ['watch']);  
}
```

Con la tarea watch se consigue que Grunt se mantenga a la espera de modificaciones de cualquier fichero SCSS, compilando un nuevo archivo CSS de manera automatizada.

#### 4.9.3. SASS / SCSS

Hay dos sintaxis disponibles para SASS. Por un lado encontramos **SCSS** (Sassy CSS), que es una extensión mejorada de la sintaxis de CSS. Cada hoja de estilo CSS válido es un fichero SCSS válido con el mismo significado. Los ficheros que utilicen esta sintaxis tienen la extensión *.scss*.

La otra sintaxis es más antigua y se conoce simplemente como **SASS** o de «sintaxis indentada». Esta utiliza la indentación en lugar de los corchetes para indicar la nidificación de los selectores y las líneas nuevas en vez de los bordes y punto para separar las propiedades. Es de escritura más ágil, puesto que para escribir el código se utilizan menos caracteres y por su indentación puede resultar más clarificadora y limpia.

#### Ved también

Si queréis saber más sobre la utilización de Grunt, dirigíos a su documentación.

Al contrario que SCSS, SASS no es compatible con CSS: si pegamos un extracto de CSS en una hoja SASS, el compilador nos devolverá un error. Los ficheros que utilicen esta sintaxis tienen la extensión `.sass`.

Cada expresión de SASS, como por ejemplo declaraciones de propiedades y selectores, se tiene que colocar en su propia línea. Además, todo lo que estaría dentro de `{ }` después de una declaración tiene que estar en una nueva línea y sangrar un nivel más profundo que esta afirmación. Por ejemplo, este CSS:

```
#main {  
  color: grey;  
  font-size: 0.9em;  
}
```

Sería este SASS:

```
#main  
  color: grey  
  font-size: 0.9em
```

O este SCSS:

```
#main {  
  color: grey;  
  font-size: 0.9em;  
  
  a {  
    font: {  
      weight: bold;  
      family: serif;  
    }  
    &:hover {  
      background-color: #eee;  
    }  
  }  
}
```

Sería este SASS:

```
#main  
  color: grey  
  font-size: 0.9em  
  
  a  
    font:  
      weight: bold  
      family: serif
```

```
&:hover
  background-color: #eee
```

En general, la mayoría de la sintaxis CSS y SCSS trabaja de manera directa en SASS utilizando líneas nuevas en lugar de punto y coma, e indentación en vez de llaves. Sin embargo, hay algunos casos en los que hay diferencias o sutilezas.

#### 4.9.4. Propiedades

La sintaxis indentada admite dos maneras de declarar propiedades CSS. La primera es como CSS, excepto sin el punto y coma. La segunda, sin embargo, coloca el color antes que el nombre de la propiedad. Por ejemplo:

```
#main
  :color grey
  :font-size 0.9em
```

Por defecto se pueden utilizar las dos maneras, a pesar de que se puede restringir el uso de una de ellas configurando el entorno mediante la opción `:property_syntax`.

#### 4.9.5. Selectores multilínea

Normalmente, en la sintaxis indentada, un selector único tiene que ocupar una sola línea. Aun así, hay una excepción: los selectores pueden contener líneas nuevas siempre que solo aparezcan después de las comas. Por ejemplo:

```
.users #userTab,
.posts #postTab
  width: 100px
  height: 30px
```

#### 4.9.6. Importaciones

La directiva `@import` de SASS no requiere comillas dobles, a pesar de que se pueden utilizar opcionalmente. Por ejemplo, este SCSS:

```
@import "themes/dark";
@import "font.sass";
```

Sería este SASS:

```
@import themes/dark
@import font.sass
```

## 4.10. LESS

LESS<sup>6</sup> nació en 2009, dos años después de SASS, y lo hizo también construido en el lenguaje Ruby, pero posteriormente fue reescrito en JavaScript. Las funcionalidades de LESS y SASS son muy similares, pero la sintaxis es ligeramente diferente.

<sup>(6)</sup>Página oficial de LESS: <http://lesscss.org>.

La principal diferencia entre LESS y otros precompiladores CSS es que LESS permite la compilación en tiempo real vía less.js en el navegador. LESS se puede ejecutar tanto del lado del cliente como del lado del servidor, o se puede compilar en CSS sin formato.

### 4.10.1. Instalación

Hay varias maneras de utilizar LESS. Como se ha comentado en el apartado anterior, LESS funciona con JavaScript y esto nos permite incluir el archivo .less directamente en nuestra web y que un fichero less.js haga la traducción a css «al vuelo».

Para trabajar de este modo, primero se tendrían que configurar las opciones de LESS antes de cargar el script:

```
<script type="text/javascript">
  less = {
    env: "development", // o "production"
    async: false,        // carga las importaciones de manera asíncrona
    fileAsync: false,    // carga las importaciones async cuando una
                        // página usa un protocolo de archivos
    poll: 1000,          // Tiempo (en milisegundos) entre las consultas
                        // mientras se encuentra en modo de seguimiento (watch)
    functions: {},       // funciones del usuario, especificadas por nombre
    dumpLineNumbers: "comments", // o "mediaQuery" o "all"
    relativeUrls: false, // ajusta las URLs relativas si false,
                        // relativas respecto al fichero less de entrada
    rootpath: ":/a.com/" // ruta para añadir al principio de las URLs
  };
</script>
<script src="less.js"></script>
```

Y finalmente enlazamos el archivo .less que contendrá todos nuestros estilos:

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

Esta forma de utilizarlo es útil y práctica en la fase de desarrollo porque así nos ahorramos compilar el archivo en cada cambio, pero a la hora de publicar el proyecto en internet se recomienda incluir solo el archivo CSS ya compilado

y minimizado para que el rendimiento de la web sea más óptimo (este paso lo podemos hacer utilizando un convertidor en línea y de este modo no hemos tenido que instalar nada).

Igual que se ha visto con SASS, hay varias aplicaciones para trabajar con LESS, pero no entraremos en detalle con cada una de ellas<sup>7</sup> sino que lo que veremos a continuación es cómo instalar LESS en nuestro equipo mediante la consola de comandos.

<sup>(7)</sup>El listado de aplicaciones disponibles para trabajar con LESS tanto en línea como en la máquina local lo encontraremos en el apartado de utilización de su página oficial: <http://lesscss.org/usage/index.html>.

### Instalación de paquetes con NPM

Cada vez más, tanto los desarrolladores frontend como los desarrolladores backend utilizan la consola de comandos para trabajar de forma más ágil en sus proyectos y gran parte de «culpa» la tienen herramientas como NPM.

NPM (*node package manager*) es un gestor de paquetes que permite administrar los módulos y dependencias que necesitamos en un proyecto local. Mediante esta herramienta podemos instalar librerías en nuestros proyectos escribiendo una sola línea en la consola de comandos. Entonces NPM lo descarga de su repositorio al servidor y lo instala allá donde nos convenga.

Para poder utilizar NPM necesitamos tener el entorno de desarrollo Node.js en nuestro equipo, lo que podemos hacer descargándonos el instalador en la siguiente dirección: <https://nodejs.org/es/>.

Desde la versión 0.6.3 de Node.js el gestor NPM ya viene integrado, así que una vez tengamos Node.js en nuestra máquina no habrá que hacer ningún paso adicional.

La manera más sencilla y rápida de instalar LESS en el equipo de desarrollo es a través de NPM, el gestor de paquetes NodeJS, con la siguiente instrucción:

```
$ npm install -g less
```

#### 4.10.2. Variables

Una pequeña diferencia respecto a SASS es que LESS usa el símbolo @ para las variables, que es el signo que SASS utiliza para mixins. Y para variables, SASS utiliza el carácter \$.

Las variables se definen de la siguiente manera:

```
@color-principal: #1D365D;  
@color-secundario: #172B4A;  
@color-texto: #222;
```

```
@anchura-imágenes: 300px;  
@radio: 5px;
```

Las variables en LESS pueden tener ámbito global o local:

```
@colorbase: red;  
body{  
    @colorbase: blue;  
    color: @colorbase;  
}  
  
//Salida CSS:  
body {  
    color: blue;  
}
```

#### 4.10.3. Mixins

Vemos un ejemplo del uso de mixins en LESS. Recordemos que los mixins nos permiten incluir reglas dentro de otras. Por ejemplo, tenemos la siguiente regla:

```
.bordered {  
    -webkit-border-radius: 5px;  
    -moz-border-radius: 5px;  
    border-radius: 5px;  
}
```

Si ahora queremos añadir las propiedades de la clase «.bordered» en otra clase, hacemos lo siguiente:

```
.box {  
    border: 1px solid #ccc;  
    color: #444;  
    .bordered;  
}
```

Al incluir la clase «.bordered» dentro de la clase «.box» hacemos que la segunda herede las propiedades definidas en la primera, de modo que el CSS generado será el siguiente:

```
.box {  
    border: 1px solid #ccc;  
    color: #444;  
    -webkit-border-radius: 5px;  
    -moz-border-radius: 5px;  
    border-radius: 5px;
```



```
}
```

Los mixins también se pueden comportar como funciones y pasar argumentos por parámetro, como se ve en el siguiente ejemplo:

```
// LESS

.rounded-corners (@radius: 5px) {
  -webkit-border-radius: @radius;
  -moz-border-radius: @radius;
  -ms-border-radius: @radius;
  -o-border-radius: @radius;
  border-radius: @radius;
}

.box {
  .rounded-corners;
}

footer {
  .rounded-corners(10px);
}

/* CSS compilado */

.box {
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
  -ms-border-radius: 5px;
  -o-border-radius: 5px;
  border-radius: 5px;
}

footer {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -ms-border-radius: 10px;
  -o-border-radius: 10px;
  border-radius: 10px;
}
```

#### 4.10.4. Anidación de selectores

Al contrario del CSS tradicional, no necesitamos escribir nombres largos de selectores para especificar una relación de herencia, sino que podemos anidar selectores dentro de los otros:

```
// LESS

header {
  h1 {
```

```
    font-size: 26px;
    font-weight: bold;
  }
  p { font-size: 12px;
    a { text-decoration: none;
      &:hover { border-width: 1px }
    }
  }
}

/* CSS compilado */
header h1 {
  font-size: 26px;
  font-weight: bold;
}
header p {
  font-size: 12px;
}
header p a {
  text-decoration: none;
}
header p a:hover {
  border-width: 1px;
}
```

#### 4.10.5. Funciones y operaciones

Las operaciones permiten sumar, restar, multiplicar y dividir los valores de las propiedades y colores, dándonos el poder de crear relaciones complejas entre propiedades. Por ejemplo:

```
// LESS
@the-border: 1px;
@base-color: #111;
@red:        #842210;

#header {
  color: (@base-color * 3);
  border-left: @the-border;
  border-right: (@the-border * 2);
}
#footer {
  color: (@base-color + #003300);
  border-color: desaturate(@red, 10%);
}

/* CSS compilado */
```

```
#header {  
  color: #333;  
  border-left: 1px;  
  border-right: 2px;  
}  
  
#footer {  
  color: #114411;  
  border-color: #7d2717;  
}
```

#### 4.10.6. Compilar archivos LESS

Ya hemos visto que podemos compilar archivos LESS en línea o con software de terceros, pero seguimos centrándonos en la utilización del compilador mediante la línea de comandos.

```
$ lessc styles.less
```

En el comando anterior, «styles.less» es el nombre de nuestro archivo LESS. Y como no le hemos dado ninguna instrucción adicional compilará el archivo .css dentro de la misma carpeta donde se encuentra el archivo .less.

Podéis visitar la página oficial para ver el resto de las opciones disponibles para la compilación mediante la consola de comandos o bien utilizar alguna herramienta automatizadora de tareas (como Grunt, que lo hemos visto con SASS) para disponer de opciones avanzadas y una mejor automatización del proceso. En este caso, la librería que necesitaríamos sería *grunt-contrib-less*.

#### 4.11. Stylus

Stylus<sup>8</sup> es un lenguaje que proporciona una forma eficiente, dinámica y expresiva de generar CSS. Apoyando tanto una sintaxis indentada como la escritura típica del CSS.

<sup>(8)</sup>Página oficial de Stylus: <http://stylus-lang.com>.

Es menos conocido y utilizado que SASS y LESS, pero también cuenta con bastantes admiradores por su agilidad a la hora de escribir la sintaxis y la simplicidad a la hora de leer.

Para instalar Stylus vía NPM, tenemos que escribir en el terminal la siguiente instrucción:

```
$ npm install -g stylus
```

La sintaxis de Stylus tiene un estilo similar a lenguajes de programación como Python o Ruby. No se usan llaves para delimitar los bloques como CSS, sino que se usa la indentación.

Para la indentación se pueden usar tantos espacios como se deseen, pero hay que ser consistente: si usamos 2 espacios, que sean 2 espacios en todo el código.

Como ya hemos visto, en Stylus se pueden omitir los dos puntos y los punto y coma. Esto hace que programar en Stylus sea más simple y cometamos menos errores por olvidar poner alguno de estos caracteres. Aun así, si no nos gusta esta sintaxis, Stylus permite usar la sintaxis tradicional de CSS, o combinar las dos. Así, podríamos hacer lo siguiente:

```
/* Código Stylus */  
body {  
  font-family Roboto  
  font-size 16px  
  color #444  
}
```

O esto:

```
/* Código Stylus */  
body  
  font-family: Roboto;  
  font-size: 16px;  
  color: #444;
```

O esto otro:

```
/* Código Stylus */  
body  
  font-family: Roboto  
  font-size: 16px  
  color: #444
```

Podemos escribir el código de las tres maneras y Stylus lo acepta porque es muy permisivo, pero eso sí, la consistencia es importante, así que la forma escogida se tendría que seguir en todo el código.

#### 4.11.1. Compilar archivos Stylus

Para compilar un fichero .styl y convertirlo en un archivo corriente CSS, habrá que ir a nuestro terminal o a nuestra línea de comandos y situarnos en la carpeta del proyecto. Una vez allá, escribiremos el siguiente código:

```
stylus estil.styl
```

Esto compilará el archivo `estil.styl` a un archivo `estil.css` una sola vez, pero si queremos que Stylus se quede observando los cambios para que vaya compilando automáticamente cada vez que se cambia el fichero `.styl`, entonces lo que escribiremos es lo siguiente:

```
stylus -w estil.styl
```

La diferencia con la línea anterior es `-w`, que quiere decir **watch** (observar), de manera similar (pero simplificada) a como se hace con SASS.

Si además queremos que el archivo resultante esté comprimido en una sola línea, añadimos también el parámetro `-c` (compress).

También podemos vigilar una carpeta entera y dar una ruta para especificar dónde se guardarán el fichero o ficheros CSS resultantes:

```
stylus -w -c ruta/carpetastylus -o ruta/carpetacss
```

En este caso, Stylus se queda observando «carpetastylus» y cuando detecte una modificación, compilará el archivo CSS dentro de `carpetacss` y además lo hará comprimiendo el fichero en una sola línea.

Igual que con SASS y LESS, para la automatización de la compilación y minificación también podemos utilizar algún gestor de automatizaciones de tareas como GruntJS. En este caso la librería que necesitaríamos sería *grunt-contrib-stylus*.

## 5. Bootstrap

### 5.1. ¿Qué es Bootstrap?

Bootstrap es un framework de presentación para el desarrollo de aplicaciones web. Nació en el año 2011 por parte del equipo de Twitter como una solución para fomentar la consistencia entre las herramientas internas. La primera versión de Bootstrap, después de meses de trabajo, era una manera de documentar, compartir bienes y patrones de diseño dentro de la compañía.

A pesar de que empezó como solución interna de la compañía Twitter, en agosto del 2011 liberaron Bootstrap como un proyecto de código abierto en GitHub, y pocos meses más tarde, en febrero del 2012, se convirtió en el proyecto de desarrollo más popular de la plataforma.

Desde su lanzamiento, la comunidad de Bootstrap ha sido muy amplia y activa, y se ha convertido en el framework de presentación más popular para el desarrollo de proyectos responsivos.

Bootstrap contiene plantillas de diseño con tipografía, formularios, botones, tablas, menús de navegación y otros elementos de diseño basados en HTML y CSS, así como extensiones adicionales de JavaScript opcionales.

### 5.2. Descarga e instalación de Bootstrap

Bootstrap se puede descargar en versión compilada o el paquete de desarrollador que incluye el código fuente. En función del tipo de descarga que se realice, la estructura de los directorios será diferente.

La versión compilada de Bootstrap es la forma más sencilla de utilizar la librería en un proyecto web. Cada archivo cuenta con dos variantes:

- los archivos compilados (su nombre empieza por bootstrap.\*)
- los archivos compilados y comprimidos (su nombre empieza por bootstrap.min\* )

Esta versión también incluye las fuentes de los iconos Glyphicons y el estilo opcional de Bootstrap. Su estructura de directorios es:

```
bootstrap/  
├─ css/  
│   ├─ bootstrap.css  
│   ├─ bootstrap.min.css  
│   ├─ bootstrap-theme.css  
│   └─ bootstrap-theme.min.css  
├─ js/  
│   ├─ bootstrap.js  
│   └─ bootstrap.min.js  
└─ fonts/  
    ├─ glyphsicons-halflings-regular.eot  
    ├─ glyphsicons-halflings-regular.svg  
    ├─ glyphsicons-halflings-regular.ttf  
    └─ glyphsicons-halflings-regular.woff
```

La versión de código fuente de Bootstrap, además de las versiones compiladas de los archivos CSS y JavaScript, incluye el código fuente y toda la documentación.

Los directorios `less/`, `js/` y `fonts/` contienen el código fuente, y el directorio `dist/` contiene la versión compilada de estos. El directorio `docs/` contiene la documentación y en su interior el directorio `examples/` contiene varios ejemplos de uso. Su estructura de archivos es la siguiente:

```
bootstrap/  
├─ less/  
├─ js/  
├─ fonts/  
├─ dist/  
│   ├─ css/  
│   ├─ js/  
│   └─ fonts/  
└─ docs/  
    └─ examples/
```

### 5.3. Descarga y enlace de Bootstrap

La versión compilada se descarga en la sección de «Download» de la web <http://getbootstrap.com>, mientras que la descarga del código fuente se descarga en GitHub: <https://github.com/twbs/bootstrap/releases>.

La descarga del código de la fuente en formato SASS, para la integración de Bootstrap en aplicaciones de Ruby On Rails se descarga desde: <http://github.com/twbs/bootstrap-sass/releases>.

Para utilizar Bootstrap en un proyecto web solo hace falta enlazar los archivos principales de CSS y JS en el proyecto.

Si se quiere utilizar el CDN de Bootstrap, las instrucciones para enlazar Bootstrap son:

```
<!-- Última versión compilada y comprimida del CSS -->
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css"
integrity="sha384-BVYiISiFeKldGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous"></link>

<!--Tema opcional -->
< link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap-theme.min.css"
integrity="sha384-rHyoNliRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp"
crossorigin="anonymous"></link>

<!--Última versión compilada y comprimida del Javascript -->
<s script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNlCPD7Txa"
crossorigin="anonymous"></script >
```

## 5.4. Diseño responsive en rejilla

Para implementar los diseños responsive, Bootstrap utiliza un diseño conocido como diseño en rejilla, en el cual una página se divide en filas y cada fila a la vez está dividida en 12 columnas.

Bootstrap utiliza las media queries para definir cuatro tamaños de pantalla:

- Pantallas grandes (llamadas LG) a partir de 1.200 píxeles
- Pantallas medianas (llamadas MD) a partir de 992 píxeles
- Pantallas tipo mesita (llamadas SM) a partir de 768 píxeles
- Pantallas de dispositivos móviles (llamadas XS), por debajo de 768 píxeles.

Cada fila se divide en 12 columnas, independientemente del tamaño de la pantalla desde la que se visualice y lo que cambia en cada dispositivo es la anchura de cada columna.



A la hora de realizar el diseño, mediante una clase se indica el tamaño deseado para cada elemento en cada uno de los dispositivos.

Para entenderlo mejor, ponemos un ejemplo, en el que tenemos una web con cuatro bloques, que queremos que se visualicen de la siguiente manera para cada tipo de pantalla:

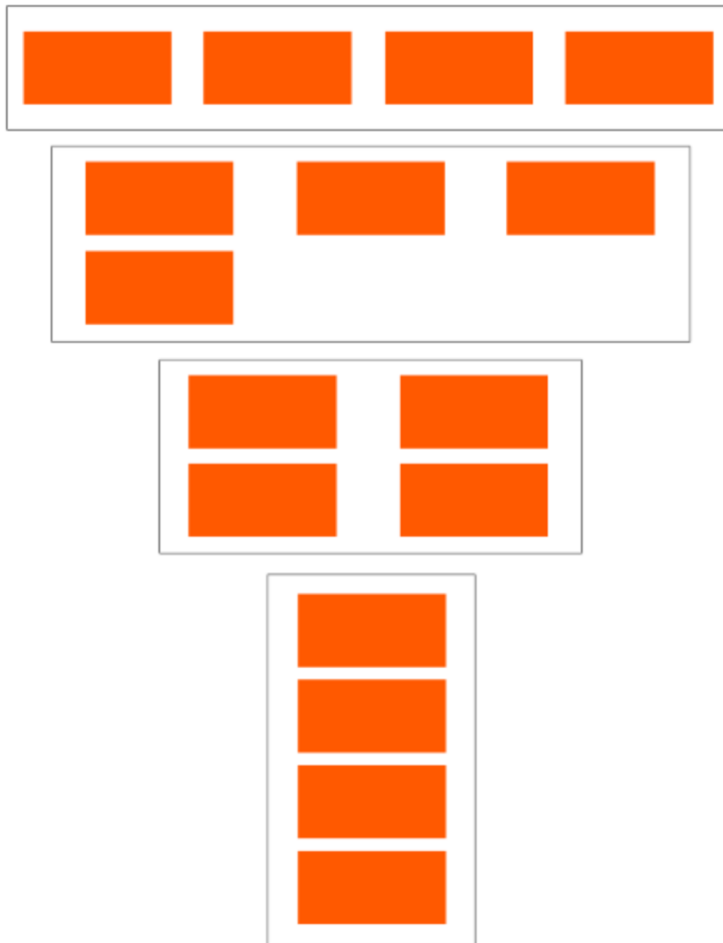
- Resolución LG: queremos los 4 bloques en fila. Como hay 12 columnas, cada bloque ocupará 3 columnas.
- Resolución MD: queremos 3 bloques por fila, Como hay 12 columnas, necesitaremos que cada bloque ocupe 4 columnas.
- Resolución SM: queremos 2 bloques por fila. Como hay 12 columnas, cada bloque ocupará 6 columnas.
- Resolución XS: queremos 1 bloque por fila: es decir, cada elemento ocupará las 12 columnas.

El código para lograr cada uno de estos requerimientos en los diferentes tipos de pantalla es:

```
<div class="row">
  <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">bloque 1</div>
  <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">bloque 2</div>
  <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">bloque 3</div>
  <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">bloque 4</div>
</div>
```

(De hecho, el «col-xs-12» nos lo podemos ahorrar, puesto que por defecto, si no se indica nada, ya tomaría el 100 % de la anchura de la pantalla).

La apariencia de los bloques con el código indicado y en cada tipo de resolución será:



Además de los elementos ya vistos, Bootstrap tiene otros elementos importantes en su diseño en rejilla:

**Para ocultar un bloque** en una determinada medida de pantalla se utiliza la clase «hidden», que en función de la medida de pantalla en la que queremos que se aplique se representará mediante las clases: hidden-xs, hidden-sm, hidden-md, hidden-lg

**Si lo que queremos es hacer visible un bloque** en una medida concreta, las clases a utilizar tienen el formato «.visible-medida-display». En versiones anteriores a 3.2.0 teníamos las clases .visible-xs, .visible-sm, .visible-md y .visible-lg. Todavía se pueden usar, pero en realidad están obsoletas, puesto que se han sustituido por las nuevas en las que indicamos el tipo de display.

El problema que tenían las clases antiguas era que, en todos los casos, tenían la propiedad de CSS «display» como block, pero a menudo nos puede interesar que se visualice con un display de tipo «inline» o «inline-block».

Grupo de clases	CSS display
.visible-*-block	display: block;
.visible-*-inline	display: inline;

Grupo de clases	CSS display
.visible-*-inline-block	display: inline-block;

Así, por ejemplo, para pantallas extrapequeñas (xs) tendríamos las clases .visible-xs-block, .visible-xs-inline y .visible-xs-inline-block.

**Para cambiar el orden de dos bloques**, en función del dispositivo se pueden utilizar las clases «push» y «pull».

En el siguiente código se intercambian las posiciones de los dos primeros elementos para pantallas de tipo SM (a partir de 768 píxeles), pasando el elemento con la clase «push» al final y el elemento con la clase «pull» al principio.

```
<div class="row">
  <div class="col-sm-4 col-sm-push-8">bloque1</div>
  <div class="col-sm-8 col-sm-pull-4">bloque2</div>
</div>
```

**Para desplazar un bloque a la derecha** y ampliar así el margen con su bloque predecesor en una línea, Bootstrap dispone de las clases col-sm-offset-\*, col-md-offset-\* y col-lg-offset-\*, donde el símbolo \* indica el número de columnas a la derecha que se quiere desplazar el bloque. El siguiente código muestra un bloque de 4 columnas, seguido de un margen de 4 columnas en el centro y otro bloque de 4 columnas para finalizar la fila.

```
<div class="row">
  <div class="col-md-4">bloque 1</div>
  <div class="col-md-4 col-md-offset-4">bloque 2</div>
</div>
```

**Evitar saltos de línea en parrillas.** Cuando se muestra un listado de elementos en línea, es muy probable que algunos de ellos tengan una altura diferente y los elementos no empiecen a la misma altura en líneas diferentes. Para evitar problemas donde las alturas sean diferentes, Bootstrap dispone de la clase «clearfix», que, unida a la clase «visible», permite **crear saltos de línea y evitar la superposición de elementos**.

```
<div class="clearfix visible-xs"></div>
```

## 5.5. El elemento container

Todas las filas de una página están englobadas por un elemento contenedor. Este elemento puede tener una anchura fija o una anchura variable.

Si la anchura es fija, se utiliza la clase «container» y si es de anchura variable, se utiliza la clase «container-fluid».

El elemento «container» tiene una anchura fija para cada uno de los tipos de pantalla definidos en Bootstrap, exceptuando el de medida más pequeña, que adquiere el 100 % de la anchura. Si una pantalla es más grande que la anchura que tiene Bootstrap definida para este tipo de dispositivo, se asigna un «margin» de forma automática a ambos lados del contenedor, que hace que el elemento aparezca centrado en la pantalla.

En cambio, para los contenedores globales de tipo «container-fluid», el diseño ocupa toda la pantalla, sin ningún margen lateral, y la anchura de cada columna se adapta para que las 12 columnas ocupen el 100 % de la pantalla del navegador.

La siguiente tabla muestra un resumen de todos los conceptos trabajados hasta el momento:

	<b>XS</b> (<768 px)	<b>SM</b> (≥768 px)	<b>MD</b> (≥992 px)	<b>LG</b> (≥ 1.200 px)
<b>Prefijo de la clase</b>	col-xs	col-sm	col-md	col-lg
<b>Anchura máxima contenedor</b>	Ninguno (auto)	728 px	940 px	1.170 px
<b>Anchura máxima columna</b>	Auto	60 px	78 px	95 px
<b>Separación entre columnas</b>	30 px (15 px a cada lado)			
<b>Permite anidación</b>	Sí	Sí	Sí	Sí
<b>Permite desplazar columnas</b>	No	Sí	Sí	Sí
<b>Permite reordenar columnas</b>	No	Sí	Sí	Sí

## 5.6. Configuración inicial

Bootstrap utiliza algunos elementos HTML y algunas propiedades CSS que requieren el uso del DOCTYPE de HTML5. Por lo tanto, la declaración de los documentos tiene que empezar con la declaración del documento HTML5:

```
<!DOCTYPE html>
<html lang="es">
```

En la cabecera, para que las páginas se muestren correctamente en dispositivos móviles y se pueda hacer un zoom en los elementos, en el <head> del documento se tiene que añadir la siguiente instrucción:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

### 5.7. Otros elementos Bootstrap

Además de la estructura en rejilla, Bootstrap ofrece muchas características que se agrupan en las siguientes secciones:

- Tipografía
- Código
- Tablas
- Formularios
- Botones
- Imágenes
- Mensajes de información y ayuda

Para todas las secciones, Bootstrap ofrece clases específicas que facilitan el diseño de cada uno de los elementos. La mayoría de las mejoras se basan en colores, medidas, formas y márgenes que permiten validar los estados y transiciones de una aplicación web, como por ejemplo, errores al introducir los datos en un formulario o la indicación de que se ha cumplimentado correctamente; mensajes de advertencia o error, tablas de gran medida adaptadas a diferentes tamaños de pantalla, etc.

Todas y cada una de las opciones para cada elemento de estilo de Bootstrap están disponibles en su especificación, que se encuentra en la URL: <http://getbootstrap.com/css/>.

