



Esta página fue traducida del inglés por la comunidad, pero no se mantiene activamente, por lo que puede estar desactualizada. Si desea ayudar a mantenerlo, descubra cómo activar las configuraciones regionales inactivas.

Cascada y herencia

El objetivo de este artículo es desarrollar la comprensión de algunos de los conceptos fundamentales de CSS (cascada, especificidad y herencia) que controlan cómo se aplica el CSS al HTML y cómo se resuelven los conflictos.

A medida que avances en este apartado verás que puede resultar menos relevante y un poco más académico que otros artículos, pero la comprensión de estas cuestiones te ahorrará problemas más adelante. Te animamos a que trabajes meticulosamente este apartado y verifiques que entiendes los conceptos antes de continuar.

Prerrequisitos: Conocimientos básicos de informática, tener el [software básico instalado](#), conocimientos básicos de [trabajar con archivos](#), HTML básico (véase [Introducción a HTML](#)) y una idea de cómo funciona el CSS (véase [Primeros pasos con CSS](#)).

Objetivo: Aprender qué son la cascada y la especificidad, y cómo funciona la herencia en CSS.

Reglas conflictivas

CSS significa **hojas de estilo en cascada** (cascading style sheets), y es muy importante entender la palabra *cascada*. La forma en que se comporta la cascada es la clave para comprender el CSS.

En algún momento trabajarás en un proyecto y encontrarás que el CSS que pensabas que

debería aplicarse a un elemento no funciona. Por lo general, el problema suele ser que has

debería aplicarse a un elemento no funciona. Por lo general, el problema suele ser que has creado dos normas que podrían aplicarse al mismo elemento. La **cascada**, y el concepto estrechamente relacionado de **especificidad** son mecanismos que controlan qué regla se aplica cuando aparecen tales conflictos. Es posible que la regla que se aplique finalmente a tu elemento no sea la que esperas, por lo que debes comprender cómo funcionan estos mecanismos.

También es significativo el concepto de **herencia**, que significa que algunas propiedades CSS heredan por defecto los valores establecidos en el elemento padre, pero otras no. Esto también puede causar una respuesta diferente a la que esperas.

Vamos a empezar por echar un vistazo rápido a los principales elementos que nos interesan, y a continuación veremos cómo interactúan entre sí y con tu CSS. Pueden resultar un poco difíciles de entender, pero a medida que practiques escribiendo CSS te resultará más fácil de entender la manera cómo funcionan.

Cascada

En un primer nivel de simplicidad, la **cascada** en las hojas de estilo significa que el orden de las reglas importa en CSS: cuando dos reglas tienen la misma especificidad, se aplica la que aparece en último lugar en el CSS.

En el ejemplo siguiente tenemos dos reglas que pueden aplicarse al `h1`. El `h1` acaba siendo de color azul porque estas normas tienen un selector idéntico y, por lo tanto, tienen la misma especificidad. Por esta razón, se aplica la última que aparece.

This is my heading.

```
h1 {  
  color: red;  
}  
h1 {  
  color: blue;  
}
```

```
<h1>This is my heading.</h1>
```

  Reset

Especificidad

La especificidad es el modo que tiene el navegador de decidir qué regla se aplica si diversas reglas tienen selectores diferentes pero podrían aplicarse a un mismo elemento. Básicamente, la especificidad mide cuán específica es la selección de un selector:


- Un selector de elemento es menos específico (selecciona todos los elementos de aquel tipo que aparecen en la página) por lo que presenta una puntuación más baja en especificidad.
- Un selector de clase es más específico (selecciona solo los elementos de una página que tienen un valor de atributo `class` dado), y por tanto recibe una puntuación mayor.

Veamos un ejemplo. Aquí abajo encontrarás dos reglas que pueden aplicarse al elemento `h1`. Este elemento `h1` termina siendo de color rojo: el selector de clase confiere a esta regla una mayor especificidad, así que se aplicará a pesar de la regla para el selector de elemento que aparece más abajo en el orden del código.

This is my heading.

```
.main-heading {  
  color: red;  
}  
  
h1 {  
  color: blue;  
}
```

```
<h1 class="main-heading">This is my heading.</h1>
```



Reset

Profundizaremos en la especificidad más adelante.

Herencia

La herencia también debe entenderse en este contexto: algunos valores de las propiedades CSS que se han establecido para los elementos padre los heredan los elementos hijo, pero otros no.

Por ejemplo, si para un elemento se establece el color (`color`) y el tipo de letra (`font-family`), cada elemento que se encuentre dentro de él también se mostrará de ese color y con ese tipo de letra, a menos que les se haya aplicado un color y un tipo de letra diferentes directamente.

As the body has been set to have a color of blue this is inherited through the descendants.

We can change the color by targetting the element with a selector, such as this `span`.

```
body {  
  color: blue;  
}  
  
span {  
  color: black;  
}
```

```
<p>As the body has been set to have a color of  
blue this is inherited through the descendants.  
</p>  
<p>We can change the color by targetting the  
element with a selector, such as this  
<span>span</span>.</p>
```

[Reset](#)

Algunas propiedades no se heredan. Por ejemplo, si para un elemento se establece un ancho [width](#) del 50%, sus descendientes no tendrán un 50% de ancho con respecto al de sus padres. Si este fuera el caso, ¡sería muy frustrante usar CSS!

Nota: En las páginas de referencia de las propiedades CSS de MDN encontrarás un cuadro con información técnica (por lo general, en la parte inferior de la sección de especificaciones) que enumera una serie de puntos sobre cada propiedad, incluyendo cuáles se heredan y cuáles no. Véase, por ejemplo, la [sección de especificaciones de la propiedad color](#).

Comprender cómo trabajan juntos estos conceptos

Estos tres conceptos controlan qué CSS se aplica a qué elemento. En las secciones siguientes veremos cómo funcionan en conjunto. A veces puede parecer un poco complicado, pero lo irás recordando a medida que ganes experiencia con el CSS, y siempre puedes consultar los detalles si se te olvidan. ¡Incluso los desarrolladores experimentados lo hacen!

Comprender la herencia

Vamos a empezar con la herencia. En el ejemplo siguiente tenemos un elemento [](#) con dos niveles de listas no ordenadas anidadas en él. Hemos establecido para el `` exterior un borde, un relleno y un color de fuente.

El color se ha aplicado a los hijos directos y también a los hijos indirectos: los elementos hijo `` inmediatos y los que están dentro de la primera lista. A continuación, hemos añadido a la segunda lista anidada una clase especial y le hemos aplicado un color diferente, que los elementos hijo de esta heredarán.

- Item One
- Item Two
 - 2.1
 - 2.2
- Item Three
 - **3.1**
 - **3.1.1**
 - **3.1.2**
 - **3.2**

```
.main {  
  color: rebeccapurple;  
  border: 2px solid #ccc;  
  padding: 1em;  
}
```

```
.special {  
  color: black;  
  font-weight: bold;  
}
```

```
<ul class="main">  
  <li>Item One</li>  
  <li>Item Two  
    <ul>  
      <li>2.1</li>  
      <li>2.2</li>  
    </ul>  
  </li>
```

La anchura (como se mencionó anteriormente), los márgenes, el relleno y los bordes no se heredan. Si los elementos hijo de nuestra lista heredaran los bordes, todas las listas y los elementos de lista ganarían un borde cada vez ¡y no es probable que vez quieras un efecto así!

Las propiedades que se heredan por defecto y las que no son cuestión, en gran medida, de sentido común.

Control de la herencia

CSS proporciona cuatro valores de propiedad universales especiales para el control de la herencia. Todas las propiedades CSS aceptan estos valores.

[inherit](#)

Establece que el valor de la propiedad que se aplica a un elemento determinado sea exactamente igual al del elemento padre. En la práctica, esto "activa la herencia".

[initial](#)

Establece que el valor de la propiedad que se aplica a un elemento seleccionado tenga el mismo valor que esté establecido para esa propiedad en la hoja de estilo por defecto del navegador.

[unset](#) [\(en-US\)](#)

Restablece la propiedad a su valor natural, lo que significa que si la propiedad se hereda de forma natural, actúa como `inherit`, y en caso contrario como `initial`.

Nota: También hay un valor más reciente, [revert](#) [\(en-US\)](#), que todavía admiten pocos navegadores.

Nota: Véase la sección [El origen de las declaraciones CSS](#) en el artículo [Introducción al concepto de cascada en CSS](#) para obtener más información sobre cada uno de estos valores y el modo en que funcionan.

A continuación veremos una lista de enlaces y exploraremos cómo funcionan los valores universales. El ejemplo en vivo de abajo te permite jugar con el CSS y ver lo que sucede cuando se hacen cambios. Jugar con el código es la mejor forma de enfrentarse al HTML y el CSS.

Por ejemplo:

1. Se ha aplicado la clase `my-class-1` al segundo elemento de lista. Esto establece por herencia el color del elemento `<a>` que está anidado en él. ¿Cómo cambia el color del enlace si quitamos esta regla?
2. ¿Entiendes por qué el tercer y el cuarto enlace se ven de este color? En caso contrario, comprueba la descripción de los valores anteriores.
3. ¿Cuál de los enlaces va a cambiar de color si se define un nuevo color para el elemento `<a>`, por ejemplo, a `{ color: red; }`?

- Default [link](#) color
- Inherit the [link](#) color
- Reset the [link](#) color
- Unset the [link](#) color

```
body {  
  color: green;  
}  
  
.my-class-1 a {  
  color: inherit;  
}  
  
.my-class-2 a {  
  color: initial;  
}  
  
.my-class-3 a {  
  color: unset;  
}
```

```
<ul>  
  <li>Default <a href="#">link</a> color</li>  
  <li class="my-class-1">Inherit the <a  
href="#">link</a> color</li>  
  <li class="my-class-2">Reset the <a  
href="#">link</a> color</li>  
  <li class="my-class-3">Unset the <a  
href="#">link</a> color</li>  
</ul>
```

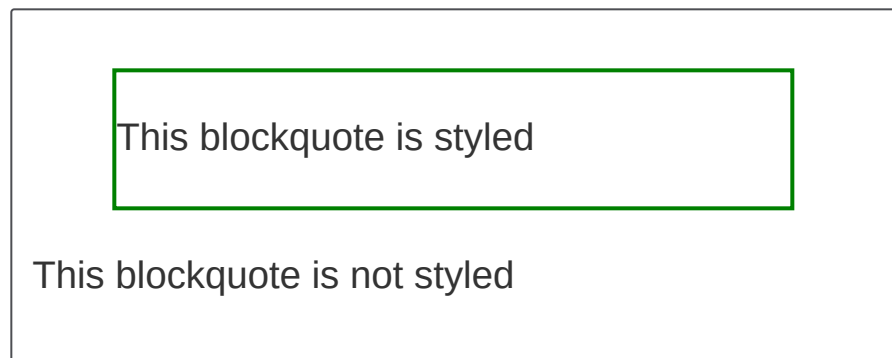
Reset

Restablecer todos los valores de propiedad

La propiedad CSS abreviada `all` se puede utilizar para aplicar uno de estos valores de herencia a (casi) todas las propiedades a la vez. Su valor puede ser cualquiera de los valores de herencia (`inherit`, `initial`, `unset`, o `revert`). Es una forma práctica de deshacer los cambios

realizados respecto al estilo para que puedas volver a un punto de partida conocido antes de empezar a introducir cambios.

En el ejemplo siguiente hay dos bloques de cita. El primero ya tiene un estilo aplicado al propio elemento de cita, mientras que el segundo tiene una clase aplicada al bloque de cita que establece el valor `all` en `unset`.



```
blockquote {  
  background-color: red;  
  border: 2px solid green;  
}  
  
.fix-this {  
  all: unset;  
}
```

```
<blockquote>  
  <p>This blockquote is styled</p>  
</blockquote>
```

```
<blockquote class="fix-this">  
  <p>This blockquote is not styled</p>  
</blockquote>
```

//

Reset

Prueba a establecer el valor de `all` al resto de valores disponibles y observa la diferencia.

Comprender la cascada

Ahora entendemos por qué un párrafo que está anidado en la estructura del HTML es del mismo color que el CSS aplicado al cuerpo (`body`) del HTML y, a partir de los artículos de introducción sabemos cómo cambiar el CSS aplicado a algo en cualquier parte del documento, ya sea mediante la asignación de CSS a un elemento o la creación de una clase. Ahora vamos a echar un vistazo a la forma en que el concepto de cascada define qué reglas CSS se aplican cuando más de un elemento de estilo puede aplicar estilo a un elemento.

Hay que considerar tres factores, que se enumeran a continuación en orden de importancia creciente. Los posteriores invalidan los anteriores:

1. **Orden en el código**
2. **Especificidad**
3. **Importancia**

Vamos a explicarlos para ver cómo los navegadores determinan exactamente que CSS deben aplicar.

Orden en el código

Ya hemos visto cómo el orden en el código es importante en el concepto de cascada. Si tienes más de una regla con exactamente el mismo peso, la que ocupa el último lugar en el CSS gana. Puedes entenderlo como que las reglas que están más cerca del elemento considerado sobreescriben las anteriores hasta que la última gana y da formato al elemento.

Especificidad

Una vez entendido el hecho de que el orden de los elementos en el código es importante, te

encontrarás en alguna situación en la que sabes cuál es la última norma en la hoja de estilo, pero se aplica una regla anterior. Esto se debe a que la regla anterior tiene **una especificidad mayor**, es decir, es más específica y, por lo tanto, el navegador la escoge como la que debe dar forma al elemento.

Como hemos visto anteriormente en este mismo artículo, un selector de clase tiene más peso que un selector de elemento, por lo que las propiedades que se definen en la clase tienen prioridad sobre las que se aplican directamente en el elemento.

Un elemento que hay que tener en cuenta es que aunque pensamos en términos de selectores y reglas que se aplican a lo que estos seleccionan, no es toda la regla lo que se sobrescribe, sino solo las propiedades que entran en conflicto.

Este comportamiento ayuda a evitar repeticiones en el CSS. Una práctica común es definir estilos genéricos para los elementos básicos y luego, crear clases para los elementos que son diferentes. Por ejemplo, en la hoja de estilo que mostramos a continuación hemos definido estilos genéricos para los encabezados de nivel 2; posteriormente hemos creado algunas clases que solo cambian algunas de las propiedades y los valores. Los valores definidos inicialmente se aplican a todos los encabezados, y entonces los valores más específicos se aplican a los encabezados con las clases.

Heading with no class

Heading with class of small

Heading with class of
bright

```
h2 {  
  font-size: 2em;  
  color: #000;  
  font-family: Georgia, 'Times New Roman',  
  Times, serif;  
}  
  
.small {  
  font-size: 1em;  
}  
  
.bright {  
  color: rebeccapurple;  
}
```

```
<h2>Heading with no class</h2>  
<h2 class="small">Heading with class of
```

Ahora vamos a echar un vistazo a cómo el navegador calcula la especificidad. Ya sabemos que un selector de elemento tiene una especificidad baja y se puede sobrescribir con un elemento de clase. Esencialmente se otorga un valor de puntos a los diferentes tipos de selectores y la suma de estos establece la importancia de ese selector en particular, que a continuación puede evaluarse ante otras posibles coincidencias.

La cantidad de especificidad de un selector se mide usando cuatro valores diferentes (o componentes), que pueden describirse como millares, centenas, decenas y unidades (cuatro dígitos individuales dispuestos en cuatro columnas):

1. **Millares:** Se suma un punto en esta columna si la declaración está en un atributo de [style](#) o, como suelen denominarse, estilos en línea. Tales declaraciones no tienen selectores, por lo que su especificidad siempre es 1000.
2. **Centenas:** Se suma un punto en esta columna por cada selector con ID particular que esté contenido en el selector general.
3. **Decenas:** Se suma un punto en esta columna por cada selector de clase, de atributo o pseudoclase que estén contenidos en el selector general.
4. **Unidades:** Se suma un punto en esta columna por cada selector o pseudoelemento que esté contenido en el selector general.

Nota: El selector universal (*), los operadores de combinación (+ , > , ~ , ' ') y la pseudo-clase de negación (: not) no tienen ningún efecto sobre la especificidad.

La tabla siguiente muestra algunos ejemplos concretos para ayudarte a entenderlo mejor. Analízalos y trata de entender por qué tienen la especificidad que les hemos dado. Aun no hemos explicado los selectores de forma detallada, pero puedes encontrar detalles de cada selector en los [selectores de referencia](#) de MDN.

Selector	Millares:	Centenas:	Decenas:	Unidades:	Especificidad total
h1	0	0	0	1	0001

Selector	0 Millares:	0 Centenas:	0 Decenas:	3 Unidades:	0003 Especificidad total
h1 + p::first-letter					
li > a[href*="en-US"] > .inline-warning	0	0	2	2	0022
#identifier	0	1	0	0	0100
Sin selector, con una regla en el atributo de un elemento style	1	0	0	0	1000

Antes de continuar, vamos a ver un ejemplo:

One

Two

```
/* specificity: 0101 */
#outer a {
  background-color: red;
}

/* specificity: 0201 */
#outer #inner a {
  background-color: blue;
}

/* specificity: 0104 */
#outer div ul li a {
  color: yellow;
}

/* specificity: 0113 */
#outer div ul li a {
```

```
<div id="outer" class="container">
  <div id="inner" class="container">
    <ul>
      <li class="nav"><a href="#">One</a>
    </li>
      <li class="nav"><a href="#">Two</a>
    </li>
    </ul>
  </div>
```

Reset

¿Qué pasa aquí? En primer lugar, estamos interesados solo en las primeras siete reglas de este ejemplo y, como te habrás dado cuenta, hemos incluido sus valores de especificidad en un comentario antes de cada una.

- Los dos primeros selectores compiten sobre el estilo del color del fondo del vínculo (el segundo gana y por eso el color de fondo es azul, porque en la cadena hay un selector con ID particular extra: la especificidad es de 201 contra 101).
- El tercer y el cuarto selector compiten sobre el estilo del color del texto del enlace (el segundo gana y hace que el texto sea blanco porque, aunque tiene un selector de elemento de menos, el selector que falta se sustituye por un selector de clase, con un valor de decena en vez de un valor de unidad). Así que la especificidad es de 113 contra 104.
- Los selectores 5-7 compiten por el estilo del borde del vínculo cuando el cursor se desplaza sobre estos. El sexto selector pierde claramente ante el quinto con una especificidad de 23 contra 24. En la cadena hay un selector de elemento de menos. El séptimo selector, sin embargo, los supera a ambos: en la cadena hay el mismo número de estos subselectores que en el quinto, pero se ha intercambiado un elemento por un selector de clase. Así que la especificidad es de 33 contra 23 y 24.

Nota: Esto solo es un ejemplo aproximado para facilitar la comprensión. En realidad, cada tipo de selector tiene su nivel de especificidad propio, que no pueden sobrescribir los selectores con un nivel de especificidad menor. Por ejemplo, un *millar* de selectores de **clase** combinados no serían capaces de sobrescribir las reglas de *un* selector **ID**.

Una forma más precisa de evaluar la especificidad sería anotar los niveles de especificidad individualmente de mayor a menor. Solo cuando hay empate entre las puntuaciones de los selectores dentro de un nivel específico será necesario evaluar el nivel inferior siguiente; de lo contrario, puedes prescindir de los selectores de especificidad de los niveles inferiores, ya que nunca pueden sobrescribir los niveles de especificidad más altos.

Propiedad `!important`

Hay una pieza especial de CSS que se puede utilizar para anular todos los cálculos anteriores, sin embargo se debe tener mucho cuidado con su uso: `!important`. Se utiliza para convertir una propiedad y un valor particular en el elemento más específico, de modo que se invalidan las reglas normales de la cascada.

Echa un vistazo a este ejemplo en el que se muestran dos párrafos, uno de los cuales tiene un elemento ID.

This is a paragraph.

One selector to rule them all!

```
#winning {  
  background-color: red;  
  border: 1px solid black;  
}  
  
.better {  
  background-color: gray;  
  border: none !important;  
}  
  
p {  
  background-color: blue;  
  color: white;  
  padding: 5px;  
}
```

```
<p class="better">This is a paragraph.</p>  
<p class="better" id="winning">One selector to  
rule them all!</p>
```

Reset

Vamos a observarlo con detenimiento para ver qué sucede. Elimina algunas de las propiedades para ver lo que sucede si te cuesta entender lo que ocurre:

1. Verás que se han aplicado los valores de `color` y `padding` de la tercera regla pero no el de `background-color`. ¿Por qué? Deberían haberse aplicado para los tres porque las reglas que se encuentran más adelante en el orden en el código fuente prevalecen sobre las reglas anteriores.
2. Sin embargo, ganan las reglas que están antes porque los selectores de clase tienen mayor especificidad que selectores de elemento.
3. En ambos bloques de código hay una clase `class` con el valor `better`, pero en el segundo bloque de código hay un `id` con el valor `winning`. Puesto que los identificadores tienen una especificidad *incluso mayor* que las clases (solo puede haber un elemento con un determinado ID en cada página, mientras que puede haber muchos elementos de la misma clase: los selectores ID son *muy específicos* con lo que delimitan), el primer bloque de código tendría un fondo de color gris y ningún borde, según lo que especifica la clase, mientras que al segundo bloque de código se aplicarían tanto el color de fondo rojo como el borde negro de 1 píxel.
4. El segundo elemento, en cambio, se muestra con el fondo de color rojo pero sin borde. ¿Por qué? Porque la declaración `!important` que hay en la segunda regla, después de `border: none` significa que esta declaración tendrá más valor que la regla anterior, aunque el ID de esta tenga mayor especificidad.

Nota: La única manera de anular la declaración `!important` sería incluir otra declaración `!important` en una declaración con la *misma especificidad* que aparezca más adelante en el orden del código fuente, o con una especificidad superior.

Es útil saber que `!important` existe para que sepas qué es cuando te lo encuentres en el código de otras personas. **Sin embargo, te recomendamos encarecidamente que no lo utilices a menos que sea absolutamente necesario.** `!important` cambia el modo en que suele funcionar la cascada, por lo que puede dificultar mucho la depuración de problemas en el CSS, especialmente en una hoja de estilo grande.

Una situación en la que puede que tengas que utilizarlo es si trabajas en un CMS en el que no es posible editar las reglas básicas de CSS y realmente tienes que anular un estilo que no puedes

posible editar los módulos básicos de CSS y realmente tienes que anular un estilo que no puede anularse de ninguna otra forma. Aun así, te recomendamos encarecidamente que evites su uso.

El efecto de la ubicación del CSS

Por último, resulta útil señalar que la importancia de una declaración CSS depende de la hoja de estilo en que se especifica (es posible que los usuarios configuren hojas de estilo personalizadas para anular los estilos de los desarrolladores, por ejemplo, porque el usuario podría tener alguna

discapacidad visual, o bien porque desea configurar el tamaño de letra de todas las páginas web que visita para que sea el doble de grande y le proporcione una mayor facilidad de lectura).

Resumen

Las declaraciones contradictorias se aplicarán en el orden siguiente (recuerda que las últimas prevalecen sobre las anteriores):

1. Declaraciones en las hojas de estilo de agente de usuario (por ejemplo, estilos predeterminados del navegador, que se utilizan cuando no hay otro estilo).
2. Declaraciones normales en las hojas de estilo del usuario (estilos personalizados creados por un usuario).
3. Declaraciones normales en las hojas de estilo de autor (los estilos que creamos nosotros, los desarrolladores web).
4. Declaraciones `!important` en las hojas de estilo de autor
5. Declaraciones `!important` en las hojas de estilo del usuario

Para los desarrolladores tiene sentido que sus hojas de estilo anulen a las de usuario para mantener el diseño según lo previsto, pero, como hemos visto, a veces los usuarios tienen buenas razones para anular las directrices de los desarrolladores web. Esto puede lograrse con el uso de `!important` en sus reglas.

Pon a prueba tus habilidades

Hemos cubierto mucho terreno en este artículo. ¿Recuerdas la información más importante? Encontrarás más pruebas para verificar que retienes esa información en [Test your skills: the Cascade](#).

¿Que sigue?

Si has entendido la mayor parte de este artículo, ¡enhorabuena! Has comenzado a familiarizarte con la mecánica fundamental del CSS. En el artículo siguiente vamos a ver con detalle los selectores.

Si todavía no tienes una comprensión completa de los conceptos de cascada, especificidad y herencia, ¡no te preocupes! Es, sin duda, lo más complejo que hemos expuesto hasta ahora y es

algo que incluso los desarrolladores web profesionales encuentran difícil. Te aconsejamos que regreses a este artículo cuantas veces necesites a medida que avances con el curso.

Regresa a esta página si empiezas a toparte con problemas extraños o con que los estilos no se aplican de la forma que esperas. Podría ser un problema de especificidad.

En este módulo

1. [La cascada y la herencia](#)
2. [Selectores CSS](#)
 - [Selectores de tipo, de clase y de ID](#)
 - [Selectores de atributo](#)
 - [Las pseudo-clases y los pseudo-elementos](#)
 - [Operadores de combinación](#)
3. [El modelo de caja](#)
4. [Fondos y bordes](#)
5. [El uso de diferentes direcciones de texto](#)
6. [El desbordamiento de los contenidos](#)
7. [Los valores y las unidades](#)
8. [Elementos de dimensionado en CSS](#)
9. [Imágenes, media y elementos de formulario](#)
10. [Aplicar estilo a las tablas](#)
11. [Depurar el CSS](#)
12. [Organizar el CSS](#)

Last modified: 2 jun 2021, [by MDN contributors](#)