# Introduccion CSS

## [¿Qué es la minificación?](https://lenguajecss.com/css/introduccion/minificar-css/" \l "qué-es-la-minificación)

La **minificación** (*en inglés, Uglify o Minification*) es la acción de eliminar carácteres o comentarios de nuestro código, con el objetivo de reducir su tamaño total, y por lo tanto, descargarlos más rápido. En archivos CSS muy grandes esto suele influir de forma considerable, por lo que es una **buena práctica** utilizar herramientas de minificación y reducir el tamaño del archivo CSS condensando toda su información, eliminando espacios, retornos de carro, etc...

Haciendo esto, conseguiremos que el archivo ocupe menos, pero a cambio, perderemos legibilidad. Por esta razón, es habitual conservar los archivos CSS originales (*sin minificar*) para trabajar con ellos, y generar los archivos reducidos con herramientas automaticas. Hay que tener en cuenta que el proceso de minification es un paso totalmente opcional, por lo que no es obligado realizarlo, pero se considera una buena práctica de optimización.

Veamos un ejemplo de un mismo archivo CSS con su contenido sin minificar y minificado:

**Ejemplo de código CSS legible (index.css)**: 96 bytes



**Ejemplo de código CSS minificado (index.min.css)**: 75 bytes

Como se puede ver, el **tamaño** y la **legibilidad** del archivo CSS se reduce considerablemente. Sería perjudicial trabajar con código del segundo ejemplo, por lo que se suele mantener un archivo legible (*el primero*) para realizar modificaciones y trabajar con él, y luego, de forma opcional, un archivo con la información minificada (*el segundo*) que será el que se utilice finalmente en nuestro proyecto cuando esté terminado.

En resumen, el primer archivo es el **código para humanos**, el que debemos mantener y trabajar con él. El segundo archivo es el **código para el navegador**, que no se debe modificar directamente por humanos y que se debe generar a partir del primero.

### [Herramientas de minificación](https://lenguajecss.com/css/introduccion/minificar-css/" \l "herramientas-de-minificación)

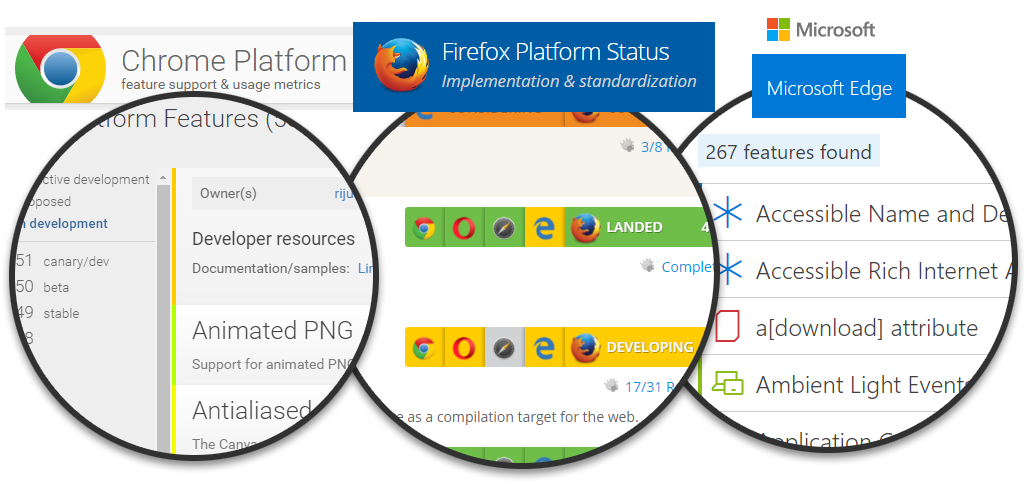
Existen múltiples herramientas para minificar código CSS. Algunas de ellas, incluso se encargan de analizar el código y, no sólo minificarlo, sino además suprimir propiedades repetidas, eliminar propiedades o valores inútiles, etc. Veamos algunas de las más populares:

|  |  |  |
| --- | --- | --- |
| Herramienta | Modalidad | Características |
| [CSS Nano](https://cssnano.co/) | PostCSS | Para automatizar desde terminal o desde PostCSS. |
| [Clean CSS](https://github.com/jakubpawlowicz/clean-css) | NodeJS/NPM | Para automatizar desde terminal. |
| [CSS Compressor](http://csscompressor.com/) | Online | Opciones variadas: grado de compresión, optimizaciones... |
| [CSSO](https://github.com/css/csso) | NodeJS/NPM | Optimizador de CSS (clean, compress and restructuring) |
| [Sqwish](https://github.com/ded/sqwish) | NodeJS/NPM | Compresor de CSS basado en Node |
| [YUI Compressor](https://yui.github.io/yuicompressor/) | Java | Compresor CSS histórico de Yahoo |

## [Ecosistema de navegadores](https://lenguajecss.com/css/introduccion/navegadores-web/" \l "ecosistema-de-navegadores)

En un mundo ideal, todas las páginas webs se verían correctamente y de la misma forma en todos los navegadores web disponibles, sin embargo, y una de las cosas que más llama la atención del diseño web cuando estamos empezando, es que no sólo debemos construir una web correctamente, sino que además **debemos ser conscientes de los navegadores más utilizados**, así como de sus carencias y virtudes.

En un principio, el consorcio **W3C** se encarga de definir unas especificaciones y «normas» de recomendación, para que posteriormente, las compañías desarrolladoras de navegadores web las sigan y puedan crear un navegador correctamente. Pero como no estamos en un mundo perfecto (*y el tiempo es un recurso limitado*), dichas compañías establecen prioridades, desarrollan características antes que otras, e incluso algunas características deciden no implementarlas por razones específicas o internas.



Las compañías más comprometidas con sus navegadores web, tienen a disposición de los diseñadores, programadores y entusiastas, una especie de diario cronológico, donde mencionan su hoja de ruta con las características que van implementando, descartando o sus planes de futuro, así como información adicional sobre el tema en cuestión:

|  |  |  |
| --- | --- | --- |
| Responsables | Producto | Página de estado de desarrollo del navegador |
| Microsoft | Navegador Microsoft Edge | [Edge Platform Status](https://developer.microsoft.com/en-us/microsoft-edge/status/) |
| (Proyecto open source) | Motor Webkit | [Webkit Feature Status](https://webkit.org/status/) |
| Fundación Mozilla | Navegador Mozilla Firefox | [Firefox Platform Status](https://platform-status.mozilla.org/) |
| Google | Navegador Google Chrome | [Chrome Platform Status](https://www.chromestatus.com/features) |

Además, existe una sección donde podemos [comparar características respecto a navegadores en CanIUse](https://caniuse.com/" \l "comparison).

### [Navegadores actuales](https://lenguajecss.com/css/introduccion/navegadores-web/" \l "navegadores-actuales)

A continuación, tenemos una lista de la rama de los **5 navegadores más populares**, que son aquellos que tienen una cuota de mercado considerable. Algunos de estos navegadores tienen varias versiones diferentes, como por ejemplo, versiones beta (*con funcionalidades aún no existentes en la versión oficial*) o versiones de desarrollador (*orientadas para el uso de programadores o diseñadores*).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Responsables | Navegador web | Propósito | Motor | Notas | URL |
| Google | Chrome | Uso habitual | Blink |  | [URL](https://www.google.com/chrome/browser/) |
| Google | Chrome beta | Early-adopters | Blink |  | [URL](https://www.google.com/chrome/browser/beta.html) |
| Google | Canary Chrome | Desarrollador | Blink |  | [URL](https://www.google.com/chrome/browser/canary.html) |
| Google | Chromium | Open Source | Blink |  | [URL](https://chromium.woolyss.com/) |
| Mozilla | Firefox | Uso habitual | Quantum |  | [URL](https://www.mozilla.org/es-ES/firefox/) |
| Mozilla | Firefox beta | Early-adopters | Quantum |  | [URL](https://www.mozilla.org/es-ES/firefox/channel/" \l "beta) |
| Mozilla | Firefox Dev Edition | Desarrollador | Quantum |  | [URL](https://www.mozilla.org/es-ES/firefox/channel/" \l "developer) |
| Mozilla | Firefox Nightly | Desarrollador | Quantum |  | [URL](https://www.mozilla.org/es-ES/firefox/channel/desktop/" \l "nightly) |
| Microsoft | Edge | Uso habitual | Blink |  | [URL](https://www.microsoft.com/en-us/windows/microsoft-edge) |
| Microsoft | Internet Explorer | Uso habitual | Trident | Sólo Windows | [URL](https://www.microsoft.com/en-us/download/internet-explorer.aspx) |
| Opera | Opera | Uso habitual | Blink |  | [URL](https://www.opera.com/es/computer) |
| Opera | Opera beta | Early-adopters | Blink |  | [URL](https://www.opera.com/es/computer/beta) |
| Opera | Opera developer | Desarrollador | Blink |  | [URL](https://www.opera.com/es/developer) |
| Opera | Opera Neon | Early-adopters | Blink |  | [URL](https://www.opera.com/es/computer/neon) |
| Apple | Safari | Uso habitual | Webkit | Sólo Apple | [URL](https://www.apple.com/safari/) |
| Apple | Safari Tech Preview | Desarrollador | Webkit | Sólo Apple | [URL](https://developer.apple.com/safari/technology-preview/) |

### [Otros navegadores](https://lenguajecss.com/css/introduccion/navegadores-web/" \l "otros-navegadores)

A continuación, tenemos una lista de otros navegadores menores, que no superan una cuota de mercado a nivel global de un 1%, pero que pueden ser interesantes en el futuro, para casos particulares o podrían experimentar un aumento de su cuota en los próximos años:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Responsables | Navegador web | Propósito | Motor | Basado en | URL |
| Tor Project | Tor | Navegación anónima | Gecko | Firefox | [URL](https://www.torproject.org/) |
| Vivaldi Tech | Vivaldi | Early-adopters | Blink |  | [URL](https://vivaldi.com/) |
| Brave Soft | Brave | Privacidad | Blink |  | [URL](https://www.brave.com/) |
| Yandex | Yandex Browser | Uso habitual | Blink |  | [URL](https://browser.yandex.com/) |
| David Rosca | QupZilla | Uso habitual | Qt WebEngine |  | [URL](http://www.qupzilla.com/) |
| Maxthon Int | Maxthon | Uso habitual | Trident/Webkit |  | [URL](http://es.maxthon.com/) |
| Fenrir Inc | Sleipnir | Uso habitual | Blink |  | [URL](http://www.fenrir-inc.com/) |
| SM Project | SeaMonkey | Uso habitual | Gecko | Mozilla AS | [URL](https://www.seamonkey-project.org/) |
| Chris Dywan | Midori | Uso habitual | Webkit |  | [URL](http://www.midori-browser.org/) |
| Comodo Group | Comodo Dragon | Uso habitual | Blink | Chromium | [URL](https://browser.comodo.com/) |
| Avant Force | Avant | Uso habitual | Trident/Gecko |  | [URL](http://avantbrowser.com/) |
| M. Patocka | Links | Navegador de texto | - |  | [URL](http://links.twibright.com/) |
| Thomas Dickey | Lynx | Navegador de texto | - |  | [URL](http://lynx.invisible-island.net/) |
| Akinori Ito | w3m | Navegador de texto | - |  | [URL](http://w3m.sourceforge.net/) |
| P. Baudis | Elinks | Navegador de texto | - |  | [URL](http://www.elinks.cz/) |

Existen muchos más navegadores, esto sólo es una lista de los que he considerado más relevantes.

## [Niveles CSS](https://lenguajecss.com/css/introduccion/niveles-y-prefijos-css/" \l "niveles-css)

A lo largo de su historia, **CSS** ha ido evolucionado en diferentes versiones denominados **niveles**:

|  |  |  |
| --- | --- | --- |
| Nivel | Año | Descripción |
| **CSS1** | 1996 | Propiedades de fuente, colores, alineación, etc... |
| **CSS2** | 1998 | Propiedades de posicionamiento, tipos de medios, etc... |
| **CSS2.1** | 2005 | Corrige errores de CSS2 y modifica ciertas propiedades |
| **CSS3** | 2011 | Inicio de características de CSS como módulos separados |

Desde 2011, la **especificación CSS** comienza a evolucionar separando sus nuevas funcionalidades en pequeños **módulos**, favoreciendo su implementación en navegadores. Si se desea información más técnica, se puede consultar la evolución de los diferentes [módulos relacionados con CSS](https://www.w3.org/standards/techs/css) en la página web del consorcio W3C.

### [Prefijos](https://lenguajecss.com/css/introduccion/niveles-y-prefijos-css/" \l "prefijos)

Algunas de las propiedades que veremos no están definidas por completo, sólo son borradores o pueden variar en la especificación definitiva, por lo que los navegadores las implementan utilizando una serie de **vendor prefixes** (prefijos por navegador), que facilitan la segmentación de funcionalidades.

De esta forma, podemos utilizar varios prefijos para asegurarnos que aunque dichas funcionalidades tengan un comportamiento o sintaxis diferente en cada navegador, podemos hacer referencia a cada una de ellas por separado:



## Herencia CSS

### [Concepto de herencia](https://lenguajecss.com/css/introduccion/herencia-css/" \l "concepto-de-herencia)

En primer lugar, debemos saber que algunas propiedades CSS se **heredan** desde los elementos padres a los elementos hijos, modificando el valor que tienen por defecto:



En el ejemplo anterior, aplicamos a la etiqueta HTML <body> el **color de texto verde**. En principio, esta propiedad aplicará dicho color a los textos que estén dentro de dicha etiqueta <body>.

Sin embargo, si tenemos más etiquetas dentro, como por ejemplo una etiqueta <div> con texto en su interior, si no tenemos aplicada una propiedad color a dicho elemento, veremos que también aparece en color verde. Esto ocurre porque la propiedad **color** es una de las propiedades CSS que, en el caso de no tener valor específico, **hereda el valor de su elemento padre**.

Ojo, porque esto no ocurre si lo hacemos con otras propiedades CSS, como por ejemplo, con **los bordes** de un elemento HTML:



Si esta propiedad aplicara herencia, todos los elementos HTML situados en el interior de <body> tendrían un borde rojo, comportamiento que no suele ser el deseado. Por esa razón, la herencia no ocurre con todas las propiedades CSS, sino sólo con algunas propiedades como **color** o **font**, donde si suele ser deseable.

### [Valores especiales](https://lenguajecss.com/css/introduccion/herencia-css/" \l "valores-especiales)

Además de los valores habituales de cada propiedad CSS, también podemos aplicar ciertos **valores especiales** que son **comunes** a todas las propiedades existentes. Con estos valores modificamos el comportamiento de la herencia en dicha propiedad:

|  |  |
| --- | --- |
| Valor | Significado |
| **inherit** | Hereda el valor de la propiedad del elemento padre. |
| **initial** | Establece el valor que tenía la propiedad inicialmente. |
| **unset** | Combinación de las dos anteriores: Hereda el valor de la propiedad del elemento padre, y en caso de no existir, su valor inicial. |

Veamos, por ejemplo, el siguiente ejemplo para forzar la herencia en una propiedad que no la aplica por defecto:



Si tenemos un elemento <div> dentro del <body>, el primero heredará los estilos del elemento <body>, ya que le hemos especificado el valor **inherit** en la propiedad **border**.

## [Cascada CSS](https://lenguajecss.com/css/introduccion/cascada-css/" \l "cascada-css)

Para saber que bloque de estilos tiene prioridad, CSS analiza (por orden) tres conceptos clave del código CSS: su **importancia**, la **especificidad** y su **orden**. Veamos en que se basa cada uno de ellos.

### [Importancia](https://lenguajecss.com/css/introduccion/cascada-css/" \l "importancia)

La **importancia** de un código CSS se determina dependiendo de las hojas de estilo donde está colocado. Generalmente, no necesitaremos preocuparnos de este factor, pero siempre es una buena idea conocer cómo funciona. Existen varios tipos de hojas de estilo, de menor a mayor importancia:

|  |  |  |
| --- | --- | --- |
| Tipo de CSS | Descripción | Definido por |
| Agente de usuario | Son los estilos CSS que aplica el navegador por defecto. | Navegador |
| CSS de usuario | Son los estilos CSS que añade el usuario, por razones específicas. | Usuario |
| CSS de autor | Son los estilos CSS que coloca el autor de la página. | Desarrollador |

1. **Los estilos por defecto que tiene definido el agente de usuario, es decir, el navegador web del usuario.** No todos los navegadores utilizan el mismo estilo por defecto (Mozilla Firefox, Internet Explorer, [Safari](https://trac.webkit.org/browser/trunk/Source/WebCore/css/html.css" \t "_blank)). Para evitar este problema, existen hojas de estilo como [css reset](https://cssreset.com/" \t "_blank) que inicializan o resetean los estilos para que una página web se visualice de la misma forma en casi todos los navegadores.
2. **Los estilos especificados por el usuario que está leyendo el documento.** Son estilos que define cada usuario y que se pueden emplear para sustituir los estilos que tiene definidos una página web. Esta característica se puede utilizar para mejorar la accesibilidad de una página web.
3. **Los estilos definidos por el autor de la página web.** Estos estilos se pueden definir en tres puntos.
   * En un fichero externo enlazado con la etiqueta link.
   * Al principio de la propia página con al etiqueta style.
   * En un elemento concreto con el atributo style.

Aunque no es recomendable utilizarlo frecuentemente (puede convertirse en una mala costumbre), se puede añadir al final de cada regla el texto **!important**, consiguiendo que la regla en cuestión tenga prioridad sobre las demás, independientemente del nivel o la altura a la que estén:



**Nota**: En el caso de que una misma propiedad del CSS de usuario y una propiedad del CSS de autor tuvieran **!important**, como caso excepcional tendría prioridad la del CSS de usuario sobre la del CSS de autor.

### [Especificidad](https://lenguajecss.com/css/introduccion/cascada-css/" \l "especificidad)

En segundo caso, y si la importancia no elimina la ambigüedad, se pasa a determinar la especificidad de los selectores CSS. Para ello, se sigue un cálculo matemático basado en 4 componentes: **A**, **B**, **C**, **D**.

|  |  |
| --- | --- |
| Componente | Descripción |
| Componente A | Número de estilos aplicados mediante un atributo style . |
| Componente B | Número de veces que aparece un id en el selector. |
| Componente C | Número de veces que aparece una clase , pseudoclase o atributo en el selector. |
| Componente D | Número de veces que aparece un elemento o un pseudoelementos en el selector. |

Para saber si un bloque de CSS es más específico que otro (*y por lo tanto, tiene prioridad*) sólo hay que calcular sus componentes. Se ordenan teniendo en cuenta los valores de cada componente, de izquierda a derecha.

Veamos algunos ejemplos, ordenados de **menor a mayor especificidad**:



En **keegan.st** tienes una excelente [calculadora de especificidad CSS](https://specificity.keegan.st/) donde podrás calcular la especificidad de un selector CSS rápida y cómodamente.

### [Orden](https://lenguajecss.com/css/introduccion/cascada-css/" \l "orden)

En CSS, es posible crear múltiples reglas CSS para definir un mismo concepto. En este caso, la que prevalece ante todas las demás depende de ciertos factores, como es la *«altura»* a la que está colocada la regla:

* El **CSS embebido** , ejemplo <p style="color:blue">embebido</p>, en un elemento HTML es el que tiene mayor precedencia, por lo que siempre será el que tenga prioridad sobre otras reglas CSS.
* En segundo lugar, el **CSS interno** definido a través de bloques style en el propio documento HTML será el siguiente a tener en cuenta en orden de prioridad.

 <style>

    selector{propiedad:valor;}

 </style>

* Por último, los documentos **CSS externos** son la tercera opción de prioridad a la hora de tomar en cuenta las reglas CSS.

 <link rel="stylesheet" href="estilos.css">

Teniendo esto en cuenta, hay que recordar que las propiedades que prevalecerán serán las que estén en último lugar, siempre respetando la prioridad de la lista anterior.

## Variables [CSS o CSS Custom Properties](https://lenguajecss.com/css/introduccion/css-custom-properties/" \l "css-custom-properties)

Las CSS Custom Properties (muchas veces conocidas por variables CSS) son un mecanismo de CSS que permite dar un valor personalizado a las propiedades, si necesitamos cambiar el valor en algún momento, podemos hacerlo en esa **propiedad personalizada** y no en múltiples partes del documento, donde nos podríamos equivocar al añadir el mismo valor o incluso hacer mucho más difícil de mantener el código.

### [Definir una variable CSS](https://lenguajecss.com/css/introduccion/css-custom-properties/" \l "definir-una-variable-css)

Para definir una **custom property** haremos uso de los dos guiones -- previos al nombre que queramos utilizar. Además, debemos fijarnos en el elemento que definimos la variable, en este ejemplo la pseudoclase :root:



Hay varios detalles que comentar sobre este fragmento de código:

En primer lugar, la pseudoclase :root hace referencia al elemento raíz del documento, o lo que es lo mismo, al elemento <html>. La diferencia de utilizar html o :root como selector es que este último tiene algo más de [especificidad CSS](https://lenguajecss.com/css/introduccion/cascada-css/" \l "especificidad). Mientras que html tiene **001**, :root tendría **010**.

Al colocarla en :root estamos definiendo que la **custom property** estará definida para el **ámbito** de esa etiqueta <html> (*o cualquier elemento hijo*), es decir, a todo el documento. Sin embargo, ya veremos que podemos aplicar estas variables sólo a partes concretas del DOM de nuestra página.

### [Utilizar una variable CSS](https://lenguajecss.com/css/introduccion/css-custom-properties/" \l "utilizar-una-variable-css)

A la hora de utilizar una **custom property**, hay que utilizarla dentro de la expresión var():



Además, es muy recomendable que la expresión var() tenga **dos parámetros**. El primero de ellos, la **custom property** en cuestión, el segundo de ellos, el valor por defecto en el caso de que esa propiedad no esté definida en el ámbito actual:



### [Ámbito de las custom properties](https://lenguajecss.com/css/introduccion/css-custom-properties/" \l "ámbito-de-las-custom-properties)

El ejemplo anterior es muy básico y puede que no se aprecie el detalle de los **ámbitos** con las **custom properties**, así que vamos a verlo con un ejemplo más específico.

Observa el marcado HTML del siguiente ejemplo, donde tenemos tres elementos con clase *child*, los dos primeros dentro de parent y el tercero fuera:





Sin embargo, ahora estamos definiendo la variable --background-color en diferentes ámbitos:

* Los dos primeros elementos .child tomarán color negro, ya que se le aplica a .parent (*e hijos*).
* El primer elemento .child se sobreescribe con color púrpura, ya que se le aplica a .first.
* El tercer elemento no tendrá ninguna variable definida, por lo que tomará color azul.

Esto nos permite mucha flexibilidad y potencia a la hora de utilizar **custom properties** en CSS, ya que son tremendamente útiles y versátiles, permitiéndonos utilizar incluso la cascada de CSS a nuestro favor.

Las **custom properties** tienen un buen soporte en navegadores en la actualidad, a excepción, como siempre, de Internet Explorer. Si necesitaras soporte para IE11, quizás podría venirte bien el polifyll [ie11CustomProperties](https://github.com/nuxodin/ie11CustomProperties).

### [Diferencia con Sass/LESS](https://lenguajecss.com/css/introduccion/css-custom-properties/" \l "diferencia-con-sassless)

También es importante mencionar que las **variables CSS** no funcionan exactamente igual que las **variables Sass** o las **variables LESS** (*en general, las variables de preprocesadores*).

***Sass*\*** y **LESS**, al igual que muchos otros preprocesadores de CSS, no trabajan directamente en el navegador, sino en una capa previa (*capa de preprocesamiento*). Por lo tanto, muchas tareas se realizan antes de llegar al navegador.

Es el caso de las **variables Sass/LESS**, por ejemplo. Dichas variables son leídas por **Sass** o **LESS** y convertidas a CSS «plano» (*sin variables*):



Es por eso que no deben tomarse las **custom properties** como equivalente de las **variables SCSS** o **variables LESS**, ya que tienen matices diferentes.

### [Variables CSS desde Javascript](https://lenguajecss.com/css/introduccion/css-custom-properties/" \l "variables-css-desde-javascript)

Existen varias formas de manipular estilos CSS desde Javascript, ya sea directamente a través de la propiedad **.style** del elemento en cuestión, o utilizando la función global de cada página **.getComputedStyle()**, que devuelve los estilos computados por el navegador.

Sin embargo, en esta ocasión nos centraremos en una serie de métodos de ayuda que nos hacen la vida más fácil, ya que podemos establecer, obtener y/o eliminar **propiedades CSS** (*custom properties incluídas*) de un elemento de forma muy sencilla.

Observen el siguiente método mediante el cuál añadimos (*o modificamos*) la propiedad border de un elemento del DOM de nuestro HTML:



Ejemplo de modificación de variable en :root desde javascript



Al igual que lo hacemos con una propiedad CSS, lo podríamos hacer con una **custom property**, la cuál quedaría añadida en el atributo HTML style del elemento.

Los métodos de ayuda que tenemos para modificar propiedades son los siguientes:

|  |  |
| --- | --- |
| Método | Descripción |
| .setProperty(name,value,priority) | Añade/cambia el **valor** de una propiedad CSS. |
| .getPropertyValue(name) | Obtiene el valor de una propiedad CSS. |
| .getPropertyPriority(name) | Devuelve important si tiene prioridad. |
| .removeProperty(name) | Elimina una propiedad CSS de un elemento. |

El método que hemos visto anteriormente es .style.setProperty(), con el cuál podemos añadir (*o modificar*) un valor a una propiedad. Además, si establecemos el tercer parámetro con el important, sería equivalente a añadirle el !important al valor de la propiedad.

Por otro lado, tenemos el método opuesto .style.getPropertyValue() mediante el cuál podemos obtener el valor de una propiedad concreta o el método .style.getPropertyPriority(), con el cuál te devuelve el string important si lo tiene definido. Por último, el método .style.removeProperty() como su propio nombre indica, elimina una propiedad de un elemento.

**Nota**: Ten en cuenta que estas propiedades solo funcionan si los estilos CSS o custom properties han sido definidas a través del atributo style de CSS o con estos mismos métodos (que lo hacen por la misma vía).

# Modelo de cajas

## Unidades css

Antes de comenzar a utilizar propiedades que utilicen medidas, como por ejemplo width (*propiedad que sirve para establecer un ancho a un elemento concreto*), es conveniente conocer los tipos de unidades que pueden utilizarse en CSS para indicar un determinado tamaño . Existen varios tipos de unidades, vamos a desglosarlas en grupos y explicar cada una de ellas. Más adelante veremos las propiedades de CSS que pueden utilizarlas.

### [Unidades absolutas](https://lenguajecss.com/css/modelo-de-cajas/unidades-css/" \l "unidades-absolutas)

Las **unidades absolutas** son un tipo de medida fija que no cambia, que no depende de ningún otro factor. Son ideales en contextos donde las medidas no varían como pueden ser en medios impresos (documentos, impresiones, etc...), pero son unidades poco flexibles y adecuadas para la web actual, ya que no tienen la capacidad de adaptarse a diferentes resoluciones o pantallas, que es lo que tendemos a hacer hoy en día.

Sin embargo, el uso de la unidad px es muy recomendable para el desarrollador (*al menos en sus primeros pasos en el diseño web*) ya que se trata de una unidad fácil de comprender, muy conocida y que nos permitirá afianzar conceptos a la vez que profundizamos en el diseño web.

Las diferentes **unidades absolutas** que pueden utilizarse en CSS son las siguientes (*de mayor a menor tamaño*):

|  |  |  |
| --- | --- | --- |
| Unidad | Significado | Medida aproximada |
| in | Pulgadas | 1in = 25.4mm |
| cm | Centímetros | 1cm = 10mm |
| pc | Picas | 1pc = 4.23mm |
| mm | Milímetros | 1mm = 1mm |
| pt | Puntos | 1pt = 0.35mm |
| px | Píxels | 1px = 0.26mm |
| Q | Cuarto de mm | 1Q = 0.248mm |

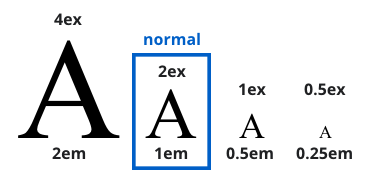
**Consejo**: El punto (**pt**) es una medida que puede utilizarse para documentos CSS en los que se fija el tamaño de las fuentes en medios impresos.

### [Unidades relativas](https://lenguajecss.com/css/modelo-de-cajas/unidades-css/" \l "unidades-relativas)

Las **unidades relativas** son un tipo de medida más potente en CSS. Al contrario que las unidades absolutas, las unidades relativas dependen de algún otro factor (*resolución, densidad de pantalla, etc...*). Tienen una curva de aprendizaje más compleja, pero son las ideales para trabajar en dispositivos con diferentes tamaños, ya que son muy flexibles y versátiles:

|  |  |  |
| --- | --- | --- |
| Unidad | Significado | Medida aproximada |
| em | «M» | 1em = tamaño de fuente establecida en navegador |
| ex | «X» (~0.5em) | 1ex = ~ mitad del tamaño de fuente del navegador |
| ch | «zero width» | 1ch = tamaño de ancho del cero ( 0 ) |
| rem | «root M» | 1rem = tamaño fuente raíz |
| % | Porcentaje | Relativa a herencia (contenedor padre) |

La unidad **em** se utiliza para hacer referencia al tamaño actual de la fuente que ha sido establecido en el navegador, que habitualmente es un valor aproximado de 16px. De esta forma, una cantidad de **1em** sería este tamaño establecido por el usuario, mientras que una cantidad de 2em sería justo el doble y una cantidad de 0.5em sería justo la mitad. Por otro lado, con **1ex** establecemos la mitad del tamaño de la fuente, ya que 1ex = 0.5em.



Realmente, la medida ex está basada en la **altura de la x minúscula**, que es aproximadamente un poco más de la mitad de la fuente actual (depende de la tipografía utilizada), o **ch** , que equivale al tamaño de ancho del 0 de la fuente actual, aunque en la práctica es un tipo de unidad que no suele ser utilizada demasiado.



Una unidad muy interesante y práctica para tipografías es la unidad **rem** (*root em*). Esta unidad es muy cómoda, ya que permite establecer un tamaño para el documento en general (*utilizando el elemento* ***body*** *o la pseudoclase* ***:root***):



Posteriormente, podemos ir utilizando la unidad **rem** en ciertas partes del documento. Con esto, estamos indicando el factor de escala (*respecto al tamaño general que indicamos en el body*). En el ejemplo anterior, los elementos **<h1>** tendrán **44 píxels** de tamaño, ya que hemos establecido **2rem**, que significa «el doble que el tamaño general». Por otro lado, los elementos **<h2>** tendrían el mismo tamaño: **22 píxels**.

Esto nos da una ventaja principal considerable: Si queremos cambiar el tamaño del texto en general, sólo tenemos que cambiar el **font-size** del elemento **body**, puesto que el resto de unidades son factores de escalado y se modificarán todas en consecuencia al cambio del body. Algo, sin duda, muy práctico y fácil de modificar.

### [Unidades flexibles (viewport)](https://lenguajecss.com/css/modelo-de-cajas/unidades-css/" \l "unidades-flexibles-viewport)

Existen unas unidades de "nueva generación" que resultan muy útiles, porque dependen del **viewport** (*región visible de la página web en el navegador*). Con estas unidades podemos hacer referencia a un porcentaje concreto del tamaño específico que tengamos en la ventana del navegador, independientemente de si es redimensionado o no. Las unidades son las siguientes:

|  |  |  |
| --- | --- | --- |
| Unidad | Significado | Medida aproximada |
| vw | viewport width | 1vw = 1% ancho de navegador |
| vh | viewport height | 1vh = 1% alto de navegador |
| vmin | viewport minimum | 1vmin = 1% de alto o ancho (el mínimo) |
| vmax | viewport maximum | 1vmax = 1% de alto o ancho (el máximo) |

La unidad **vw** hace referencia al ancho del viewport, mientras que **vh** hace referencia al alto. Por ejemplo, si utilizamos 100vw estaremos haciendo referencia al 100% del ancho del navegador, o sea, todo lo que se está viendo de ancho en pantalla, mientras que si indicamos 50vw estaremos haciendo referencia a la mitad del ancho del navegador.

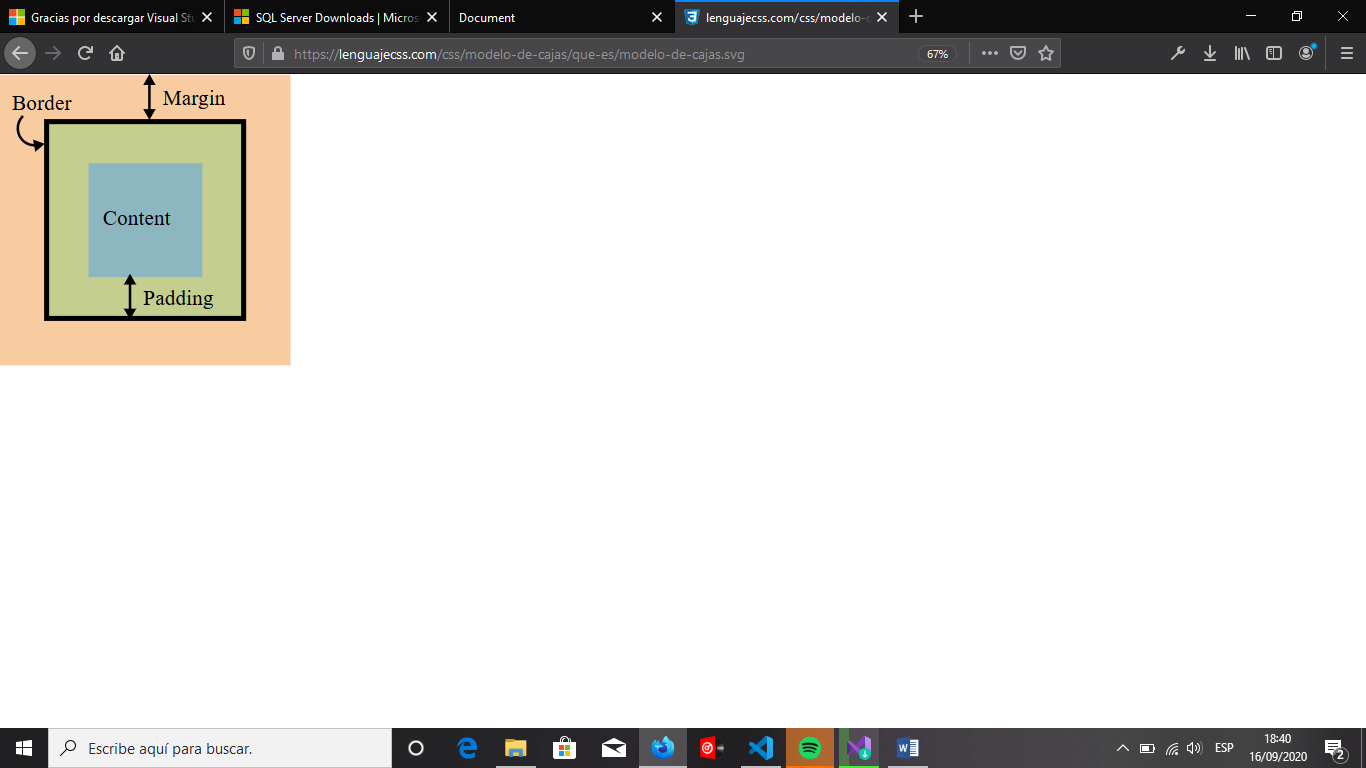
Por último tenemos **vmin** y **vmax** , que simplemente se utilizan para utilizar el porcentaje de ancho o alto del **viewport**, dependiendo cual sea más pequeño o más grande de los dos, lo que puede ser útil en algunas situaciones donde quieres flexibilidad con diseños adaptables.

Es interesante tener en cuenta que existen una serie de funciones para hacer cálculos con unidades CSS. Son las funciones calc(), min(), max() o clamp(), entre otras. Las veremos más adelante, en el capítulo de [funciones CSS](https://lenguajecss.com/css/modelo-de-cajas/funciones-css).

### Modelo de cajas

La representación básica del **modelo de cajas** es la siguiente, donde podemos observar varios conceptos importantes a diferenciar:

* El **borde** (*border*). En negro, es el límite que separa el interior del exterior del elemento.
* El **márgen** (*margin*). En naranja, es la parte exterior del elemento, por fuera del borde.
* El **relleno** (*padding*). En verde, es la parte interior del elemento, entre el contenido y el borde.
* El **contenido** (*en azul*). En azul, es la parte interior del elemento, excluyendo el relleno.

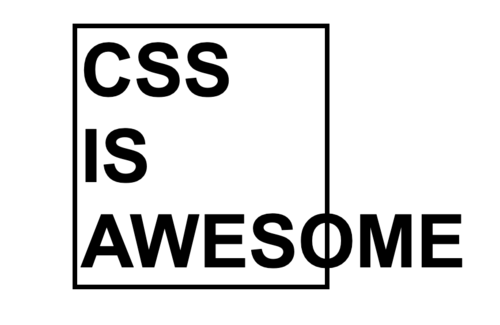


### [Dimensiones (ancho y alto)](https://lenguajecss.com/css/modelo-de-cajas/que-es/" \l "dimensiones-ancho-y-alto)

Para dar tamaños específicos a los diferentes elementos de un documento HTML, necesitaremos asignarles valores a las propiedades width (ancho) y height (alto).

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| width | **auto** | size | Tamaño de ancho de un elemento. |
| height | **auto** | size | Tamaño de alto de un elemento. |

En el caso de utilizar el valor **auto** en las propiedades anteriores (*que es lo mismo que no indicarlas, ya que es el valor que tienen por defecto*), el navegador se encarga de calcular el ancho o alto necesario, dependiendo del contenido del elemento, mientras que si indicamos un ancho y alto concretos, **estamos obligando a CSS tener un aspecto concreto** y podemos obtener resultados similares al siguiente (*conocida broma de CSS*) si su contenido es más grande que el tamaño que hemos definido:



Otra forma de lidiar con esto, es utilizar las propiedades hermanas de **width**: min-width y max-width y las propiedades hermanas de **height**: min-height y max-height. Con estas propiedades, en lugar de establecer un tamaño fijo, establecemos unos máximos y unos mínimos, donde el ancho o alto podría variar entre esos valores.



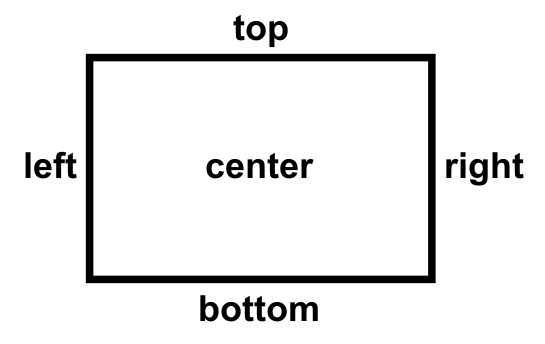
En este caso, por ejemplo, a pesar de estar indicando un tamaño de 800px, le aplicamos un max-width de 500px, por lo que estamos limitando el elemento a un tamaño de ancho de 500 píxeles como máximo y nunca superará ese tamaño.

Por un lado tenemos las propiedades de mínimos min-width y min-height, que por defecto tienen valor **0**, mientras que por otro lado, tenemos las propiedades de máximos max-width y max-height, que por defecto tienen valor **none**:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| max-width | **none** | size | Ancho máximo que puede ocupar un elemento. |
| min-width | **0** | size | Ancho mínimo que puede ocupar un elemento. |
| max-height | **none** | size | Alto máximo que puede ocupar un elemento. |
| min-height | **0** | size | Alto mínimo que puede ocupar un elemento. |

### [Zonas de un elemento](https://lenguajecss.com/css/modelo-de-cajas/que-es/" \l "zonas-de-un-elemento)

Antes de continuar, es importante saber que en CSS existen ciertas palabras clave para hacer referencia a una zona u orientación concreta sobre un elemento. Son conceptos muy sencillos y prácticamente lógicos, por lo que no tendrás ningún problema en comprenderlos. Son los siguientes:



### [Desbordamiento](https://lenguajecss.com/css/modelo-de-cajas/que-es/" \l "desbordamiento)

Volvamos a pensar en la situación de la imagen anterior: Damos un tamaño de ancho y alto a un elemento HTML, pero su contenido de texto es tan grande que no cabe dentro de ese elemento. ¿Qué ocurriría? Probablemente lo que vimos en la imagen: el contenido se desbordaría.

Podemos modificar ese comportamiento con la propiedad de CSS overflow, o con alguna de sus propiedades específicas overflow-x o overflow-y:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| overflow | **visible** | hidden | scroll | auto | Establece el comportamiento de desbordamiento. |
| overflow-x | **visible** | hidden | scroll | auto | Establece el desbordamiento sólo para el eje X (*horizontal*). |
| overflow-y | **visible** | hidden | scroll | auto | Establece el desbordamiento sólo para el eje Y (*vertical*). |

Dichas propiedades pueden tomar varios valores, donde **visible** es el valor que tiene por defecto, que permite que haya desbordamiento. Otras opciones son las siguientes, donde **no se permite desbordamiento**:

|  |  |  |
| --- | --- | --- |
| Valor | ¿Qué ocurre si se desborda el contenedor? | ¿Desbordamiento? |
| visible | Se muestra el contenido que sobresale (*comportamiento por defecto*) | Sí |
| hidden | Se oculta el contenido que sobresale. | No |
| scroll | Se colocan barras de desplazamiento (horizontales y verticales). | No |
| auto | Se colocan barras de desplazamiento (sólo las necesarias). | No |

**Nota**: CSS3 añade las propiedades overflow-x y overflow-y para cada eje individual, que antiguamente solo era posible hacerlo con overflow para ambos ejes. Estas propiedades son útiles cuando no quieres mostrar alguna barra de desplazamiento, habitualmente, la barra de desplazamiento horizontal.

## Márgenes y rellenos

### [Márgenes](https://lenguajecss.com/css/modelo-de-cajas/margenes-y-rellenos/" \l "márgenes)

Dichos márgenes se pueden considerar en conjunto (*de forma general*) o de forma concreta en cada una de las zonas del elemento. Veamos primero las propiedades específicas para cada zona:

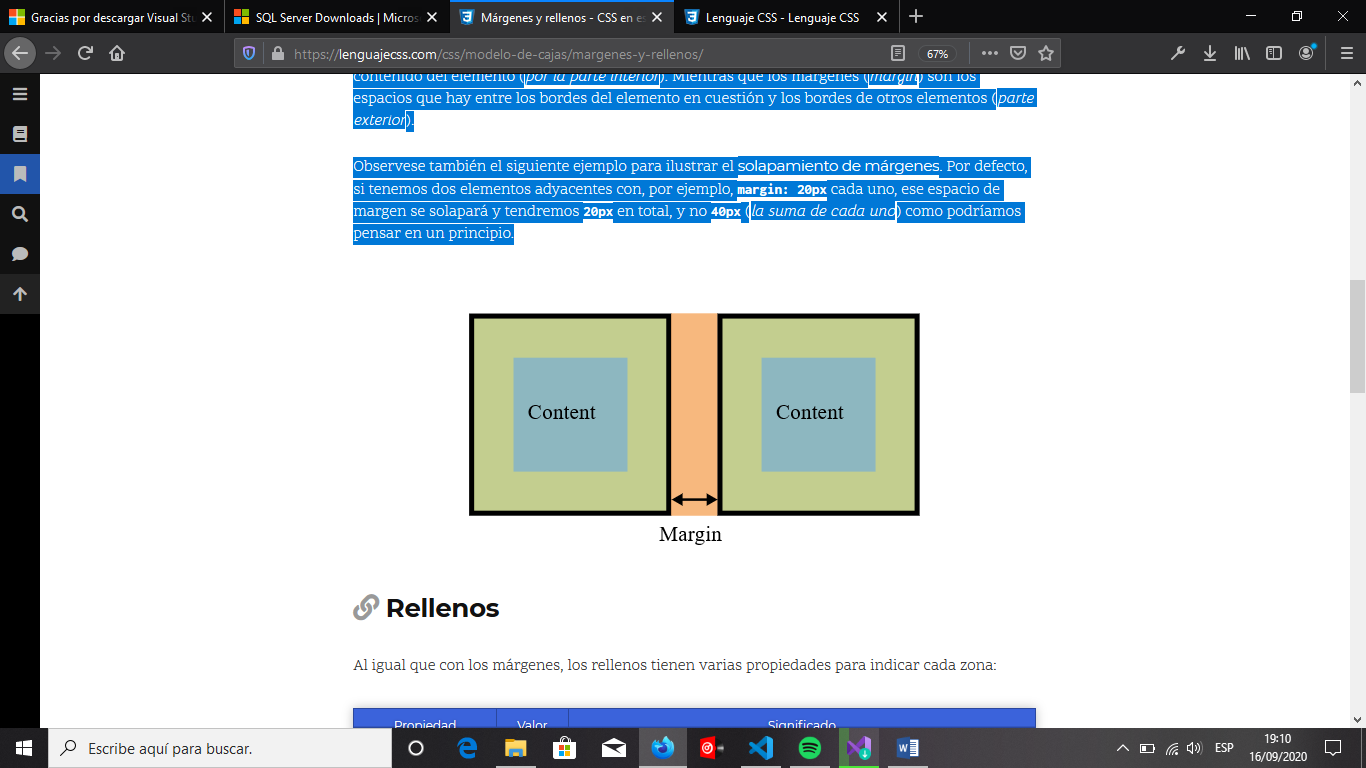
|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| margin-top | **auto** | size | Establece un tamaño de margen superior. |
| margin-left | **auto** | size | Establece un tamaño de margen a la izquierda. |
| margin-right | **auto** | size | Establece un tamaño de margen a la derecha. |
| margin-bottom | **auto** | size | Establece un tamaño de margen inferior. |

Podemos aplicar diferentes márgenes a cada zona de un elemento utilizando cada una de estas propiedades, o dejando al nevegador que lo haga de forma automática indicando el valor **auto**.

**Truco: Existe un truco muy sencillo y práctico para centrar un elemento en pantalla. Basta con aplicar un ancho fijo al contenedor, width:500px (por ejemplo) y luego aplicar un margin:auto. De esta forma, el navegador, al conocer el tamaño del elemento (y por omisión, el resto del tamaño de la ventana) se encarga de repartirlo equitativamente entre el margen izquierdo y el margen derecho, quedando centrado el elemento.**

Hay que recordar diferenciar bien los **márgenes** de los **rellenos**, puesto que no son la misma cosa. Los **rellenos** (*padding*) son los espacios que hay entre los bordes del elemento en cuestión y el contenido del elemento (*por la parte interior*). Mientras que los márgenes (*margin*) son los espacios que hay entre los bordes del elemento en cuestión y los bordes de otros elementos (*parte exterior*).

Observese también el siguiente ejemplo para ilustrar el **solapamiento de márgenes**. Por defecto, si tenemos dos elementos adyacentes con, por ejemplo, margin: 20px cada uno, ese espacio de margen se solapará y tendremos 20px en total, y no 40px (*la suma de cada uno*) como podríamos pensar en un principio.



### [Rellenos](https://lenguajecss.com/css/modelo-de-cajas/margenes-y-rellenos/" \l "rellenos)

Al igual que con los márgenes, los rellenos tienen varias propiedades para indicar cada zona:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| padding-top | **0** | size | Aplica un relleno interior en el espacio superior de un elemento. |
| padding-left | **0** | size | Aplica un relleno interior en el espacio izquierdo de un elemento. |
| padding-right | **0** | size | Aplica un relleno interior en el espacio derecho de un elemento. |
| padding-bottom | **0** | size | Aplica un relleno interior en el espacio inferior de un elemento. |

Como se puede ver en la tabla, por defecto no hay relleno (*el relleno está a cero*), aunque puede modificarse tanto con las propiedades anteriores como la propiedad de atajo que veremos a continuación.

### [Atajo: Modelo de cajas](https://lenguajecss.com/css/modelo-de-cajas/margenes-y-rellenos/" \l "atajo-modelo-de-cajas)

Al igual que en otras propiedades de CSS, también existen atajos para los márgenes y los rellenos:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valores | Significado |
| margin | size | 1 parámetro. Aplica el mismo margen a **todos** los lados. |
|  | size|size | 2 parámetros. Aplica margen **top/bottom** y **left/right**. |
|  | size|size|size | 3 parámetros. Aplica margen **top**, **left/right** y **bottom**. |
|  | size|size|size|size | 4 parámetros. Aplica margen **top**, **right**, **bottom** e **left**. |

Con las propiedades padding y border-width pasa exactamente lo mismo, actuando en relación a los **rellenos**, en lugar de los márgenes en el primer caso, y en relación al **grosor del borde** de un elemento en el segundo.

**Ojo**: Aunque al principio es muy tentador utilizar márgenes negativos para ajustar posiciones y colocar los elementos como queremos, se aconseja no utilizar dicha estrategia salvo para casos muy particulares, ya que a la larga es una mala práctica que hará que nuestro código sea de peor calidad.

## Bordes css

En CSS es posible especificar el aspecto que tendrán los bordes de cualquier elemento, pudiendo incluso, dar valores distintos a las diferentes **zonas** predeterminadas del elemento (zona superior, izquierda, derecha o zona inferior).

Las propiedades básicas existentes de los bordes en CSS son las siguientes:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| border-color |  | Especifica el color que se utilizará en el borde. |
| border-width | thin | **medium** | thick | | Especifica un tamaño predefinido para el grosor del borde. |
| border-style | **none** | | Define el estilo para el borde a utilizar (ver más adelante). |

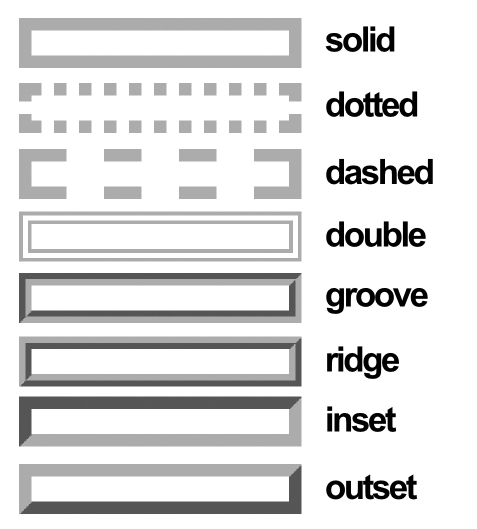
En primer lugar, **border-color** establece el color del borde, de la misma forma que lo hicimos en apartados anteriores de colores. En segundo lugar, con **border-width** podemos establecer la anchura o grosor del borde utilizando tanto **palabras clave** predefinidas como un tamaño concreto con cualquier tipo de las **unidades** ya vistas.

### [Estilos de borde](https://lenguajecss.com/css/modelo-de-cajas/bordes/" \l "estilos-de-borde)

Por último, con **border-style** podemos aplicar un estilo determinado al borde de un elemento. En **estilo de borde** podemos elegir cualquiera de las siguientes opciones:

|  |  |
| --- | --- |
| Valor | Descripción |
| hidden | Oculto. Idéntico al anterior salvo para conflictos con tablas. |
| dotted | Establece un borde basado en puntos. |
| dashed | Establece un borde basado en rayas (línea discontínua). |
| solid | Establece un borde sólido (línea contínua). |
| double | Establece un borde doble (dos líneas contínuas). |
| groove | Establece un borde biselado con luz desde arriba. |
| ridge | Establece un borde biselado con luz desde abajo. Opuesto a **groove**. |
| inset | Establece un borde con profundidad «hacia dentro». |
| outset | Establece un borde con profundidad «hacia fuera». Opuesto a **inset**. |

Sin embargo, el borde más frecuente suele ser **solid**, que no es más que un borde liso. Pueden utilizarse cualquiera de los estilos indicados en la tabla anterior. Veamos como se verían los diferentes estilos de borde utilizando **10 píxels** de grosor y color **gris**:



### [Bordes múltiples (diferentes)](https://lenguajecss.com/css/modelo-de-cajas/bordes/" \l "bordes-m%C3%BAltiples-diferentes)

Hasta ahora, sólo hemos utilizado un parámetro en cada propiedad, lo que significa que se aplica el mismo valor para cada borde de un elemento (*borde superior, borde derecho, borde inferior y borde izquierdo*). Sin embargo, podemos especificar uno, dos, tres o cuatro parámetros, dependiendo de lo que queramos hacer:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| border-color | color | 1 parámetro. Aplica el mismo color a todos los bordes. |
|  | color|color | 2 parámetros. Aplica al borde **top/bottom**, y al **left/right**. |
|  | color|color|color | 3 parámetros. Aplica al **top**, al **left/right** y al **bottom**. |
|  | color|color|color|color | 4 parámetros. Aplica al **top**, **right**, **bottom** y **left**. |

De la misma forma, podemos hacer exactamente lo mismo con las propiedades **border- width**

(*respecto al ancho del borde*) y **border-style** (*respecto al estilo del borde*).

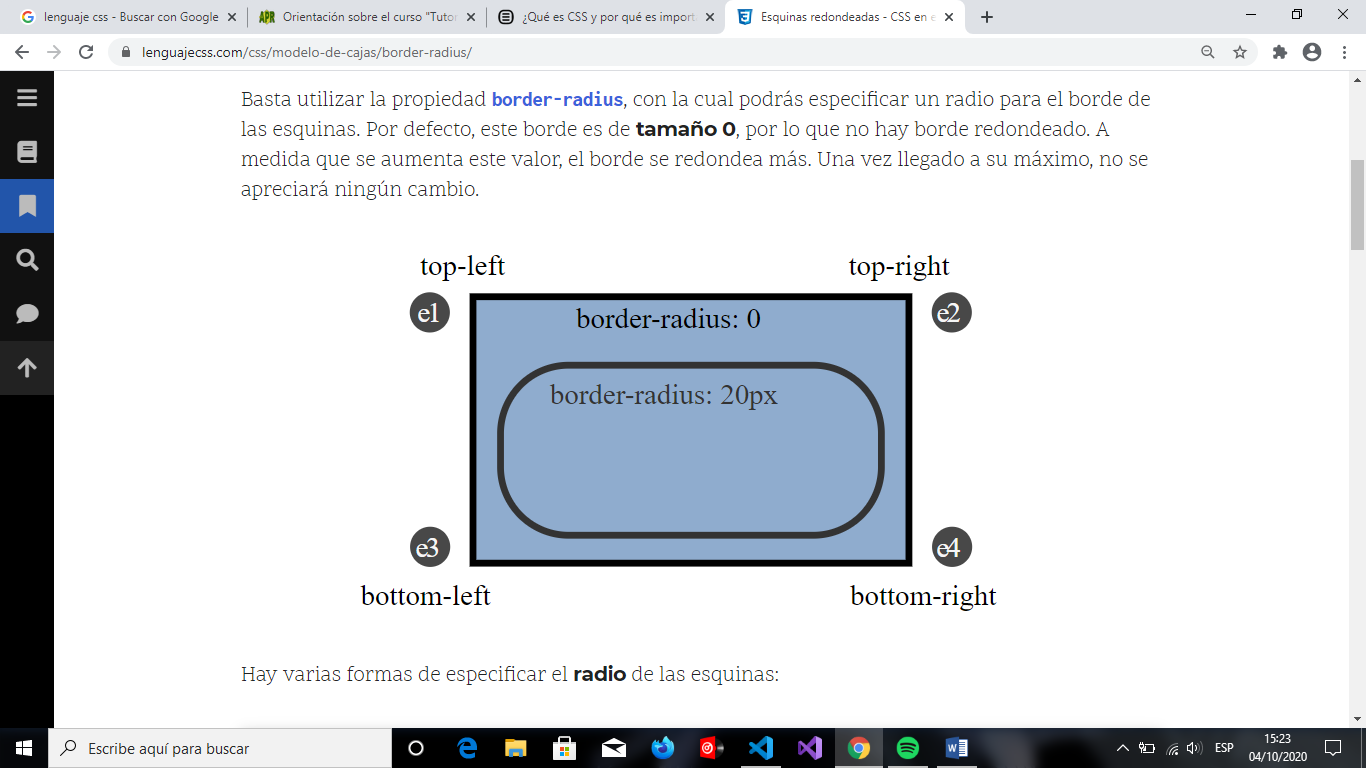
### [Atajo: Bordes](https://lenguajecss.com/css/modelo-de-cajas/bordes/" \l "atajo-bordes)

Pero ya habremos visto que con tantas propiedades, para hacer algo relativamente sencillo, nos pueden quedar varias lineas de código complejas y difíciles de leer. Al igual que con otras propiedades CSS, podemos utilizar la propiedad de atajo border, con la que podemos hacer un resumen y no necesitar utilizar las propiedades individuales por separado, realizando el proceso de forma más corta:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| border | **Size|style|color** | **Propiedad de atajo para simplificar valores.** |

## Esquinas redondeadas

**CSS3** añade interesantes características en materia de bordes, como la posibilidad de crear **bordes con esquinas redondeadas**, basta utilizar la propiedad **border-radius**, con la cual podrás especificar un radio para el borde de las esquinas.



Hay varias formas de especificar el **radio** de las esquinas:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| border-radius | size | 1 parámetro. Aplica el radio a **todas** y cada una de las esquinas. |
|  | size|size | 2 parámetros: **top-left** + **bottom-right** y a **top-right** + **bottom-left**. |
|  | size|size|size | 3 parámetros: **top-left**, a **top-right** y **bottom-left** y a **bottom-right**. |
|  | size|size|size|size | 4 parámetros. Orden de las agujas del reloj, empezando por **top-left**. |

### [Esquinas irregulares](https://lenguajecss.com/css/modelo-de-cajas/border-radius/" \l "esquinas-irregulares)

**Truco**: Es posible diferenciar el **radio horizontal** del **radio vertical** de una esquina determinada, creando una esquina redondeada irregular.

Para conseguirlo, no hay más que añadir una barra (/) y repetir nuevamente el número de parámetros escogido. De esta forma, los parámetros a la izquierda de la barra representan el radio horizontal, mientras que los que están a la derecha, representan el radio vertical.

### [Esquinas específicas](https://lenguajecss.com/css/modelo-de-cajas/border-radius/" \l "esquinas-específicas)

De la misma forma que hemos visto con anterioridad en otras propiedades CSS similares, también es posible especificar los valores de cada esquina mediante propiedades por separado:

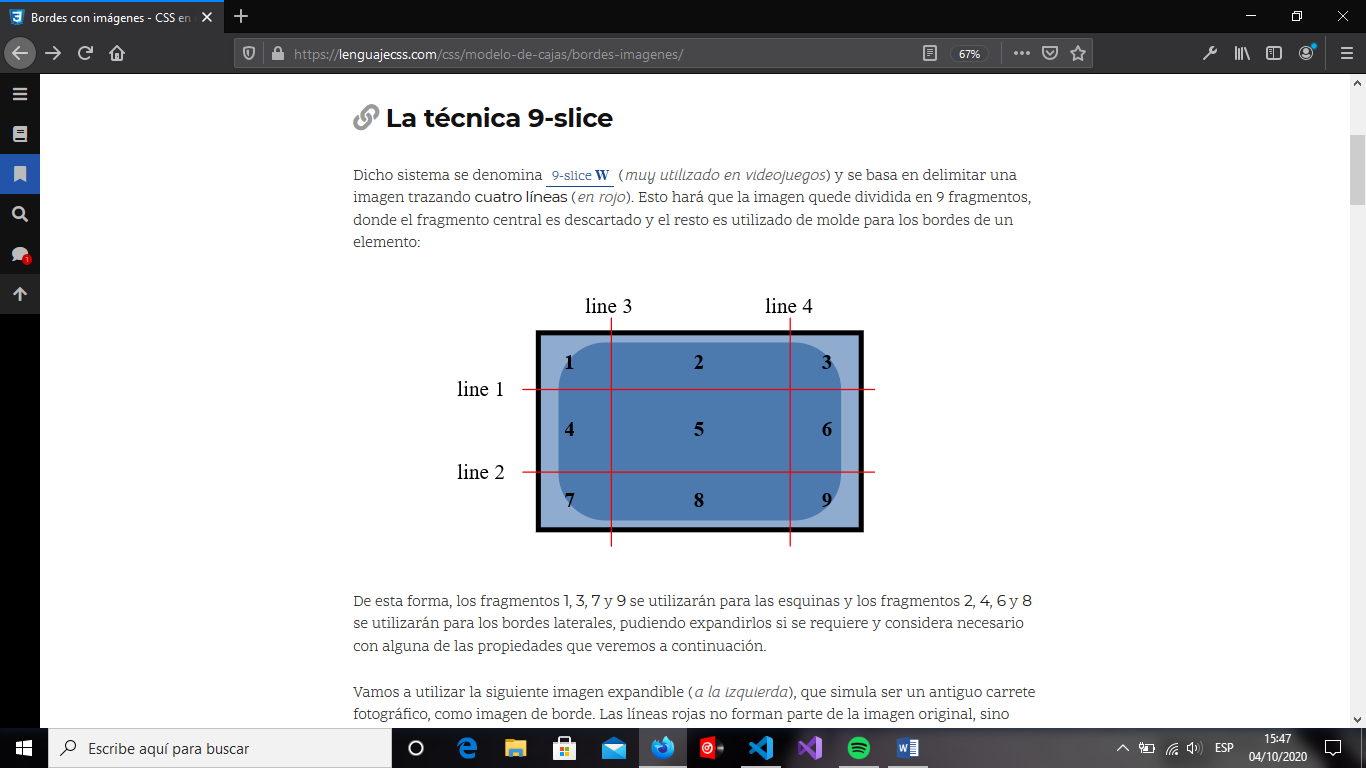
|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| border-top-left-radius |  | Indica un radio para redondear la esquina **top-left**. |
| border-top-right-radius |  | Indica un radio para redondear la esquina **top-right**. |
| border-bottom-left-radius |  | Indica un radio para redondear la esquina **bottom-left**. |
| border-bottom-right-radius |  | Indica un radio para redondear la esquina **bottom-right**. |

## Bordes con imágenes

Una de las limitaciones que tenía CSS, es que, si en lugar de utilizar los bordes de los que disponemos en CSS (*sólidos, punteados, etc...*) quisieramos hacer algo un poco más complejo con imágenes, podría volverse una tarea muy complicada. Por esa razón, **CSS3** incorporó en su momento un sistema para crear bordes extensibles basados en una imagen de molde.

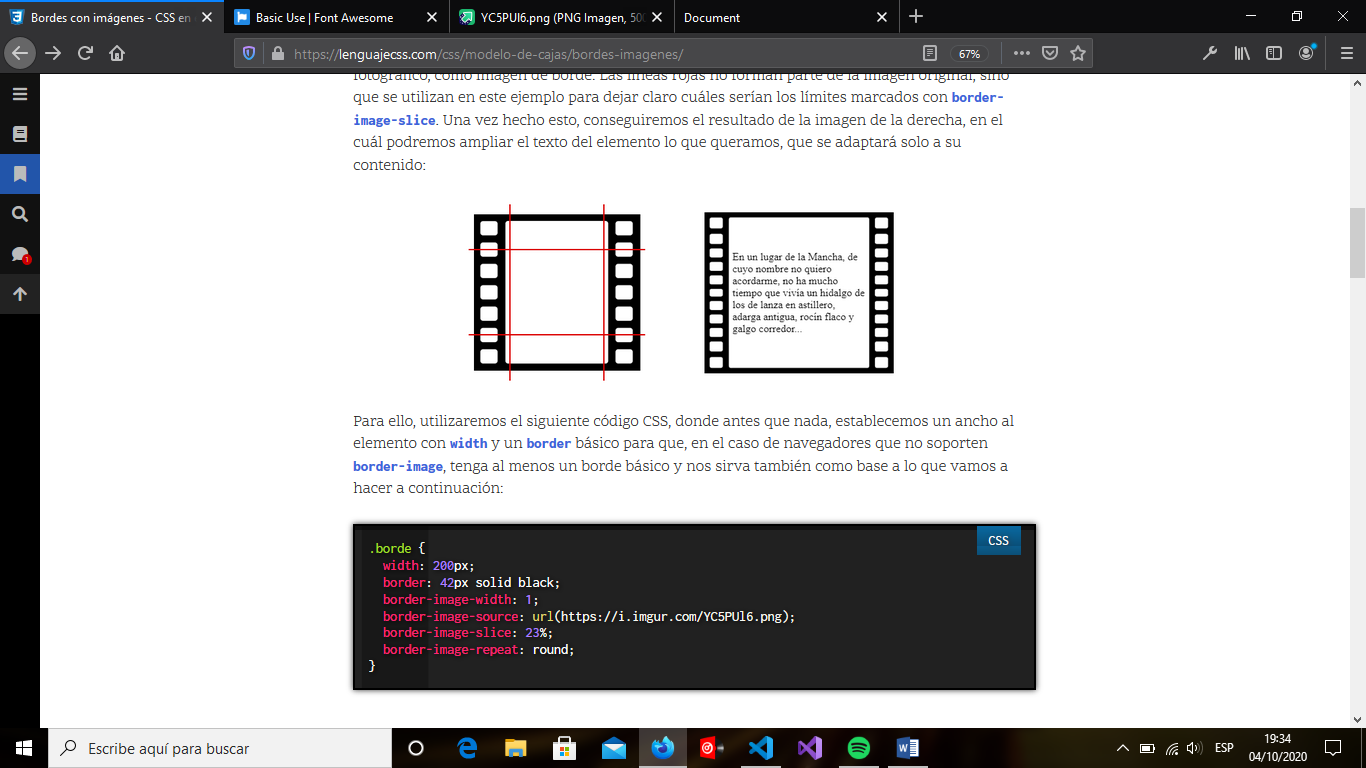
### [La técnica 9-slice](https://lenguajecss.com/css/modelo-de-cajas/bordes-imagenes/" \l "la-técnica-9-slice)

Dicho sistema se denomina [9-slice](https://en.wikipedia.org/wiki/9-slice_scaling) (*muy utilizado en videojuegos*) y se basa en delimitar una imagen trazando **cuatro líneas** (*en rojo*). Esto hará que la imagen quede dividida en 9 fragmentos, donde el fragmento central es descartado y el resto es utilizado de molde para los bordes de un elemento:



De esta forma, los fragmentos **1**, **3**, **7** y **9** se utilizarán para las esquinas y los fragmentos **2**, **4**, **6** y **8** se utilizarán para los bordes laterales, pudiendo expandirlos si se requiere y considera necesario con alguna de las propiedades que veremos a continuación.

Vamos a utilizar la siguiente imagen expandible (*a la izquierda*), que simula ser un antiguo carrete fotográfico, como imagen de borde. Las líneas rojas no forman parte de la imagen original, sino que se utilizan en este ejemplo para dejar claro cuáles serían los límites marcados con *border-image-slice*. Una vez hecho esto, conseguiremos el resultado de la imagen de la derecha, en el cuál podremos ampliar el texto del elemento lo que queramos, que se adaptará solo a su contenido:



Para ello, utilizaremos el siguiente código CSS, donde antes que nada, establecemos un ancho al elemento con **width** y un **border** básico para que, en el caso de navegadores que no soporten **border-image**, tenga al menos un borde básico y nos sirva también como base a lo que vamos a hacer a continuación:



Para establecer los límites y poder utilizar imágenes en los bordes, hemos hecho uso de las siguientes propiedades de CSS:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| border-image-width | **1** |size|number|auto | Grosor de la imagen de borde. |
| border-image-source | **none**|url(*imagen.png*) | Imagen a utilizar de molde para el 9-slice. |
| border-image-slice | **100%** |size|number| fill | Tamaño de los bordes en la imagen. |
| border-image-outset | **0** | size | Tamaño en el que el borde crece hacia fuera. |
| border-image-repeat | [*repetición en X e Y*] | 1 parámetro. Comportamiento de repetición en ambos ejes. |
|  | [*rep. en X*] [*rep. en Y*] | 2 parámetros. Comportamiento de repetición por separado. |

1. La propiedad **border-image-width** indica el **tamaño** que tendrá el borde de la imagen. El tamaño puede ser indicado con unidades (*píxeles o porcentajes, por ejemplo*) o sin ellas, lo que lo tomará como múltiplo del tamaño indicado en **border-width**.

En nuestro ejemplo, tanto indicarle un valor de **1** como dejarlo sin especificar, sería decirle que el tamaño del borde de la imagen será de 42px, pero por ejemplo, si indicamos **border-image-width**: 2, le estaremos indicando que use un tamaño de 84px.

**Consejo:** No olvides que hay que indicar también un border-width y un border-style para que el borde CSS esté definido y se pueda visualizar.

1. La propiedad **border-image-source** establece, mediante la expresión **url()**, la imagen que vamos a utilizar para crear nuestro borde con imágenes.

**Truco**: ¡Como imagen de fondo puedes utilizar un [gradiente de CSS](https://lenguajecss.com/css/colores-y-fondos/gradientes-css/), ya que internamente se interpretan como imágenes!

1. La propiedad **border-image-slice** define la posición de las líneas divisorias de la imagen, o lo que es lo mismo, el tamaño de los bordes. Por defecto, el valor es de **100%** (*tamaño de ancho completo de la imagen*), pero también podemos usar números sin unidad, que simbolizan píxeles de recorte. Se pueden especificar 1, 2, 3 ó 4 parámetros.

Por otro lado, el valor opcional **fill** sirve para indicar que quieres rellenar el elemento con el fondo del fragmento 5, que por defecto es descartado. Útil en casos que quieras aprovechar el fondo.

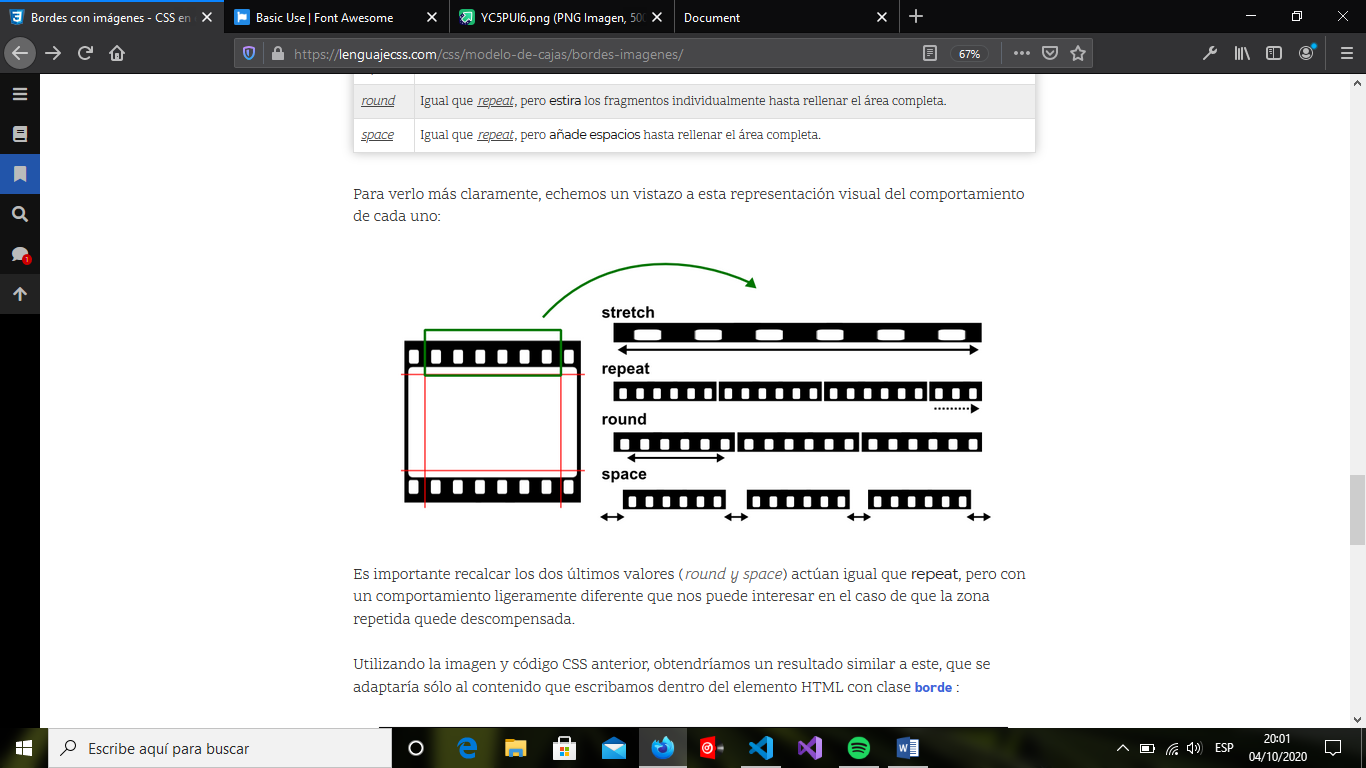
En nuestro caso, nos podría valer tanto con **110** (110 píxeles de recorte) como con **23%**, ya que es más o menos la cantidad apropiada para establecer el límite tanto de ancho como de alto.

1. La propiedad **border-image-outset** establece el factor de crecimiento (*hacia fuera*) de la imagen. Muy útil para compensar la imagen si se extiende hasta el contenido. Usar con cuidado, ya que puede desplazar el contenido. Por defecto no tiene desplazamiento.
2. La propiedad **border-image-repeat** establece como deben comportarse los fragmentos del borde y el tipo de repetición que deben efectuar. Se puede usar la modalidad de un parámetro en la que se aplica a todos los bordes, o la modalidad de dos parámetros donde estableces diferente comportamiento para los bordes horizontales y verticales.

Esta propiedad puede tomar los siguientes valores:

|  |  |
| --- | --- |
| Valor | Significado |
| stretch | Los bordes se **estiran** hasta rellenar el área. Es el valor por defecto. |
| *repeat* | Los bordes se **repiten** hasta rellenar el área. |
| *round* | Igual que *repeat*, pero **estira** los fragmentos individualmente hasta rellenar el área completa. |
| *space* | Igual que *repeat*, pero **añade espacios** hasta rellenar el área completa. |

Para verlo más claramente, echemos un vistazo a esta representación visual del comportamiento de cada uno:



Utilizando la imagen y código CSS anterior, obtendríamos un resultado similar a este, que se adaptaría sólo al contenido que escribamos dentro del elemento HTML con clase **borde** :



Como suele ser costumbre, este tipo de propiedades tienen una propiedad para ahorrar espacio y escribirlo todo de una sola vez. En este caso, la sintaxis es la siguiente:



Link para probar 9-slice

*<https://border-image.com/>*

## Funciones css

En CSS, muchas veces necesitaremos un sistema de apoyo para realizar operaciones más cercanas a un lenguaje de programación que a un lenguaje de estilos, como pueden ser **cálculos** o utilizar **valores precalculados.**

Un resumen de las **funciones CSS** que veremos:

|  |  |
| --- | --- |
| Función CSS | Descripción |
| calc() | Permite **calcular operaciones** con unidades CSS como px, %, vw, vh u otras. |
| min() | Permite calcular el **valor mínimo** de las unidades indicadas. |
| max() | Permite calcular el **valor máximo** de las unidades indicadas. |
| clamp() | Permite calcular valores «ajustados». Equivalente a max(MIN, min(VAL, MAX)). |
| attr() | Permite obtener el **valor de un atributo** desde CSS. |

### [Valores calculados: Función calc()](https://lenguajecss.com/css/modelo-de-cajas/funciones-css/" \l "valores-calculados-función-calc)

Es posible que en algunas ocasiones necesitemos indicar **valores precalculados** por el navegador. Por ejemplo, la suma de dos valores que a priori desconocemos o no sabemos exactamente cuanto suman, pero que el navegador si debería conocer.

Esto es posible hacerlo con la función calc() de CSS, como se muestra a continuación:



Como se puede ver, se pueden usar operaciones como sumas ( + ), restas ( - ), multiplicaciones ( \* ) o divisiones ( / ) que utilicen alguna de las unidades soportadas por CSS, como por ejemplo, números, dimensiones, porcentajes, tiempos, ángulos, etc...

Ten en cuenta que también se pueden anidar funciones calc() una dentro de otra. Resulta especialmente interesante su utilización junto a variables (custom properties CSS).

### [Valor mínimo: Función min()](https://lenguajecss.com/css/modelo-de-cajas/funciones-css/" \l "valor-mínimo-función-min)

En algunas ocasiones, necesitaremos que en una propiedad se aplique un valor de varias posibilidades posibles. De dichas posibilidades, no sabremos exactamente cuál necesitamos, pero si sabremos una de sus cualidades: **es la más pequeña**, es decir, necesitamos el valor mínimo.

CSS permite realizar esto utilizando la función min() que nos permite elegir el valor más pequeño de 2 o más posibilidades aplicadas por parámetro:



Esto aplicaría un width de 200px, siempre y cuando el 25% del elemento padre sea más grande de 200px. Todo este cálculo, generalmente realizado en Javascript debido a su cualidad de lógica de cálculo, puede realizarse desde CSS si sólo se utiliza exclusivamente para esta finalidad.

### [Valor máximo: Función max()](https://lenguajecss.com/css/modelo-de-cajas/funciones-css/" \l "valor-máximo-función-max)

De la misma forma que tenemos una función min(), también tenemos una función max(). Como podemos imaginar, dicha función sirve para lo opuesto de la función min(): obtener el valor máximo entre varias posibilidades.

Observa el siguiente ejemplo, donde utilizamos **más de 2 parámetros** (*ambas funciones permiten múltiples valores, no necesariamente dos*):



Observa que como se ve en el ejemplo en 200px - 100px, tanto en min() como en max(), puedes realizar operaciones directamente en su interior sin necesidad de utilizar la función calc().

### [Valor ajustado: Función clamp()](https://lenguajecss.com/css/modelo-de-cajas/funciones-css/" \l "valor-ajustado-función-clamp)

Ahora pensemos en un caso que podríamos necesitar y que se puede volver algo poco legible. Imagina que necesitas obtener el valor mínimo

Con la función clamp() podemos realizar, de forma concisa la operación max(MIN, min(VAL, MAX)). Veamos un ejemplo:



Observa que existen dos líneas con un width. Ambas líneas son equivalentes. El primer valor de clamp() sería el **valor mínimo**, mientras que el tercero es el **valor máximo**. De esta forma, el navegador realiza lo siguiente:

* Obtiene el **valor mínimo** entre el segundo y tercer parámetro.
* Obtiene el **valor máximo** entre el primer parámetro y el resultado anterior.
* Utiliza el resultado de la operación anterior en el width.

### [Valor del atributo: Función attr()](https://lenguajecss.com/css/modelo-de-cajas/funciones-css/" \l "valor-del-atributo-función-attr)

Aunque no es de género matemático como las anteriores, existe la función CSS attr(), que es algo más particular, ya que permite obtener el **valor de un atributo HTML** y utilizarlo desde CSS, generalmente para temas de decoración.

Por ejemplo, observa el siguiente fragmento de código:



Por medio de una [pseudoclase CSS](https://lenguajecss.com/css/selectores/pseudoclases/), utilizando la propiedad **content**, le pedimos al navegador que utilice el [metadato](https://lenguajehtml.com/html/introduccion/atributos-comunes-html/) del atributo **data-author** del elemento donde se le está aplicando el CSS para mostrarlo como contenido.

# Colores y fondos

## Colores css

Uno de los primeros cambios de estilo que podemos pensar realizar en un documento HTML es hacer variaciones en los colores de primer plano y de fondo. Esto es posible con las primeras dos propiedades que veremos a continuación:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| color |  | Cambia el **color del texto** que está en el interior de un elemento. |
| background-color |  | Cambia el **color de fondo** de un elemento. |

Todas las propiedades CSS donde existen valores , establecen la posibilidad de indicar **4 formas alternativas** (*con algunos derivados*) para especificar el color deseado:

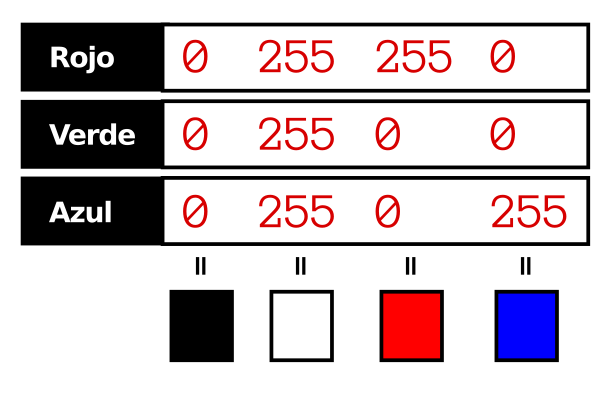
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | | Formato | | Ejemplo |
| Palabra clave predefinida | *[palabra clave]* | | red | |
| Esquema RGB | rgb(*rojo*, *verde*, *azul*) | | rgb(255, 0, 0) | |
| Esquema RGB con canal alfa | rgba(*rojo*, *verde*, *azul*, *alfa*) | | rgba(255, 0, 0, 0.25) | |
| Esquema RGB hexadecimal | #*RRGGBB* | | #ff0000 | |
| Esquema RGB hexadecimal con alfa | #*RGBBBAA* | | #ff000040 | |
| Esquema HSL | hsl(*color*, *saturación*, *brillo*) | | hsl(0, 100%, 100%) | |
| Esquema HSL con canal alfa | hsla(*color*, *saturación*, *brillo*, *alfa*) | | hsla(0, 100%, 100%, 0.25) | |

Además, existen algunos valores especiales que puedes utilizar cuando quieras especificar un color, como colores **transparentes** o el **color actual del texto**, muy útil para SVG, por ejemplo:

|  |  |
| --- | --- |
| Valor | Significado |
| transparent | Establece un color completamente transparente (valor por defecto de background-color) |
| currentColor | Establece el mismo color que se está utilizando para el texto (CSS3 y SVG) |

### [Formato RGB](https://lenguajecss.com/css/colores-y-fondos/colores-css/" \l "formato-rgb)

Uno de los métodos más conocidos por los diseñadores gráficos es utilizar el **formato RGB**. Las siglas RGB significan **rojo**, **verde** y **azul**, por lo que cada cifra (*del 0 al 255*) representa la intensidad de cada componente de color. Como se puede ver en la siguiente imagen, si utilizamos una cantidad (0, 0, 0) de cada canal, obtenemos el **color negro**. En cambio, si utilizamos una cantidad (255, 0, 0), obtendremos el **color rojo**.



De esta forma, mezclando las cantidades de cada canal, se puede obtener prácticamente cualquier color. Existen muchos esquemas de colores, pero en diseño web nos interesa particularmente el esquema RGB (*junto al HSL*).

### [Formato hexadecimal](https://lenguajecss.com/css/colores-y-fondos/colores-css/" \l "formato-hexadecimal)

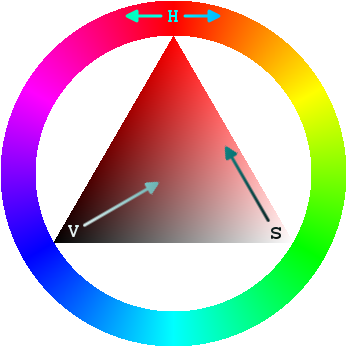
El **formato hexadecimal** es el más utilizado por los desarrolladores web, aunque en principio puede parecer algo extraño y complicado, sobre todo si no has oído hablar nunca del sistema hexadecimal (*sistema en base 16 en lugar del que utilizamos normalmente, en base 10*).

Cada par de letras simboliza el valor del RGB en el sistema de numeración hexadecimal, así pues, el color **#FF0000**, o sea HEX(FF,00,00), es equivalente al RGB(255,0,0), que es también equivalente al HSL(0, 100%, 100%). Veamos algunos ejemplos para clarificarlo:

|  |  |  |  |
| --- | --- | --- | --- |
| Hexadecimal | Hex. abreviado | Color RGB | Palabra clave |
| #FF0000 | #F00 | 255,0,0 | red (rojo) |
| #000000 | #000 | 0,0,0 | black (negro) |
| #00FFFF | #0FF | 0, 255, 255 | cyan (azul claro) |
| #9370DB | #97D | 147,112,219 | mediumpurple (lila) |

### [Formato HSL](https://lenguajecss.com/css/colores-y-fondos/colores-css/" \l "formato-hsl)

Las siglas HSL significan **color** (*o matiz*), **saturación** y **brillo**. La primera cifra selecciona el matiz de color (*una cifra de 0 a 360 grados*), seleccionando el color del círculo exterior de la imagen. Por su parte, las dos siguientes, son el porcentaje de saturación y el brillo del color, respectivamente (*ambos, porcentajes de 0% a 100%*).



### [Canales Alfa](https://lenguajecss.com/css/colores-y-fondos/colores-css/" \l "canales-alfa)

Es posible que deseemos indicar un color que tenga cierto grado de transparencia, y de esta forma, refleje el contenido, color o imágenes que se encuentren detrás. Hasta ahora solo conocemos la palabra clave transparent, que es un color de transparencia total (*totalmente transparente*).

Sin embargo, existe la posibilidad de utilizar los denominados **canales alfa**, que permiten establecer una **transparencia parcial** en determinados colores. Estos se pueden establecer en cualquier formato, salvo en los colores con palabras clave (red, blue, Orange, etc…). Vamos a ver cómo hacerlo en cada caso:

* **Formato RGB**: En lugar de rgb() indicamos rgba() para establecer que usaremos un canal alfa. Posteriormente, en lugar de establecer 3 parámetros (*rojo, verde, azul*), añadiremos uno más, que será el canal alfa. Dicho canal alfa será un valor (*del 0 al 1 con decimales*) o un porcentaje (*del 0% al 100%*).
* **Formato HSL**: Prácticamente idéntico al anterior. En lugar de hsl() indicamos hsla().
* **Formato Hexadecimal**: Es posible indicar (*al final*) un par adicional que indique el grado de transparencia. Por ejemplo, el color #FF0000 reescrito como #FF000077 se trataría de dicho color, con un grado de transparencia casi del 50% (00 es 0%, 80 es 50%, FF es 100%).

Veamos algunos ejemplos de cada caso:



## Fondos css

Es posible que, buscando hacer un diseño más avanzado, en lugar de utilizar un color de fondo quieras utilizar imágenes. Para ello, CSS te proporciona la propiedad ***background-image***, con la cuál puedes indicar imágenes de fondo o, como veremos más adelante, incluso degradados de varios colores.

### [Imágenes de fondo](https://lenguajecss.com/css/colores-y-fondos/fondos-css/" \l "imágenes-de-fondo)

En el caso de querer utilizar una imagen de fondo, como ya hemos dicho, utilizaremos la propiedad ***background-image*** y en el valor, el nombre de la imagen (*o la dirección donde está alojada*), siempre rodeada del texto url().

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| background-image | **none** | No utiliza ninguna imagen de fondo. |
| background-image | url(*imagen.jpg*) | Usa la imagen de nombre **imagen.jpg** como fondo. |
|  |  |  |

En el caso de que sólo se coloque el nombre de la imagen (*por ejemplo, imagen.jpg*), el navegador buscará la imagen en la misma carpeta donde está el archivo CSS. Esto es lo que se llama una **ruta relativa**. En el caso de que se coloque la ruta completa, por ejemplo https://i.emezeta.com/img/logo.png, se accederá a la imagen alojada en esa dirección web. Esto es lo que se llama **ruta absoluta**.

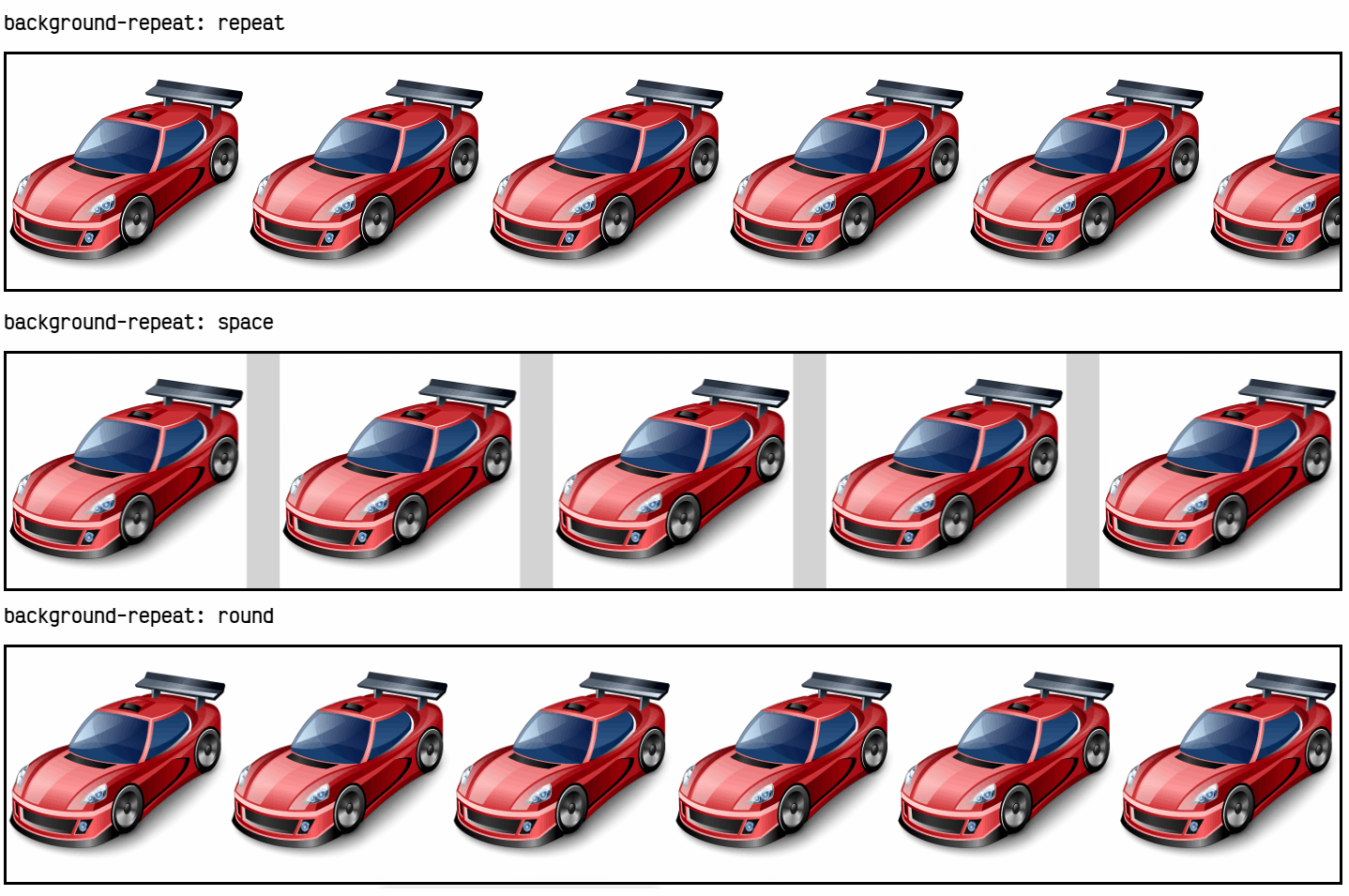
Una vez establecida una imagen de fondo con **background-image**, se puede personalizar la forma en la que se mostrará dicha imagen mediante propiedades como **background-repeat, background-attachment o background-position**, entre otras:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| background-repeat | **repeat** | Repite la imagen de fondo horizontal y verticalmente. |
|  | repeat-x | Repite la imagen de fondo sólo horizontalmente (eje x). |
|  | repeat-y | Repite la imagen de fondo sólo verticalmente (eje y). |
|  | space | Repite la imagen y rellena con espacio los huecos. |
|  | round | Repite la imagen y amplia cada repetición para ajustar. |
|  | no-repeat | La imagen de fondo no se repite. |
| background-attachment | **scroll** | Cuando hacemos scroll la imagen de fondo se desplaza. |
|  | fixed | Cuando hacemos scroll, la imagen de fondo permanece fija. |
| background-position | posx | 1 parámetro. Desplaza la imagen de fondo al punto (x, 50%). |
|  | Posx posy | 2 parámetros. Desplaza la imagen de fondo al punto (x, y). |

La propiedad background-repeat especifica si la imagen se repetirá horizontalmente (*repeat-x*), si se repetirá verticalmente (*repeat-y*), si lo hará en ambas direcciones (*repeat*) o en ninguna (*no-repeat*). Por defecto, si no se indica nada, esta propiedad está ajustada en **repeat**.

Existen también dos valores interesantes, **space** y **round**, los cuales asumen implícitamente que se repite el fondo. En el caso de que tengamos una imagen de fondo que se repita varias veces en mosaico, space evita que se corte la imagen, introduciendo un espacio entre las repeticiones individuales.

Por su parte, round lo que hace es ajustar la imagen individual, de modo que la expande o contrae para ajustarla al espacio disponible. En ambos casos la repetición de los fondos nunca se mostrará cortada.



Siempre se podrá combinar en cada eje, con valores mixtos, por ejemplo, utilizando background-repeat: **space** **round**, lo que aplicará space al eje X y round al eje Y. Si sólo se especifica uno, se aplicará a ambos ejes.

La propiedad ***background-attachment*** especificará si la imagen de fondo seguirá el desplazamiento del usuario (*scroll, el comportamiento por defecto*) o por el contrario, se quedará fijado y no se moverá (*fixed*), mientras el usuario se desplaza por la página.

Por último, ***background-position*** coloca la imagen en la zona especificada por y por . Por defecto, esos valores son **0% 0%**, pero pueden especificarse con unidades (*porcentajes, píxels, etc...*) o mediante palabras clave que representan zonas predefinidas (*top, left, right, bottom y center*).

Si sólo se especifica un valor, se tomará para el eje x, mientras que el valor del eje Y será automáticamente establecido a **center** (*o 50%*).

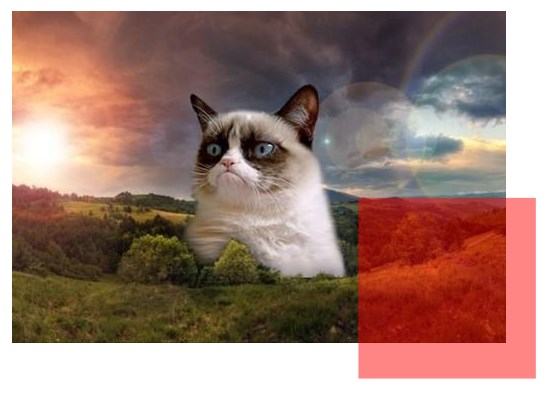
### [Opacidad](https://lenguajecss.com/css/colores-y-fondos/fondos-css/" \l "opacidad)

Es posible utilizar la propiedad *CSS3* opacity para establecer una transparencia total sobre el elemento indicado. Cuando decimos «**transparencia total**» nos referimos a que la transparencia se aplicará al elemento en cuestión y a todos los elementos HTML que estén en su interior.

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| opacity | *number* | Establece una transparencia (0 = 100% transparente, a 1 = 100% opaco) |

El valor a indicar es un número entre 0 (*completamente transparente*) y 1 (*completamente visible*), pudiendo establecer decimales para valores intermedios:





Como se puede ver en la imagen, esto hará que la capa div (el recuadro rojo) se muestre al **50% de opacidad**, con color de texto blanco (en el caso de existir texto) y fondo de color rojo. Si buscamos una transparencia parcial (color de fondo transparente que no afecte al texto) debemos utilizar los valores **RGBA** o **HSLA** (canales alfa) en la propiedad **background-color**:



De este modo, sólo aplicamos la transparencia al color de fondo, mientras que con **opacity** se aplica a toda la capa en sí, por lo que afecta a todos los elementos que están dentro de la capa.

### [Fondos múltiples](https://lenguajecss.com/css/colores-y-fondos/fondos-css/" \l "fondos-múltiples)

**CSS3** ofrece nuevas características a la hora de utilizar imágenes de fondo, como por ejemplo la posibilidad de establecer **múltiples imágenes de fondo** de forma simultánea:



De esta forma, se pueden utilizar varias imágenes y superponerlas una sobre la otra, algo especialmente interesante si la primera imagen de fondo está en formato PNG (*la cuál soporta transparencias*). Al establecer imágenes de fondo múltiples, las propiedades complementarias a los fondos como ***background-repeat, background-position*** y otras, pueden actuar de forma personalizada para cada fondo.

Mientras que en el fragmento de código anterior, el navegador le indica a cada una de las tres imágenes que no debe repetirse, en el siguiente fragmento de código veremos que es posible indicar individualmente el comportamiento de cada una, separando por comas:



Además, **CSS3** también añade nuevas propiedades para especificar como cubrirá la imagen de fondo al elemento en cuestión:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| background-clip | **border-box** | padding-box | content-box | Modo de relleno de la imagen de fondo |
| background-origin | border-box | **padding-box** | content-box | Origen del modo de relleno del fondo |

La propiedad **background-clip** establece la forma en la que el color o la imagen de fondo cubrirá el elemento, mientras que la propiedad **background-origin** intenta posicionar el comienzo de la imagen de fondo, útil con imágenes. La primera utiliza **border-box** como valor por defecto, mientras que la segunda utiliza **padding-box**.

Ambas propiedades pueden tomar uno de los siguientes valores:

|  |  |
| --- | --- |
| Valor | Significado |
| padding-box | La imagen o color de fondo cubrirá la zona del espaciado y contenido. |
| border-box | La imagen o color de fondo cubrirá la zona del borde, espaciado y contenido. |
| content-box | La imagen o color de fondo cubrirá sólo la zona del contenido. |

**Consejo**: Una buena forma de darse cuenta del funcionamiento de estas propiedades es establecer un borde grueso punteado. Usando border-box la imagen de fondo se extenderá en todo el elemento, incluyendo borde, espaciado y contenido. El valor padding-box extenderá la imagen de fondo sólo mediante el padding y el contenido, y por último, la propiedad content-box extenderá la imagen de fondo sólo en la zona del contenido.

### [Tamaño de fondos](https://lenguajecss.com/css/colores-y-fondos/fondos-css/" \l "tamaño-de-fondos)

Una de las últimas incorporaciones a la familia de propiedades de fondos de imagen es la propiedad background-size, la cuál ajusta el tamaño (*ancho y alto*) de la imagen de fondo, por si deseamos escalarla a un tamaño diferente. Por defecto, una imagen de fondo toma automáticamente el tamaño de la imagen (*que podría ser demasiado grande, por ejemplo*). Para no tener que modificar la imagen original de forma manual con un editor de imágenes, podemos utilizar esta propiedad y ajustarla a nuestro agrado mediante CSS:

|  |  |  |
| --- | --- | --- |
| Propiedad | Valor | Significado |
| background-size |  | 1 parámetro. Aplica un de (*ancho* × **auto**) a la imagen de fondo. |
|  |  | 2 parámetros. Aplica un de (*ancho* × *alto*) a la imagen de fondo. |

Los valores de tamaño que podemos utilizar, son los siguientes:

|  |  |
| --- | --- |
| Valor | Significado |
| auto | No escala la imagen. Utiliza el tamaño original. Es el valor por defecto. |
| *unidad* | Indicamos el tamaño específico que queremos usar (*píxels o porcentaje, por ej.*). |
| cover | Escala el ancho de la imagen de fondo al ancho del elemento. |
| contain | Escala el alto de la imagen de fondo al alto del elemento. |

Los dos últimos valores (***cover y contain***) sólo pueden utilizarse en el caso de que se especifique un sólo parámetro como valor en la propiedad **background-size**.

### [Atajo moderno: Fondos](https://lenguajecss.com/css/colores-y-fondos/fondos-css/" \l "atajo-moderno-fondos)

Los navegadores modernos, incluyen una nueva propiedad de atajo **background** que permite incluir los valores de propiedades CSS3 como **background-clip** , **background-origin** o **background-size**, que fueron incorporados más tarde que los demás. Es por ello, que el orden aconsejado para adquirir buenas prácticas es el siguiente y varía un poco respecto a la otra propiedad de atajo tradicional:



## Gradientes css

Otra funcionalidad que se echaba mucho de menos en versiones anteriores de CSS era la de utilizar fondos con **gradientes**, o lo que es lo mismo, un color inicial que se transforma poco a poco en uno o varios colores sucesivamente. Sin embargo, en versiones anteriores a CSS3 no era posible hacerlo desde código.

En CSS3 se introducen algunas expresiones relacionadas que harán esta tarea algo muy sencillo, pudiendo utilizarlas fácilmente mediante la propiedad **background-image** e indicando el tipo de gradiente deseado:

|  |  |  |
| --- | --- | --- |
| Propiedad | Función de gradiente | Significado |
| background-image | linear-gradient(...) | Define un gradiente lineal, en una dirección específica. |
|  | radial-gradient(...) | Define un gradiente radial, en forma de círculo o elipse. |
|  | conic-gradient(...) | Define un gradiente cónico, desde un punto de vista superior. |

Veamos cada uno de estos gradientes por separado para entender como funcionan.

### [Gradientes lineales](https://lenguajecss.com/css/colores-y-fondos/gradientes-css/" \l "gradientes-lineales)

El primer tipo de gradiente es el **gradiente lineal**, que permite crear fondos degradados que van en una cierta dirección y cambian de un color a otro, dependiendo de la lista de colores indicada por parámetro.

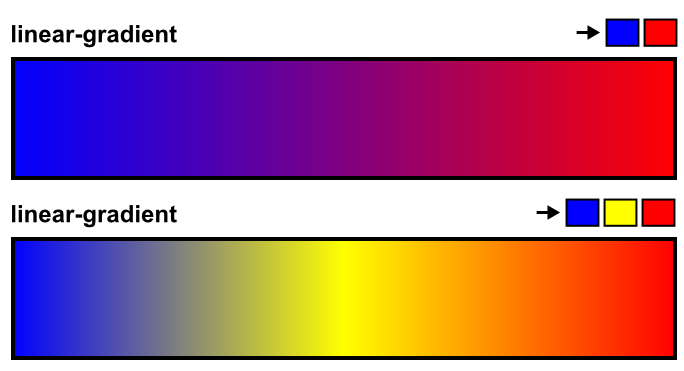
La sintaxis a utilizar debe ser una de las siguientes (*es la misma, pero con menor o mayor cantidad opcional de información*):

|  |  |
| --- | --- |
| Valor | Significado |
| linear-gradient(color,color , ...) | Gradiente básico entre colores. |
| linear-gradient( direction| angle, color , color , ...) | Gradiente con dirección o ángulo. |
| linear-gradient(direction | angle,color size , color size , ...) | Gradiente indicando el tamaño del color. |

El ejemplo más básico de los anteriores es el primero, donde simplemente indicaremos los colores del gradiente (*para que se aprecie el degradado, como mínimo deben ser 2 colores*):



Esto dibujará un degradado con los colores indicados en dirección desde **arriba hacia abajo**, que es la dirección por defecto. Se pueden ir añadiendo más colores al gradiente si se desea, o cambiar la dirección del degradado, como veremos a continuación:



Pero los parámetros que se pueden utilizar son los siguientes:

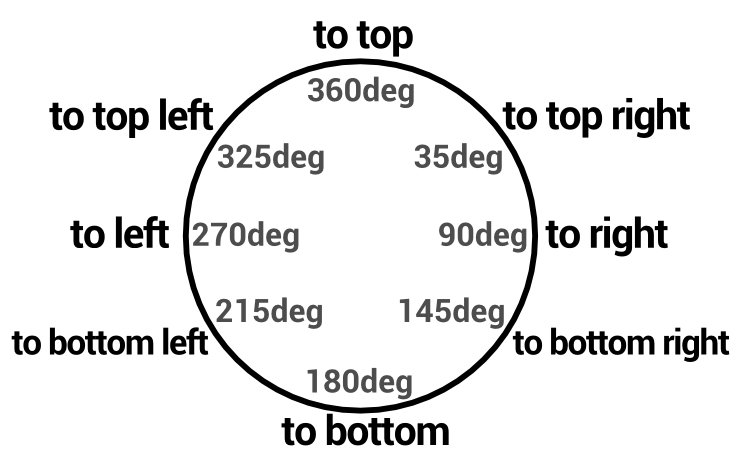
|  |  |
| --- | --- |
| Parámetros | Valor |
| direction | Dirección del gradiente lineal (*ver valores posibles más adelante*). El valor por defecto es **to bottom**. |
| angle | Número de ángulos de la dirección del gradiente lineal. Alternativa a la opción anterior. |
| color | El primer color del gradiente |
| color | El siguiente color del gradiente |
| size | De forma opcional, se puede indicar a que altura comienza a cambiar el gradiente. |
| ... | (Se pueden seguir añadiendo colores) |

### [Dirección del gradiente](https://lenguajecss.com/css/colores-y-fondos/gradientes-css/" \l "dirección-del-gradiente)

En el ejemplo visual anterior, la dirección del gradiente es *horizontal* (hacia la derecha). El primer ejemplo tiene sólo 2 colores (azul y rojo), mientras que el segundo tiene 3 colores (azul, amarillo y rojo). Ambos tienen como dirección una orientación hacia la derecha:



Como vemos, el primer parámetro utilizado fue *to right*, que es lo que indica la orientación del gradiente. En este parámetro se puede indicar tanto una **palabra clave** *direction* como un número de grados *angle* que represente la dirección:



Para que quede aún más claro, veamos una tabla de equivalencias:

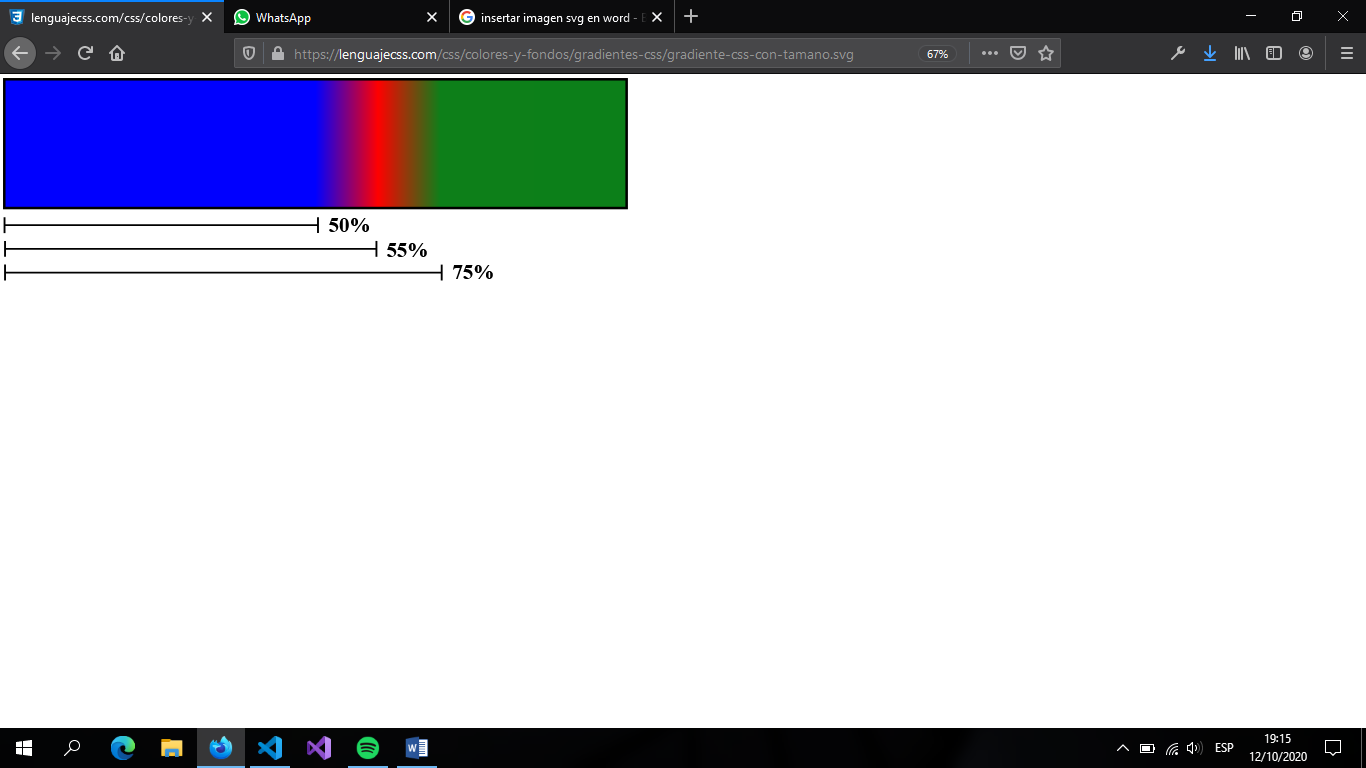
|  |  |  |
| --- | --- | --- |
| Dirección | Grados | Significado |
| to top | 0 ó 360deg | Gradiente lineal hacia arriba. |
| to top right | 35deg | Gradiente diagonal hacia derecha-arriba. |
| to right | 90deg | Gradiente lineal hacia derecha. |
| to bottom right | 145deg | Gradiente diagonal hacia abajo-derecha. |
| to bottom | 180deg | Gradiente lineal hacia abajo. |
| to bottom left | 215deg | Gradiente diagonal hacia abajo-izquierda. |
| to left | 270deg | Gradiente lineal hacia izquierda. |
| to top left | 325deg | Gradiente diagonal hacia izquierda-arriba. |

### [Colores del gradiente](https://lenguajecss.com/css/colores-y-fondos/gradientes-css/" \l "colores-del-gradiente)

Por defecto, las distancias entre colores se ajustan automáticamente con proporciones equivalentes, salvo que especifiquemos un tamaño (*porcentaje o píxeles, por ejemplo*) justo después del color en cuestión, que se puede indicar de forma opcional:

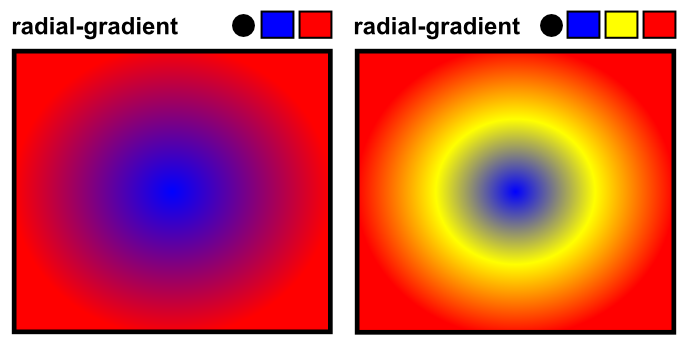


Este tamaño representa el momento en el que el gradiente comienza a cambiar el color, respecto al tamaño del gradiente completo. En este fragmento de código, nos mostraría un gradiente similar al que vemos a continuación, donde se aprecia que los tamaños o distancias entre colores no son equivalentes, sino que corresponden **azul** (*0-50%*), **rojo** (*50%-55%*) y **verde** (*55%-75%*), el cuál permanece hasta el final al no haber ningún color más:



### [Gradiente radial](https://lenguajecss.com/css/colores-y-fondos/gradientes-css/" \l "gradiente-radial)

Como vimos al principio, a parte de gradientes lineales, también podemos crear otros tipos de gradientes, como por ejemplo, los **gradientes radiales**, que permite crear degradados con formas circulares. Para ello, sólo tenemos que utilizar la expresión *radial-gradient* en lugar de *linear-gradient*:



De la misma forma, también tenemos diferentes modalidades para crear **gradientes radiales**, desde una muy básica donde le indicamos sólo los colores, hasta otra más avanzada donde le damos forma, tamaño e incluso ubicación:

|  |  |
| --- | --- |
| Valor | Significado |
| radial-gradient(color,color , ...) | Gradiente básico entre colores. |
| radial-gradient(*forma*, color, color, ...) | Gradiente con forma circular o elíptica. |
| radial-gradient(*forma* size,color ,color , ...) | Gradiente con tamaño o dimensión. |
| radial-gradient(*forma* size at *ubicación*, color ,color , ...) | Gradiente con colocación inicial. |
| radial-gradient(*forma* size at *ubicación*, color size, color size, ...) | Gradiente con tamaños entre colores. |

Algunos ejemplos aplicando estos parámetros:



Donde:

|  |  |
| --- | --- |
| Parámetros | Valor |
| *[forma]* | **ellipse** | circle |
| size | **farthest-corner** | closest-corner | farthest-side | closest-side | size |
| *[ubicación]* | **center** | top | left | right | bottom | top left | top right | bottom left | bottom right |
| color | El primer color del gradiente |
| color | El siguiente color del gradiente |
| size | De forma opcional, se puede indicar a que altura comienza a cambiar el gradiente. |
| ... | (Se pueden seguir añadiendo colores) |

En el caso de la **forma** podemos elegir circle o ellipse, dependiendo de si queremos un gradiente con forma circular o uno con forma de elipse (*que se adapta a la forma del elemento*). En el caso del **tamaño** del elipse o círculo, tenemos varias opciones para indicar el tamaño que tendrá nuestro gradiente:

|  |  |
| --- | --- |
| Valor | Significado |
| farthest-corner | Intenta cubrir hasta la esquina más lejana (*expande hacia ambos lados*). |
| farthest-side | Intenta cubrir hasta el lado más lejano (*expande hacia un sólo lado*). |
| closest-corner | Intenta cubrir hasta la esquina más cercana. |
| closest-side | Intenta cubrir hasta el lado más cercano. |
| size | Le da un tamaño específico al gradiente (*píxeles, porcentajes, etc...*). Sólo usable en circle. |

En el caso de la **ubicación**, simplemente especificamos la posición en la que queremos que esté colocado el centro del gradiente.

### [Gradiente cónico](https://lenguajecss.com/css/colores-y-fondos/gradientes-css/" \l "gradiente-cónico)

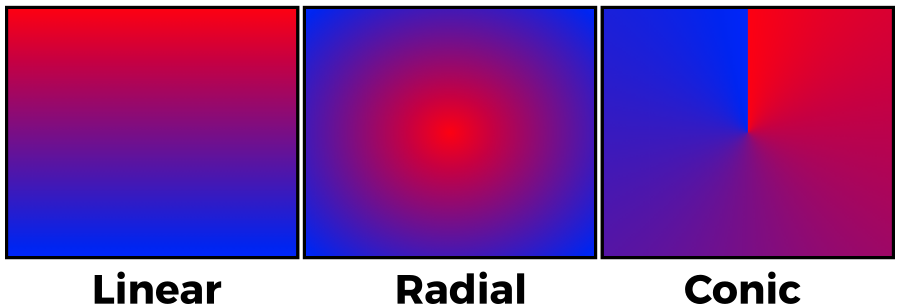
Recientemente, se ha incorporado un nuevo tipo de gradiente muy interesante en CSS, visualmente muy similar al **gradiente radial**, pero tratándose de una visualización de un cono desde un plano superior, incluyendo sus sombras.

|  |  |
| --- | --- |
| Valor | Significado |
| conic-gradient(color,color , ...) | Gradiente básico entre colores. |
| conic-gradient(from angle,color size ,color size , ...) | Gradiente indicando punto de inicio. |
| conic-gradient(from angle at *ubicación*,color size ,color size, ...) | Gradiente indicando ubicación del inicio. |

Donde:

|  |  |
| --- | --- |
| Parámetros | Valor |
| from *angle* | Ángulo desde donde comienza el gradiente cónico. Por defecto **from 0deg**. |
| at *[ubicación]* | **center** | top | left | right | bottom | top left | top right | bottom left | bottom right |
| *color* | El primer color del gradiente |
| *color* | El siguiente color del gradiente |
| *size* | De forma opcional, se puede indicar a que altura comienza a cambiar el gradiente. |
| ... | (Se pueden seguir añadiendo colores) |

A continuación podemos ver la diferencia de un gradiente *linear-gradient*, un *radial-gradient* y un *conic-gradient*:



Poco hay que añadir en este tipo de gradiente, ya que funciona muy similar a los anteriores. La única diferencia mencionable es el uso de la palabra clave **from** antes de indicar los ángulos de dirección, si se requiere.

### [Gradientes repetitivos](https://lenguajecss.com/css/colores-y-fondos/gradientes-css/" \l "gradientes-repetitivos)

Por último, todos los gradientes que hemos visto permiten la posibilidad de añadir el prefijo *repeating*- para conseguir que el efecto del gradiente, en lugar de adaptarse a la región completa, realice una repetición constantemente. Comprueba los siguientes ejemplos de forma individual:



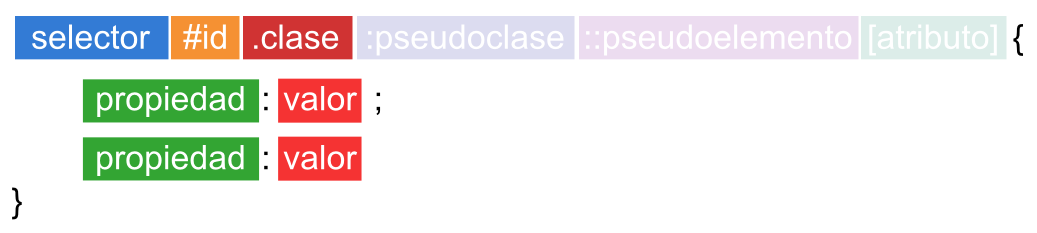
Es importante que al usar el prefijo repeating- se especifiquen tamaños a los colores, puesto que se necesita saber cuando comenzará a repetirse. En los ejemplo los colores seguirían el patrón hasta el último color y volverían a repetirse sucesivamente, de forma que tendríamos:

* Primera repetición: **blue** 10%, **yellow** 20%, **red** 30%
* Segunda repetición: **blue** 40%, **yellow** 50%, **red** 60%
* Tercera repetición: **blue** 70%, **yellow** 80%, **red** 90%
* Cuarta repetición: **blue** 100%

# Selectores

## Selectores básicos

Cuando comenzamos a trabajar con CSS, es habitual dar estilo con ejemplos muy sencillos, donde generalmente utilizamos un **selector genérico** que representa (*por ejemplo*) una etiqueta HTML. Sin embargo, lo que estamos haciendo en realidad es seleccionar todos los elementos del documento que sean dicha etiqueta. La verdadera potencia de CSS radica en la capacidad de poder **seleccionar sólo los elementos que nos interesen**



### [Seleccionar por etiquetas](https://lenguajecss.com/css/selectores/selectores-basicos/" \l "seleccionar-por-etiquetas)

Como ya hemos mencionado, la forma más básica de seleccionar elementos en CSS es indicar el elemento al cuál queremos aplicarle los estilos. Esto se comportará como parece lógico: aplicando el estilo CSS a **TODOS** los elementos **de ese tipo**:



En el ejemplo anterior, todos los elementos marcados con la etiqueta HTML <strong> se visualizarán de color rojo.

### [Seleccionar por ID (únicos)](https://lenguajecss.com/css/selectores/selectores-basicos/" \l "seleccionar-por-id-únicos)

Todas las etiquetas HTML pueden tener un atributo id con un valor concreto. Un buen símil con la vida real es la de un **DNI** o tarjeta de identificación, ya que la particularidad clave de los **ID** es que no se deben repetir, Veamos un ejemplo:



En la práctica no suelen utilizarse demasiado los **IDs**, ya que en la mayoría de los casos utilizar una **clase** es perfectamente válido y mucho más mantenible a la larga. La situación en la que los **IDs** están bien utilizados, es cuando se usan para designar una zona del documento que sabemos perfectamente que no se va a repetir, y debe ser identificada como una zona única.



Como podemos ver en el ejemplo, en CSS la forma de hacer referencia a los IDs es con el símbolo *#,* mientras que en el HTML se indica el atributo *id="saludo".*

### [Seleccionar por clases](https://lenguajecss.com/css/selectores/selectores-basicos/" \l "seleccionar-por-clases)

Las etiquetas HTML pueden tener otro atributo interesante: class. La diferencia principal respecto a los **IDs** es que las clases no se requiere que sean únicas.



Esto nos permite indicar en la parte de CSS un sólo selector con los estilos y reutilizarlos en todos los elementos HTML que se desee.



En CSS se hace referencia a las clases con un punto: .classic, mientras que en el HTML se escribiría el atributo class="classic".

### [Selecciones mixtas](https://lenguajecss.com/css/selectores/selectores-basicos/" \l "selecciones-mixtas)

Al margen de todo lo que hemos visto, es posible utilizar varias clases en un mismo elemento HTML, simplemente separando por espacios dentro del atributo class.

De esta forma, a dicho elemento se le aplicarán los estilos de cada una de las clases indicadas, las cuales suelen tener un grupo de características relacionadas con su nombre, lo cuál es interesantes y muy práctico de recordar, dándonos mucha soltura a la hora de crear clases y reutilizarlas:

