

1. Introducción a CSS

Puedes consultar los apuntes completos en <https://lenguajecss.com/css/>

1. Introducción a CSS

¿Qué es CSS?

¿Qué es realmente CSS?

¿Cómo usar CSS?

Enlace a CSS externo (link)

Incluir CSS en el HTML(style)

Estilos en línea (atributo style)

Estructura de CSS

Minificar CSS

¿Qué es la minificación?

Herramientas de minificación

Los navegadores web

Ecosistema de navegadores

Historia de los navegadores

Navegadores actuales

Versiones de los navegadores

Otros navegadores

Niveles de CSS y prefijos

Niveles CSS

Prefijos

Herencia en CSS

Concepto de herencia

Valores especiales

Guía interactiva de propiedades

2. Modelo de cajas

Unidades CSS

Unidades absolutas

Unidades relativas

Unidades flexibles (viewport)

Modelo de cajas

Dimensiones (ancho y alto)

Zonas de un elemento

Desbordamiento

Márgenes y rellenos

Márgenes

Rellenos

Atajo: Modelo de cajas

Bordes CSS

Estilos de borde

Bordes múltiples (diferentes)

Atajo: Bordes

Bordes específicos

Esquinas redondeadas

Esquinas irregulares

Esquinas específicas

3. Colores y fondos

Colores CSS

Palabras clave de color

Formato RGB

Formato hexadecimal

Formato HSL

Canales Alfa

Fondos en CSS

Imágenes de fondo

Atajo clásico: Fondos

Opacidad

Fondos múltiples

Tamaño de fondos

Atajo moderno: Fondos

4. Selectores CSS

Selectores CSS básicos

Seleccionar por etiquetas

Seleccionar por ID (únicos)

Seleccionar por clases

Selecciones mixtas

Pseudoclases CSS

Pseudoclases de enlaces

Pseudoclases de ratón

Pseudoclases de interacción

 Pseudoclases de activación

 Pseudoclases de validación

Pseudoclases de negación

Otras pseudoclases

5. Fuentes y tipografías

Tipografías CSS

Detalles de una tipografía

Propiedades básicas

 Familia tipográfica

 Tamaño de la tipografía

 Estilo de la tipografía

 Peso de la tipografía

Tipografías externas

 La regla @font-face

 Google Fonts

 Atajo para tipografías

Textos y alineaciones

 Alineaciones

 Variaciones

6. Representación de datos

Tablas CSS

 Propiedades CSS para tablas

Listas CSS

Listas CSS

 Atajo: Listas

7. Maquetación y colocación

Tipos de elementos

 Otros tipos de elementos

 Ocultar elementos

Posicionamiento

 Posicionamiento relativo

 Posicionamiento absoluto

 Posicionamiento fijo

 Otros posicionamientos

 Profundidad (niveles)

Desplazamientos

 Elementos flotantes

 Limpiar flujo flotante

8. Responsive Web Design

[¿Qué es Responsive Design?](#)
[Conceptos básicos](#)
[Preparación previa](#)
[Estrategias de diseño](#)
[Bases del Responsive Design](#)
[Diseño con porcentajes](#)
[Tamaños máximos y mínimos](#)
[El viewport](#)
[Media Queries](#)
[¿Qué son las media queries?](#)
[Ejemplos de media queries](#)
[Tipos de características](#)
[Condicionales CSS](#)
[Medios impresos](#)
[¿Qué son los medios impresos?](#)
[Ejemplo de medios impresos](#)

¿Qué es CSS?

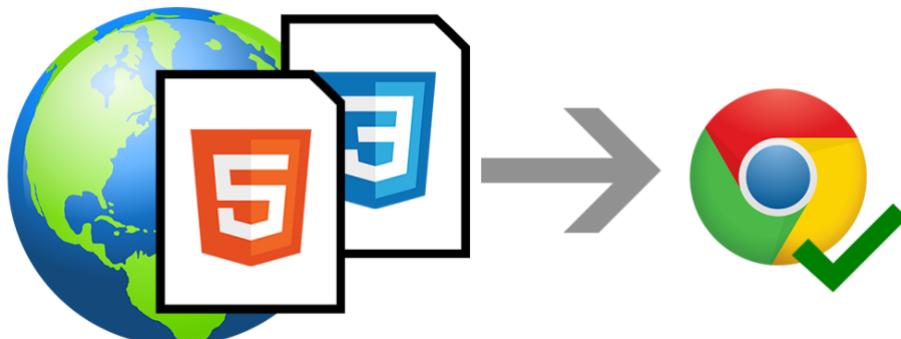
Si te gusta el mundo del diseño web o tienes curiosidad por empezar en este ámbito, probablemente ya habrás escuchado el término **CSS**. Se trata de una tecnología utilizada para dotar de **cualidades visuales y estéticas** a una página web. Si nunca has tocado esta materia, comprobarás que se trata de una forma analítica, lógica y casi matemática de crear páginas web, pero gracias a ella podemos simplificar la creación de páginas y conseguir exactamente lo que buscamos.

La **curva de aprendizaje** de CSS suele ser sencilla (*puede ser algo dura si nunca has programado o eres totalmente ajeno a estas temáticas*), pero a medida que cometes errores y vas practicando, tu capacidad para escribir código CSS mejora de forma exponencial, permitiéndonos avanzar a un ritmo cada vez más veloz.

¿Qué es realmente CSS?

Antes de comenzar, debes tener claro un concepto clave: una página web es realmente **un documento de texto**. En dicho documento se escribe **código HTML**, con el que se crea el contenido de una web. Por otro lado, existe el **código CSS**, que unido al código HTML permite darle forma, color, posición (*y otras características visuales*) a una página.

En resumen, se trata de un idioma como podría ser el inglés o el alemán, que los navegadores web como **Chrome** o **Firefox** conocen y pueden entender. Nuestro objetivo como diseñadores y programadores web es precisamente ese: aprender el idioma.



Las siglas **CSS** (*Cascading Style Sheets*) significan «Hojas de estilo en cascada» y parten de un concepto simple pero muy potente: aplicar **estilos** (colores, formas, márgenes, etc...) a uno o varios documentos (*generalmente documentos HTML, páginas webs*) de forma masiva.

Se le denomina estilos **en cascada** porque se aplican de arriba a abajo (*siguiendo un patrón denominado **herencia** que trataremos más adelante*) y en el caso de existir ambigüedad, se siguen una serie de normas para resolverla.

La idea de CSS es la de utilizar el concepto de **separación de presentación y contenido**, intentando que los documentos HTML incluyan sólo información y datos, relativos al significado de la información a transmitir (*el contenido*), y todos los aspectos relacionados con el estilo (diseño, colores, formas, etc...) se encuentren en un documento CSS independiente (*la presentación*).



De esta forma, se puede unificar todo lo relativo al diseño visual en **un solo documento CSS**, y con ello, varias ventajas:

- Si necesitamos hacer modificaciones de presentación lo hacemos en un sólo lugar y no tenemos que editar todos los documentos HTML por separado.
- Se reduce la duplicación de estilos en diferentes lugares, por lo que la información a transmitir es considerablemente menor (las páginas se descargan más rápido).
- Es más fácil crear versiones diferentes de presentación para otros tipos de dispositivos: tablets, smartphones o dispositivos móviles, etc...

¿Cómo usar CSS?

Antes de comenzar a trabajar con **CSS** hay que conocer las diferentes formas para incluir estilos en nuestros **documentos HTML**, ya que hay varias, cada una con sus particularidades y diferencias.

En principio, tenemos **tres** formas diferentes de hacerlo, siendo la primera la más común y la última la menos habitual:

Nombre	Método	Descripción
CSS Externo	Etiqueta <code><link></code>	El código se escribe en un archivo <code>.css</code> a parte. Método más habitual.
CSS Interno	Etiqueta <code><style></code>	El código se escribe en una etiqueta <code><style></code> en el documento HTML.
Estilos en línea	Atributo <code>style="..."</code>	El código se escribe en un atributo HTML de una etiqueta.

Veamos cada una de ellas detalladamente:

Enlace a CSS externo (link)

En la cabecera de nuestro documento HTML, más concretamente en el bloque `<head></head>`, podemos incluir una etiqueta `<link>` con la que establecemos una relación entre el documento actual y el archivo CSS que indicamos en el atributo `href`:

```
<link rel="stylesheet" href="index.css" />
```

De esta forma, los navegadores sabrán que deben aplicar los estilos que se encuentren en el archivo `index.css`. Se aconseja escribir esta línea lo antes posible (*sobre todo, antes de los scripts*), obligando así al navegador a aplicar los estilos cuanto antes y eliminar la **falsa percepción visual** de que la página está en blanco y no ha sido cargada por completo.

■ Esta es la manera recomendada de utilizar **estilos CSS** en nuestros documentos.

Incluir CSS en el HTML(style)

Otra de las formas habituales que existen para incluir estilos CSS en nuestra página es la de añadirlos directamente en el documento HTML, a través de una etiqueta `<style>` que contendrá el código CSS:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título de la página</title>
    <style>
      div {
        background: hotpink;
        color: white;
      }
    </style>
  </head>
  ...
</html>
```

Este sistema puede servirnos en ciertos casos particulares, pero hay que darle prioridad al método anterior (*CSS externo*), ya que incluyendo el código CSS en el interior del archivo HTML arruinamos la posibilidad de tener el código CSS en un documento a parte, pudiendo **reutilizarlo y enlazarlo** desde otros documentos HTML mediante la etiqueta `<link>`.

■ Aunque no es obligatorio, es muy común que las etiquetas `<style>` se encuentren en la cabecera `<head>` del documento HTML, ya que antiguamente era la única forma de hacerlo.

Estilos en línea (atributo style)

Por último, la tercera forma de aplicar estilos en un documento HTML es hacerlo directamente, a través del atributo `style` de la propia etiqueta donde queramos aplicar el estilo:

```
<p>¡Hola <span style="color:red">amigo lector</span>!</p>
```

De la misma forma que en el método anterior, se recomienda no utilizarse salvo casos muy específicos, ya que los estilos se asocian a la etiqueta en cuestión y no pueden reutilizarse. Sin embargo, es una opción que puede venir bien en algunos casos.

Consejo: Si quieras comenzar a hacer pruebas rápidas con HTML, CSS y Javascript puedes utilizar [CodePen](#), una plataforma web que te permite crear contenido HTML, CSS y Javascript, previsualizando al vuelo el resultado del documento final, sin necesidad editores.

Estructura de CSS

Al igual que los documentos HTML, los documentos CSS son archivos de texto donde se escribe una serie de órdenes y el cliente (navegador) las interpreta y aplica a los documentos HTML asociados.

La **estructura CSS** se basa en reglas que tienen el siguiente formato:



- **Selector:** El selector es el elemento HTML que vamos a seleccionar del documento para aplicarle un estilo concreto. *Por ejemplo, el elemento p.* Realmente, esto es mucho más complejo, pero ya dedicaremos una serie de capítulos exclusivamente a este tema.
- **Propiedad:** La propiedad es una de las diferentes características que brinda el lenguaje CSS e iremos aprendiendo.
- **Valor:** Cada propiedad CSS tiene una serie de valores concretos, con los que tendrá uno u otro comportamiento.

Con todo esto le iremos indicamos al navegador que, para cada etiqueta (selector especificado) debe aplicar las reglas (propiedad y valor) indicadas.

Vamos a verlo con un ejemplo para afianzar conceptos. Supongamos que este es el código HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Título de página</title>
    <link rel="stylesheet" type="text/css" href="index.css" />
</head>
<body>
    <div id="first">
        <p>Párrafo</p>
    </div>
    <div id="second">
        <span>Capa</span>
    </div>
</body>
</html>
```

Y además, por otro lado, este sería el código CSS del archivo `index.css`:

```
p {
    color:red;          /* Color de texto rojo */
}
```

De esta forma, a todas las etiquetas `<p>` se le aplicará el estilo especificado: **el color rojo**.

Truco: Se pueden incluir comentarios entre los caracteres `/*` y `*/`, los cuales serán ignorados por el navegador y pueden ser utilizados por legibilidad y para documentar nuestros documentos CSS.

Sin embargo, esto es sólo un ejemplo muy sencillo. Se pueden aplicar muchas más reglas (*no sólo el color del ejemplo*), consiguiendo así un conjunto de estilos para la etiqueta indicada en el selector. Cada una de estas reglas se terminará con el carácter **punto y coma** (`;`).

En el siguiente esquema se puede ver las diferentes partes del código CSS con sus respectivos nombres:



Truco: El último `;` de un selector (*en naranja*) no es obligatorio y se puede omitir.

Además, también se pueden especificar agrupaciones de etiquetas, clases de etiquetas o cosas más complejas, pero eso lo veremos más adelante. De momento, vamos a centrarnos en las diferentes reglas que podemos utilizar.

Un buen consejo, para hacer más legible nuestro código CSS, es utilizar la siguiente estructura visual (*indentar el código mediante espacios, con una propiedad por línea*). Es una buena práctica, indispensable a la larga, que nos facilitará la lectura del código:

```
selector {  
    propiedad : valor ;  
    propiedad : valor  
}
```

Esto mejora sustancialmente la legibilidad del código y se considera un convenio a utilizar para evitar la complejidad de entender el código que no se encuentre correctamente indentado.

Más adelante, en un capítulo dedicado expresamente a ello, veremos que la estructura CSS puede ser más compleja, pero de momento trabajaremos con el esquema simplificado.

Minificar CSS

Normalmente, cuando el desarrollador escribe **código** (*y no sólo CSS, sino también HTML o Javascript*), lucha en todo momento con varios factores clave, que podríamos delimitar en los siguientes:

- **Funcionamiento:** El código debe estar bien escrito para funcionar correctamente. Además, se debe garantizar el correcto funcionamiento en diferentes navegadores, diferentes sistemas operativos (*incluyendo dispositivos de escritorio, móviles y/o tablets*), los cuales suelen/pueden tener algunas diferencias entre ellos.
- **Legibilidad:** Correcta indentación o colocación de las diferentes partes del código, facilitando la legibilidad por humanos, favoreciendo la velocidad de modificación e introducción de cambios (*mantenibilidad*).
- **Tamaño:** Cuanto más texto tenga un archivo **CSS, HTML o Javascript** (*espacios, líneas en blanco, comentarios, código no usado...*), más grande será el tamaño final del archivo, por lo que más tiempo tardará en descargarse y procesarse por el navegador.

- **Rendimiento:** Cuanto más costosas sean las operaciones que vamos a obligar a hacer al navegador, más tardará en procesarse y por lo tanto, más tiempo tardará en pintar y renderizar (*dibujar*) la página.

¿Qué es la minificación?

La **minificación** (*en inglés, Uglify o Minification*) es la acción de eliminar caracteres 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 **buenas prácticas** 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 automáticas. 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

```
#main {  
    background-color: black;  
    color: white;  
    padding: 16px;  
    border: 2px solid blue;  
}
```

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

```
#main{background-color:#000;color:#fff;padding:16px;border:2px solid #00f;}
```

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

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	PostCSS	Para automatizar desde terminal o desde PostCSS.
Clean CSS	NodeJS/NPM	Para automatizar desde terminal.
CSS Compressor	Online	Opciones variadas: grado de compresión, optimizaciones...
CSSO	NodeJS/NPM	Optimizador de CSS (clean, compress and restructuring)
Squish	NodeJS/NPM	Compresor de CSS basado en Node
YUI Compressor	Java	Compresor CSS histórico de Yahoo

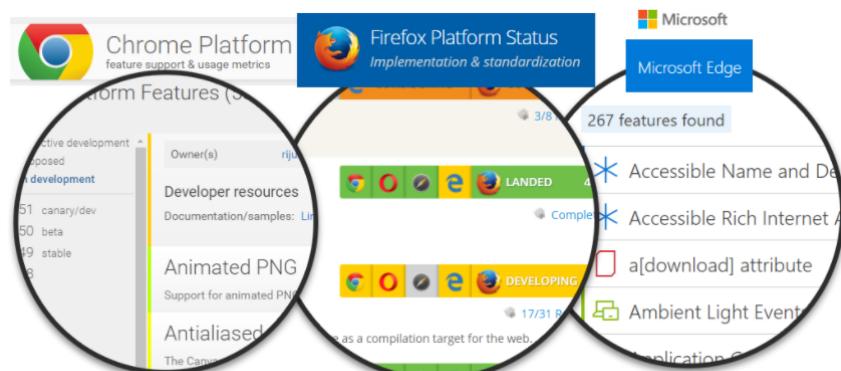
Los navegadores web

Los **navegadores web** (también *llamados clientes*) son esos programas que utilizamos para acceder a Internet y visualizar páginas en nuestros dispositivos. Todos los usuarios conocen al menos uno o varios navegadores web, aunque sea los más populares como **Google Chrome** o **Mozilla Firefox**. Sin embargo, existen muchos más. Para ser un buen diseñador o desarrollador web es recomendable conocer bien el ecosistema de navegadores existente y sus características, que no son pocas.

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:

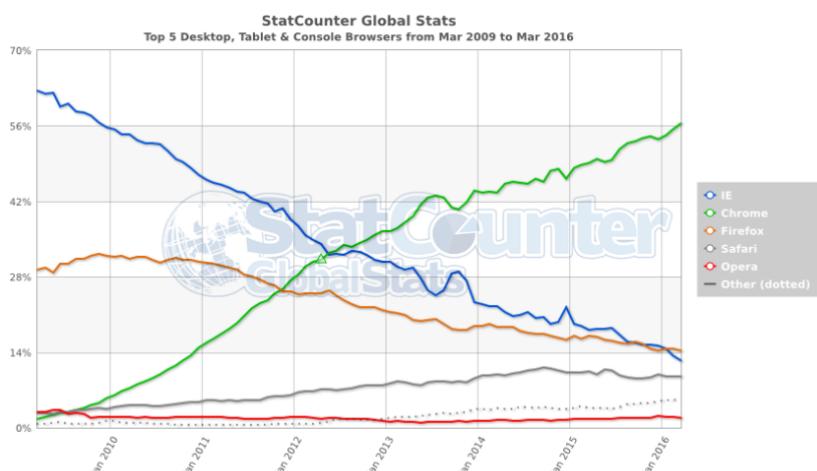
Responsables	Producto	Página de estado de desarrollo del navegador
Microsoft	Navegador Microsoft Edge	Edge Platform Status
(Proyecto open source)	Motor Webkit	Webkit Feature Status
Fundación Mozilla	Navegador Mozilla Firefox	Firefox Platform Status
Google	Navegador Google Chrome	Chrome Platform Status

Además, existe una sección donde podemos [comparar características respecto a navegadores en CanIUse](#).

Historia de los navegadores

Si echamos un vistazo atrás, la historia de los navegadores ha variado muchísimo. Quizás, el cambio más importante en los últimos 10 años ha sido el reemplazo de **Internet Explorer**, como navegador más popular, a **Google Chrome**. Antiguamente, **Internet Explorer** fue un navegador que se había estancado y no implementaba nuevas características y funcionalidades, al contrario que sus competidores. Pero además, para empeorar la situación, era el navegador más utilizado por los usuarios, debido al liderazgo de Windows como sistema operativo. Esto impedía que las nuevas tecnologías webs se adoptaran y frenaba su avance. Por suerte, esto ha ido cambiando a lo largo de los años y la situación hoy en día es bastante diferente.

A continuación se puede ver la evolución de los navegadores más populares durante esta última década (*desde 2009 hasta 2016*). Vemos que los navegadores más perjudicados son Internet Explorer y Mozilla Firefox, mientras que Chrome ha experimentado un incremento muy grande. Safari también ha experimentado un ligero incremento, probablemente debido al éxito de dispositivos como iPhone o iPad.



Como toda estadística, debe ser tomada con precaución porque existen sesgos en sus datos. Esta estadística ha sido extraída de [Global StatCounter](#). También puedes echar un vistazo a algunas estadísticas más en [w3counter](#), aunque quizás la más interesante y adecuada sea [CanIUse: Usage table](#), donde podemos encontrar los navegadores más utilizados, separado por versiones y mostrado con porcentajes.

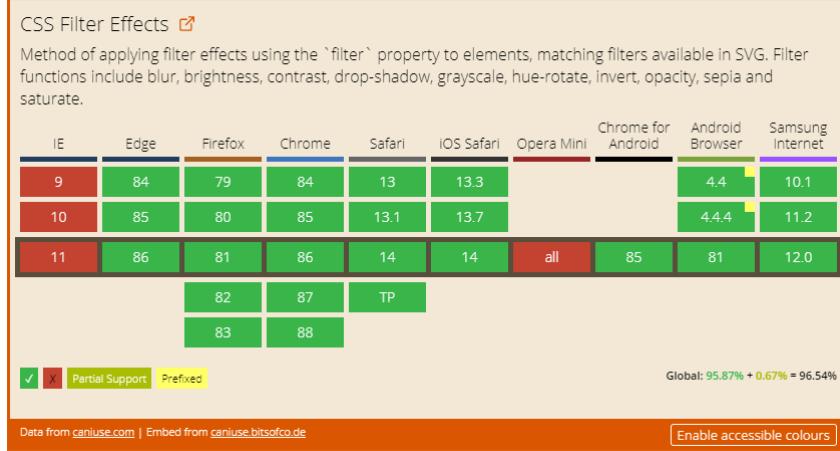
Navegadores actuales

A continuación, tenemos una lista de la rama de los **5 navegadores más populares**, con 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
Google	Chrome beta	Early-adopters	Blink		URL
Google	Canary Chrome	Desarrollador	Blink		URL
Google	Chromium	Open Source	Blink		URL
Mozilla	Firefox	Uso habitual	Quantum		URL
Mozilla	Firefox beta	Early-adopters	Quantum		URL
Mozilla	Firefox Dev Edition	Desarrollador	Quantum		URL
Mozilla	Firefox Nightly	Desarrollador	Quantum		URL
Microsoft	Edge	Uso habitual	Blink		URL
Microsoft	Internet Explorer	Uso habitual	Trident	Sólo Windows	URL
Opera	Opera	Uso habitual	Blink		URL
Opera	Opera beta	Early-adopters	Blink		URL
Opera	Opera developer	Desarrollador	Blink		URL
Opera	Opera Neon	Early-adopters	Blink		URL
Apple	Safari	Uso habitual	Webkit	Sólo Apple	URL
Apple	Safari Tech Preview	Desarrollador	Webkit	Sólo Apple	URL

Versiones de los navegadores

Es también muy importante conocer la versión del navegador que utiliza la mayoría de nuestro público (*datos que se pueden obtener con herramientas como Google Analytics, por ejemplo*), ya que de una versión a otra se añaden nuevas características y funcionalidades, de las cuales los usuarios de las versiones anteriores no podrán disfrutar. En la herramienta [CanIUse](#) se muestra, a lo largo de las columnas de cada navegador, si las funcionalidades están implementadas en la versión concreta del mismo, o cuando empezarán a funcionar:



Es por tanto, lógico pensar, que si tenemos un alto porcentaje de usuarios que utilizan una versión de un navegador que no soporta la funcionalidad que queremos utilizar, haya que buscar alternativas o abstenerse a utilizarla hasta que ese porcentaje se reduzca.

Por suerte, desde hace ya bastante tiempo los navegadores han comenzado a implementar una estrategia de **actualización silenciosa** (*en inglés los denominan evergreen browser*), con la cuál consigues que el usuario tenga siempre el navegador actualizado a la última versión (*si tiene acceso a Internet, claro*), ya que el grueso de los usuarios no suele actualizar manualmente la versión de su navegador, y esto hacía que existiera una gran cuota de usuarios con navegadores sin actualizar.

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	Observaciones	URL
Tor Project	Tor	Navegación anónima	Gecko	Basado en Firefox	URL
Vivaldi Tech	Vivaldi	Early-adopters	Blink		URL
Maxthon Int	Maxthon	Uso habitual	Trident/Webkit		URL
Yandex	Yandex Browser	Uso habitual	Blink		URL
Fenrir Inc	Sleipnir	Uso habitual	Blink		URL
SM Project	SeaMonkey	Uso habitual	Gecko	Basado en Mozilla AS	URL
Chris Dywan	Midori	Uso habitual	Webkit		URL
David Rosca	QupZilla	Uso habitual	Qt WebEngine		URL
Comodo Group	Comodo Dragon	Uso habitual	Blink	Basado en Chromium	URL
Avant Force	Avant	Uso habitual	Trident/Gecko		URL
Brave Soft	Brave	Uso habitual	Blink		URL
M. Patocka	Links	Navegador de texto	-		URL
Thomas Dickey	Lynx	Navegador de texto	-		URL
Akinori Ito	w3m	Navegador de texto	-		URL
P. Baudis	Elinks	Navegador de texto	-		URL

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

Niveles de CSS y prefijos

El **lenguaje CSS** es una especificación desarrollada y mantenida por el World Wide Web Consortium (W3C), una comunidad internacional que se encarga de desarrollar estándares para asegurar el crecimiento y la neutralidad de la web, independizándolo de tecnologías propietarias e intentando aunar esfuerzos para satisfacer la demanda de características útiles e interesantes.

En el consorcio participan y colaboran prácticamente casi todas las empresas relacionadas con Internet, como por ejemplo Apple, Adobe, Akamai, Cisco, Google, Facebook, HP, Intel, LG, Microsoft, Nokia, Twitter, Yahoo, entre [muchos otros](#).

Desde hace algunos años, la comunidad [WHATWG](#) participa también en el desarrollo y evolución de especificaciones como las de [HTML](#), [DOM](#) u otras tecnologías relacionadas.

Niveles CSS

A lo largo de su historia, **CSS** ha 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](#) en la página web del consorcio W3C.

Prefijos

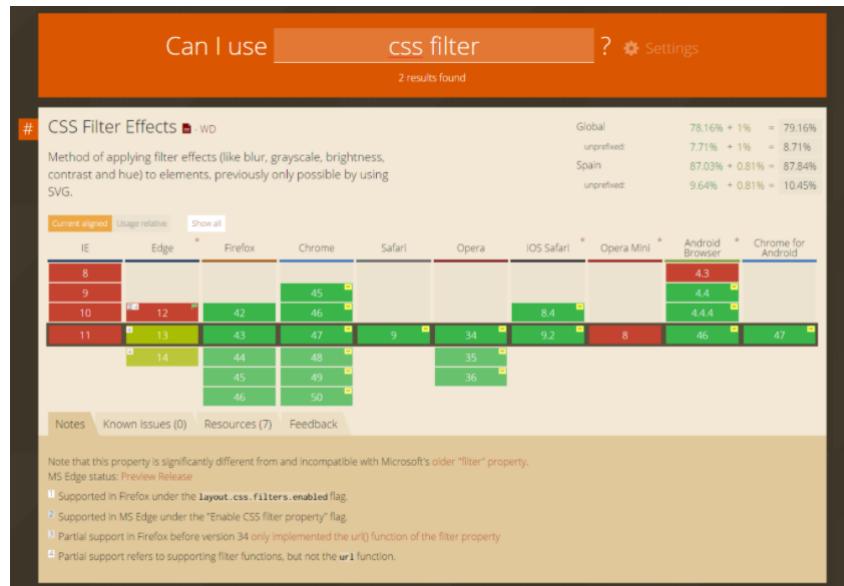
Algunas de las propiedades que veremos no están definidas por completo, sólo son borradores o pueden variar en la especificación definitiva, por lo que los navegadores las implementan utilizando una serie de **vendor prefixes** (*prefijos por navegador*), que facilitan la segmentación de funcionalidades.

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:

```
div {  
    transform: ... /* Navegadores que implementan especificación oficial */  
    -webkit-transform: ... /* Versiones antiguas de Chrome (Motor WebKit) */  
    -moz-transform: ... /* Versiones antiguas de Firefox (Motor Gecko) */  
    -ms-transform: ... /* Versiones antiguas de IE (Motor Trident) */  
    -o-transform: ... /* Versiones antiguas de Opera (Motor Presto) */  
}
```

En el ejemplo anterior, la propiedad `transform` se refiere a los navegadores que tengan implementada la especificación definitiva por completo e ignorará el resto de propiedades. Por otro lado, otro navegador (*o el mismo en una versión más antigua*) puede tener implementada una versión anterior a la definitiva, por lo que hará caso a las propiedades con un prefijo concreto.

Debido al ritmo y la rápida velocidad de Internet en estas cuestiones, es muy complicado obtener una lista de funcionalidades implementadas en cada navegador, algo que puede variar incluso en cuestión de semanas. Aconsejo utilizar la página [Can I Use](#), una web colaborativa para saber el estado actual, previo e incluso futuro de las propiedades CSS o elementos HTML en cada navegador.



En esta página se puede buscar, a través de un buscador y de forma rápida y cómoda, el estado de ciertas características por parte de las diferentes versiones de los navegadores.

Actualmente, los **vendor prefixes** están en proceso de desaparecer. Las principales compañías de navegadores han optado por favorecer el uso de flags en el navegador del usuario para activar o desactivar opciones experimentales o crear especificaciones más pequeñas y breves que puedan ser estables mucho más rápido. Por esta razón, se aconseja utilizar vendor prefixes solo cuando necesitas soporte específico en navegadores muy antiguos.

En el caso de querer utilizar *vendor prefixes*, recomiendo encarecidamente utilizar sistemas como [autoprefixer](#) (*el más popular, que forma parte de PostCSS*) o [prefix-free](#) que añaden de forma automática y transparente los prefijos, basándose en información de herramientas como [Can I Use](#). Busca extensiones en el **editor** que utilices o la opción para activarlas, ya que te ahorrará mucho tiempo y te permitirá tener un código más legible y modular al no tener que repetir código.

Herencia en CSS

Otro detalle que hay que dejar claro antes de empezar es el concepto de **herencia** y el concepto de **cascada**, pues son los que más problemas suelen dar y los que, sin lugar a dudas, mayor frustración acarrean cuando no comprendemos lo que está pasando.

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:

```
body {  
    color:green; /* color de texto verde */  
}
```

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:

```
body {  
    border-width: 2px;  
    border-style: solid;  
    border-color: red;  
}
```

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

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, de su valor inicial.

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

```

body {
  border-width: 2px;
  border-style: solid;
  border-color: red;
}

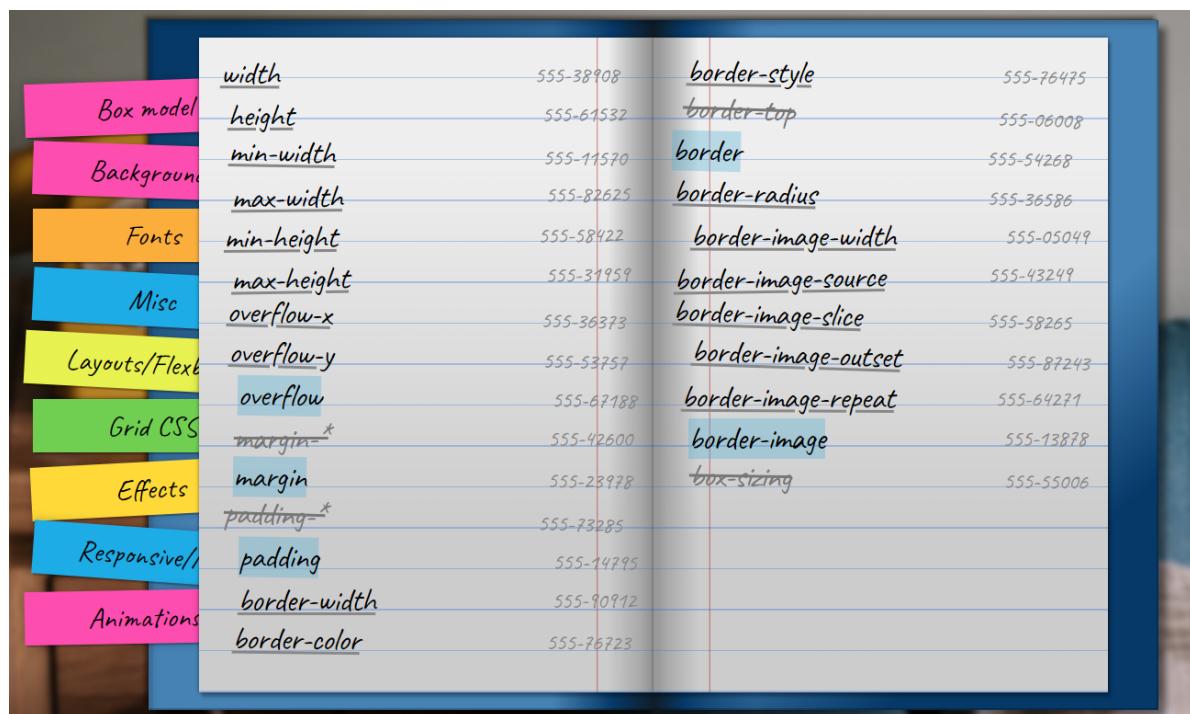
div {
  border: inherit;
}

```

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**.

Guía interactiva de propiedades

<https://lenguajecss.com/css/introduccion/guia-css/>



2. Modelo de cajas

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 CSS

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
<code>in</code>	Pulgadas	1in = 25.4mm
<code>cm</code>	Centímetros	1cm = 10mm
<code>pc</code>	Picas	1pc = 4.23mm
<code>mm</code>	Milímetros	1mm = 1mm
<code>pt</code>	Puntos	1pt = 0.35mm
<code>px</code>	Píxels	1px = 0.26mm
<code>Q</code>	Cuarto de mm	1Q = 0.248mm

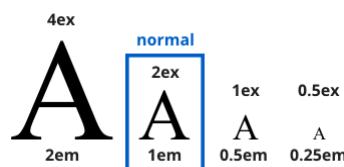
Consejo: El punto (`pt`) es una medida que puede utilizarse para documentos CSS en los que se fija el tamaño de las fuentes en medios impresos.

Unidades relativas

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
<code>em</code>	«M»	1em = tamaño de fuente establecida en navegador
<code>ex</code>	«X» (~0.5em)	1ex = ~ mitad del tamaño de fuente del navegador
<code>ch</code>	«zero width»	1ch = tamaño de ancho del cero (0)
<code>rem</code>	«root M»	1rem = tamaño fuente raíz
<code>%</code>	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` estableceremos 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`*):

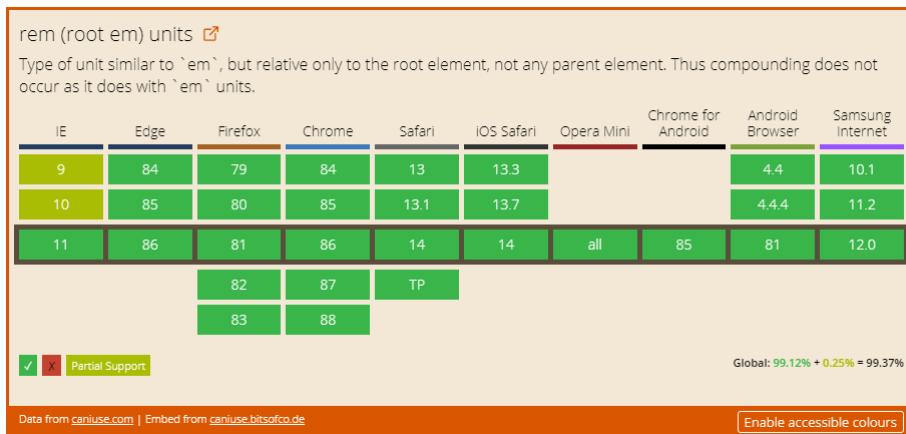
```
body {
    font-size: 22px;      /* Tamaño general */
}

h1 {
    font-size: 2rem;     /* El doble del tamaño general: 44px */
}

h2 {
    font-size: 1rem;     /* El mismo tamaño general: 22px */
}
```

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 píxeles** 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 píxeles**.

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.



En general, en diseño web, se recomienda utilizar **unidades relativas** siempre que sea posible, ya que son unidades mucho más flexibles.

Truco: Cuando se especifican valores de unidades iguales a `0`, como por ejemplo `0px`, `0em` o `0%`, podemos omitir las unidades y escribir simplemente `0`, ya que en este caso particular las unidades son redundantes y no aportan valor.

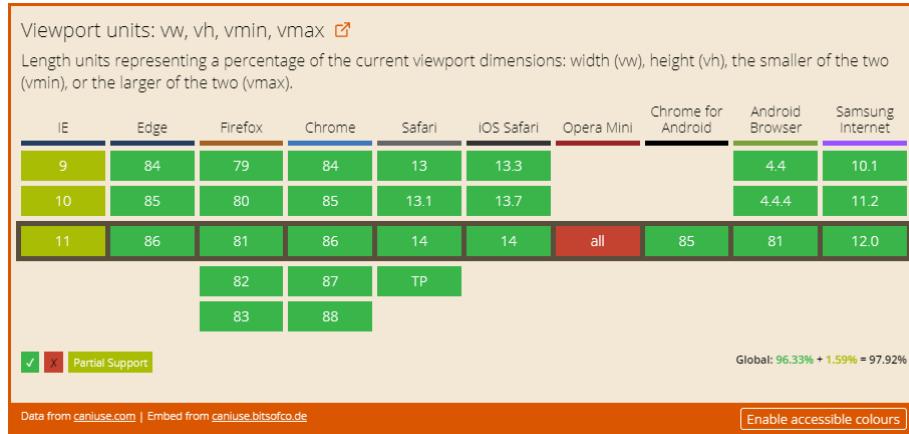
Unidades flexibles (viewport)

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
<code>vw</code>	viewport width	$1vw = 1\%$ ancho de navegador
<code>vh</code>	viewport height	$1vh = 1\%$ alto de navegador
<code>vmin</code>	viewport minimum	$1vmin = 1\%$ de alto o ancho (el mínimo)
<code>vmax</code>	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 quieras flexibilidad con diseños adaptables.



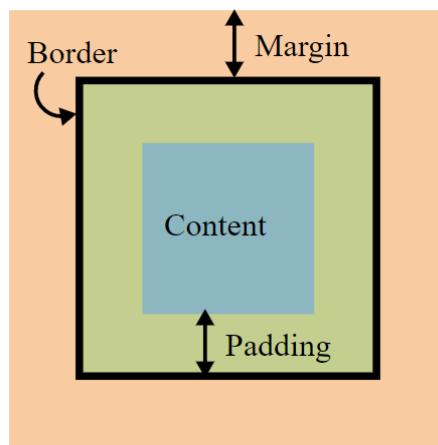
Es interesante tener en cuenta que existen una serie de funciones para hacer cálculos con unidades CSS. Son las funciones `calc()`, `min()`, `max()` o `clamp()`, entre otras. Las veremos más adelante, en el capítulo de [funciones CSS](#).

Modelo de cajas

Durante varios años, el denominado **modelo de cajas** fue una pesadilla para los desarrolladores web, puesto que se mostraba visualmente de forma diferente en **Internet Explorer** respecto a los demás navegadores. Por fortuna, todos los navegadores actuales ya interpretan de la misma forma el modelo de cajas, pero conviene aprender bien la diferencia para no ser como Internet Explorer.

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** (*content*). En azul, es la parte interior del elemento, excluyendo el relleno.



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
<code>width</code>	<code>auto</code>	Tamaño de ancho de un elemento.
<code>height</code>	<code>auto</code>	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. Esto es algo que también puede variar, dependiendo del tipo de elemento que estemos usando, y que veremos más adelante, en el apartado de maquetación.

Hay que ser muy conscientes de que, sin indicar valores de ancho y alto para la caja, el elemento generalmente toma el tamaño que debe respecto a su contenido, 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.

```
div {
  width: 800px;
  height: 400px;
  background: red;
  max-width: 500px;
}
```

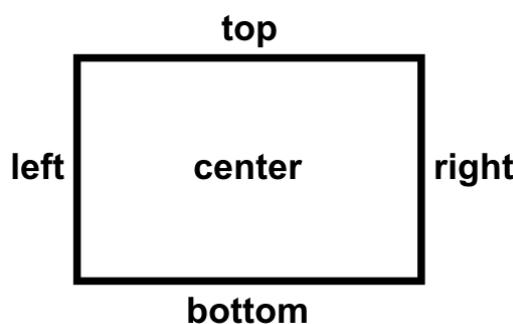
En este caso, por ejemplo, a pesar de estar indicando un tamaño de `800px`, le aplicamos un `max-width` de `500px`, por lo que estamos limitando el elemento a un tamaño de ancho de 500 píxeles como máximo y nunca superará ese tamaño.

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
<code>max-width</code>	<code>none</code> <code>0</code>	Ancho máximo que puede ocupar un elemento.
<code>min-width</code>	<code>0</code>	Ancho mínimo que puede ocupar un elemento.
<code>max-height</code>	<code>none</code> <code>0</code>	Alto máximo que puede ocupar un elemento.
<code>min-height</code>	<code>0</code>	Alto mínimo que puede ocupar un elemento.

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:



- **Top:** Se refiere a la parte superior del elemento.
- **Left:** Se refiere a la parte izquierda del elemento.
- **Right:** Se refiere a la parte derecha del elemento.
- **Bottom:** Se refiere a la parte inferior del elemento.
- **Center:** En algunos casos se puede especificar el valor `center` para referirse a la posición central entre los extremos horizontales o verticales.

Estas palabras clave las utilizaremos muy a menudo en diferentes propiedades CSS para hacer referencia a una zona particular.

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.

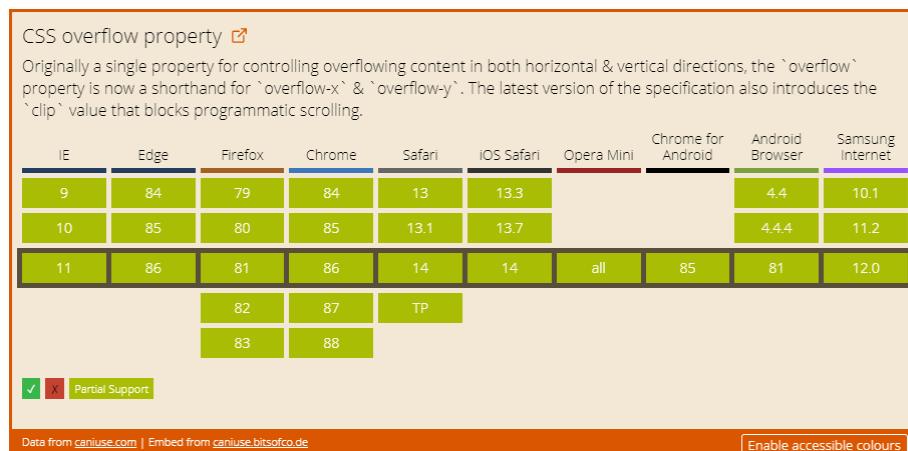
Podemos modificar ese comportamiento con la propiedad de CSS `overflow`, o con alguna de sus propiedades específicas `overflow-x` o `overflow-y`:

Propiedad	Valor	Significado
<code>overflow</code>	<code>visible</code> <code>hidden</code> <code>scroll</code> <code>auto</code>	Establece el comportamiento de desbordamiento.
<code>overflow-x</code>	<code>visible</code> <code>hidden</code> <code>scroll</code> <code>auto</code>	Establece el desbordamiento sólo para el eje X (<i>horizontal</i>).
<code>overflow-y</code>	<code>visible</code> <code>hidden</code> <code>scroll</code> <code>auto</code>	Establece el desbordamiento sólo para el eje Y (<i>vertical</i>).

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 (<i>comportamiento por defecto</i>)	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 quieras mostrar alguna barra de desplazamiento, habitualmente, la barra de desplazamiento horizontal.



Márgenes y rellenos

En el modelo de cajas, los **márgenes** (*margin*) son los espacios exteriores de un elemento. El espacio que hay entre el borde de un elemento y el borde de otros elementos adyacentes, es lo que se considera margen.

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:

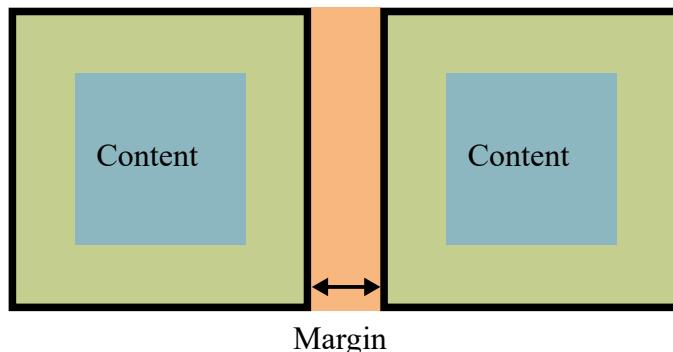
Propiedad	Valor	Significado
<code>margin-top</code>	<code>auto</code> <code>size</code>	Establece un tamaño de margen superior.
<code>margin-left</code>	<code>auto</code> <code>size</code>	Establece un tamaño de margen a la izquierda.
<code>margin-right</code>	<code>auto</code> <code>size</code>	Establece un tamaño de margen a la derecha.
<code>margin-bottom</code>	<code>auto</code> <code>size</code>	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 navegador 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

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

Propiedad	Valor	Significado
<code>padding-top</code>	<code>0</code> <code>size</code>	Aplica un relleno interior en el espacio superior de un elemento.
<code>padding-left</code>	<code>0</code> <code>size</code>	Aplica un relleno interior en el espacio izquierdo de un elemento.
<code>padding-right</code>	<code>0</code> <code>size</code>	Aplica un relleno interior en el espacio derecho de un elemento.
<code>padding-bottom</code>	<code>0</code> <code>size</code>	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

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

Propiedad	Valores	Significado
<code>margin</code> <code>1234</code>	<code>size</code>	1 parámetro. Aplica el mismo margen a todos los lados.
	<code>size size</code>	2 parámetros. Aplica margen top/bottom y left/right .
	<code>size size size</code>	3 parámetros. Aplica margen top, left/right y bottom .
	<code>size size size</code> <code>size</code>	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
<code>border-color</code> <code>1234</code>	<code>COLOR</code>	Especifica el color que se utilizará en el borde.
<code>border-width</code> <code>1234</code>	<code>thin medium thick</code> <code>size</code>	Especifica un tamaño predefinido para el grosor del borde.
<code>border-style</code> <code>1234</code>	<code>none style</code>	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

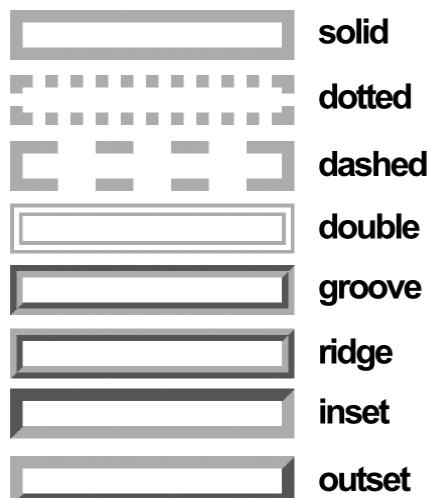
Por último, con `border-style` podemos aplicar un estilo determinado al borde de un elemento. En **estilo de borde** podemos elegir cualquiera de las siguientes opciones:

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 discontinua).
solid	Establece un borde sólido (línea continua).
double	Establece un borde doble (dos líneas continuas).
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 .

Veamos un ejemplo sencillo:

```
div {  
    border-color: gray;  
    border-width: 1px;  
    border-style: dotted;  
}
```

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 píxeles** de grosor y color **gris**:



Bordes múltiples (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
<code>border-color</code> 1234	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). Teniendo en cuenta esto, disponemos de mucha flexibilidad a la hora de especificar esquemas de bordes más complejos:

```
div {  
    border-color: red blue green;  
    border-width: 2px 10px 5px;  
    border-style: solid dotted solid;  
}
```

En el ejemplo anterior hemos utilizado 3 parámetros, indicando un elemento con borde superior rojo sólido de 2 píxeles de grosor, con borde izquierdo y derecho punteado azul de 10 píxeles de grosor y con un borde inferior verde sólido de 5 píxeles de grosor.

Atajo: Bordes

Pero ya habremos visto que con tantas propiedades, para hacer algo relativamente sencillo, nos pueden quedar varias líneas 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
<code>border</code>	SIZE STYLE COLOR	Propiedad de atajo para simplificar valores.

Por ejemplo:

```
div {  
    border: 1px solid #000000;  
}
```

Así pues, estamos aplicando un borde de **1 píxel** de grosor, estilo **sólido** y color **negro** a todos los bordes del elemento, ahorrando mucho espacio y escribiéndolo todo en una sola propiedad.

Consejo: Intenta organizarte y aplicar siempre los atajos si es posible. Ahorrarás mucho espacio en el documento y simplificarás la creación de diseños. El orden, aunque no es obligatorio, si es recomendable para mantener una cierta coherencia con el estilo de código.

Bordes específicos

Otra forma, quizás más intuitiva, es la de utilizar las propiedades de bordes específicos (*por zonas*) y aplicar estilos combinándolos junto a la [herencia de CSS](#). Para utilizarlas bastaría con indicarle la zona justo después de `border`:

```
div {  
    border-bottom-width: 2px;  
    border-bottom-style: dotted;  
    border-bottom-color: black;  
}
```

Esto dibujaría **sólo un borde inferior** negro de 2 píxeles de grosor y con estilo punteado. Ahora imaginemos que queremos un elemento con todos los bordes en rojo a 5 píxeles de grosor, salvo el borde superior, que lo queremos con un borde de 15 píxeles en color naranja. Podríamos hacer lo siguiente:

```
div {  
    border: 5px solid red;  
    border-top-width: 15px;  
    border-top-color: orange;  
    border-top-style: solid; /* Esta propiedad no es necesaria (se hereda) */  
}
```

El ejemplo anterior conseguiría nuestro objetivo. La primera propiedad establece todos los bordes del elemento, sin embargo, las siguientes propiedades modifican sólo el borde superior, cambiándolo a las características indicadas.

Recuerda que también existen atajos para estas propiedades de bordes en zonas concretas, lo que nos permite simplificar aún más el ejemplo anterior, haciéndolo más fácil de comprender:

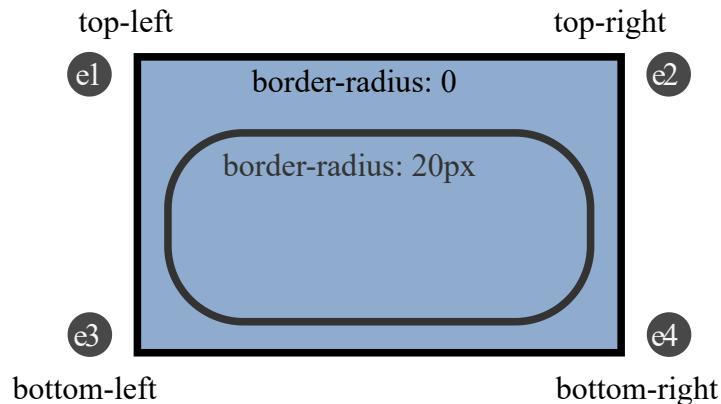
```
div {  
    border: 5px solid red;  
    border-top: 15px solid orange;  
}
```

Ojo: Es muy importante entender como se está aplicando la herencia en los ejemplos anteriores, puesto que es una de las características más complejas de dominar de CSS junto a la cascada. Por ejemplo, si colocáramos el `border-top` antes del `border`, este último sobrescribiría los valores de `border-top` y no funcionaría de la misma forma.

Esquinas redondeadas

CSS3 añade interesantes características en materia de bordes, como la posibilidad de crear **bordes con esquinas redondeadas**, característica que en versiones anteriores de CSS era muy complicado de lograr, necesitando recurrir al apoyo de imágenes gráficas. Por su parte, en **CSS3** es realmente sencillo hacerlo mediante código.

Basta utilizar la propiedad `border-radius`, con la cual podrás especificar un radio para el borde de las esquinas. Por defecto, este borde es de **tamaño 0**, por lo que no hay borde redondeado. A medida que se aumenta este valor, el borde se redondea más. Una vez llegado a su máximo, no se apreciará ningún cambio.



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

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

El **primer formato**, un único parámetro, aplica ese tamaño a **todas** las esquinas del borde.

El **segundo formato**, con dos parámetros, aplica el primer valor, **e1**, a las esquinas superior-izquierda e inferior-derecha, y el segundo valor, **e2**, a las esquinas superior-derecha e inferior-izquierda.

En el **tercer formato**, se aplica el parámetro **e1** a la esquina superior-izquierda, el parámetro **e2** a las esquinas superior-derecha e inferior-izquierda y el parámetro **e3** a la esquina inferior-derecha.

Y por último, en el **cuarto formato**, se aplica el tamaño de cada valor a cada esquina por separado, en el sentido de las agujas del reloj. O lo que es lo mismo, **e1** a la esquina superior-izquierda, **e2** a la esquina superior-derecha, **e3** a la esquina inferior-derecha y **e4** a la esquina inferior-izquierda.

A modo de ejemplo teórico, pueden ver un ejemplo de la aplicación de varios formatos:

```
div {
    border-radius: 25px;           /* Formato con un parámetro */
    border-radius: 25% 50%;       /* Formato con dos parámetros */
    border-radius: 50px 25px 10px; /* Formato con tres parámetros */
    border-radius: 25px 0 15px 50px; /* Formato con cuatro parámetros */
}
```

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.

```
div {  
    /* Usando el segundo formato */  
    border-radius: 5px 50px / 50px 15px;  
}
```

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
<code>border-top-left-radius</code>	<code>SIZE</code>	Indica un radio para redondear la esquina top-left .
<code>border-top-right-radius</code>	<code>SIZE</code>	Indica un radio para redondear la esquina top-right .
<code>border-bottom-left-radius</code>	<code>SIZE</code>	Indica un radio para redondear la esquina bottom-left .
<code>border-bottom-right-radius</code>	<code>SIZE</code>	Indica un radio para redondear la esquina bottom-right .

Estas propiedades son ideales para aplicar junto a la herencia de CSS y sobreescribir valores específicos.

3. 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
<code>color</code>	<code>COLOR</code>	Cambia el color del texto que está en el interior de un elemento.
<code>background-color</code>	<code>COLOR</code>	Cambia el color de fondo de un elemento.

La propiedad `color` establece el color del texto, mientras que la propiedad `background-color` establece el color de fondo del elemento.

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

Nombre	Formato	Ejemplo
Palabra clave predefinida	<i>[palabra clave]</i>	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 canal alfa	#RGBBA	#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)

A continuación iremos explicando cada uno de estos formatos para entender como se especifican los colores en CSS y utilizar el método que más se adapte a nuestras necesidades.

Consejo: Si lo que buscamos es un sistema para extraer colores (*eye dropper*) de una página web, podemos utilizar la extensión para Chrome de [ColorZilla](#) o el propio [Chrome Developer Tools](#), que integra dicha funcionalidad.

Palabras clave de color

El primer caso (*y más limitado*) permite establecer el color utilizando palabras reservadas de colores, como **red**, **blue**, **orange**, **white**, **navy**, **yellow** u otras. Existen más de 140 palabras clave para indicar colores:

black	navy	darkblue	mediumblue
blue	darkgreen	green	teal
darkcyan	deepskyblue	darkturquoise	mediumspringgreen
lime	springgreen	aqua	cyan
midnightblue	dodgerblue	lightseagreen	forestgreen
seagreen	darkslategray	darkslategrey	limegreen
mediumseagreen	turquoise	royalblue	steelblue
darkslateblue	mediumturquoise	indigo	darkolivegreen
cadetblue	cornflowerblue	mediumaquamarine	dimgray
dimgray	slateblue	olivedrab	slategray
slategrey	lightslategray	lightslategrey	mediumslateblue
lawngreen	chartreuse	aquamarine	maroon
purple	olive	gray	grey
skyblue	lightskyblue	blueviolet	darkred
darkmagenta	saddlebrown	darkseagreen	lightgreen
mediumpurple	darkviolet	rebeccapurple	palegreen
darkorchid	yellowgreen	sienna	brown
darkgray	darkgrey	lightblue	greenyellow
paleturquoise	lightsteelblue	powderblue	firebrick
darkgoldenrod	mediumorchid	rosybrown	darkkhaki
silver	mediumvioletred	indianred	peru
chocolate	tan	lightgray	lightgrey
thistle	orchid	goldenrod	palevioletred
crimson	gainsboro	plum	burlywood
lightcyan	lavender	darksalmon	violet
palegoldenrod	lightcoral	khaki	aliceblue
honeydew	azure	sandybrown	wheat
beige	whitesmoke	mintcream	ghostwhite
salmon	antiquewhite	linen	lightgoldenrodyellow
oldlace	red	fuchsia	magenta
deeppink	orangered	tomato	hotpink
coral	darkorange	lightsalmon	orange
lightpink	pink	gold	peachpuff
navajowhite	moccasin	bisque	mistyrose
blanchedalmond	papayawhip	lavenderblush	seashell
cornsilk	lemonchiffon	floralwhite	snow
yellow	lightyellow	ivory	white

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)

Veamos algunos ejemplos de palabras clave de color:

```
div {
  background-color: blue;
  background-color: transparent;
  background-color: lightpink;
  background-color: rebeccapurple; /* En honor a Rebeca, la hija de Eric Meyer */
}
```

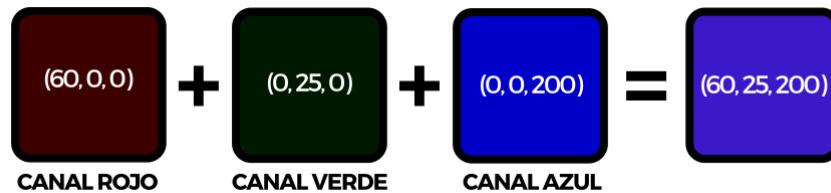
Formato RGB

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**.

Rojo	0	255	255	0
Verde	0	255	0	0
Azul	0	255	0	255

|| || || ||

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*).



La mayoría de los editores tienen los denominados **ColorPicker**, que no son más que un sistema cómodo y rápido para elegir un color a base de clics por una paleta o círculo visual. También podemos hacerlo directamente en buscadores como [Duck Duck Go](#) o [Google](#).

Veamos algunos ejemplos de colores en formato RGB:

```
div {  
background-color: rgb(125, 80, 10);  
background-color: rgb(0, 0, 34);  
color: rgb(255, 255, 0)  
}
```

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	#OFF	0, 255, 255	cyan (azul claro)
#9370DB	#97D	147,112,219	mediumpurple (lila)

[HailPixel](#) nos proporciona una manera muy sencilla y rápida de seleccionar tonalidades de color en **formato hexadecimal** con sólo mover el ratón. Por otro lado, en [ColorSpire](#) puedes seleccionar el color deseado y observar como varían tanto los valores hexadecimales como los valores RGB o HSL (*ver a continuación*).

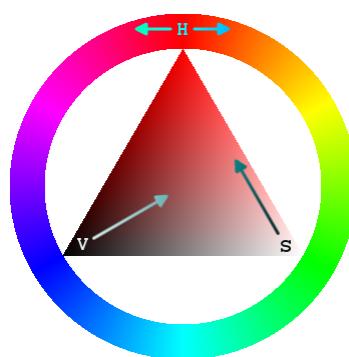
Truco: Como se puede ver en la segunda columna, para ahorrar espacio puedes utilizar el **formato hexadecimal abreviado**, especificando sólo las primeras tres cifras de cada par. Por ejemplo, `#9933AA` como `#93A`. El color abreviado sólo será fiel cuando los pares de cifras sean idénticos (o al menos, aproximados).

Veamos algunos ejemplos del formato hexadecimal (RGB abreviado):

```
div {
    background-color: #512592;
    background-color: #000000;
    background-color: #451;           /* Equivalente a #445511; */
}
```

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%*).



Veamos algunos ejemplos del formato HSL:

```
div {
    background-color: hsl(35deg, 0%, 100%);
    background-color: hsl(120deg, 25%, 75%);
    background-color: hsl(5deg, 20%, 20%);
}
```

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. Vamos a ver como 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()`. Añadimos un nuevo valor como canal alfa (*valor o porcentaje*).
- **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:

```
div {  
background-color: rgba(0, 0, 0, 0.5);  
background-color: rgba(0, 0, 0, 50%);  
background-color: hsla(180deg, 50%, 25%, 0.75);  
background-color: hsla(180deg, 50%, 25%, 75%);  
background-color: #aa44ba80;  
}
```

¡OJO!: El formato de [transparencia en formato hexadecimal](#) se encuentra actualmente bien soportado, pero puede no ser compatible en versiones más antiguas u otros navegadores.



Consejo: Una herramienta genial para seleccionar varios colores en nuestros proyectos es [Adobe Color CC](#). Nos permite escoger entre colores análogos, monocromáticos, tríadas, colores complementarios, compuestos o tonos similares. Muy visual e intuitiva. Otra herramienta, más simple pero muy práctica es [HSL Picker](#), donde puedes elegir el color deseado utilizando el **formato de colores HSL** y pudiendo seleccionar incluso los canales alfa.

Fondos en 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 cual puedes indicar imágenes de fondo o, como veremos más adelante, incluso degradados de varios colores.

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
<code>background-image</code>	<code>none</code>	No utiliza ninguna imagen de fondo.
<code>background-image</code>	<code>url(imagen.jpg)</code>	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**.

NOTA: En el caso de que no se encuentre la imagen o el valor de `background-image` se haya establecido a `none`, no se utilizará ninguna imagen de fondo, y en su lugar se mostrará el color establecido con `background-color`.

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
<code>background-repeat</code>	<code>repeat</code>	Repite la imagen de fondo horizontal y verticalmente.
	<code>repeat-x</code>	Repite la imagen de fondo sólo horizontalmente (eje x).
	<code>repeat-y</code>	Repite la imagen de fondo sólo verticalmente (eje y).
	<code>space</code>	Repite la imagen y rellena con espacio los huecos.
	<code>round</code>	Repite la imagen y amplia cada repetición para ajustar.
	<code>no-repeat</code>	La imagen de fondo no se repite.
<code>background-attachment</code>	<code>scroll</code>	Cuando hacemos scroll la imagen de fondo se desplaza.
	<code>fixed</code>	Cuando hacemos scroll, la imagen de fondo permanece fija.
<code>background-position</code>	<code>POSX</code>	1 parámetro. Desplaza la imagen de fondo al punto (x, 50%).
	<code>POSX</code> <code>POSY</code>	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.

`background-repeat: repeat`



`background-repeat: space`



`background-repeat: round`



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 `POSX` y por `POSY`. Por defecto, esos valores son **0% 0%**, pero pueden especificarse con unidades (*porcentajes, pixels, 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%).

Atajo clásico: Fondos

Es posible establecer todas estas propiedades anteriores en una sola regla de CSS a modo de atajo, y así ahorrar mucho espacio en escribir las propiedades anteriores por separado. Aunque no es estrictamente obligatorio, se aconseja respetar el siguiente orden (*acostumbrarse a usar el mismo orden es una buena práctica*):

```
div {  
    /* background: <color> <image> <repeat> <attachment> <position> */  
    background: #FFF url(imagen.jpg) repeat-x scroll top left;  
}
```

Y con esto, ya conocemos las ventajas básicas de CSS a través de propiedades tan interesantes como `background-color` o `background-image`. Sin embargo, la llegada de CSS3 incorporó novedades muy interesantes en nuestros navegadores, que mediante versiones anteriores a CSS3 no era posible realizar (*o era algo bastante complejo*).

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
<code>opacity</code>	<code>NUMBER</code>	Establece una transparencia (<code>0</code> = 100% transparente, a <code>1</code> = 100% opaco)

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

```
div {  
    background-color: #FF0000;  
    color: #FFFFFF;  
    opacity: 0.5;  
}
```



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**:

```
div {  
    color: white;  
    background-color: RGBA(255, 0, 0, 0.5);  
}
```

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

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:

```
div {  
    background-image: url(manz.png), url(fondo2.jpg), url(fondo3.jpg);  
    background-repeat: no-repeat;  
}
```

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:

```
div {  
    background-image: url(manz.png), url(fondo2.jpg), url(fondo3.jpg);  
    background-repeat: no-repeat, repeat-x, repeat;  
}
```

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

Propiedad	Valor	Significado
<code>background-clip</code>	<code>border-box</code> <code>padding-box</code> <code>content-box</code>	Modo de relleno de la imagen de fondo
<code>background-origin</code>	<code>border-box</code> <code>padding-box</code> <code>content-box</code>	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
<code>padding-box</code>	La imagen o color de fondo cubrirá la zona del espaciado y contenido.
<code>border-box</code>	La imagen o color de fondo cubrirá la zona del borde, espaciado y contenido.
<code>content-box</code>	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

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
<code>background-size</code>	<code>SIZE</code>	1 parámetro. Aplica un <code>SIZE</code> de (<i>ancho × auto</i>) a la imagen de fondo.
	<code>SIZE</code> <code>SIZE</code>	2 parámetros. Aplica un <code>SIZE</code> de (<i>ancho × alto</i>) a la imagen de fondo.

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

Valor	Significado
<code>auto</code>	No escala la imagen. Utiliza el tamaño original. Es el valor por defecto.
<code>unidad</code>	Indicamos el tamaño específico que queremos usar (<i>píxeles o porcentaje, por ej.</i>).
<code>cover</code>	Escala el ancho de la imagen de fondo al ancho del elemento.
<code>contain</code>	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

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:

```
div {  
    /* background: <color> <position> <size> <repeat> <origin> <clip> <attachment>  
    <image> */  
    background: #FFF top left cover repeat-x padding-box border-box scroll  
    url(imagen.jpg);  
}
```

4. Selectores CSS

Selectores CSS 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.

Por ejemplo, consideremos el siguiente caso:

```
div {  
    background-color: red;  
}
```

En este ejemplo, le decimos al navegador que a todas las etiquetas `div` que encuentre en la página, le ponga color de fondo rojo. La parte donde he colocado `div` es lo que se denomina **selector**, y puede llegar a ser **mucho más complejo y potente**, como veremos en los siguientes capítulos.

La verdadera potencia de CSS radica en la capacidad de poder **seleccionar sólo los elementos que nos interesen**, ya que a medida que el documento HTML crece, aparecerán nuevos elementos que podrían adoptar ese estilo, y ser eso algo que no nos interese.

```
selector #id .clase :pseudoclase ::pseudoelemento [atributo] {  
    propiedad : valor ;  
    propiedad : valor  
}
```

Aunque el esquema general es mucho más amplio (*lo iremos viendo todo, poco a poco*), para empezar, vamos a centrarnos en la sintaxis básica de los selectores CSS, y la más utilizada: los **ID** y las **clases**.

Seleccionar por etiquetas

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

```
strong {  
    color: red;  
}
```

En el ejemplo anterior, todos los elementos marcados con la etiqueta HTML `` se visualizarán de color rojo. Este pequeño ejemplo es didáctico pero no suele ser práctico, ya que no nos permite diferenciar entre todos los elementos de ese tipo. Para ello tenemos los selectores que veremos a continuación.

Seleccionar por ID (únicos)

Todas las etiquetas HTML pueden tener un atributo `id` con un valor concreto. Este valor será el nombre que le daremos a la etiqueta. 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, es decir, que **sólo debe haber una etiqueta con el mismo ID** por documento. Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Documento de ejemplo</title>
</head>
<body>
    <div id="saludo">
        <p>¡Hola, visitante! ¡Bienvenido a esta página!</p>
    </div>

    <div id="main">
        <p>En esta página encontrarás los siguientes temas:</p>
    </div>
</body>
</html>
```

En el documento anterior, encontramos dos elementos `<div>`. El primero de ellos, tiene ID **saludo**, una capa donde se le da la bienvenida al usuario. La segunda, con ID **main**, es una capa donde hay contenido. Sería incorrecto crear otra etiqueta con ID **saludo** o **main** en este mismo documento, ya que ya existe una con esos nombres.

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.

```
#saludo {
    background-color: blue;
    color: white;
}
```

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

A medida que vamos codificando y creando nuestros documentos HTML, comprobaremos que necesitamos cosas más flexibles y cómodas que los IDs, ya que los elementos tienden a repetirse y no deben ser únicos. Aquí es donde entran en escena las **clases de CSS**.

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, sino que pueden repetirse a lo largo del documento HTML:

```
<!DOCTYPE html>
<html>
<head>
```

```

<title>Documento de ejemplo</title>
</head>
<body>
  <div id="main">
    <p>Escoge uno de los siguientes botones:</p>

    <button class="classic">Opción 1</button>
    <button class="classic">Opción 2</button>
    <button class="classic">Opción 3</button>
    <button class="back">Volver</button>
  </div>
</body>
</html>

```

En el documento HTML anterior, tenemos cuatro botones HTML. Los 3 primeros son botones de opciones, por lo que establecemos una misma clase llamada `classic`, mientras que el último botón (*un botón para volver hacia atrás*), le hemos indicado la clase `back` para que tenga un estilo diferente a los anteriores.

Para hacer esto con **IDs** (*recordemos que son únicos, no deberían repetirse*) tendríamos que crear **3 IDs con nombres diferentes**, mientras que con las **clases** (*las cuales si se pueden repetir*) podemos reutilizar el mismo nombre. 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.

```

.classic {
  background-color: green;
  color: white;
}

.back {
  background-color: orange;
  color: white;
}

```

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

Además, en el caso de las clases, podemos incluso diferenciar el tipo de elemento del que se trata. Gracias a esto podríamos utilizar la clase `.classic` en `<button>` para dar estilo visual a botones, y la clase `.classic` en `<p>` para dar otros estilos diferentes a los párrafos de texto:

```

button.classic {
  background-color: green;
  color: white;
}

p.classic {
  color: red;
}

```

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:

```
<button class="classic green-border big">Botón</button>
```

Este enfoque se considera uno de los principios del denominado **CSS atómico**, en el que se basan frameworks CSS como [Tachyons](#) o [TailwindCSS](#).

Por último, mencionar que también tenemos la posibilidad de ser más específicos y aplicar estilo sólo a los elementos HTML que contengan todas las clases indicadas, colocando cada clase de forma consecutiva en el CSS, de la siguiente forma:

```
button.classic.green-border.big {  
    /* Estilos CSS */  
}
```

En este ejemplo, sólo se aplicarán los estilos cuando el elemento HTML `<button>` tenga una clase `classic`, una clase `green-border` y una clase `big`. Si falta alguna de ellas, no se aplicará el estilo.

Pseudoclases CSS

Las **pseudoclases** se utilizan para hacer referencia a ciertos comportamientos de los elementos HTML. Así como los combinadores CSS se utilizan para dar estilos dependiendo de donde estén colocados en la estructura del HTML, las pseudoclases se utilizan para dar estilos a elementos **respecto al comportamiento** que experimentan en determinado momento.

Volvamos a recordar el esquema general de sintaxis de CSS:

```
selector #id .clase :pseudoclase ::pseudoelemento [atributo] {  
    propiedad : valor ;  
    propiedad : valor  
}
```

Las pseudoclases se definen añadiendo **dos puntos** antes de la pseudoclase concreta. En el caso de existir selectores de etiqueta, id o clases, estas se escribirían a su izquierda.

Pseudoclases de enlaces

Existen algunas pseudoclases orientadas a los enlaces o hipervínculos. En este caso, permiten cambiar los estilos dependiendo del comportamiento del enlace:

Pseudoclase	Descripción
<code>:link</code>	Aplica estilos cuando el enlace no ha sido visitado todavía.
<code>:visited</code>	Aplica estilos cuando el enlace ha sido visitado anteriormente.

A continuación veremos un ejemplo donde seleccionamos mediante un simple selector `a` los enlaces que **aún no han sido visitados**, cambiando el color de los mismos o su formato, lo que mostrará dichos enlaces de color verde y en negrita:

```
a:link {
  color: green;
  font-weight: bold
}
```

Por otro lado, la pseudoclase `:visited` puede utilizarse para dar estilo a los enlaces que **hayan sido visitados previamente** en el navegador del usuario:

```
a:visited {
  color: purple;
  font-weight: bold
}
```

Pseudoclases de ratón

Originalmente, las siguientes pseudoclases se utilizaban solamente en enlaces (*Internet Explorer no los soportaba en otros elementos*). Sin embargo, actualmente pueden ser utilizadas con seguridad en cualquier otro elemento, sin necesidad de ser `<a>`.

Pseudoclase	Descripción
<code>:hover</code>	Aplica estilos cuando pasamos el ratón sobre un elemento.
<code>:active</code>	Aplica estilos cuando estamos pulsando sobre el elemento.

La primera de ellas, `:hover`, es muy útil e interesante, ya que permite aplicar estilos a un elemento justo cuando el usuario está pasando el ratón sobre él. Es una de las pseudoclases más utilizadas:

```
/* Usuario mueve el ratón sobre un enlace */
a:hover {
  background-color: cyan;
  padding: 2px
}

/* Usuario mueve el ratón sobre un div y resalta todos los enlaces que contiene */
div:hover a {
  background-color: steelblue;
  color: white;
}
```

Obsérvese que podemos realizar acciones un poco más específicas, como el segundo ejemplo anterior, donde al movernos sobre un elemento div (`div:hover`), aplicaremos los estilos a los enlaces (`a`) que están dentro del mencionado `div`.

Por otro lado, la segunda pseudoclase, `:active`, permite resaltar los elementos que se encuentran activos, donde el usuario está pulsando de forma activa con el ratón:

```
a:active {  
    border: 2px solid #FF0000;  
    padding: 2px  
}
```

Nota: Aunque las pseudoclases anteriores se inventaron para interactuar con **un ratón** en un sistema de escritorio, pueden funcionar en dispositivos táctiles. Aún así, ten en cuenta que, por ejemplo, el `:hover` no tiene mucho sentido en dispositivos móviles, ya que, aunque podría hacerlo, un usuario no navega por móvil arrastrando el dedo por la pantalla.

Pseudoclases de interacción

Existen pseudoclases orientadas principalmente a los campos de formulario de páginas webs y la interacción del usuario con ellos, veamos otro par interesante:

Pseudoclase	Descripción
<code>:focus</code>	Aplica estilos cuando el elemento tiene el foco.
<code>:checked</code>	Aplica estilos cuando la casilla está seleccionada.

Cuando estamos escribiendo en un campo de texto de un formulario de una página web, generalmente pulsamos TAB para cambiar al siguiente campo y SHIFT+TAB para volver al anterior. Cuando estamos posicionados en un campo se dice que ese campo **tiene el foco**, mientras que al pulsar TAB y saltar al siguiente, decimos que **pierde el foco**.

El comportamiento de «ganar el foco» puede gestionarse mediante la pseudoclase `:focus`:

```
/* El campo ha ganado el foco */  
input:focus {  
    border: 2px dotted #444  
}
```

Nota: Aunque estas pseudoclases suelen utilizarse con elementos de formularios como `<input>`, también pueden utilizarse con otros elementos, como por ejemplo los enlaces `<a>`. Esta es una excelente oportunidad para personalizar el estilo de los campos de texto de un formulario (`<input>` y `<textarea>`) mientras el usuario escribe y se mueve por ellos.

Por otro lado, la pseudoclase `:checked` permite aplicar el estilo especificado a los elementos `<input>` (casillas de verificación o botones de radio) u `<option>` (la opción seleccionada de un `<select>`).

Por ejemplo, se podría utilizar el siguiente fragmento de código:

```
input:checked + span {  
    color: green;  
}
```

Este ejemplo añade el selector hermano `+` para darle formato al `` que contiene el texto y se encuentra colocado a continuación de la casilla `<input>`. De esta forma, los textos que hayan sido seleccionados, se mostrarán en verde.

Pseudoclases de activación

Por norma general, los elementos de un formulario HTML están siempre activados, aunque se pueden desactivar añadiendo el atributo `disabled` (es un atributo booleano, no lleva valor) al elemento HTML en cuestión. Esto es una práctica muy utilizada para impedir al usuario escribir en cierta parte de un formulario porque, por ejemplo, no es aplicable.

Existen varias pseudoclases para detectar si un campo de un formulario está activado o desactivado:

Pseudoclase	Descripción
<code>:enabled</code>	Aplica estilos cuando el campo del formulario está activado.
<code>:disabled</code>	Aplica estilos cuando el campo del formulario está desactivado.
<code>:read-only</code>	Aplica estilos cuando el campo es de sólo lectura.
<code>:read-write</code>	Aplica estilos cuando el campo es editable por el usuario.

Utilizando las dos primeras pseudoclases, bastante autoexplicativas por si solas, podemos seleccionar elementos que se encuentren activados (*comportamiento por defecto*) o desactivados:

```
/* Muestra en fondo blanco las casillas que permiten escribir */
input:enabled {
    background-color: white;
}

/* Muestra en fondo gris las casillas que no permiten escribir */
input:disabled {
    background-color: grey;
}
```

Por otro lado, las pseudoclases `read-only` y `read-write` nos permiten seleccionar y diferenciar elementos que se encuentran en modo de sólo lectura (*tienen especificado el atributo `readonly` en el HTML*) o no:

```
input:read-only {
    background-color: darkred;
    color: white
}
```

En el ejemplo anterior, la pseudoclase `:read-only` le da estilo a aquellos campos `<input>` de un formulario que están marcados con el atributo de sólo lectura `readonly`. La diferencia entre un campo con atributo `disabled` y un campo con atributo `readonly` es que la información del campo con `readonly` se enviará a través del formulario, mientras que la del campo con `disabled` no se enviará. Aún así, ambas no permiten modificar el valor.

Nota: Ten en cuenta que `:read-only` aplicará los estilos a todos los elementos HTML que no puedan ser modificados por el usuario.

Por otro lado, la pseudoclase `:read-write` es muy útil para dar estilos a todos aquellos elementos que son editables por el usuario, sean campos de texto `<input>` o `<textarea>`.

```
input:read-write {  
    background-color: green;  
    color: white  
}
```

Nota: La pseudoclase `read-write` da estilo también a elementos HTML que contengan el atributo `contenteditable`, como por ejemplo un párrafo editable por el usuario con dicho atributo.

Pseudoclases de validación

En HTML5 es posible dotar de capacidades de validación a los campos de un formulario, pudiendo interactuar desde Javascript o incluso desde CSS. Con estas validaciones podemos asegurarnos de que el usuario escribe en un campo de un formulario el valor esperado que debería. Existen algunas pseudoclases útiles para las validaciones, como por ejemplo las siguientes:

Pseudoclase	¿Cuándo aplica estilos?
<code>:required</code>	Cuando el campo es obligatorio, o sea, tiene el atributo <code>required</code> .
<code>:optional</code>	Cuando el campo es opcional (por defecto, todos los campos).
<code>:invalid</code>	Cuando los campos no cumplen la validación HTML5.
<code>:valid</code>	Cuando los campos cumplen la validación HTML5.
<code>:out-of-range</code>	Cuando los campos numéricos están fuera del rango.
<code>:in-range</code>	Cuando los campos numéricos están dentro del rango.

En un formulario HTML es posible establecer un campo obligatorio que será necesario llenar para enviar el formulario. Por ejemplo, el DNI de una persona que va a matricularse en un curso, o el nombre de usuario de alta en una plataforma web para identificarse. Campos que son absolutamente necesarios.

Para hacer obligatorios dichos campos, tenemos que indicar en el HTML el atributo `required`, al cuál será posible darle estilo mediante la pseudoclase `:required`:

```
input:required {  
    border: 2px solid blue;  
}
```

Por otra parte, los campos opcionales (*no obligatorios, sin el atributo `required`*) pueden seleccionarse con la pseudoclase `:optional`:

```
input:optional {  
    border: 2px solid grey;  
}
```

Las **validaciones en formularios HTML** siempre han sido un proceso tedioso, hasta la llegada de HTML5. HTML5 brinda un excelente soporte de validaciones desde el lado del cliente, pudiendo comprobar si los datos especificados son correctos o no antes de realizar las validaciones en el lado del servidor, y **evitando la latencia** de enviar la información al servidor y recibirla de vuelta.

Ojo: Ten en cuenta que la validación de cliente es apropiada solo para reducir la latencia de envío/recepción al servidor, pero nunca como estrategia para evitar problemas de seguridad o similares, para la cual se debe tener **validación en el servidor** siempre. Las validaciones utilizadas en frontend, es posible falsearlas o saltárselas.

Imaginemos un campo de entrada en el que queremos obtener la **edad del usuario**. Nuestra intención es que solo se puedan introducir números. Para ello hacemos uso de la **expresión regular** `[0-9]+`, que significa «una o más cifras del 0 al 9»:

```
<input type="text" name="age" pattern="[0-9]+" />
```

Sin embargo, el atributo `pattern` permite expresiones regulares realmente complejas, como por ejemplo, una **expresión regular** para validar el formato de un DNI, ya sea en el formato nacional de España (12345678L) o en formato NIE (X1234567L), aceptando guiones si se indican:

```
<input type="text" name="dni"
       pattern="(([x-z]{1})([-?])(\d{7})([-?])([A-z]{1}))|((\d{8})([-?])([A-z]
{1}))" />
```

Se pueden aplicar ciertos estilos dependiendo de si se cumple o no el patrón de validación, utilizando las siguientes pseudoclases:

```
input:invalid {
    background-color: darkred;
    color: white;
}

input:valid {
    background-color: green;
    color: white;
}
```

Sin embargo, en la validación numérica que vimos anteriormente, un usuario podría escribir **500**, que es una edad imposible, porque en el patrón de validación indicamos «una o más cifras del 0 al 9». Lo ideal sería establecer un rango, algo que se suele hacer muy a menudo si tenemos campos numéricos de formulario:

```
<input type="number" name="age" min="18" max="100" />
```

Este campo permite al usuario especificar su edad, utilizando los atributos de validación `min` y `max`, que sólo permiten valores entre 18 y 100 años. Los valores fuera de este rango, no serán válidos.

De la misma forma que antes, es posible aplicar estilos para los valores fuera de rango, como dentro de rango:

```
input:out-of-range {  
    background-color: darkred;  
    color: white;  
}  
  
input:in-range {  
    background-color: green;  
    color: white;  
}
```

Pseudoclases de negación

Existe una pseudoclase muy útil, denominada **pseudoclase de negación**. Permite seleccionar todos los elementos que no cumplan los selectores indicados entre paréntesis.

Veamos un ejemplo:

```
p:not(.general) {  
    border: 1px solid #DDD;  
    padding: 8px;  
    background: #FFF;  
}
```

Este pequeño fragmento de código nos indica que todos los párrafos (*elementos <p>*) que no pertenezcan a la clase **general**, se les aplique el estilo especificado.

Consejo: Las reglas de negación pueden ser complejas, ineficientes y poco escalables.
Intenta utilizarlas sólo en los casos que sea absolutamente necesario.

Otras pseudoclases

Para finalizar el apartado de pseudoclases, quiero mencionar algunas que no encajan en los apartados anteriores, pero que pueden ser muy útiles en algunos casos:

Pseudoclase	Significado
<code>:lang(es)</code>	Aplica estilo a los elementos con el atributo <code>lang="es"</code> .
<code>:target</code>	Aplica estilo al elemento que coincide con el ancla de la URL.
<code>:root</code>	Aplica estilo al elemento raíz (padre) del documento.
<code>:default</code>	Experimental. Aplica estilo al elemento por defecto. Útil en formularios.
<code>:dir(A)</code>	Experimental. Aplica estilo al elemento que coincide con la dirección <code>ltr</code> o <code>rtl</code> .
<code>:indeterminate</code>	Experimental. Aplica estilo a la casilla checkbox o al elemento <code><progress></code> sin definir.
<code>:fullscreen</code>	Experimental. Aplica estilo si la página está en el modo de pantalla completa.
<code>:scope</code>	Experimental. Aplica estilo a los elementos en el ámbito indicado.
<code>:any(A)</code>	Experimental. Aplica estilo si coincide con algún elemento indicado en <code>A</code> .

Otras pseudoclases como `:first`, `:left`, `:right` o `:blank` las mencionamos en el capítulo de [medios paginados](#).

5. Fuentes y tipografías

Tipografías CSS

Las **tipografías** (*también denominadas fuentes*) son una parte muy importante del mundo de CSS. De hecho, son uno de los pilares del diseño web. La elección de una **tipografía adecuada**, su tamaño, color, espacio entre letras, interlineado y otras características pueden variar mucho, de forma consciente o inconsciente, la percepción en la que una persona interpreta o accede a los contenidos de una página.

Detalles de una tipografía

Existen multitud de características en las tipografías que convendría conocer antes de continuar, por lo que vamos a ver algunas de ellas:



- **Serifa:** Las fuentes o tipografías que utilizan **serifa** o **gracia**, son aquellas que incorporan unos pequeños adornos o remates en los extremos de los bordes de las letras. Muchas de estas tipografías suelen terminar su nombre en «Serif» (*con serifa*).
- **Paloseco:** Las fuentes o tipografías de paloseco son las opuestas a la anterior: unas tipografías lisas, sin adornos o remates en los extremos de los bordes de las letras. Muchas de estas tipografías suelen terminar su nombre en «Sans Serif» (*sin serifa*).

Tradicionalmente, se han utilizado tipografías con **serifa** en **medios impresos** argumentando que dichos bordes ofrecen una mayor legibilidad que las tipografías de paloseco, ya que ayudan a reconocer más rápidamente las letras. En **medios digitales**, las tipografías de **paloseco** suelen ser más comunes puesto que dan un aspecto más limpio y ayudan a que se canse menos la vista del usuario. No obstante, todo esto puede ser muy subjetivo y está sujeto a diferentes interpretaciones.

- **Monoespaciada:** Por otro lado, existe un tipo de tipografía denominada fuente monoespaciada, que se basa en que cada una de sus letras tienen exactamente el mismo ancho. Son muy útiles para tareas de programación o emuladores de [terminal](#), donde se leen mejor líneas con estas características, ya que no queremos que una línea sea más corta dependiendo de su contenido.

 fuente no monoespaciada

 fuente monoespaciada

Propiedades básicas

Existe un amplio abanico de propiedades CSS para modificar las características básicas de las tipografías a utilizar. Aunque existen muchas más, a continuación, veremos las propiedades CSS más básicas para aplicar a cualquier tipo de tipografía:

Propiedad	Valor	Significado
<code>font-family</code>	<i>fuente</i>	Indica el nombre de la fuente (tipografía) a utilizar.
<code>font-size</code>	<code>SIZE</code>	Indica el tamaño de la fuente.
<code>font-style</code>	<code>normal</code> <code>italic</code> <code>oblique</code>	Indica el estilo de la fuente.
<code>font-weight</code>	<i>peso</i>	Indica el peso (grosor) de la fuente (100-800).

Con ellas podemos seleccionar tipografías concretas, especificar su tamaño, estilo o grosor.

Familia tipográfica

Empezaremos por la más lógica, la propiedad CSS para seleccionar una **familia tipográfica** concreta. Con esta propiedad, denominada `font-family`, podemos seleccionar seleccionar cualquier tipografía simplemente escribiendo su nombre.

Si dicho nombre está compuesto por varias palabras separadas por un espacio, se aconseja utilizar comillas simples para indicarla (*como se ve en el segundo ejemplo*):

```
body {  
    font-family: Verdana;  
    font-family: 'PT Sans'; /* Otro ejemplo */  
}
```

Esta es la forma más básica de indicar una tipografía. Sin embargo, hay que tener en cuenta un detalle muy importante: estas fuentes **sólo se visualizarán si el usuario las tiene instaladas en su sistema o dispositivo**. En caso contrario, se observarán los textos con otra tipografía «suplente» que esté disponible en el sistema, pero que puede ser visualmente muy diferente.

Esto convierte una tarea a priori simple, en algo muy complejo, puesto que los sistemas operativos (*Windows, Mac, GNU/Linux*) tienen diferentes tipografías instaladas. Si además entramos en temas de licencias y tipografías propietarias, la cosa se vuelve aún más compleja.

Consejo: La página [FontFamily.io](https://fontfamily.io) incorpora un sencillo formulario para mostrar información sobre determinadas tipografías y como se mostrarian en diferentes sistemas (*Windows, Mac OS, GNU/Linux, Android, iOS, Windows Phone, etc...*). Más adelante, veremos la regla `@font-face` de **CSS3**, que permite solucionar este problema y la usan sistemas como **Google Fonts**.

Un primer y sencillo paso para paliar (*en parte*) este problema, es añadir varias tipografías alternativas, separadas por comas, lo que además se considera una buena práctica de CSS:

```
div {  
    font-family: Vegur, 'PT Sans', Verdana, sans-serif;  
}
```

De esta forma, el navegador busca la fuente **Vegur** en nuestro sistema, y en el caso de no estar instalada, pasa a buscar la siguiente (*PT Sans*), y así sucesivamente. Se recomienda especificar al menos 2 ó 3 tipografías diferentes.

Consejo: Como última opción de **font-family** se recomienda utilizar una palabra clave denominada «web-safe fonts» (*fuente segura*). Esta fuente segura no es una tipografía específica, sino una **palabra clave** con la que se denomina una categoría. Esto indica al navegador que debe buscar una tipografía instalada en el sistema que entre dentro de la misma categoría.

Las palabras clave de **fuentes seguras** son las siguientes:

Fuente	Significado	Fuentes de ejemplo
serif	Tipografía con serifa	Times New Roman, Georgia...
sans-serif	Tipografía sin serifa	Arial, Verdana, Tahoma...
cursive	Tipografía en cursiva	Sanvito, Corsiva...
fantasy	Tipografía decorativa	Critter, Cottonwood...
monospace	Tipografía monoespaciada	Courier, Courier New...

Tamaño de la tipografía

Otra de las propiedades más utilizadas con las tipografías es **font-size**, una tipografía que permite especificar el tamaño que tendrá la fuente que vamos a utilizar:

Propiedad	Valor	Tipo de medida
font-size	xx-small x-small small medium large x-large xx-large	Absoluta (tamaño predefinido)
font-size	smaller larger	Relativa (más pequeña/más grande)
font-size	SIZE	Específica (tamaño exacto)

Se pueden indicar tres tipos de valores:

- **Medidas absolutas:** Palabras clave como **medium** que representan un tamaño medio (*por defecto*), **small**: tamaño pequeño, **x-small**: tamaño muy pequeño, etc...
- **Medidas relativas:** Palabras clave como **smaller** que representan un tamaño un poco más pequeño que el actual, o **larger** que representa un tamaño un poco más grande que el actual.
- **Medida específica:** Simplemente, indicar píxeles, porcentajes u otra unidad para especificar el tamaño concreto de la tipografía. Para tipografías se recomienda empezar por píxeles (*más fácil*) o utilizar estrategias con [unidades rem](#) (*mejor, pero más avanzado*).

Estilo de la tipografía

A las tipografías elegidas se les puede aplicar ciertos estilos, muy útil para maquetar los textos, como por ejemplo **negrita** o **cursiva** (*italic*). La propiedad que utilizamos es `font-style` y puede tomar los siguientes valores:

Valor	Significado
normal	Estilo normal, por defecto. Sin cambios aparentes.
italic	Cursiva. Estilo caracterizado por una ligera inclinación de las letras hacia la derecha.
oblique	Oblícua. Idem al anterior, salvo que esta inclinación se realiza de forma artificial.

Con la propiedad `font-style` podemos aplicarle estos estilos. En la mayoría de los casos, se aprecia el mismo efecto con los valores **italic** y **oblique**, no obstante, **italic** muestra la versión cursiva de la fuente, específicamente creada por el diseñador de la tipografía, mientras que **oblique** es una representación forzosa artificial de una tipografía cursiva.

Peso de la tipografía

Por otro lado, tenemos el **peso** de la fuente, que no es más que el grosor de la misma. También depende de la fuente elegida, ya que no todas soportan todos los tipos de grosor. De forma similar a como hemos visto hasta ahora, se puede especificar el peso de una fuente mediante tres formas diferentes:

Propiedad	Valor	Significado
<code>font-weight</code>	normal bold	Medidas absolutas (predefinidas)
<code>font-weight</code>	bolder lighter	Medidas relativas (dependen de la actual)
<code>font-weight</code>	peso	Medida específica (número del peso concreto)



- **Valores absolutos:** Palabras claves para indicar el peso de la fuente: *normal* y *bold*. **Normal** es el valor por defecto.
- **Valores relativos:** *Bolder* (más gruesa) o *Lighther* (más delgada).
- **Valor numérico:** Un número del **100** (menos gruesa) al **900** (mas gruesa). Generalmente, se incrementan en valores de 100 en 100.

OJO: Ten en cuenta que los diferentes pesos de una tipografía son diseñados por el creador de la tipografía. Algunas tipografías carecen de diferentes pesos y sólo tienen uno específico. Esto es algo muy sencillo de ver en Google Fonts (*al seleccionar una tipografía*).

Tipografías externas

Antiguamente, utilizar tipografías en CSS tenía una gran limitación. Usando la propiedad `font-family` y especificando el nombre de la tipografía a utilizar, en principio deberían visualizarse. Pero fundamentalmente, existían dos problemas:

Las tipografías especificadas mediante `font-family` debían estar instaladas en el sistema donde se visualiza la página web.

```
p {  
    font-family: vegur, Georgia, "Times New Roman", sans-serif;  
}
```

En el ejemplo superior, se han indicado las tipografías [Vegur](#) (*tipografía personalizada*), [Georgia](#) y [Times New Roman](#) (*tipografías de Microsoft Windows*) y la categoría segura [sans-serif](#). Un usuario con la tipografía [Vegur](#) instalada, vería sin problema el diseño con dicha tipografía, mientras que un usuario de Windows la vería con Georgia (*o si no la tiene, con Times New Roman*), mientras que un usuario de Linux o Mac, la vería con otra tipografía diferente (*una tipografía sans-serif del sistema*).

Esto es un problema ya que no permite hacer diseños consistentes, pero hay formas de solucionarlo, como veremos a continuación.

Por otro lado, muchas tipografías genéricas tienen [derechos de autor](#) y puede que algunos sistemas no tengan permiso para tenerlas instaladas.

Mientras que las tipografías que vienen en sistemas como Microsoft Windows de serie (*Times New Roman, Verdana, Tahoma, Trebuchet MS...*) se verían correctamente en navegadores con dicho sistema operativo, no ocurriría lo mismo en dispositivos con GNU/Linux o Mac. Y lo mismo con tablets o dispositivos móviles, o viceversa. Esto ocurre porque muchas tipografías son propietarias y tienen licencias que permiten usarse sólo en dispositivos de dicha compañía.

En definitiva, aunque teníamos los mecanismos, vivimos en un mundo complicado en el que no es tan sencillo establecer una fuente específica para obtener el mismo resultado de diseño en todos los navegadores y sistemas disponibles.... al menos hasta que llegó [@font-face](#).

La regla @font-face

La regla [@font-face](#) permite descargar una fuente o tipografía, cargarla en el navegador y utilizarla en nuestras páginas. Todo ello de forma transparente al usuario sin que deba instalar o realizar ninguna acción.

Veamos un ejemplo de como se puede utilizar:

```
@font-face {  
    font-family: 'Open Sans';  
    font-style: normal;  
    font-weight: 400;  
    src: local('Open Sans'),  
        url(/fonts/opensans.woff2) format('woff2'),  
        url(/fonts/opensans.woff) format('woff'),  
        url(/fonts/opensans.ttf) format('truetype'),  
        url(/fonts/opensans.otf) format('opentype'),  
        url(/fonts/opensans.eot) format('embedded-opentype');  
}
```

La regla [@font-face](#) suele colocarse al principio del fichero CSS para preparar el navegador para descargar la tipografía en el caso de no disponer de ella. En el ejemplo superior lo hemos hecho con la fuente [Open Sans](#), una tipografía libre creada por [Steve Matteson](#) para Google y disponible en Google Fonts.

Basicamente, abrimos un bloque `@font-face`, establecemos su nombre mediante `font-family` y definimos sus características mediante propiedades como `font-style` o `font-weight`. El factor clave viene a la hora de **indicar** la tipografía, que se hace mediante la propiedad `src` (*source*) con los siguientes valores:

Valor	Significado	Soporte
<code>local('Nombre')</code>	¿Está la fuente 'Nombre' instalada? Si es así, no hace falta descargarla.	Todos
<code>url(file.woff2)</code>	Formato Web Open Font Format 2 . Mejora de <code>WOFF</code> con Brotli .	No IE
<code>url(file.woff)</code>	Formato Web Open Font Format . Es un <code>TTF</code> comprimido, ideal para web.	Bueno
<code>url(file.ttf)</code>	Formato True Type . Uno de los formatos más conocidos.	Bueno
<code>url(file.otf)</code>	Formato Open Type . Mejora del formato <code>TTF</code> .	Bueno
<code>url(file.eot)</code>	Formato Embedded OpenType . Mejora de <code>OTF</code> , propietaria de Microsoft.	Sólo IE
<code>url(file.svg)</code>	Tipografías creadas como formas SVG. No usar , considerada obsoleta .	Malo

Consejo: Actualmente, una buena práctica es utilizar la expresión `local()` seguida de la expresión `url()` con los formatos `WOFF2`, `WOFF` y `TTF` (*en dicho orden*), dando así soporte a la mayoría de navegadores. Para dar soporte a versiones antiguas de Internet Explorer, podría ser adecuado incluir también el formato `EOT`.

Google Fonts

En la actualidad, es muy común utilizar [Google Fonts](#) como repositorio proveedor de tipografías para utilizar en nuestros sitios web por varias razones:

- **Gratis:** Disponen de un amplio catálogo de fuentes y tipografías libres y/o gratuitas.
- **Cómodo:** Resulta muy sencillo su uso: Google nos proporciona un código y el resto lo hace él.
- **Rápido:** El servicio está muy extendido y utiliza un CDN, que brinda ventajas de velocidad.

En la propia página de **Google Fonts** podemos seleccionar las fuentes con las características deseadas y generar un código HTML con la tipografía (*o colección de tipografías*) que vamos a utilizar.



Open Sans

Designed by Steve Matteson

[Download family](#)[Select styles](#)[Glyphs](#)[About](#)[License](#)

Styles

Type here to preview text

Almost before we knew it, we had left the gro

Size: 30px



Light 300

Almost before we knew it, we had left the gro [+ Select this style](#)

Light 300 italic

Almost before we knew it, we had left the gro [+ Select this style](#)

Regular 400

Almost before we knew it, we had left the gro [- Remove this style](#)

Selected family

[Review](#)[Embed](#)

To embed a font, copy the code into the <head> of your html

<link href="https://fonts.googleapis.com/css2?family=Open+Sans&display=swap" rel="stylesheet">

CSS rules to specify families

font-family: 'Open Sans', sans-serif;

Todo esto nos generará el siguiente código, que aparece en la zona derecha de la web (*en la zona «embed»*), y que será el fragmento de código que tendremos que insertar en nuestro documento HTML, concretamente, antes de finalizar la sección `<head>`:

```
<link rel="stylesheet"
      href="https://fonts.googleapis.com/css2?
family=Open+Sans:wght@300;400&display=swap">
```

Cómo se puede ver el ejemplo anterior, al añadir este código estamos enlazando nuestro documento HTML con un documento CSS del repositorio de Google, que incluye los `@font-face` correspondientes. Esto hará que incluyamos automáticamente todo ese código CSS necesario para las tipografías escogidas, en este caso la tipografía **Open Sans** con los pesos **300** y **400**.

Si además, añadimos también la familia de tipografías **Roboto** (con grosor 400) y **Lato** (con grosor 300 y 400), el código necesario sería el siguiente:

```
<link rel="stylesheet"
      href="https://fonts.googleapis.com/css2?
family=Lato:wght@300;400&family=Open+Sans:wght@300;400;600&family=Roboto&display=swap">
```

De esta forma conseguimos cargar varias tipografías desde el repositorio de Google **de una sola vez**, sin la necesidad de varias líneas de código diferentes, que realizarían varias peticiones diferentes a **Google Fonts**.

Nota que en este nuevo ejemplo, en caso de no tener instaladas ninguna de las tipografías anteriores, estaríamos realizando **6 descargas**: (*el css de Google Fonts*), (*las 2 tipografías con los diferentes pesos de Open Sans*), (*la de Roboto*), (*y las 2 tipografías con los diferentes pesos de Lato*).

Por último y para terminar, sólo necesitaremos añadir la propiedad `font-family: "Open Sans"`, `font-family: "Lato"` o `font-family: "Roboto"` a los textos que queramos dar formato con dichas tipografías. No te olvides de añadir tipografías alternativas y fuente segura para mejorar la compatibilidad con navegadores antiguos.

El parámetro `display` con valor `swap` que aparece en la última versión de Google Fonts, lo explicamos aquí [Propiedades avanzadas de tipografías](#).

No hay que dejar de tener en cuenta que **cuantas más tipografías** (y/o más pesos) incluyamos en nuestra página, **más lenta será la experiencia del usuario**, ya que más contenido tendrá que descargar. Salvo excepciones particulares, lo habitual suele ser elegir entre **2-3 tipografías como máximo**, cada una con una finalidad concreta: encabezados o titulares, tipografía de lectura normal y tipografía secundaria, por ejemplo.

Atajo para tipografías

Finalmente, algunas de las propiedades más utilizadas de tipografías y fuentes se pueden resumir en una propiedad de atajo, como viene siendo habitual. El esquema es el siguiente:

```
div {  
    font: <style> <variant> <weight> <size/line-height> <family>;  
}
```

Por ejemplo, utilizar la tipografía `Arial`, con la fuente alternativa `Verdana` o una fuente segura sin serifa, a un tamaño de 16 píxeles, con un interlineado de 22 píxeles, un peso de 400, sin utilizar versalitas y con estilo cursiva:

```
div {  
    font: italic normal 400 16px/22px Arial, Verdana, sans-serif;  
}
```

Textos y alineaciones

CSS dispone de ciertas propiedades relacionadas con el texto de una página, pero alejándose de criterios de tipografías, y centrándose más en objetivos de alineación o tratamiento de espaciados. Veamos algunas de estas propiedades:

Propiedad	Valor	Significado
<code>letter-spacing</code>	<code>normal</code> <code>SIZE</code>	Espacio entre letras (tracking)
<code>word-spacing</code>	<code>normal</code> <code>SIZE</code>	Espacio entre palabras
<code>line-height</code>	<code>normal</code> <code>SIZE</code>	Altura de una línea (interlineado)
<code>text-indent</code>	<code>SIZE</code>	Indentación de texto (sangría)
<code>white-space</code>	<code>normal</code> nowrap pre pre-line pre-wrap	Comportamiento de los espacios
<code>tab-size</code>	<code>NUMBER</code>	<code>SIZE</code>
<code>direction</code>	<code>ltr</code> <code>rtl</code>	Dirección del texto

Las tres primeras propiedades, determinan el espacio en diferentes zonas del texto. Por ejemplo la primera de ellas, `letter-spacing`, especifica el espacio de separación que hay entre cada letra de un texto, denominado comúnmente **interletraje** o **tracking**. Con números negativos tendremos más unidas las letras y con números positivos, las tendremos más separadas unas de otras.



La propiedad `line-height` especifica la **altura** que tendrá cada línea de texto, una característica que puede facilitar muchísimo la lectura, puesto que un interlineado excesivo puede desorientar al lector, mientras que uno insuficiente puede hacer perder al visitante el foco en el texto.

La propiedad `word-spacing` permite establecer el espacio que hay entre una palabra y otra en un texto determinado, lo que puede facilitar la legibilidad de los textos de una página web y da flexibilidad y control sobre ciertas tipografías.

La propiedad `text-indent` establece un tamaño de indentación (*por defecto, 0*), o lo que es lo mismo, hace un sangrado, desplazando el texto la longitud especificada hacia la derecha (*o izquierda en cantidades negativas*).

Al utilizar `white-space` podemos indicar el comportamiento que tendrán los espacios en blanco en una página web. Por defecto, el valor es `normal` (*transforma múltiples espacios en blanco en un solo espacio consecutivo*), pero tiene otras opciones posibles:

Valor	Espacios en blanco consecutivos	Contenido
<code>normal</code>	Los espacios se transforman en uno solo.	Se ajusta al contenedor.
<code>nowrap</code>	Los espacios se transforman en uno solo.	Ignora saltos de línea.
<code>pre</code>	Respeta literalmente los espacios.	Ignora saltos de línea.
<code>pre-wrap</code>	Respeta literalmente los espacios.	Se ajusta al contenedor.
<code>pre-line</code>	Respeta literalmente los espacios y suprime los espacios del final.	Se ajusta al contenedor.

Nota: La diferencia entre `pre-wrap` y `pre-line` es que este último respeta literalmente los espacios que están antes del texto, mientras que si sobran después del texto, los suprime.

Probablemente, a medida que realices diferentes diseños, te encontrarás con la desagradable situación en la que un texto concreto (*por ejemplo, un enlace demasiado largo*) no cabe dentro de un contenedor, por lo que el texto se desborda y provoca efectos no deseados como salirse de su lugar.

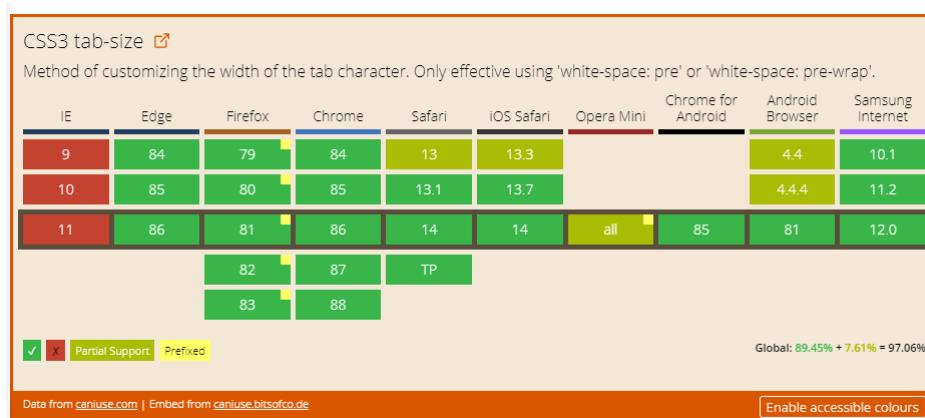
Propiedad	Valor	Significado
<code>hyphens</code>	<code>manual</code> <code>none</code> <code>auto</code>	Indica si se debe dividir las palabras por guiones.
<code>overflow-wrap</code>	<code>normal</code> <code>break-word</code> <code>anywhere</code>	Dividir palabras para evitar desbordamientos.
<code>line-break</code>	<code>auto</code> <code>loose</code> <code>normal</code> <code>strict</code> <code>anywhere</code>	Determina como dividir líneas.
<code>word-break</code>	<code>normal</code> <code>keep-all</code> <code>break-all</code>	Establece si está permitido partir palabras.

OJO: La propiedad `overflow-wrap` sólo funciona cuando `white-space` está establecida a valores que respeten espacios. Además, la propiedad `word-wrap` es un alias de `overflow-wrap` por temas de retrocompatibilidad.

Existen formas de mitigar este problema, como la propiedad `word-break`, `word-wrap` o la propiedad `hyphens`, sin embargo, aún están en fase de desarrollo y su soporte está poco extendido en la actualidad. Aún así, si quieras probar una combinación de varias propiedades que suele dar resultado para paliar este comportamiento, puedes probar lo siguiente:

```
.container {
    hyphens: auto;
    word-wrap: break-word;
    word-break: break-word;
}
```

Por otra parte, la propiedad `tab-size` permite establecer el número de espacios que se mostrarán en el cliente o navegador al representar el carácter de un TAB (*tabulador*), que generalmente se convierten a un espacio en blanco, pero sin embargo son visibles en elementos HTML como `<textarea>` o `<pre>`.



Por último, la propiedad `direction` permite establecer la dirección del texto: de izquierda a derecha (*ltr, left to right*) o de derecha a izquierda (*rtl, right to left*).

Alineaciones

También existen varias propiedades CSS que permiten modificar las diferentes alineaciones de los textos en su conjunto. Veamos un resumen de ellas:

Propiedad	Valor	Significado
<code>text-align</code>	<code>left</code> <code>center</code> <code>right</code> <code>justify</code>	Justificación del texto
<code>text-justify</code>	<code>auto</code> <code>inter-word</code> <code>inter-character</code> <code>none</code>	Método de justificación de textos
<code>text-overflow</code>	<code>clip</code> <code>ellipsis</code> <code>text</code>	Comportamiento cuando el texto no cabe «[...]»

En el primer caso, se puede establecer los valores `left`, `right`, `center` o `justify` a la propiedad `text-align` para alinear horizontalmente el texto a la izquierda, a la derecha, en el centro o justificar el texto, respectivamente, de la misma forma que lo hacemos en un procesador de texto.

En la propiedad `text-justify` indicamos el tipo de justificación de texto que el navegador realizará: automática (*el navegador elige*), ajustar el espacio entre palabras (*el resultado de ajustar con la propiedad word-spacing*), ajustar el espacio entre par de caracteres (*el resultado de ajustar con la propiedad letter-spacing*) y justificación desactivada.



Por su parte, la propiedad `text-overflow` cambia el comportamiento del navegador cuando detecta que un texto no cabe y se desborda. En ella podemos utilizar los valores `clip`, desbordar el contenedor (*comportamiento por defecto*), `ellipsis`, que muestra el texto «...» cuando no cabe más texto y por último indicar el texto que queremos utilizar en lugar de «...».

Al igual que existe `text-align` para alinear horizontalmente, también existe la propiedad `vertical-align`, que se encarga de la alineación vertical de un elemento, pudiendo establecer como valor las siguientes opciones:

Valor	¿Cómo hace la alineación?
baseline	La base del elemento con la base del elemento padre.
sub	El elemento como un subíndice.
super	El elemento como un superíndice.
top	La parte superior del elemento con la parte superior del elemento más alto de la línea.
middle	El elemento en la mitad del elemento padre.
bottom	La parte inferior del elemento con la parte inferior del elemento más bajo de esa línea.
text-top	La parte superior del elemento con la parte superior del texto padre.
text-bottom	La parte inferior del elemento con la parte inferior del texto padre.
<i>tamaño</i>	Sube o baja un elemento el tamaño o porcentaje especificado.

Consejo: Cuidado con `vertical-align`. Esta propiedad puede querer utilizarse para centrar verticalmente un elemento, sin embargo, su utilización es un poco menos intuitiva de lo que en un principio se cree, ya que se debe utilizar para alinear textos respecto a elementos. Para alinear bloques de contenido o crear estructuras de diseño, véase [Flexbox](#).

Variaciones

Por último, existen varias propiedades aplicables a los textos para variar su naturaleza. Echemos un vistazo:

Propiedad	Valor	Significado
<code>text-decoration</code>	<code>none</code> <code>underline</code> <code>overline</code> <code>line-through</code>	Indica el tipo de subrayado (decoración)
<code>text-transform</code>	<code>none</code> <code>capitalize</code> <code>uppercase</code> <code>lowercase</code>	Transforma a mayús/minús o texto capitalizado

La propiedad `text-decoration` permite establecer **subrayados** (`underline`), subrayados por encima del texto (`overline`) y tachados (`line-through`). Indicando el valor `none` se puede eliminar cualquiera de los formatos anteriores. Es muy utilizado, por ejemplo, para eliminar el subrayado de los textos que tienen un enlace o hipervínculo.

Por último, la propiedad `text-transform` es muy útil para convertir textos a mayúsculas (`uppercase`) o minúsculas (`lowercase`), o incluso capitalizar el texto (`capitalize`), es decir, poner sólo la primera letra en mayúscula, independientemente de como esté escrito en el documento HTML.

6. Representación de datos

Tablas CSS

Las [tablas de HTML](#) son un grupo de etiquetas de HTML que sirven para mostrar datos tabulados. En el pasado, se utilizaron de forma errónea para crear un diseño, debido a su facilidad de colocación al ser un esquema rectángular. Afortunadamente, hoy en día ya se ha perdido esa percepción incorrecta, aunque nunca está de más conocer un poco de historia para no repetir los errores del pasado.

Cuando queremos dar estilo a una tabla de HTML, podemos utilizar propiedades genéricas como `border`, `background`, `color`, `font-family`, `padding`, `margin`, entre otras. Combinándolas en los diferentes elementos HTML de tablas, en particular, celdas `<td>`, encabezados `<th>`, filas `<tr>` y tablas `<table>`, podemos personalizar mucho el aspecto visual de la misma.

Propiedades CSS para tablas

Sin embargo, también existen varias propiedades CSS específicas para alterar o modificar el comportamiento de ciertas características de una tablas HTML. Veamos cuáles son esas propiedades específicas para tablas:

Propiedad	Valor	Significado
<code>border-collapse</code>	<code>separate</code> <code>collapse</code>	Aplicado sobre la tabla, elimina el espacio de relleno entre celdas.
<code>border-spacing</code>	<code>0</code> <code>SIZE</code>	Amplia el espacio de relleno entre tabla y celdas.
<code>caption-side</code>	<code>top</code> <code>bottom</code>	Mueve el elemento <code><caption></code> del interior de una tabla.
<code>empty-cells</code>	<code>show</code> <code>hide</code>	Hace desaparecer visualmente una celda vacía (sin contenido).
<code>table-layout</code>	<code>auto</code> <code>fixed</code>	Indica si las celdas deben ajustarse o tener un tamaño fijo.

La propiedad `border-collapse` permite especificar si los bordes de una tabla y sus celdas deben estar unidos (`collapse`) o separados (`separate`). En el segundo caso, se puede también aplicar la propiedad `border-spacing`, que especifica el tamaño que medirán los espacios exteriores entre celdas.

La propiedad `caption-side` permite especificar donde se colocará el título de la tabla (*en el caso de haber utilizado el elemento HTML*): al principio de la tabla (`top`) o al final (`bottom`).

La propiedad `empty-cells` establece si mostrar (`show`) o no (`hide`) las celdas vacías, haciéndolas desaparecer en el último caso.

Por último, la propiedad `table-layout` permite especificar si el navegador debe adaptar el tamaño de las celdas automáticamente (`auto`) o establecer un tamaño fijo (`fixed`).

Listas CSS

De la misma forma que las tablas, las listas también poseen sus propias propiedades específicas para alterar el estilo o características de listas HTML, tanto listas ordenadas `<o1>` como listas sin orden ``.

Listas CSS

Por ejemplo, las siguientes propiedades:

Propiedad	Valor	Significado
<code>list-style-image</code>	<code>none</code> <code>url(image.png)</code>	Especifica una imagen para usar como «punto» o viñeta de ítem.
<code>list-style-position</code>	<code>inside</code> <code>outside</code>	Establece o elimina indentación de ítems sobre la lista.

La primera propiedad, `list-style-image` permite indicar la URL de una imagen para utilizar de icono o viñeta en cada ítem de la lista. Con el valor `none` eliminamos el uso de cualquier imagen.

Por otro lado, la propiedad `list-style-position` permite establecer una indentación a todos los ítems de la lista, estableciéndolos desplazados a la derecha (`inside`) o sin desplazar (`outside`).

La tercera propiedad, `list-style-type` nos permite indicar qué tipo de numeración tendrán las listas (*en el caso de no estar utilizando ningún imagen*). Se establecen tres grupos. El primero indicado para las listas que no requieren orden, el segundo para las listas numeradas y el tercero para las listas numeradas que queremos especificar con letras (*números romanos, letras griegas, etc...*).

Veamos cada uno de los valores posibles:

Propiedad	Valor	Significado
<code>list-style-type</code>	<code>disc</code> <code>circle</code> <code>square</code> <code>none</code>	Viñetas sin orden
<code>list-style-type</code>	<code>decimal</code> <code>decimal-leading-zero</code> <code>lower-roman</code> <code>upper-roman</code>	Viñetas numéricas
<code>list-style-type</code>	<code>lower-alpha</code> <code>upper-alpha</code> <code>lower-greek</code>	Viñetas alfabéticas

El primer grupo, indicado para listas sin orden ``:

- **disc**: Un pequeño círculo relleno.
- **circle**: Un círculo vacío.
- **square**: Un cuadrado relleno.
- **none**: No muestra ninguna marca a la izquierda de los ítems.

Si lo que queremos es establecer una lista numerada `<o1>`, podemos utilizar valores como:

- **decimal**: Numeración decimal: 1, 2, 3, 4, 5...
- **decimal-leading-zero**: Numeración decimal con ceros: 01, 02, 03, 04, 05...
- **lower-roman**: Números romanos en minúsculas: i, ii, iii, iv, v...
- **upper-roman**: Números romanos en mayúsculas: I, II, III, IV, V...
- **lower-alpha / lower-latin**: Minúsculas: a, b, c, d, e...
- **upper-alpha / upper-latin**: Mayúsculas: A, B, C, D, E...

- Otras menos utilizadas generalmente, como **lower-greek** (letras griegas minúsculas), **arabic-indic** (números árabes), **upper-armenian / armenian** (letras armenias en mayúsculas), **lower-armenian** (letras armenias en minúsculas) o **georgian** (letras georgianas).

Existen otros valores como `bengali`, `cambodian`, `hebrew`, `devanagari`, `gujarati`, `gurmukhi`, `kannada`, `lao`, `malayalam`, `mongolian`, `myanmar`, `oriya`, `persian`, `tamil`, `telugu`, `thai`, `tibetan`, `hiragana`, `hiragana-iroha`, `katakana`, `katakana-iroha`, entre otros.

Atajo: Listas

Es posible utilizar la propiedad de atajo `list-style` para especificar varias propiedades en una sola. El orden aconsejado es el siguiente:

```
div {  
    /* list-style: <type> <position> <image> */  
    list-style: circle inside none;  
}
```

7. Maquetación y colocación

Tipos de elementos

Una de las partes más complejas de CSS, probablemente, sea la colocación y distribución de los elementos de una página. Sin embargo, suele ser difícil porque se desconocen los detalles particulares que componen CSS. Una vez se analiza y se comprenden los detalles, todo resulta más fácil.

Para comenzar, hay que saber que cada etiqueta HTML tiene un tipo de representación visual en un navegador, lo que habitualmente se suele denominar el **tipo de caja**. En principio, se parte de dos tipos básicos: **inline** y **block**.

Valor	Denominación	Significado	Ejemplo
inline	Elemento en línea	El elemento se coloca en horizontal (un elemento a continuación del otro).	<code></code>
block	Elemento en bloque	El elemento se coloca en vertical (un elemento encima de otro).	<code><div></code>

Obsérvese que por defecto, todos los elementos `<div>` son elementos de bloque (*block*) y todos los elementos `` son elementos en línea (*inline*). Para entender esto fácilmente, vamos a crear un HTML con 3 etiquetas `<div>` como las siguientes:

```
<div>Elemento 1</div>
<div>Elemento 2</div>
<div>Elemento 3</div>
```

A estas etiquetas HTML le vamos a aplicar el siguiente código CSS:

```
div {
    background: blue;
    color: white;
    margin: 1px;
}
```

Con esto observaremos que en nuestro navegador nos aparecen 3 cajas azules colocadas en vertical (*una debajo de otra*) que cubren todo el ancho disponible de la página. Esto ocurre porque la etiqueta `<div>` es un elemento en bloque, o lo que es lo mismo, que tiene un tipo de representación **block** por defecto. Cada etiqueta HTML tiene un tipo de representación concreta.

Sin embargo, este comportamiento de elementos puede cambiarse con la propiedad CSS `display`. Tan sencillo como añadir `display: inline` en el ejemplo anterior y veremos como pasan a ser 3 cajas azules colocadas en horizontal (*una al lado de la otra*) que cubren sólo el ancho del contenido de cada una. Ahora los `<div>` de esa página son **elementos en línea** (*el tipo de representación visual que tienen los *).

Otros tipos de elementos

A medida que vamos cambiando el tipo de representación de estos elementos, nos damos cuenta que es insuficiente para realizar tareas y vamos necesitando más tipos de caja.

Vamos a llenar un poco más la tabla, con las características más importantes de las opciones que puede tomar la propiedad CSS `display`:

Tipo de caja	Características
block	Se apila en vertical. Ocupa todo el ancho disponible de su etiqueta contenedora.
inline	Se coloca en horizontal. Se adapta al ancho de su contenido. Ignora <code>width</code> o <code>height</code> .
inline-block	Combinación de los dos anteriores. Se comporta como <code>inline</code> pero no ignora <code>width</code> o <code>height</code> .
flex	Utiliza el modelo de cajas flexibles Flexbox . Muy útil para diseños adaptables.
inline-flex	La versión en línea (ocupa sólo su contenido) del modelo de cajas flexibles flexbox.
grid	Utiliza cuadrículas o rejillas con el modelo de cajas Grid CSS .
inline-grid	La versión en línea (ocupa sólo su contenido) del modelo de cajas grid css.
list-item	Actúa como un ítem de una lista. Es el comportamiento de etiquetas como <code></code> .
table	Actúa como una tabla. Es el comportamiento de etiquetas como <code><table></code> .
table-cell	Actúa como la celda de una tabla. Es el comportamiento de etiquetas como <code><th></code> o <code><td></code> .
table-row	Actúa como la fila de una tabla. Es el comportamiento de etiquetas como <code><tr></code> .

Ocultar elementos

En la lista anterior, falta un valor de la propiedad `display`. Mediante la mencionada propiedad, es posible aplicar un valor `none` y ocultar completamente elementos que no queramos que se muestren, los cuales desaparecen por completo. Es muy útil para hacer desaparecer información cuando el usuario realiza alguna acción, por ejemplo.

Tipo de caja	Características
none	Hace desaparecer visualmente el elemento, como si no existiera.

No obstante, también existe una propiedad CSS llamada `visibility` que realiza la misma acción, con la ligera diferencia de que no sólo oculta el elemento, sino que además mantiene un vacío con el mismo tamaño de lo que antes estaba ahí.

Dicha propiedad `visibility` tiene los siguientes valores posibles:

Valor	Significado
<code>visible</code>	El elemento es visible. Valor por defecto.
<code>hidden</code>	El elemento no es visible pero sigue ocupando su espacio y posición.
<code>collapse</code>	Sólo para tablas. El elemento se contrae para no ocupar espacio.

Utilizar `visibility:hidden` es muy interesante si queremos que un elemento y su contenido se vuelva invisible, pero siga ocupando su espacio y así evitar que los elementos adyacentes se desplacen, lo que suele ser un comportamiento no deseado en algunas ocasiones cuando se aplica `display: none`.

Otra opción interesante es utilizar la propiedad `opacity` junto a transiciones o animaciones, desplazarse desde el valor `0` al `1` o viceversa. De esta forma conseguimos una animación de aparición o desvanecimiento.

Posicionamiento

A grandes rasgos, y como aprendimos en temas anteriores, si tenemos varios **elementos en línea** (*uno detrás de otro*) aparecerán colocados de **izquierda hacia derecha**, mientras que si son **elementos en bloque** se verán colocados desde **arriba hacia abajo**. Estos elementos se pueden ir combinando y anidando (*incluyendo unos dentro de otros*), construyendo así esquemas más complejos.

Hasta ahora, hemos estado trabajando sin saberlo en lo que se denomina posicionamiento **estático** (*static*), donde todos los elementos aparecen con un orden natural según donde estén colocados en el HTML. Este es el **modo por defecto** en que un navegador renderiza una página.

Sin embargo, existen otros modos alternativos de posicionamiento, que podemos cambiar mediante la propiedad `position`, que nos pueden interesar para modificar la posición en donde aparecen los diferentes elementos y su contenido.

A la propiedad `position` se le pueden indicar los siguientes valores:

Valor	Significado
<code>static</code>	Posicionamiento estático. Utiliza el orden natural de los elementos HTML.
<code>relative</code>	Posicionamiento relativo. Los elementos se mueven ligeramente en base a su posición estática.
<code>absolute</code>	Posicionamiento absoluto. Los elementos se colocan en base al contenedor padre.
<code>fixed</code>	Posicionamiento fijo. Idem al absoluto, pero aunque hagamos scroll no se mueve.

Si utilizamos un modo de posicionamiento diferente al estático (*absolute, fixed o relative*), podemos utilizar una serie de propiedades para modificar la posición de un elemento. Estas propiedades son las siguientes:

Propiedad	Valor	Significado
top:	auto SIZE	Empuja el elemento una distancia desde la parte superior hacia el inferior.
bottom:	auto SIZE	Empuja el elemento una distancia desde la parte inferior hacia la superior.
left:	auto SIZE	Empuja el elemento una distancia desde la parte izquierda hacia la derecha.
right:	auto SIZE	Empuja el elemento una distancia desde la parte derecha hacia la izquierda.
z-index:	auto NUMBER	Coloca un elemento en el eje de profundidad, más cerca o más lejos del usuario.

Antes de pasar a explicar los tipos de posicionamiento, debemos tener claras las propiedades `top`, `bottom`, `left` y `right`, que sirven para mover un elemento desde la orientación que su propio nombre indica hasta su extremo contrario. Esto es, si utilizamos `left` e indicamos `20px`, estaremos indicando mover **desde la izquierda** 20 píxeles **hacia la derecha**.

Pero pasemos a ver cada tipo de posicionamiento por separado y su comportamiento:

Posicionamiento relativo

Si utilizamos la palabra clave `relative` activaremos el modo de **posicionamiento relativo**, que es el más sencillo de todos. En este modo, los elementos se colocan exactamente igual que en el posicionamiento estático (*permanecen en la misma posición*), pero dependiendo del valor de las propiedades `top`, `bottom`, `left` o `right` variaremos ligeramente la posición del elemento.

Ejemplo: Si establecemos `left:40px`, el elemento se colocará 40 píxeles a la derecha **desde la izquierda** donde estaba colocado en principio, mientras que si especificamos `right:40px`, el elemento se colocará 40 píxeles a la izquierda **desde la derecha** donde estaba colocado en principio.

Posicionamiento absoluto

Si utilizamos la palabra clave `absolute` estamos indicando que el elemento pasará a utilizar **posicionamiento absoluto**, que no es más que utilizar el documento completo como referencia. Esto no es exactamente el funcionamiento de este modo de posicionamiento, pero nos servirá como primer punto de partida para entenderlo.

Ejemplo: Si establecemos `left:40px`, el elemento se colocará 40 píxeles a la derecha del extremo izquierdo de la página. Sin embargo, si indicamos `right:40px`, el elemento se colocará 40 píxeles a la izquierda del extremo derecho de la página.

Como mencionaba anteriormente, aunque este es el funcionamiento del posicionamiento absoluto, hay algunos detalles más complejos en su modo de trabajar. Realmente, este tipo de posicionamiento coloca los elementos **utilizando como punto de origen el primer contenedor con posicionamiento diferente a estático**.

Por ejemplo, si el contenedor padre tiene posicionamiento estático, pasamos a mirar el posicionamiento del padre del contenedor padre, y así sucesivamente hasta encontrar un contenedor con posicionamiento no estático o llegar a la etiqueta `<body>`, en el caso que se comportaría como el ejemplo anterior.

Posicionamiento fijo

Por último, el **posicionamiento fijo** es hermano del **posicionamiento absoluto**. Funciona exactamente igual, salvo que hace que el elemento se muestre en una posición fija **dependiendo de la región visual del navegador**. Es decir, aunque el usuario haga scroll y se desplace hacia abajo en la página web, el elemento seguirá en el mismo sitio posicionado.

Ejemplo: Si establecemos `top:0` y `right:0`, el elemento se colocará justo en la esquina superior derecha y se mantendrá ahí aunque hagamos scroll hacia abajo en la página.

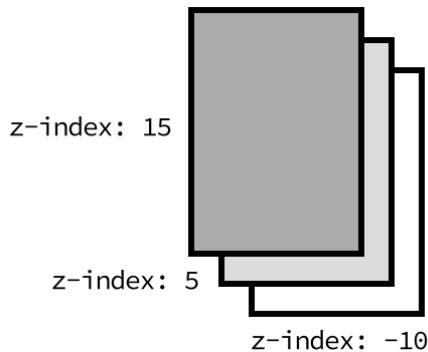
Otros posicionamientos

Existen otros valores de `position` como `sticky`, `page` o `center`, sin embargo, su soporte e implementación aún es muy temprana y no se sabe si su funcionalidad terminará ofreciéndose desde los diferentes navegadores.

Profundidad (niveles)

Es interesante conocer también la existencia de la propiedad `z-index`, que establece el **nivel de profundidad** en el que está un elemento sobre los demás. De esta forma, podemos hacer que un elemento se coloque encima o debajo de otro.

Su funcionamiento es muy sencillo, sólo hay que indicar un número que representará el nivel de profundidad del elemento. Los elementos un número más alto estarán por encima de otros con un número más bajo, que permanecerán ocultos detrás de los primeros.



Nota: Los niveles `z-index`, así como las propiedades `top`, `left`, `bottom` y `right` no funcionan con elementos que estén utilizando posicionamiento estático. Deben tener un tipo de posicionamiento diferente a estático.

Desplazamientos

Es posible que en algún momento necesitemos algo más de control sobre nuestra página y realizar cambios en determinados elementos. Existe una propiedad denominada `float` que tiene un funcionamiento peculiar con el que cambiamos el flujo de ordenación de elementos.

Con `float` podemos conseguir que un elemento «flete» a la izquierda o a la derecha de otro elemento. Para ello podemos utilizar las siguientes propiedades:

Propiedad	Valor	Significado
<code>float</code>	<code>none</code> <code>left</code> <code>right</code>	Cambia el flujo para que el elemento flote a la izquierda o derecha.
<code>clear</code>	<code>none</code> <code>left</code> <code>right</code> <code>both</code>	Impide que los elementos puedan flotar en la orientación indicada.

Elementos flotantes

Con la propiedad `float` puedes conseguir que los elementos que quieras, alteren su comportamiento y floten a la izquierda (`left`) o a la derecha (`right`). Con el valor `none` (*valor por defecto*) eliminas esta característica de desplazamiento.

Imaginemos que tenemos un párrafo de texto, seguido de una lista, seguida de otro párrafo de texto:

```
ul {
  background: grey;
}

li {
  background: blue;
  width: 100px;
  padding: 8px;
  margin: 8px;
  color: white;
}

ul, li {
  float: left;
}
```

Con esto conseguimos que los ítems de la lista floten uno a continuación de otro. No obstante, para conseguir este comportamiento siempre recomiendo utilizar `display` en lugar de `float`. Cambiando la representación de elementos se suele conseguir una solución más limpia y organizada.

Limpiar flujo flotante

Por otro lado, la propiedad `clear` se encarga de impedir elementos flotantes en la zona indicada, a la izquierda del elemento (`left`), a la derecha (`right`) o en ambos lados (`both`).

En el ejemplo anterior, el segundo párrafo de texto aparecería a continuación de la lista, cuando probablemente, nuestra intención es que apareciera en la parte inferior. Se podría solucionar simplemente añadiendo el siguiente texto:

```
p {
  clear:both
}
```

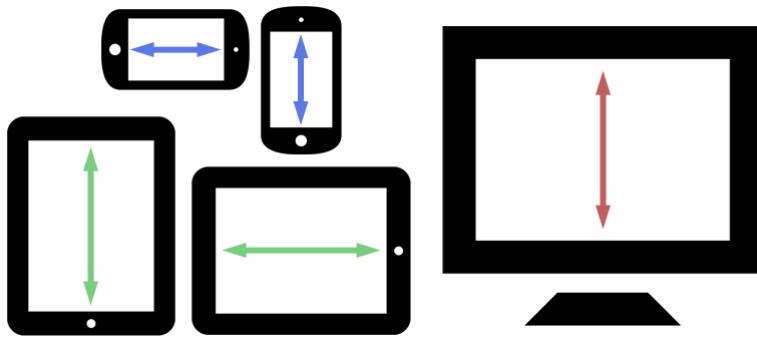
OJO: La propiedad `float` es una propiedad que podría ser interesante en determinadas condiciones, sin embargo, el código resultante suele ser más sucio y complejo de lo que sería mediante otros métodos actuales, por lo que se recomienda sólo utilizarlo por razones de retrocompatibilidad y darle preferencia a métodos como [Flexbox](#) o [Grid CSS](#).

8. Responsive Web Design

¿Qué es Responsive Design?

En la actualidad, el uso de todo tipo de **dispositivos móviles** se ha disparado, no sólo de «smartphones», sino también de tablets, «smartwatches», lectores de ebooks y múltiples tipos de dispositivos con capacidad de conexión a Internet.

Cada vez es más frecuente acceder a Internet con diferentes tipos de dispositivos, que a su vez tienen **diferentes pantallas y resoluciones**, con distintos tamaños y formas, que hacen que se consuman las páginas webs de formas diferentes, apareciendo por el camino también diferentes necesidades, problemas y soluciones.

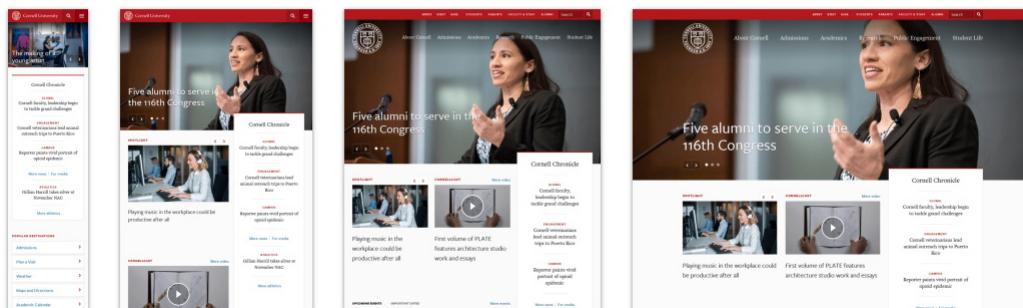


Por lo tanto, en la actualidad, cuando diseñamos una web, esta debe estar preparada para verse correctamente en diferentes resoluciones, cosa que, a priori no es sencilla. Antiguamente, se llegó al punto de preparar una web diferente dependiendo del dispositivo o navegador que utilizaba el usuario, pero era algo que se terminó descartando, ya que no era práctico.

Por suerte, esos tiempos han quedado atrás, y la máxima que se sigue hoy es **diseñar una sola web, que se adapte visualmente al dispositivo utilizado**.

Hoy en día se le denomina **Responsive Web Design** (o RWD) a los diseños web que tienen la capacidad de adaptarse al tamaño y formato de la pantalla en la que se visualiza el contenido, respecto a los diseños tradicionales en los que las páginas web estaban diseñadas sólo para un tamaño o formato específico, y no tenían esa capacidad de adaptación.

Cornell University

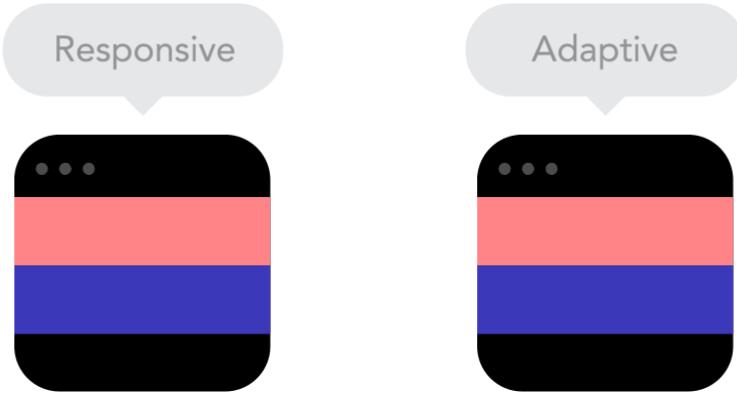


Aunque en principio el concepto de **web adaptativa** es muy sencillo de comprender, aplicarlo puede ser todo un quebradero de cabeza si no se conocen bien las bases y se adquiere experiencia. En [MediaQuer.es](#) puedes encontrar algunos ejemplos de páginas que utilizan Responsive Web Design para tener clara la idea.

Conceptos básicos

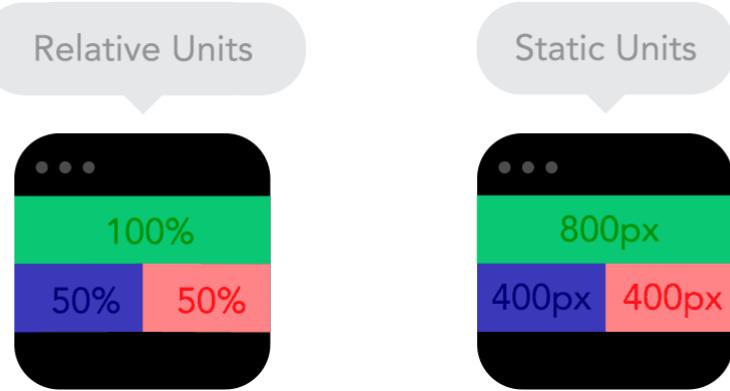
En el excelente artículo [9 basic principles of responsive web design, de Froont](#), hay una estupenda explicación visual de algunos conceptos básicos necesarios para entender correctamente el **Responsive Web Design**. Son los siguientes:

El primero de ellos es la diferencia entre **diseño responsive** y **diseño adaptativo**. Como se puede ver en la imagen a continuación, un diseño **responsive** responde (*valga la rebuznancia*) en todo momento a las dimensiones del dispositivo, mientras que un diseño adaptable es aquel que se adapta, pero no necesariamente responde en todo momento:



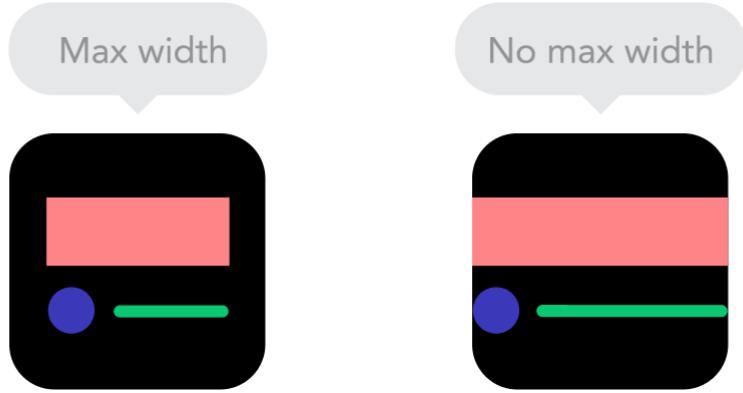
Veremos que esto puede ser la diferencia entre aplicar correctamente conceptos como media queries, porcentajes y propiedades de ancho máximo y mínimo, que veremos más adelante.

Por otro lado, para trabajar correctamente en diseños **responsive** hay que tener en cuenta que debemos trabajar con unidades relativas e intentar evitar las unidades fijas o estáticas, las cuales no responden a la adaptación de nuestros diseños flexibles:



Otra forma interesante de trabajar esa respuesta de los diseños **responsive** es utilizar propiedades como `min-width` o `max-width`, donde definimos tamaños mínimos o máximos, para que los elementos de nuestra página puedan ampliar o reducirse según sea necesario dependiendo de la pantalla del dispositivo utilizado.

Con estas propiedades podemos crear diseños que aprovechen al máximo toda la pantalla de dispositivos pequeños (*como móviles o tablets*), mientras que establecemos unos máximos en pantallas de dispositivos grandes, para crear unos espacios visuales que hacen que el diseño sea más agradable:



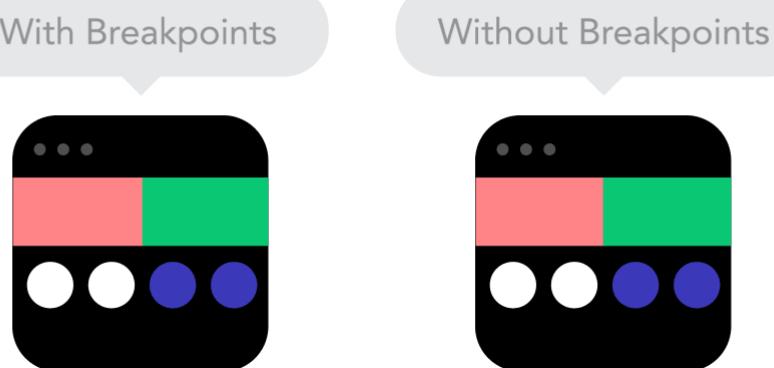
Otro concepto, que a la misma vez es una característica muy atractiva en nuestros diseños responsive es la de mantener el flujo de los elementos cuando cambian de tamaño y evitar que estos se solapen unos con otros.

Si estamos habituados a trabajar en diseños más estáticos que no están preparados para móviles, suele ser duro hacer ese cambio. Sin embargo, una vez lo conseguimos, todo resulta mucho más fácil y conseguiremos transmitir una buena respuesta y fluidez visual:



Esto último va muy de la mano del sistema habitual de recolocación de elementos que se suele seguir en los diseños **Responsive Design**. Como se puede ver en la siguiente imagen, en un diseño responsive se utilizan ciertos «puntos de control».

Por ejemplo, se suele pensar que en una resolución de escritorio queremos mostrar la información dentro de una cuadrícula (*grid*) de 4 ó 5 celdas de ancho, mientras que en la versión de tablet será sólo de 3 celdas de ancho (*el resto se desplazará a la siguiente fila*) y en móviles será una sola celda de ancho, mostrándose el resto de celdas haciendo scroll hacia abajo:



Esta forma de trabajar nos proporciona múltiples ventajas:

- Es mucho más sencillo mostrar la misma información desde diseños de pantalla grande.
- Ayuda a evitar la mala práctica de ocultar bloques de información en dispositivos móviles.
- Incentiva a diseñar siguiendo buenas prácticas para facilitar la creación responsive.

Preparación previa

Antes de comenzar a crear un diseño web preparado para móviles, es importante tener claro ciertos detalles:

- A priori, ¿Cuál es tu público objetivo? ¿móvil o escritorio? ¿ambos?
- Debes conocer las resoluciones más utilizadas por tu público potencial
- Debes elegir una estrategia acorde a los datos anteriores

En primer lugar, es importante **conocer los formatos** de pantalla más comunes con los cuales nos vamos a encontrar. Podemos consultar páginas como [MyDevices](#), la cuál tiene un apartado de [comparación de dispositivos](#), donde se nos muestra un listado de dispositivos categorizados en smartphones, tablets u otros dispositivos con las características de cada uno: dimensiones de ancho, alto, radio de píxeles, etc...

Una vez estés familiarizado con estos detalles, es importante **conocer el público de tu sitio web**. ¿Acceden más usuarios desde móvil o desde escritorio? ¿Predominan las tablets o los móviles? ¿Tu objetivo es tener más usuarios de móvil o de escritorio?

Consulta con algún sistema de estadísticas como [Google Analytics](#) para comprobar que tipo de público tienes actualmente. También es aconsejable echar un vistazo a información externa como las que nos proporcionan estadísticas globales anónimas de [Global StatCounter](#), para hacernos una idea de los atributos más comunes.

Estrategias de diseño

Por último, es aconsejable decidirse por una estrategia de diseño antes de comenzar. Aunque existen otras estrategias, las dos vertientes principales más populares son las siguientes:

Estrategia	Descripción
Mobile first	Primero nos enfocamos en dispositivos móviles y luego pensamos en otros.
Desktop first	Primero nos enfocamos en dispositivos de escritorio, y luego pensamos en otros.

La estrategia **Mobile-first** es la que utilizan los diseñadores de sitios webs en las que su público objetivo es mayoritariamente usuario de móvil. Ejemplos como una web para comprar billetes de transporte, la web de un juego o aplicación móvil o una web para pedir cita en un restaurante podrían ser, a priori, una buena elección para utilizar **Mobile-first**.

Esta estrategia hace que el desarrollo en escritorio sea muy sencillo, ya que se reduce a tener un diseño de móvil en escritorio e ir añadiendo nuevas secciones o partes para «completar» el diseño en resoluciones grandes.

Por otro lado, la estrategia **Desktop-first** suele interesar más a los diseñadores de sitios webs en las que el público objetivo son usuarios de escritorio. Por ejemplo, una página de una aplicación para PC/Mac o similares, podría ser una buena opción para la estrategia **Desktop-first**. En ella, hacemos justo lo contrario que en la anterior, lo primero que diseñamos es la versión de escritorio, y luego vamos descargando detalles o recolocando información hasta tener la versión para dispositivos móviles.

Bases del Responsive Design

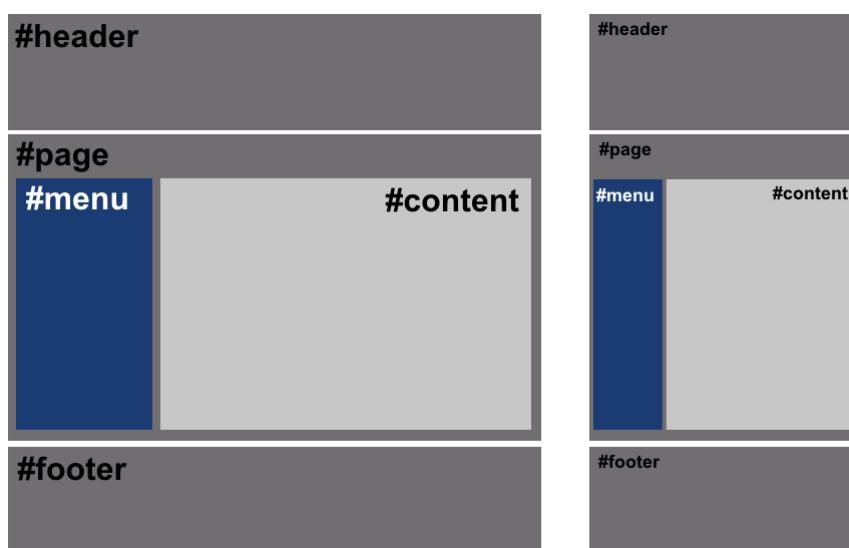
Como explicamos en el capítulo anterior, hay ciertos conceptos que hay que tener claros antes de comenzar con el **Responsive Design**. En esta sección vamos a ver como llevarlos a la práctica con código.

Diseño con porcentajes

El primer paso para crear un diseño que se adapte correctamente, es comenzar a familiarizarse con un tipo de unidades relativas: **los porcentajes**. Recordemos que los porcentajes son relativos al contenedor padre, por lo que si especificamos un porcentaje a un elemento, el navegador va a tomar dicho porcentaje del contenedor.

Nota: Al comenzar, algunos diseñadores tienen una percepción incorrecta de que los porcentajes toman el tamaño dependiendo de lo que mide la ventana del navegador. Realmente los porcentajes dependen siempre del tamaño del elemento padre que los contiene. Si queremos basarnos en el tamaño del navegador, hay que usar [unidades de viewport](#).

Podemos comenzar usando porcentajes con las propiedades `width` en un ejemplo sencillo. Si establecemos un ancho de `100%` (*valor por defecto en elementos de tipo block, no hace falta indicarlo*) a los elementos grises que vemos a continuación (`#header`, `#page` y `#footer`), un `30%` al azul (`#menu`) y un `70%` al gris claro (`#content`) podríamos obtener este diseño:



El código utilizado sería algo parecido a lo siguiente:

```
<div id="header"></div>
<div id="page">
  <div id="menu"></div>
  <div id="content"></div>
</div>
<div id="footer"></div>
```

Nótese que los elementos `#menu` y `#content` se encuentran dentro de `#page`. Tengan en cuenta que, estamos utilizando **id** en este ejemplo, aunque habíamos comentado que lo ideal quizás sería utilizar clases.

Por su parte, el código CSS tendría esta pinta:

```
div {
  /* Ponemos un alto mínimo, por defecto es 0 */
  min-height: 200px;
```

```

/* Dibujamos un borde para ver límites */
border: 2px solid black;
}

#header, #page, #footer {
background: grey;
}

#menu, #content {
/* Necesario para que los elementos estén en horizontal */
display: inline-block;
}

#menu {
background: blue;
width: 30%;
}

#content {
background: lightgrey;
width: 70%;
}

```

Sin embargo, utilizar porcentajes no nos garantiza un diseño adaptativo de calidad, hay que comprender otros detalles. El primer problema que encontraremos será que si sumamos el tamaño de los elementos ($70\% + 30\%$) junto a los bordes (*2px por cada lado*), la suma es superior al 100% del contenedor padre, por lo que **no cabe en su interior** y el segundo elemento se desplaza a la zona inferior, descuadrando todo el diseño. Lo mismo puede ocurrir si intentamos añadir `margin` o `padding`. Esto es algo muy habitual en CSS. Y frustrante al principio.

Hay varias formas de solucionar esto:

- Eliminar los bordes y reducir los porcentajes hasta que quepan en el 100% del padre.
- Usar `box-sizing: border-box` para cambiar el modo en el que se gestionan los tamaños.
- Utilizar un sistema moderno como [Flexbox](#) o [Grid](#) (*recomendado*).

Una forma simple de solucionar el problema en el ejemplo anterior, es hacer los siguientes cambios en el CSS del documento:

```

/* Eliminamos este bloque */
#menu, #content {
display: inline-block;
}

/* Añadimos este */
#page {
display: flex;
}

```

De esta forma, conseguimos que nuestro diseño se adapte de forma adecuada a la página, sin necesidad de tener que ajustar los márgenes, rellenos, bordes o tamaño de los contenidos.

Tamaños máximos y mínimos

Si buscamos un cierto grado de control aún mayor, podríamos recurrir a las propiedades `max-width` y `min-width`, con las que podemos indicar el ancho de un elemento como máximo y el ancho de un elemento como mínimo respectivamente, consiguiendo así garantizar cierto control del diseño:

```
.picture {  
    min-height: 200px;      /* Por defecto, height es 0 */  
    background: grey;      /* Simplemente, para verlo visualmente */  
  
    max-width: 1024px;  
    min-width: 800px;  
}
```

En este caso, el elemento tiene un tamaño máximo de 1024 píxeles, y un tamaño mínimo de 800 píxeles, por lo que si ajustamos el ancho de la ventana del navegador, dicho elemento iría variando en un rango de 800 a 1024 píxeles, nunca haciéndose más pequeño de 800 o más grande de 1024.

Nota: Es importante darse cuenta de que este ejemplo funciona porque no hay definido un `width` (*por omisión, es igual a `width: 100%`*). Desde que existe un `width`, las otras propiedades pierden efecto porque se está obligando a que tenga un tamaño fijo concreto.

Con las imágenes, videos y contenidos multimedia, se puede hacer lo mismo, consiguiendo así que las imágenes se escalen y adapten al formato especificado o incluso al tamaño de pantalla de los diferentes dispositivos utilizados:

```
img,  
video,  
object,  
embed {  
    max-width: 100%;  
    height: auto;  
}
```

El `viewport`

En muchos casos puede que oigas hablar del **viewport** del navegador. Esa palabra hace referencia a la **región visible del navegador**, o sea, la parte de la página que está visualizándose actualmente en el navegador. Los usuarios podemos redimensionar la ventana del navegador para reducir el tamaño del viewport y simular que se trata de una pantalla y dispositivo más pequeño.

Si queremos editar ciertos comportamientos del viewport del navegador, podemos editar el documento HTML para especificar el siguiente campo meta, antes de la parte del `</head>`:

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

Con esta etiqueta `<meta>`, estamos estableciendo unos parámetros de comportamiento para el **viewport** del navegador. Veamos que significan y cuales más existen:

Propiedades	Valor	Significado
width	device-width	Indica un ancho para el viewport.
height	device-height	Indica un alto para el viewport.
initial-scale	1	Escala inicial con la que se visualiza la página web.
minimum-scale	0.1	Escala mínima a la que se puede reducir al hacer zoom.
maximum-scale	10	Escala máxima a la que se puede aumentar al hacer zoom.
user-scalable	no/fixed yes/zoom	Posibilidad de hacer zoom en la página web.

Las propiedades `initial-scale`, `minimum-scale` y `maximum-scale` permiten valores desde el `0.1` al `10`, aunque ciertos valores se traducen automáticamente a ciertos números determinados:

- yes = 1
- no = 0.1
- device-width = 10
- device-height = 10

Por otra parte, `user-scalable` permite definir si es posible que el usuario pueda «pellizcar» la pantalla para ampliar o reducir el zoom.

Ojo: Aunque es posible utilizar algunos de estos parámetros, es aconsejable revisar detalladamente las consecuencias de especificar estos parámetros. Lo recomendable es utilizar sólo los que se mencionan en el fragmento de código superior, para evitar problemas de accesibilidad que impidan ciertas acciones.

Media Queries

Una vez nos adentramos en el mundo del **Responsive Design**, nos damos cuenta en que hay situaciones en las que determinados aspectos o componentes visuales deben aparecer en un tipo de dispositivos, o deben existir ciertas diferencias.

Por ejemplo, una zona donde se encuentra el buscador de la página puede estar colocada en un sitio concreto en la versión de escritorio, pero en móvil quizás nos interesa que ocupe otra zona (*o que tenga otro tamaño o forma*) para aprovechar mejor el poco espacio que tenemos en la versión móvil de la página.

Para ello, utilizaremos un concepto denominado **media queries**, con los que podemos hacer esas excepciones para que sólo se apliquen a un tipo de diseño concreto.

¿Qué son las media queries?

Las reglas **media queries** (*también denominadas MQ a veces*) son un tipo de reglas de CSS que permiten crear un bloque de código que sólo se procesará en los dispositivos que cumplan los criterios especificados como condición:

```

@media screen and (*condición*) {
    /* reglas CSS */
    /* reglas CSS */
}

@media screen and not (*condición*) {
    /* reglas CSS */
    /* reglas CSS */
}

```

Con este método, especificamos que queremos aplicar los estilos CSS para tipos de medios concretos (`screen`: sólo en pantallas, en este caso) que cumplan las condiciones especificadas entre paréntesis. De esta forma, una estrategia aconsejable es crear reglas CSS generales (como *hemos hecho hasta ahora*) aplicadas a todo el documento: colores, tipo de fuente, etc. y luego, las particularidades que se aplicarían sólo en el dispositivo en cuestión.

Aunque suele ser menos habitual, también se pueden indicar reglas `@media` negadas mediante la palabra clave `not`, que aplicará CSS siempre y cuando no se cumpla una determinada condición. También pueden separarse por comas varias condiciones de media queries.

Truco: Al igual que `not`, también existe una palabra clave `only` que, suele usarse a modo de **hack**. El comportamiento por defecto ya incluye los dispositivos que encajan con la condición, pero con la palabra clave `only` conseguimos que navegadores antiguos que no la entienden, no procesen la información, dejándola sólo para navegadores modernos.

Existen los siguientes **tipos de medios**:

Tipo de medio	Significado
<code>screen</code>	Monitores o pantallas de ordenador. Es el más común.
<code>print</code>	Documentos de medios impresos o pantallas de previsualización de impresión.
<code>speech</code>	Lectores de texto para invidentes (Antes <code>aural</code> , el cuál ya está obsoleto).
<code>all</code>	Todos los dispositivos o medios. El que se utiliza por defecto .

Otros tipos de medios como `braille`, `embossed`, `handheld`, `projection`, `tty` o `tv` aún son válidos, pero están marcados como obsoletos a favor de utilizar tipos de medios de la lista anterior y restringir sus características posteriormente.

Recordemos que con el siguiente fragmento de código HTML estamos indicando que el nuevo ancho de la pantalla es **el ancho del dispositivo**, por lo que el aspecto del viewport se va a adaptar consecuentemente:

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



Con esto conseguiremos preparar nuestra web para dispositivos móviles y prepararnos para la introducción de reglas **media query** en el documento CSS.

Ejemplos de media queries

Veamos un ejemplo clásico de **media queries** en el que definimos diferentes estilos dependiendo del dispositivo que estamos utilizando. Observese que en el código existen 3 bloques `@media` donde se definen estilos CSS para cada uno de esos tipos de dispositivos.

El código sería el siguiente:

```
@media screen and (max-width: 640px) {  
    .menu {  
        background: blue;  
    }  
}  
  
@media screen and (min-width: 640px) and (max-width: 1280px) {  
    .menu {  
        background: red;  
    }  
}  
  
@media screen and (min-width: 1280px) {  
    .menu {  
        background: green;  
    }  
}
```

El ejemplo anterior muestra un elemento (*con clase menu*) con un color de fondo concreto, dependiendo del tipo de medio con el que se visualice la página:

- **Azul** para resoluciones menores a **640 píxeles** de ancho (*móviles*).
- **Rojo** para resoluciones entre **640 píxeles y 1280 píxeles** de ancho (*tablets*).
- **Verde** para resoluciones mayores a **1280 píxeles** (*desktop*).

El número de bloques de reglas `@media` que se utilicen depende del desarrollador web, ya que no es obligatorio utilizar un número concreto. Se pueden utilizar desde un sólo media query, hasta múltiples de ellos a lo largo de todo el documento CSS.

Hay que tener en cuenta que los **media queries** también es posible indicarlos desde HTML, utilizando la etiqueta `<link>`:

```
<link rel="stylesheet" href="mobile.css"
      media="screen and (max-width: 640px)">

<link rel="stylesheet" href="tablet.css"
      media="screen and (min-width: 640px) and (max-width: 1280px)">

<link rel="stylesheet" href="desktop.css"
      media="screen and (min-width: 1280px)">
```

Estos estilos quedarán separados en varios archivos diferentes. Ten en cuenta que todos serán descargados al cargar la página, sólo que no serán aplicados al documento hasta que cumplan los requisitos indicados en el atributo `media`.

Tipos de características

En los ejemplos anteriores solo hemos utilizado `max-width` y `min-width` como tipos de características a utilizar en condiciones de media query. Sin embargo, tenemos una lista de tipos de características que podemos utilizar:

Tipo de característica	Valores	¿Cuándo se aplica?
<code>width</code>	<code>SIZE</code>	Si el dispositivo tiene el tamaño indicado exactamente.
<code>min-width</code>	<code>SIZE</code>	Si el dispositivo tiene un tamaño de ancho mayor al indicado.
<code>max-width</code>	<code>SIZE</code>	Si el dispositivo tiene un tamaño de ancho menor al indicado.
<code>aspect-ratio</code>	<code>aspect-ratio</code>	Si el dispositivo encaja con la proporción de aspecto indicada.
<code>orientation</code>	<code>landscape portrait</code>	Si el dispositivo está en colocado en modo vertical o apaisado.

Existen otras características minoritarias que en algunos casos límite pueden ser interesantes, como por ejemplo `scan`, `resolution`, `monochrome`, `grid`, `color-index`, `color`, etc...

Condicionales CSS

Aunque no forman parte de las media queries en sí, podemos utilizar la regla `@supports` para establecer condicionales y crear reglas similares a `@media` pero dependiendo de si el navegador del usuario soporta una característica concreta.

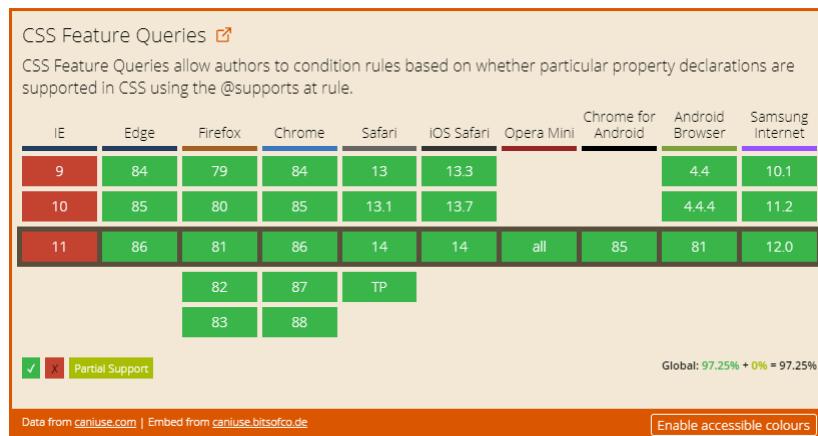
```

@supports not (display: grid) and (display: flex) {
  .content {
    display: flex;
    justify-content: center;
  }
}

@supports not (display: flex) {
  .content {
    display: block;
  }
}

```

Estas reglas son muy interesantes para casos particulares donde queremos dar soporte a navegadores antiguos, pero hay que tener en cuenta que navegadores como **Internet Explorer** (quizás uno de los más interesantes para utilizarlo) no tienen soporte para la regla `@supports`, aún ni en su versión 11.



Medios impresos

Ahora que ya disponemos de un conocimiento amplio y extenso de CSS, debemos saber que CSS no se limita sólo a páginas webs. Por ejemplo, muy probablemente conozcas uno de los formatos más populares de **ebooks** (*libros electronicos*): el **formato EPUB**. Este formato no es más que un archivo comprimido en formato `.zip`, que en su interior contiene HTML (*más concretamente, XML*) y CSS.

Algunos programas o aplicaciones, como por ejemplo el editor [Visual Studio Code](#), la versión de escritorio de [WhatsApp](#) o [Slack](#) están hechos con [Electron](#), que básicamente utilizan HTML y Javascript para crear la interfaz visual que utiliza el usuario y mediante CSS cambian el aspecto visual de la aplicación. Esto sólo son algunos casos en los que CSS forma parte de nuestra vida casi sin darnos cuenta.

¿Qué son los medios impresos?

Otra de las características de CSS que nos puede interesar es la de para preparar hojas de estilo destinadas a la **impresión de documentos**. Esto es, un documento CSS especialmente diseñado para que se aplique únicamente cuando el usuario desee **imprimir una página**.

Este sistema no es más que una de las posibilidades de los [media queries](#). En lugar de aplicar una regla `@media` a `screen` (*pantallas*), lo aplicaremos a `print` (*documentos de impresión*):

```
@media print {  
    /* Reglas de CSS */  
}
```

También se puede hacer desde el HTML, a través de la etiqueta <link>:

```
<link rel="stylesheet" media="print" href="print.css">
```

La razón es obvia, en una impresión nos pueden interesar ciertos detalles:

- **Eliminar** fondos negros o intentar que predomine el texto blanco (*gastar menos tinta*).
- Usar tipografías **apropiadas para impresión** (*cansar menos la vista, reducir gasto de tinta*).
- **Ocultar** publicidad, menús, navegación, etc... (*no tienen sentido en un documento impreso*).
- **Mostrar** ciertos detalles necesarios (*la URL de los enlaces, por ejemplo*).

Ejemplo de medios impresos

Quizás, la forma más sencilla de añadir **medios impresos** a un documento CSS ya existente, es simplemente añadir las reglas de impresión @media print al final del documento, para que se apliquen sobre las anteriores que ya existen. De esta forma partimos del diseño actual que ya tenemos, haciendo pequeños cambios en la versión de impresión:

```
@media print {  
    /* Ocultar zonas no relevantes */  
    .navigation,  
    .banner,  
    .menu {  
        display: none;  
    }  
  
    /* Cambiar tipografía o alternar colores */  
    body,  
    .content {  
        font-family: Ecofont, sans-serif;  
        background: white;  
        color: black;  
    }  
  
    /* Mostrar enlaces */  
    a::after {  
        content: "(" attr(href) ")";  
        padding: 0 5px;  
    }  
}
```

Si imprimes muy a menudo documentos, puedes utilizar **tipografías ecológicas** como [EcoFont](#) o [EcoSans](#), una **tipografía** que tiene pequeños agujeros en su interior para ahorrar tinta en la impresión de borradores y que mantiene casi el mismo aspecto visual:



El veloz murciélagos hindú
El veloz murciélagos hindú
El veloz murciélagos hindú
El veloz murciélagos hindú

Obviamente, depende del desarrollador preferir escribir unas reglas `@media print` al final del documento CSS aplicando herencia con el resto de los estilos de la página, o crear un documento CSS individual a parte, donde colocará todos los estilos de impresión y cargará a través de su elemento `<link media="print">` correspondiente.

Publicado por Manz - [LenguajeCSS](#)