

Introducción a CSS3



Contenido

1. Evolución hasta CSS3	4
2. Añadir hojas de estilo a nuestro HTML.....	5
Estilos In-line.....	5
Etiqueta Style.....	5
Directiva @import	6
Etiqueta Link	6
3. El modelo de caja. Etiquetas en línea y en bloque	7
Comportamiento de las cajas	7
4. Selectores CSS.....	9
Algunos selectores.....	9
5. Hoja de estilos ¿en “Cascada”?	12
6. Propiedades más usadas	13
Colores	14
Fondo	15
Transparencias de color o de imagen.....	15
Dimensiones y unidades	15
Márgenes, Bordes y Paddings	16
Bordes redondeados	17
Estilos para el texto	17
Texto en varias columnas	18
Flotar y posicionar	18
Sombras en cajas y texto	19
7. Google Fonts con CSS	20
Problemas usando las fuentes por defecto	20
Qué es y cómo utilizar Google Fonts	20
Ejemplo de uso de Google Fonts	21
8. Pseudoselectores.....	21
Pseudoselectores.....	22
Pseudoelementos	24
La pseudoclase :root.....	24
Propiedades personalizadas.....	25
9. Práctica: Pseudoselectores	25
10. Función calc().....	25
11. Variables en CSS	26
12. Estilos por defecto y reseteo de propiedades	28

13.	Prefijos específicos para navegadores	28
14.	Optimización de CSS	29
15.	Herramientas relacionadas con CSS	29

1. Evolución hasta CSS3

CSS es un lenguaje de diseño gráfico usado para definir y crear la presentación de un documento estructurado escrito en lenguaje de marcado. Es muy usado para establecer el diseño visual de aplicaciones web e interfaces de usuario escritas en HTML.

Versión 1.0 (1996), ..., Versión 3 (2011).

Nivel	Año	Descripción
CSS1	1996	Se crean propiedades para tipografías, 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	Se comienzan a crear características de CSS como módulos independientes

La actual versión incluye algunas novedades, por ejemplo:

- Es modular, ya que la especificación de estilos fue creciendo tanto, que **hubo que desarrollarla modularmente**. Así, se decide separar sus nuevas funcionalidades en pequeños **módulos** en lugar de aumentar versiones, para facilitar la implementación en navegadores. De esta forma, las características avanzan independientemente del lenguaje. Por ejemplo, el módulo **Flex** puede encontrarse en el nivel 2, mientras que el módulo **Grid** puede encontrarse en el nivel 3.
- Cada módulo tiene un grado de madurez o estandarización propio. **Podemos tener un módulo que sea ya estándar y otro módulo que esté todavía en un proceso de estandarización**. De hecho, se estima que no va a haber una versión de CSS4, sino que habrá módulos en su versión 4 y módulos que todavía no habrán progresado.

Entre las cosas **que podemos hacer con CSS3** destacamos algunas:

- **Permite hacer muy fácilmente cosas como los bordes redondeados**, que con versiones anteriores de CSS3 eran bastante difíciles de conseguir.
- Permite dar **gradientes de colores**, es decir, si queremos dar a una etiqueta o a una parte de la página web un color de fondo, ya no nos vemos limitados a darle un único color, sino que podemos establecer un gradiente, una transición de un color a otro.
- Se pueden hacer **transformaciones** en muchas de las propiedades de los elementos de la página web e incluso **animaciones** utilizando solo CSS3.
- Permite **maquetar de manera mucho más fácil** utilizando contenedores **Flex** y **Grid**.
- Permite utilizar las **media-queries**, que permiten elegir una u otra hoja de estilos dependiendo de las propiedades en la pantalla que tengamos, para que la página web cambie su presentación si se muestra en una pantalla grande, en un móvil, en una tableta e incluso en la televisión.

2. Añadir hojas de estilo a nuestro HTML

En el tema anterior hemos ido, de manera esporádica, añadiendo estilos en algunos ejercicios y ejemplos.

Ahora vamos a ver que existen otras formas de añadir estos estilos, y vamos a explicar qué formas son más recomendables y por qué.

De manera general podremos añadir CSS a nuestro HTML de cuatro formas:

- **Estilos In-line**
- **Etiqueta Style**
- **Directiva @import**
- **Etiqueta link**

Estilos In-line

La inclusión de estilos In-line consiste en la inclusión de estilos visuales usando el atributo **style** de las etiquetas.

Este atributo es un atributo general que está presente en todas aquellas etiquetas que sirven para describir el contenido de la página.

Un ejemplo de estilos In-Line sería:

```
<h1 style="color:white; background-color:black;">
...
</h1>
...
<h2 style="color:red; font-family:Arial;">
...
</h2>
```

Es directo, pero si quieres realizar un cambio en todos los elementos que tienen ese color de letra en éste y en todos los demás ficheros de tu sitio, tendrías un montón de trabajo.

Etiqueta Style

La etiqueta **<style>** me permite definir todos los estilos de una misma página web en un único sitio que, normalmente, suele estar en la cabecera (**<head>**).

Un ejemplo del uso de esta etiqueta sería:

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

Si hubiera algún cambio en el estilo de mi sitio sólo tendría que hacerlo una vez por página. Sigue siendo mucho trabajo si mi sitio web está compuesto, por ejemplo, por miles de páginas.

Directiva @import

Nos permite importar un fichero css directamente en nuestro HTML o dentro de nuestro CSS.

Se utiliza dentro de la etiqueta **style** de esta manera:

```
<style>
  @import 'css/import.css';
</style>
```

import.css sería el nombre de nuestro fichero de estilos pudiendo ser el que nosotros queramos.

También se puede usar dentro de nuestro fichero CSS, se colocaría arriba del todo. Además, permite especificar una **media query** (se verán más adelante) que, en caso de cumplirse, se usará este CSS importado.

Ejemplo:

```
@import "printstyle.css" print;

@import "mobstyle.css" screen and (max-width: 768px);
```

Utilizar esta técnica me permite realizar cambios en el estilo de toda mi web únicamente tocando un fichero. Sin embargo, hay un inconveniente: **el rendimiento**, ya que es bloqueante en algunos navegadores, y se para la descarga del resto de la página hasta que no se ha descargado la hoja de estilos.

Por lo tanto, hay que usar esta opción con cuidado.

Etiqueta Link

Es la opción que se recomienda. Tendremos un único punto de cambio y a la vez no es bloqueante.

Un ejemplo de uso de esta etiqueta, que se pone dentro de la cabecera, es:

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

Tiene tres atributos en los que indicamos qué fichero quiero (*href*), que tipo de fichero es (*type*) y relación que existe entre el documento y el documento enlazado (*rel*). El atributo *type* es opcional.

Otros atributos que se pueden poner y sería interesante conocer son:

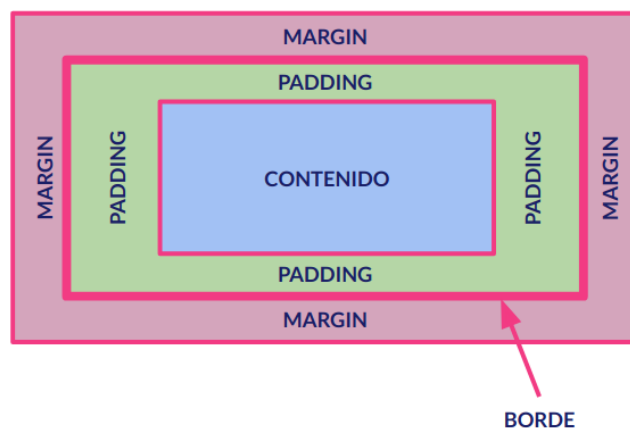
- **media**: indica el tipo de medio en el que se van a ejecutar los estilos del archivo CSS. Se verá más adelante.
- **crossorigin="anonymous | use-credentials"**. Especifica cómo el elemento maneja las solicitudes de origen cruzado (enviando o sin enviar credenciales (cookies, certificados, datos, ...)).

Por último, tenemos varios validadores CSS disponibles en la red. Uno de ellos es:
<https://jigsaw.w3.org/css-validator/>

3. El modelo de caja. Etiquetas en línea y en bloque

Una de los conceptos más importantes que debemos recordar a la hora de elaborar CSS es que todos y cada uno de los elementos de mi página web son cajas.

Eso exactamente, ¿qué significa? Significa que todos los elementos que representan algo en HTML tiene la siguiente estructura visual:



Con ese modelo debemos por lo tanto distinguir las siguientes zonas:

- El **contenido** que es lo “importante”, el texto en las etiquetas de texto, la imagen en las etiquetas de imagen etc..
- El **padding** que es la distancia que existe entre el contenido y el borde de la caja.
- El **borde** que es el elemento que marca la división entre el elemento y el resto de los elementos de la página.
- El **margen** que es la distancia entre el elemento y el resto de los elementos de la página.

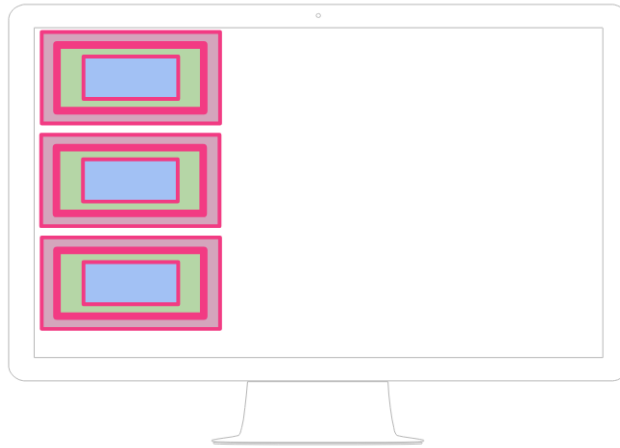
Podemos comprobar estas propiedades, y el hecho de que por defecto los navegadores les dan valor, utilizando las herramientas para desarrolladores de los mismos (F12 – Pestaña **Elements**).

Comportamiento de las cajas

Además de conocer la estructura de las cajas, en CSS debemos conocer cómo se comportan estas cajas cuando las representamos en nuestro navegador.

Los **elementos en bloque** son elementos que, independientemente de la anchura que tengan, se separan verticalmente de los elementos anteriores y posteriores. Es como si “*provocaran*” un salto de línea antes y uno después, figuradamente hablando.

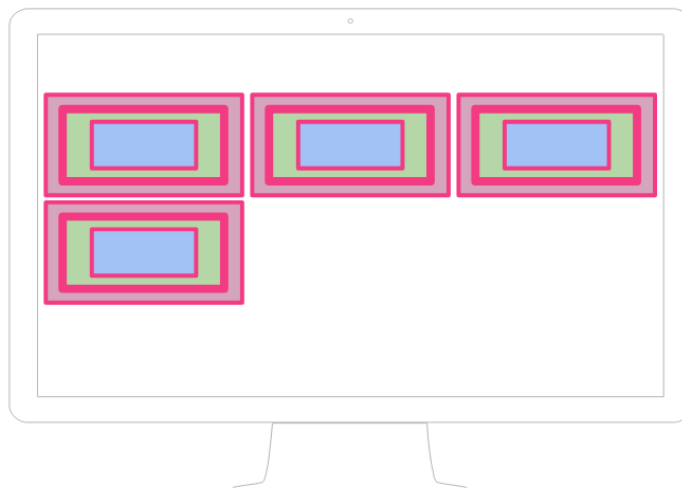
Representado esto gráficamente sería algo así:



Son elementos en bloque: ***address, blockquote, dir, div (y las semánticas: header, main, aside, footer, ...), dl, fieldset, form, hx, menu, p, pre, table, ul, ol.***

En cambio, los **elementos en línea** se van sucediendo a lo largo de la misma línea, mientras caben, uno detrás de otro y de izquierda a derecha (*al menos en nuestro idioma*). Cuando no caben pasan a la línea siguiente. Digamos que “*fluyen*” dependiendo de la anchura de la pantalla de nuestro navegador. Este fluir es precisamente la clave a la hora de **maquetar** páginas web que es algo que veremos en otro tema.

Representado este comportamiento gráficamente sería algo así:



Son elementos en línea: ***a, abbr, acronym, b, bdo, big, br, cite, code, dfn, em, font, i img, input, kbd, label, q, s, samp, selec, small, span, strike, strong, sub, sup, textarea, tt, u, var.***

Es muy importante saber qué tipo de elemento es cada etiqueta que usemos.

4. Selectores CSS

Ya sabemos incluir estilos en nuestro HTML, ya sabemos que todos los elementos de nuestra página web son cajas y ahora vamos a dar el siguiente paso, seleccionar los elementos que queremos, de entre todos los que hay, para posteriormente darles estilos.

Para ello utilizaremos selectores CSS.

Los **selectores CSS** son **reglas** o **patrones** que nos van a permitir seleccionar los distintos elementos de mi página web para poder modificar sus propiedades o estilos.

Tener buenos selectores CSS es muy importante. Un dominio correcto de los mismos te ahorrará trabajo.

Estos selectores es lo que vamos a incluir en nuestros archivos CSS y su sintaxis de manera general es la siguiente:

```
selector {  
    prop1: valor1;  
    prop2: valor2;  
    ....  
    propn: valorn;  
}
```

- **selector** hace referencia a la regla o patrón mediante cuya aplicación elegiremos uno o varios elementos de mi página web.
- **propX** son las propiedades que queremos modificar en los elementos seleccionados.
- **valorX** es el valor que daremos a cada una de las propiedades modificadas.

Todo con la sintaxis que tenemos expresada en la imagen.

Algunos selectores

- ***** es el selector universal. Selecciona todos los elementos.
- **#id** selecciona el elemento que tenga ese valor (id en este ejemplo) en el atributo id.
- **.class** selecciona los elementos que tengan ese valor (class en este ejemplo) en el atributo class.
- **etiqueta** selecciona esas etiquetas concretas.
- **selector1, selector2** (separados por comas) sirve para cambiar las propiedades de los elementos seleccionados por ambos selectores.
- **selector1 selector2** (separados por un espacio) sirve para cambiar las propiedades de los elementos seleccionados por selector2 que se encuentran **dentro** de aquellos seleccionados por selector1.

- **selector1>selector2** sirve para cambiar las propiedades de los elementos seleccionados por selector2 que tienen como padre a aquellos seleccionados por selector1.
Ej: div>p → todos los <p> que tienen como padre un div.
- **selector1+selector2** sirve para cambiar las propiedades de los elementos seleccionados por selector2 que están **justo después** de aquellos que se seleccionan mediante selector1.
- **selector1~selector2** igual que el anterior pero **justo antes**. Es decir, aplica las reglas CSS a los elementos seleccionados con *selector2* donde justo antes aparece el selector *selector1*.
- **elem[atrib]** elementos que especifiquen algún valor para el atributo **atrib**.
- **[atributo expr valor]** siendo expr (=,~=,|=,\$=,*..) para seleccionar elementos atendiendo al valor de sus atributos.

- **Expresión:**

- = → todos los elementos que tiene ese valor exacto para el atributo.
- ~= → selecciona a todos los elementos que contienen la palabra de la derecha para el atributo indicado a la izquierda de la expresión. Ejemplo:

```
/* Elementos <a> cuyo atributo class contenga la palabra "logo" */
a[class~= "logo"] {
    padding: 2px;
}
```

- |= → Selecciona los elementos cuyo atributo tenga **exactamente el valor indicado o empiece** por ese valor seguido de un guión “-“. Ejemplo:

```
/* Todos los divs en chino son rojos, ya sean
   simplificados (zh-CN) o tradicionales (zh-TW). */
div[lang|= "zh"] {
    color: red;
}
```

- ^= → selecciona todos los elementos cuyo atributo **empiece por el valor especificado**. Ejemplo:

```
/* Elementos <a> con un href que comience con "#" */
a[href^= "#"] {
    color: #001978;
}
```

- \$= → selecciona todos los elementos cuyo valor de atributo **finalice por el valor especificado**. Ejemplo:

```
/* Elementos <a> con un href que termine en ".org" */
a[href$= ".org"] {
    font-style: italic;
}
```

```
}
```

- *= → selecciona **todos los elementos** cuyo valor de atributo contenga la subcadena especificada. Ejemplo:

```
/* Elementos <a> con un href que contenga "example" */  
a[href*="example"] {  
    font-size: 2em;  
}
```

Un ejemplo de algunos de ellos podemos verlo en el siguiente HTML:

```
* {  
    font-family: "Courier New", Courier, monospace;  
}  
  
#main {  
    background-color: grey;  
}  
  
h1.especial {  
    color: blue;  
}  
  
h2,h3 {  
    border: 1px solid orange;  
}  
  
li {  
    color: red;  
}  
  
li li {  
    color: green;  
}  
  
p > img {  
    border: 5px solid black;  
}  
  
img[alt="segunda"] {  
    border: 5px solid red;  
}  
  
p ~ h3.antes {  
    background-color: pink;  
}  
  
p + h3 {  
    background-color: blue;  
}
```

Además, pueden combinarse de todas las maneras que se nos ocurran para realizar selecciones más complejas.

Práctica: Selectores

Probar TODOS los selectores antes vistos y su prioridad en distintos casos que puedan ser conflictivos.

5. Hoja de estilos ¿en “Cascada”?

Ya desde la misma definición de CSS dijimos que la C hacía referencia a “*en Cascada*”. Pero exactamente, ¿qué significa eso?

Básicamente dos cosas:

- a) Los estilos se van **propagando o heredando** hacia abajo o lo que es lo mismo, si especificamos una propiedad en un elemento padre los hijos tienen el mismo valor para esas propiedades.

Hay propiedades que no se heredan: ***size, float, margin.***

- b) Si hay más de una regla que se puede aplicar al mismo elemento y hay **conflicto**, entonces se aplica la **regla más específica**.
- c) Si las propiedades entran en conflicto (*por ejemplo, el color del fondo del elemento está definido en varios sitios con colores distintos*), existen reglas para decidir qué propiedad tiene preferencia. Si se define la misma propiedad para la misma etiqueta, con el mismo selector en dos sitios distintos, las reglas de precedencia son las siguientes:
 - Las propiedades definidas en un atributo **style** (en la etiqueta HTML) se imponen a las propiedades definidas en la etiqueta **<style>** (en el <head> de HTML).
 - Las propiedades definidas en la etiqueta **<style>** (en el <head> de HTML) se imponen a las propiedades definidas en una hoja de estilo enlazada.
 - Las propiedades definidas en un atributo **style** (en la etiqueta HTML) se imponen a las propiedades definidas en una hoja de estilo enlazada.
 - Además de estas propiedades definidas por el creador de la página web, hay que tener en cuenta que también se aplican las **propiedades definidas en la hoja de estilo por defecto del navegador**.
 - Si las propiedades se encuentran definidas en diferentes hojas de estilo, el navegador aplica el valor definido en la **última hoja de estilo enlazada** (es decir, en el último enlace <link> del <head>).

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

```
div {  
  color: red !important;  
  padding: 8px  
}  
  
div {  
  color: blue;  
  background-color: grey  
}
```

Las propiedades que se heredan entre componentes padre e hijos son: *color*, *font*, *letter-spacing*, *direction*, *word-spacing*, *line-height*.

También hay que decir que existe un mecanismo para indicar que un elemento herede el valor de una propiedad que, por defecto, no se hereda. Para ello se usan los valores:

- *Inherit*: aplica el valor de la propiedad del elemento padre.
- *Initial*: aplica el valor inicial de la propiedad.
- *unset*: hereda el valor del padre, si este no existe aplica el valor inicial o por defecto.

Ejemplo:

```
h1 {  
  border: inherit;  
}
```

6. Propiedades más usadas

Ya sabemos incluir estilos, ya sabemos sobre cajas y ya sabemos cómo seleccionar los elementos de mi página web para poder aplicarles estilos.

Ahora, en este apartado, vamos a ver que propiedades podemos modificar, pero antes, vamos a recordar la sintaxis general de los archivos CSS:

```
selector {  
  prop1: valor1;  
  prop2: valor2;  
  ....  
  propn: valorn;  
}
```

Hay muchísimas propiedades que podemos modificar, algunas son comunes a todas las etiquetas y otras sólo se pueden modificar en una etiqueta concreta.

Podéis acceder a lista completa aquí:

[Propiedades CSS - Lista completa](#)

Es casi imposible conocerla entera, pero es una referencia que debemos de tener siempre presente en nuestra barra de favoritos.

Si queremos reducir la lista a aquellas propiedades que son consideradas fundamentales o esenciales para poder empezar a aplicar estilos es mejor que nos fijemos en este enlace:

[Propiedades CSS básicas](#)

No obstante, vamos a reducir aún más la lista a una serie de propiedades referentes a los siguientes aspectos:

- **Color**
- **Fondos**
- **Transparencias**
- **Dimensiones y Unidades**
- **Márgenes, Bordes y Padding**
- **Texto**

Colores

Para establecer el **color del texto** de nuestra web lo podemos establecer usando la propiedad *color*. Por ejemplo:

```
/* Notación mediante colores */
p {
    color: red;
}

/* Notación hexadecimal */
h1 {
    color: #cccccc;
}

/* Notación RGB */
h3 {
    color: rgb(255,255,255);
}
```

Se puede establecer el color de tres formas distintas: expresando el color con su nombre en inglés, mediante un código hexadecimal de seis cifras empezando con el carácter '#' y con formato *rgb* de esta forma: *rgb(255,255,255)*. Esta última forma también acepta tantos por ciento, por ejemplo: *rgb(100%, 0%, 0%)*.

Existen otros formatos para especificar el color: *rgba()*, *hsl()* o *hsla()*. Investiga su significado.

Fondo

Usando CSS podemos también establecer el **fondo** de nuestros elementos. Hay diversas propiedades, las más destacadas:

- **background-color** para establecer el color de fondo.
- **background-image** para establecer una imagen de fondo. Necesitaremos usar la propiedad *url(...)*.
- **background-repeat** para especificar cómo se repite la imagen de fondo. Puede tomar diversos valores: *repeat*, *repeat-x*, *repeat-y*, *space*, *round*, *no-repeat*. Además, se pueden usar varios a la vez separados por espacios en blanco. Investiga cómo funciona cada una de estas opciones.
- **background-origin** especifica la posición de origen de un fondo o imagen. Sus posibles valores son: *padding-box* (valor por defecto), *border-box* y *content-box*: *esquina superior izquierda del padding, del borde y del contenido respectivamente*.
- **background-clip**: permite definir qué parte del fondo se muestra, con respecto al borde, si debe mostrarse por debajo de este o no. Tiene los mismos valores que la propiedad anterior. Se puede confundir con la anterior, de echo, son muy parecidas.
- **background-size**: permite definir el tamaño de las imágenes de fondo. Por defecto es auto, pero se puede especificar ancho y alto separados por espacio, *cover* (para que ocupe el contenedor entero), *contain* (redimensiona la imagen para asegurar que es visible completamente), ...
- **background-position**: establece dónde empieza la imagen de fondo con respecto al borde superior izquierdo. Valores posibles: *left*, *right*, *top*, *bottom*, *center*, *pares px para especificar cuanto left y top ó pares % con el mismo significado*.
- **background-attachment**: especifica si la imagen de fondo debe desplazarse o ser fija (no se desplazará con el resto de la página al hacer scroll). Valores: *fixed* o *scroll*.

Transparencias de color o de imagen

Existe la posibilidad de añadir transparencias al color o a una imagen; desde un color totalmente sólido a totalmente transparente. La propiedad en cuestión es **opacity** y toma un valor entre 0.0 y 1.0.

Dimensiones y unidades

Las dimensiones de los elementos de nuestra página se establecen usando las siguientes propiedades:

- **width** para la anchura de nuestro elemento.
- **height** para altura de nuestro elemento.

Y ambas las podemos determinar usando varios tipos de unidades:

- **px:** En píxeles.
- **%:** En relación a lo que ocupe el padre del elemento dentro del árbol DOM.
- **em:** En relación al tamaño de la fuente que se use en ese instante. Por ejemplo, si se usa un *font-size:12px*, 1em = 12px. Ten en cuenta que *font-size* se hereda de elementos padre a hijos.
- **ex:** relativa respecto a la altura de la letra **x** minúscula.
- **rem:** En relación al tamaño por defecto de la letra que tiene la etiqueta HTML.
- **vh y vw:** Medidas relativas de acuerdo a la altura y anchura respectivamente del viewport. Ejemplo: 1vh = 1% de la altura del viewport.

Un ejemplo básico:

```
#first img {  
  width: 50%;  
}  
  
#second {  
  width: 600px;  
  ...;  
}
```

También podemos usar medidas absolutas como: cm (centímetros), mm (milímetros), in (pulgadas), pt (puntos: 0.35mm, la más usual), pc (picas: 1pc == 12 pt)

Márgenes, Bordes y Paddings

Son propiedades para establecer las dimensiones de los elementos de la caja.

Para márgenes y paddings tenemos varias formas de hacerlo. Si tenemos en cuenta que A(Arriba)- D (Derecha) - AB (Abajo) - IZQ (Izquierda) van en el sentido del reloj:

```
selector {  
  /* A -D -AB -IZQ */  
  margin: 20px 50px 20px 50px;  
  
  /* A y AB - DCHA e IZQDA */  
  margin: 20px 50px;  
  
  /* Todos */  
  margin: 50px;  
  
  /* O de manera individual */  
  margin-left: 10px;  
  margin-top: 10px;  
  margin-bottom: 10px;  
  margin-right: 10px;  
}
```

Si solamente se especifican dos márgenes, se entiende que el primero se usa para arriba y para abajo, y el segundo para derecha e izquierda.

Para el padding, sería exactamente lo mismo. Sólo tenemos que sustituir *margin* por *padding*.

En relación al borde de un elemento tenemos también varias posibilidades. De igual manera lo vamos a ilustrar mediante un ejemplo:

```
/* De manera general */
border: 1px solid black;

/* Sólo el borde */
border-color: black;

/* Sólo la anchura del borde */
border-width: 1px;

/* Sólo el estilo de la línea del borde */
/* posibles valores: solid, dashed, dotted */
border-style: solid;
```

Bordes redondeados

Permite redondear las esquinas de las cajas de los elementos. Para que sea visible debe tener definida la propiedad *border*. La propiedad principal es *border-radius*, pero se pueden aplicar individualmente para cada esquina (***border-top-left-radius***, ***border-top-right-radius***, ***border-bottom-left-radius***, ***border-bottom-right-radius***).

Estilos para el texto

Hay multitud de propiedades para establecer los estilos del texto de mi página web. Algunas de las más destacables son:

```
/* Para establecer el tipo de fuente */
font-family: "Times New Roman", Times, serif;

/* Para establecer el tamaño de la fuente */
font-size: 2em;

/* Para establecer la anchura de los caracteres */
/* Posibles valores: bold, bolder, lighter, 100, 200, ..., 900 */
font-weight: bold;

/* Para establecer la alineación texto */
/* Posibles valores: center, left, right, justify */
text-align: center;

/* Para establecer la decoración de texto */
/* Posibles valores: underline, overline, none, line-through */
text-decoration: underline;

/* Para establecer tabulaciones */
```

```
text-indent: 10px;

/* Para transformar un texto todo a mayúscula o minúsculas */
/* Posibles valores: uppercase, lowercase, capitalize */
text-transform: uppercase;

/*Para poner el texto en cursiva*/
font-style: normal | italic | oblique;

/*Para establecer el interlineado del texto*/
line-height: 120%;
```

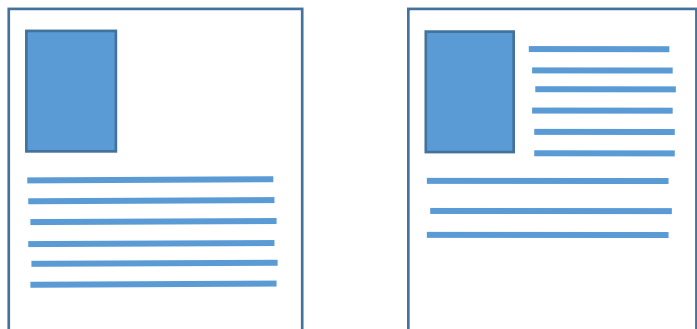
Texto en varias columnas

Podremos poner el texto en varias columnas con los atributos:

- **column-count:** para especificar el número de columnas que tendrá el elemento contenedor.
- **column-width:** si queremos fijar en ancho de las columnas del contenedor. El navegador calculará cuántas columnas, al menos, caben con ese ancho.
- **column-gap:** permite establecer la distancia que separa las distintas columnas.
- **column-rule:** funciona de manera muy similar a borde y me permite especificar el estilo, color y anchura de la línea que separa las columnas.
- **column-span:** con valores *all* o *none* para indicar si el elemento en cuestión, que estará dentro del contenedor donde hemos especificado que habrá columnas, sigue el flujo en columnas o no.
- **column-fill:** para establecer cómo se rellenan las columnas. El contenedor debe tener altura. Los valores son *auto* o *balance* (todas las columnas la misma altura)
- **break-inside:** con valor *avoid* si queremos que el elemento no quede roto de una columna a otra.

Flotar y posicionar

Cuando se trabaja a través de cajas para el modelado del código de la página web, debe tenerse en cuenta la posición de cada una de ellas; normalmente, estas se sitúan la una a continuación de la otra, bien en horizontal o en vertical, pero existen ocasiones en las que es deseable que diversas cajas contenedoras de información, ya sea texto, imagen o cualquier otro elemento, se adapten entre sí.



La existencia de cajas flotantes condiciona la situación del resto de cajas. Esto es debido a que aquellos elementos no flotantes, es decir, los que están en línea, adaptan su posición para evitar que se solapen las cajas en función a la colocación de las flotantes.

El funcionamiento concreto de esta propiedad se basa en el desplazamiento total hacia la derecha o hacia la izquierda de una caja contenedora con respecto de su posición inicial.

```
etiqueta {  
    float: left | inherit | right | none;  
}
```

Los tres valores más usados que puede tomar esa propiedad son hacia la izquierda, hacia la derecha o mantenerse en la posición original; este último es el valor por defecto y es usado en aquellos casos en los que se desea eliminar la propiedad 'float' definida para una caja.

En los casos *left* y *right*, el resto de los elementos adyacentes se adaptan para mostrarse respectivamente a la derecha o a la izquierda de la caja flotante, es decir, siguen flotando. Si no se desea que el resto de los elementos tomen este papel, se utilizará la propiedad **clear**. Esta propiedad acepta valores: *right*, *left* o *both*. Tras aplicar *clear*, ese elemento ya se añadirá tras el fin en vertical del elemento flotante.

NOTA: Los elementos con *position="absolute"* ignoran *float*.

clearfix hack → Si tenemos un contenedor y empezamos a flotar elementos, no podemos fijar el alto de ese contenedor. Una solución es usar la propiedad **overflow-y** o fijar una altura suficiente como para que los elementos no se salgan del contenedor:

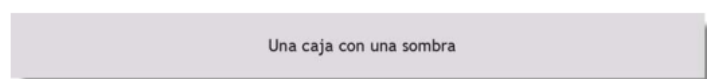
```
contenedor {  
    overflow-y: auto;  
    height: altura suficiente; /*Usar una de las dos*/  
}
```

Sombras en cajas y texto

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

```
#example1 { box-shadow: 10px 10px 5px #888; }
```

generará:



La propiedad **box-shadow** admite una lista de 5 elementos, que definen, en orden, desplazamiento horizontal, desplazamiento vertical, desenfocado, extensión y color de la sombra. Opcionalmente, se puede añadir la palabra clave **inset**, que nos permitirá hacer una sombra interna en lugar de una externa:

Algunos ejemplos:

```
#Example_F { box-shadow: inset 0 0 5px 5px #888; }
```

La propiedad **text-shadow** se comporta de igual modo, y nos permite generar sombras en los textos.

Por otro lado, cabe decir que podemos tener más de una sombra por elemento, es decir, que las podemos superponer como efecto de papel cebolla. Para hacerlo:

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

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



7. Google Fonts con CSS

Problemas usando las fuentes por defecto

Al desarrollar páginas web nos encontramos con un problema al utilizar las fuentes que por defecto utilizan los navegadores, ya que o bien resultan feas o bien son demasiado comunes, incluso pueden parecer feas por ser tan comunes.

Si queremos utilizar fuentes diferentes a las habituales, distintos tipos de letra, debemos asegurarnos que ese tipo de letra esté instalada en todos los navegadores de todos los usuarios, lo cual es bastante difícil.

Qué es y cómo utilizar Google Fonts

Ante esa situación hay diversas soluciones, pero la que vamos a ver es una que consiste en utilizar *Google Fonts*, una herramienta que Google pone a nuestra disposición de manera gratuita.

Para incluir en una página web fuentes de Google Fonts básicamente hay que hacer dos cosas:

- Añadir un enlace a una hoja de estilos en la cabecera de la página, con el nombre de la fuente que se quiera utilizar, de esta forma:

```
link href="https://fonts.googleapis.com/css?family=XXXXX" rel="stylesheet"
```

- Colocar la propiedad font-family a los elementos donde queremos aplicar ese tipo de fuente:

Ejemplo de uso de Google Fonts

Como ejemplo vamos a elegir una fuente muy llamativa, ya que no tiene porqué ser bonita, y vamos a ver cómo se cambia y cómo queda en una página de prueba.

Accedemos a la web de Google Fonts a través de <https://fonts.google.com>. Una vez dentro elegimos un tipo de fuente, que en nuestro ejemplo es la que se denomina “*Pacífico*”. Pulsamos en el botón rojo con el símbolo + para seleccionarla.

Nos aparece en la parte inferior de la pantalla el mensaje “Family Selected”, clicamos sobre el mismo y se despliegan una serie de opciones. Seleccionamos el enlace que aparece en este desplegable, que es el siguiente:

```
link href="https://fonts.googleapis.com/css?family=Pacifico" rel="stylesheet"
```

Abrimos el archivo HTML de nuestra web y pegamos el código que hemos copiado previamente dentro de la zona del head.

Volvemos a la web de Google Apis y en la pantalla de la fuente elegida copiamos el segundo código que aparece para utilizar la fuente, que en este caso es el siguiente:

```
font-family: 'Pacifico', cursive;
```

Una vez copiado abrimos el **archivo de estilos CSS** que usamos en nuestra web, y pegamos el mismo dentro de los atributos de body, para aplicar ese tipo de fuente al mismo y a todos sus elementos hijos, ya que esa propiedad se propaga por los mismos.

Finalmente cargamos la página web y podemos observar que en la misma se está utilizando la fuente **Pacifico de Google Fonts** que habíamos elegido.

8. Pseudoselectores

Además de los selectores tradicionales que hemos presentado anteriormente CSS nos proporciona unos “selectores” especiales, los llamados **pseudoselectores** y los **pseudoelementos**.

Los **pseudoselectores** son palabras clave que se añaden a los selectores y que nos indican un **ESTADO** determinado de los elementos seleccionados.

Los **pseudoelementos** son palabras clave que se añaden a los selectores y que nos indican una **PARTE** de un elemento y nos permiten añadir contenido.

Pseudoselectores

Tienen la siguiente sintaxis:

```
selector:pseudoselector {
  prop1: valor1;
  prop2: valor2;
  ....
  propn: valorn;
}
```

Y podemos dividirlos en los de **estado** y los de **posición**.

Los de posición son:

Pseudo-clase	Descripción
:first-child	Primer hijo
:last-child	Último hijo
:first-of-type	Primer hermano de su tipo
:last-of-type	Último hermano de su tipo
:only-child	Hijos únicos
:only-of-type	Únicos hermanos de su tipo
:empty	Elementos que no tienen hijos
:nth-child(n)	Enésimo elemento hijo
:nth-last-child(n)	Enésimo elemento hijo contando desde el último
:nth-of-type(n)	Enésimo hermano de su tipo
:nth-last-of-type(n)	Enésimo hermano de su tipo comenzando desde el último
:not()	<p>Coincide con cualquier elemento que NO sea el elemento o selector especificado. Ejemplo:</p> <pre>:not(p) { color: red; /*Aplica color rojo a todos los elementos que no sean de tipo párrafo*/ }</pre>
:has()	<p>Selecciona un elemento solo si contiene, al menos, un descendiente que coincida con el selector que se pasa como argumento ("selector padre")</p> <pre>/* Selecciona cualquier 'article' que TENGA DENTRO una 'img' */ article:has(img) { background-color: #f0f8ff; border: 1px solid #add8e6; }</pre>

:nth-child (argumento), puede usarse de tres formas, según sea el argumento:

- **Una palabra clave:**
 - **odd (impar):** Selecciona los elementos hermanos que ocupan una posición impar (el 1º, 3º, 5º, etc.).
 - **even (par):** Selecciona los elementos hermanos que ocupan una posición par (el 2º, 4º, 6º, etc.).
- **Un número entero:** esta es la forma de seleccionar un único elemento por su posición exacta. El conteo siempre empieza en 1.
- **La Fórmula: nth-child($An+B$),** donde:
 - **A:** Es el "tamaño del ciclo" o "paso". Indica cada cuántos elementos se repetirá el patrón.
 - **n:** Es un contador que empieza en 0 y va aumentando (0, 1, 2, 3...). No lo cambiamos, es parte de la fórmula.
 - **B:** Es el "punto de partida" o "desplazamiento". Indica desde dónde empezamos a contar en el primer ciclo. El + es parte de la sintaxis.

Ejemplos:

:nth-child(3n) → Seleccionamos cada 3 elementos, como no indicamos offset, empieza en 0.

:nth-child(3n+4) → Cada 3 elementos dejando un offset de 4.

:nth-child(4n+1) → Cada 4 elementos empezando por el 1.

:nth-child(-n+4) → Los 4 primeros elementos.

:nth-child(n+5) → Todos a partir del 5º elemento.

Los de estado más usados son:

Pseudo-clase	Descripción
:link	No visitado por el usuario
:visited	Visitado por el usuario
:hover	Modifica el estilo cuando un elemento apuntador pasa por encima
:active	Se activa cuando el usuario pulsa el elemento
:focus	Se activa cuando tiene el foco sobre el elemento

Otros:

- **:enabled**
- **:default**
- **:disabled**
- **:checked**
- **:required**
- **:invalid**
- **:optional**
- **:read-only**
- **:required**
- **:visited**
- **:empty**

Pseudoelementos

Tienen la siguiente sintaxis:

```
selector::pseudoelemento {  
  prop1: valor1;  
  prop2: valor2;  
  ....  
  propn: valorn;  
}
```

Los más destacados son:

Pseudo-clase	Descripción
::first-line	Primera línea de texto de un elemento
::first-letter	Primera letra de la primera línea de texto de un elemento
::before	Añade contenido al principio del documento
::after	Añade contenido al final del documento

Otros:

- **::selection** → texto seleccionado por el usuario.
- **::placeholder** → texto de la propiedad placeholder.
- **::marker** → es la forma moderna y correcta de aplicar estilos a la viñeta (como un punto • o un guion -) o al número de un elemento de una lista ()..

Ejemplos:

```
<style>  
  p::before {  
    content: "Read this -";  
  }  
  
  ::selection {  
    color: red;  
    background: yellow;  
  }  
  
  ::placeholder {  
    color: red;  
  }  
</style>  
  
<p>My name is Donald</p>  
<p>I live in Ducksburg</p>
```

La pseudoclase :root.

La regla :root en CSS es una pseudoclase que representa al elemento raíz (raíz) del documento. En un documento HTML, el elemento raíz es siempre la etiqueta <html>. Aunque html { ... } es

exactamente lo mismo que escribir `:root { ... }`, en la práctica, hay dos razones clave por las que se usa `:root`:

- **Mayor Especificidad:** La pseudoclase `:root` tiene una especificidad más alta que un selector de etiqueta como `html`. Esto significa que si tuvieras una regla en `html` y otra en `:root`, la de `:root` ganaría en caso de conflicto.
- **El Estándar para Variables CSS:** Su uso más importante y extendido hoy en día es para declarar Variables CSS Globales, también conocidas como "Propiedades Personalizadas" (CSS Custom Properties).

Propiedades personalizadas.

Declaración: la sintaxis para crear una variable en CSS es empezar su nombre con dos guiones (`--`).

Por ejemplo: `--primary-color`.

Uso: una vez declarada una variable, se puede acceder a su valor utilizando la función `var()`.

Por ejemplo: `var(--primary-color)`.

9. Práctica: Pseudoselectores

Prueba los distintos pseudoselectores y pseudoelementos y comprueba su funcionamiento.

10. Función `calc()`

La función `calc()` es una función CSS que se utiliza para realizar cálculos para determinar valores de propiedades CSS de forma dinámica. Esta función necesita como parámetro una expresión matemática cuyo resultado se aplicará al valor de la propiedad asignada. Permite, además, combinar diferentes unidades, como porcentajes y píxeles, mediante operaciones aritméticas básicas como suma, resta, multiplicación y división (`-`, `+`, `*` y `/`). Por ejemplo:

```
#div1 {
  position: absolute;
  left: 50px;
  width: calc(100% - 100px);
  border: 1px solid black;
  background-color: yellow;
  padding: 5px;
  text-align: center;
}

.parent{
  background-color: skyblue;
  width: calc(80% - 2em);
}
```

Hay que tener en cuenta algunos detalles para poder aplicar correctamente `calc()`:

- El operador de suma (+) y resta (-) requiere que el número tenga las unidades de longitud y también debe estar separado por un espacio. Por ejemplo:
`margin: calc(8px + 10px);`
- El operador de división (/) requiere que el segundo número no tenga unidades. Por ejemplo: `margin: calc(80px / 5);`
- El operador de multiplicación (*) requiere que uno de los operandos no tenga unidades. El orden de los operandos no importa. Por ejemplo,

```
margin: calc(20px * 3);  
margin: calc(3 * 20px);
```

- `calc()` se puede anidar para realizar cálculos complejos y crear estilos más dinámicos y flexibles. Por ejemplo,

```
div.second {  
    width: calc(500px - calc(100px + 60px));  
}  
  
div.third {  
    width: calc(calc(100% - 50px) / 2);  
}
```

Hay muchos usos útiles de `calc()`, por ejemplo, aquí tienes uno en el que crea contenido en el main para que ocupe todo el alto de la pantalla excepto el tamaño del encabezado:

```
header {  
    width : 100% ;  
    height : 4rem ;  
    background-color : darkseagreen;  
}  
  
main {  
    width : 100% ;  
    height : calc ( 100vh - 4rem );  
    background-color : #d1d5db ;  
}
```

NOTA: Existe también la función `calc-size()`. Investiga su uso y posibles aplicaciones prácticas.

11. Variables en CSS

En CSS, las **propiedades personalizadas** (también conocidas como **variables**) son entidades definidas por autores de CSS que contienen valores específicos que se pueden volver a utilizar en un documento. Se establecen mediante la notación de propiedades personalizadas (por

ejemplo, `--main-color: black;`) y se acceden mediante la función `var()` (por ejemplo, `color: var(--main-color);`).

Para declarar variables usamos un nombre que comience con dos guiones (--), y un valor que puede ser cualquier valor válido de CSS. Como cualquier otra propiedad, la escribimos dentro de un set de reglas de estilo, algo así:

```
elemento {
  --main-bg-color: #ff7799;
}
```

Ten en cuenta que el selector que usemos para las reglas de estilo define el ámbito (scope) en el que podremos usar la propiedad personalizada (variable). Una buena práctica común es declarar variables en la pseudo-clase `:root`, y así aplicarlas globalmente al documento HTML:

```
:root {
  --main-bg-color: brown;
}
```

Como mencionamos anteriormente, para acceder al valor de una propiedad personalizada usamos el nombre de la propiedad dentro de la función `var()`, en lugar de cualquier otro valor normal:

```
elemento {
  background-color: var(--main-bg-color);
}
```

Utilizando `var()` podemos definir **múltiples valores** de sustitución que se usarán cuando la variable dada no está definida aún. El primer argumento a la función es el nombre de variable que se va a sustituir. El segundo argumento a la función, si se proporciona, es un valor de reserva, que se utiliza como valor de sustitución cuando la variable no es válida. Ejemplo:

```
.PersORojo {
  color: var(--my-var, red); /* Rojo (red) si --my-var no está definida */
}
```

Además, se pueden anidar:

```
.PersORosa {
  background-color: var(
    --my-var,
    var(--my-background, pink)
  ); /* Rosa (pink) si my-var y --my-background no están definidas */
}
```

12. Estilos por defecto y reseteo de propiedades

Una cosa que parece obvia pero que con frecuencia se olvida cuando empezamos a trabajar con HTML y CSS es el hecho de que los navegadores tienen sus propias hojas de estilos que se aplican a los distintos elementos con el objetivo de hacer las páginas sin estilos más visibles.

Estos estilos no tienen por qué coincidir de un navegador a otro y si queremos consistencia a través de los distintos navegadores se suelen usar las llamadas **hojas de RESETEO CSS**

Estas hojas sirven para eliminar ciertas características que imponen los navegadores.

Hay ciertos aspectos sobre este tema que debemos considerar:

- Este tipo de hojas de estilos para Resetear son usadas por Frameworks CSS (como Bootstrap) que buscan que todas las páginas siempre luzcan igual independientemente del navegador.
- Si las usas deben ser adaptadas a cada proyecto.
- Hay que usarlas con cuidado porque podemos eliminar estilos que dábamos por sentado.
- El trabajo posterior tras usarlas es mayor pero el resultado es más personalizado.

Hay dos hojas de reseteo que son muy usadas:

[Hoja de Reseteo de ERIC Meyer](#)

[Hoja de Reseteo de HTML5 Doctor](#)

13. Prefijos específicos para navegadores

En uno de los apartados anteriores, dijimos que CSS3 era una especificación modular, extensa y que se iba implantando poco a poco.

Los distintos navegadores conforme dan soporte a propiedades experimentales lo que hacen es añadirles un prefijo para indicar que ellos sí les han dado soporte antes de que sean soportadas por los demás o antes de que sean parte del estándar.

Estos son los llamados **prefijos de navegadores** y tienen una sintaxis similar a la siguiente:

```
a {  
  -webkit-transition: -webkit-transform 1s;  
  transition: -ms-transform 1s;  
  transition: transform 1s;  
}
```

Y los valores más comunes de esos prefijos son:

- **-webkit** para Chrome, Safari, nuevas versiones de Opera y para Firefox para iOS
- **-moz** para navegadores Firefox que no sean para iOS
- **-o** para versiones antiguas de Opera
- **-ms** para Internet Explorer y Microsoft Edge

El problema que surge es que nosotros, como desarrolladores, no sabemos cuándo usarlo. CSS3 es una especificación larga con muchos módulos y es prácticamente imposible saber cuándo hay que añadir estos prefijos o cuándo las propiedades han dejado de ser experimentales.

Tenemos la suerte, no obstante, de tener varias herramientas para ello.

- [ShoudlPrefix](#)
- [Autoprefixer](#)

14. Optimización de CSS

Conforme vayamos avanzando es nuestro conocimiento y empecemos a trabajar en proyectos reales y en proyectos grandes oiremos hablar de un término como **optimización de CSS**.

Pero, ¿qué es eso realmente? Si tenemos que resumirlo de manera concisa diremos que optimizar nuestro CSS es:

- Minimizar el CSS para subirlo a producción (quitar cualquier espacio en blanco).
- Borrar reglas innecesarias y duplicadas.
- Ordenar las propiedades de nuestras reglas o selectores por orden alfabético para que todo sea más legible y fácil de mantener.
- Poner el CSS en la cabecera (aunque es eso algo sobre lo que ya hemos insistido).
- Si el CSS es muy grande lo partiremos en varios (varias etiquetas **link**).
- Organizar las reglas poniendo las que tengan relación juntas.
- Documentar (o poner comentarios) tus ficheros CSS.

Para ayudaros con estas tareas existen muchas herramientas, algunas de estas son:

- [CSS Minifier](#) → minimiza el tamaño de los ficheros CSS.
- [CSS Optimizer](#) → quita reglas redundantes, ...
- [Unused CSS](#) → indica si hay reglas que no se están usando.

Por lo tanto, antes de subir a Producción los ficheros CSS, habrá que minimizarlos y optimizarlos.

15. Herramientas relacionadas con CSS

De igual manera, veremos que al trabajar en proyectos grandes con CSS hay tareas que son tediosas y que favorecen la aparición de errores. Para evitar esto han ido surgiendo muchas

herramientas, pero aquellas que han tenido más éxito y que deberías conocer son los preprocesadores CSS.

Un **preprocesador CSS** es un programa que te permite generar CSS de forma automática, haciendo que la estructura sea más legible y más fácil de mantener.

Añaden al CSS tradicional características como:

- Las **variables**
- Los **selectores anidados**
- Los **bucles**
- Los **mixins**: permiten que los desarrolladores creen patrones de propiedades llave-valor los cuales puedan ser reutilizados en cualquier parte del documento.

Ejemplo:

```
@mixin .espacio {
  overflow: hidden;
  _overflow: visible;
  zoom: 1;
}

.contenido {
  include: .espacio;
}
```

- Etc.

Y los preprocesadores más conocidos son: **{less}**, *Sass* y  **PostCSS** que no es exactamente un preprocesador pero nos va a facilitar mucho el trabajo con CSS desde JavaScript y es fácilmente extensible.

Un ejemplo con Sass, sería:

```
$font-stack: Helvetica, sans-serif;

$primary-color: #3334AA;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Donde estamos usando variables, la variable **\$font-stack** y **\$primary-color**.