

# CSS2 y CSS3

## Características y ventajas de las CSS

El modo de funcionamiento de las CSS consiste en definir, mediante una sintaxis especial, la forma de presentación que le aplicaremos a:

- Un documento HTML o página, se puede definir la forma, desde un pequeño trozo de código en la cabecera, a toda la página.
- Una porción del documento, aplicando estilos visibles en un trozo de la página. **(No es buena práctica)**
- Una etiqueta en concreto, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta. Esto es muy importante ya que ofrece potencia en nuestra programación. Podemos definir, por ejemplo, varios tipos de párrafos: en rojo, en azul, con márgenes, sin ellos...
- Podemos definir la distancia entre líneas del documento.
- Se puede aplicar indentado (sangrado) a las primeras líneas del párrafo.
- Podemos colocar elementos en la página con mayor precisión, y sin lugar a errores.
- Definir la visibilidad de los elementos, márgenes, subrayados, tachados...

Con HTML tan sólo podíamos definir atributos en las páginas con píxeles y porcentajes, ahora podemos definirlos utilizando muchas más unidades como:

- Píxeles (px) y porcentaje (%), como antes.
- Pulgadas (in).
- Puntos (pt).
- Centímetros (cm).

## 1- Creamos el fichero con la declaración de estilos

Es un fichero de texto normal, que puede tener cualquier extensión, aunque le podemos asignar la extensión .css para aclararnos qué tipo de archivo es. El texto que debemos incluir debe ser escrito exclusivamente en sintaxis CSS, es decir, sería erróneo incluir código HTML en él. Un ejemplo:

```
p {
  font-size : 12pt;
  font-family : arial, helvetica;
  font-weight : normal;
}
h1 {
  font-size : 36pt;
  font-family : verdana, arial;
  text-decoration : underline;
  text-align : center;
  background-color : Teal;
}

td {
  font-size : 10pt;
  font-family : verdana, arial;
  text-align : center;
  background-color : 666666;
}

body {
  background-color : #006600;
  font-family : arial;
  color : White;
}
```

## 2- Enlazamos la página web con la hoja de estilos

Para ello, vamos a colocar la etiqueta <link> con los atributos

- **rel="stylesheet"** indicando que el enlace es con una hoja de estilos
- **type="text/css"** porque el archivo es de texto, con sintaxis CSS
- **href="estilos.css"** indica el nombre del fichero fuente de los estilos y el path correspondiente

Veamos una página web entera que enlaza con la declaración de estilos anterior:

```
<!DOCTYPE html>

<html>
<head>
  <meta charset="utf-8" />
  <link rel="stylesheet" type="text/css" href="estilos.css">
  <title>Página que lee estilos</title>
</head>
```

### Reglas de importancia en los estilos

Los estilos se heredan de una etiqueta a otra. Por ejemplo, si tenemos declarado en el <body> unos estilos, por lo general, estas declaraciones también afectarán a etiquetas que estén dentro de esta etiqueta

En muchas ocasiones, más de una declaración de estilos afecta a la misma porción de la página. Siempre se tiene en cuenta la declaración más particular (el elemento hijo sobre el elemento padre).

Se puede ver a continuación esta jerarquía, primero ponemos las formas de declaración más generales, y por tanto menos respetadas en caso de conflicto:

- Declaración de estilos con fichero externo. (Para todo un sitio web)
- Declaración de estilos para toda la página. (Con la etiqueta <style> en la cabecera de la página). **(No es buena práctica)**
- Definidos en una etiqueta en concreto. (Utilizando el atributo style en la etiqueta en cuestión) **(No es buena práctica)**
- Declaración de estilo para una porción pequeña del documento. (Con la etiqueta <span>)
- Para definir un estilo se utilizan atributos como font-size, text-decoration... seguidos de dos puntos y el valor que le deseamos asignar. Podemos definir un estilo a base de definir muchos atributos separados por punto y coma.

#### Ejemplo:

font-size: 10pt; text-decoration: underline; color: black; (el último punto y coma de la lista de atributos es opcional pero es mejor ponerlo para seguir una buena práctica)

- Para definir el estilo de una etiqueta se escribe la etiqueta seguida de la lista de atributos encerrados entre llaves.
- **Ejemplo:** h1{text-align: center; color:black;}
- Los valores que se pueden asignar a los atributos de estilo se pueden ver en una tabla en el siguiente capítulo. Muchos de estos valores son unidades de medida (**Unidades CSS**), por ejemplo, el valor del tamaño de un margen o el tamaño de la fuente. Las unidades de medida CSS se pueden clasificar en dos grupos, **relativas** y **absolutas**. Más la posibilidad de expresar valores en porcentaje.

**Relativas:** Se llaman así porque son unidades relativas al medio o soporte sobre el que se está viendo la página web, que dependiendo de cada usuario puede ser distinto, puesto que existen muchos dispositivos que pueden acceder a la web, como ordenadores o teléfonos móviles. En principio las unidades relativas son más aconsejables, porque se ajustarán mejor al medio con el que el usuario está accediendo a nuestra web. Son las siguientes:

em	Relative to the font-size of the element (2em means 2 times the size of the current font). The size refers to the <b>M</b> capital letter width
ex	Relative to the x-height of the current font (rarely used)
ch	Relative to width of the "0" (zero)
rem	Relative to font-size of the root element
vw	Relative to 1% of the width of the viewport*
vh	Relative to 1% of the height of the viewport*
vmin	Relative to 1% of viewport's* smaller dimension
vmax	Relative to 1% of viewport's* larger dimension
%	Relative to the parent element

**Absolutas:** Las unidades absolutas son medidas fijas, que deberían verse igual en todos los dispositivos. Como los centímetros, que son una convención de medida internacional. Pese a que en principio pueden parecer más útiles, puesto que se verían en todos los sistemas igual, tienen el problema de adaptarse menos a las distintas particularidades de los dispositivos que pueden acceder a una web y restan accesibilidad a nuestro web. Puede que en tu ordenador 1 centímetro sea una medida razonable, pero en un móvil puede ser un espacio exageradamente grande, puesto que la pantalla es mucho menor. Se aconseja utilizar, por tanto, medidas relativas.

cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

**Porcentaje:** el porcentaje se utiliza para definir una unidad en función de la que esté definida en un momento dado. Imaginemos que estamos trabajando en 12pt y definimos una unidad como 150%. Esto sería igual al 150% de los 12pt actuales, que equivale a 18pt.

Porcentaje %

Por ejemplo 120% es el 120 por cien de la unidad que estuviera anteriormente.

- Los colores se expresan con valores RGB. Un color se puede especificar también con tres números hexadecimales, en lugar de 6, como era obligatorio en HTML. Por ejemplo #fff equivaldría a #ffffff, o #123 sería #112233. Además, los colores se pueden especificar también en valores RGB decimales, con la notación rgb(r,g,b), siendo los valores de r, g, b números entre 0 y 255, por ejemplo rgb(100,0,255). Otra notación posible es rgb(r%,g%,b%), siendo cada uno de los r%,g%, b% un valor entre 0 y 100, por ejemplo rgb(100%,50%,0%), que sería todo de rojo, la mitad de verde y cero de azul.
- Otro tipo de valores que se pueden utilizar en las hojas de estilo en cascada son las URL, que sirven para especificar rutas hacia elementos, como imágenes a colocar en fondos de elementos. Las URL en CSS se especifican con la notación **url(valor)**, siendo valor la URL a la que queremos dirigirnos, que puede ser absoluta o relativa. Si es relativa, el navegador la interpreta desde el documento CSS donde estamos, si es que es un archivo CSS, o desde el documento HTML donde estamos, si es que los estilos los estamos colocando directamente en el archivo HTML. La URL se puede indicar con comillas dobles, simples o sin comillas. Ejemplo: url(http://www.desarrolloweb.com/images/miimagen.gif) o bien: url("../images/otraimagen.jpg")

#### Notación hexadecimal RGB

Esta notación es la que ya conocemos. Se especifican los tres valores de color (rojo, verde y azul) con valores en hexadecimal entre 00 y FF.

```
background-color: #ff8800;
```

#### Notación hexadecimal abreviada

Esta notación es muy parecida a la anterior, pero permite abreviar un poco la declaración del color, indicando sólo un número para cada valor rojo, verde y azul. Por ejemplo, para especificar el color de antes (#ff8800) podríamos haber escrito:

```
background-color: #f80;
```

#### Nombre del color

También podemos definir un color por su nombre. Los nombres de colores son en inglés, los mismos que sirven para especificar colores con HTML.

```
color: red;
border-color: Lime;
```

#### Notación de color con porcentajes de RGB

Se puede definir un color por los distintos porcentajes de valores RGB. Si todos los valores están al 100% el color es blanco. Si todos están al 0% obtendríamos el negro y con combinaciones de distintos porcentajes de RGB obtendríamos cualquier matiz de color.

```
color: rgb(33%, 10%, 0%);
```

#### Notación por valores decimales de RGB, de 0 a 255

De una manera similar a la notación por porcentajes de RGB se puede definir un color directamente con valores decimales en un rango desde 0 a 255.

```
color: rgb(200,255,0);
```

### Color transparente

También existe el color transparente, que no es ningún color, sino que especifica que el elemento debe tener el mismo color que el fondo donde está. Este valor, **transparent**, sustituye al color. Podemos indicarlo en principio sólo para fondos de elementos, es decir, para el atributo `background-color`. `background-color: transparent;`

### FUENTES - FONT

**color**      valor RGB o nombre de color      `color: #009900;`  
    `color: red;`

Sirve para indicar el color del texto. Lo admiten casi todas las etiquetas de HTML. No todos los nombres de colores son admitidos en el estándar, es aconsejable entonces utilizar el valor RGB.

**font-size**      `xx-small | x-small | small | medium | large | x-large | xx-large`      `font-size: 12pt;`  
    `large`      `font-size: x-large;`  
    Unidades de CSS

Sirve para indicar el tamaño de las fuentes de manera más rígida y con mayor exactitud.

**font-family**      `serif | sans-serif | cursive | fantasy | monospace`      `font-family: arial, helvetica;`  
    Todas las fuentes habituales      `font-family: fantasy;`

Con este atributo indicamos la familia de tipografía del texto. Los primeros valores son genéricos, es decir, los exploradores los comprenden y utilizan las fuentes que el usuario tenga en su sistema.

También se pueden definir con tipografías normales, como ocurría en HTML. Si el nombre de una fuente tiene espacios se utilizan comillas para que se entienda bien.

**font-weight**      `normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900`      `font-weight: bold;`  
    `font-weight: 200;`

Sirve para definir la anchura de los caracteres, o dicho de otra manera, para poner negrillas con total libertad.

Normal y 400 son el mismo valor, así como bold y 700.

**font-style**      `normal | italic | oblique`      `font-style: normal;`  
    `font-style: italic;`

Es el estilo de la fuente, que puede ser normal, itálica u oblicua. El estilo oblique es similar al italic.

**font-variant**      `normal | small-caps | initial | inherit`      `font-variant: normal;`

**PÁRRAFOS - TEXT**

**line-height** normal y unidades CSS `line-height: 12px;`  
`line-height: normal;`

El alto de una línea, y por tanto, el espaciado entre líneas. Es una de esas características que no se podían modificar utilizando HTML.

**text-decoration** none | [ underline || overline || line-through ] `text-decoration: none;`  
`text-decoration: underline;`

Para establecer la decoración de un texto, es decir, si está subrayado, subrayado o tachado.

**text-align** left | right | center | justify `text-align: right;`  
`text-align: center;`

Sirve para indicar la alineación del texto. Es interesante destacar que las hojas de estilo permiten el justificado de texto, aunque recuerda que no tiene por qué funcionar en todos los sistemas.

**text-indent** Unidades CSS `text-indent: 10px;`  
`text-indent: 2in;`

Un atributo que sirve para hacer sangrado o márgenes en las páginas. Muy útil y novedosa.

**text-transform** capitalize | uppercase | lowercase | none `text-transform: none;`  
`text-transform: capitalize;`

Nos permite transformar el texto, haciendo que tenga la primera letra en mayúsculas de todas las palabras, todo en mayúsculas o minúsculas.

**FONDO - BACKGROUND**

**Background-color** Un color, con su nombre o su valor RGB `background-color: green;`  
`background-color: #000055;`

Sirve para indicar el color de fondo de un elemento de la página.

**Background-image** El nombre de la imagen con su camino relativo o absoluto `background-image: url(mármol.gif) ;`  
`background-image:`  
`url(http://www.x.com/fondo.gif)`

**BOX - CAJA**

**Margin-left**    Unidades CSS    `margin-left: 1cm;`  
`margin-left: 0,5in;`

Indicamos con este atributo el tamaño del margen a la izquierda

**Margin-right**    Unidades CSS    `margin-right: 5%;`  
`margin-right: 1in;`

Se utiliza para definir el tamaño del margen a la derecha

**Margin-top**    Unidades CSS    `margin-top: 0px;`  
`margin-top: 10px;`

Indicamos con este atributo el tamaño del margen arriba de la página

**Margin-bottom**    Unidades CSS    `margin-bottom: 0pt;`  
`margin-top: 1px;`

Con el se indica el tamaño del margen en la parte de abajo de la página

**Padding-left**    Unidades CSS    `padding-left: 0.5in;`  
`padding-left: 1px;`

Indica el espacio insertado, por la izquierda, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas.

El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.

**Padding-right**    Unidades CSS    `padding-right: 0.5cm;`  
`padding-right: 1pt;`

Indica el espacio insertado, en este caso por la derecha, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas.

El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.

**Padding-top**    Unidades CSS    `padding-top: 10pt;`  
`padding-top: 5px;`

Indica el espacio insertado, por arriba, entre el borde del elemento-continente y el contenido de este.

**Padding-bottom**    Unidades CSS    `padding-right: 0.5cm;`  
`padding-right: 1pt;`

Indica el espacio insertado, en este caso por abajo, entre el borde del elemento-continente y el contenido de este.

**Padding-bottom**    Unidades CSS    `padding-right: 0.5cm;`  
`padding-right: 1pt;`

Indica el espacio insertado, en este caso por abajo, entre el borde del elemento-continente y el contenido de este.

**Border-color**    color RGB y nombre de color    `border-color: red;`  
`border-color: #ffccff;`

Para indicar el color del borde del elemento de la página al que se lo aplicamos. Se puede poner colores por separado con los atributos border-top-color, border-right-color, border-bottom-color, border-left-color.

**Border-style**    none | dotted | solid | double | groove | ridge | inset | outset    `border-style: solid;`  
`border-style: double;`

El estilo del borde, los valores significan: none=ningun borde, dotted=punteado (no parece funcionar), solid=sólido, double=doble borde, y desde groove hasta outset son bordes con varios efectos 3D.

**border-width**    Unidades CSS    `border-width: 10px;`  
`border-width: 0.5in;`

El tamaño del borde del elemento al que lo aplicamos.

**float** none | left | right

float: right;

Sirve para alinear un elemento a la izquierda o la derecha haciendo que el texto se agrupe alrededor de dicho elemento. Igual que el atributo align en imágenes en sus valores right y left.

**clear** none | right | left

clear: right;

Si este elemento tiene a su altura imágenes u otros elementos alineados a la derecha o la izquierda, con el atributo clear hacemos que se coloque en un lugar donde ya no tenga esos elementos a el lado que indiquemos.

### Shorthand

Escribir de forma reducida nuestras reglas CSS para que nuestros archivos de estilo tengan menos peso y sean más entendibles a la hora de una actualización.

Según la W3C hay dos formas de escribir la misma regla de CSS: la **estándar** y la **shorthand**. Una es la larga y la otra es la reducida.

#### Propiedad **font** (fuente)

font-style || font-variant || font-weight || font-size / line-height || familia de fuente

**Ejemplo:** `p {font: italic normal bold 12px/14pt Verdana, Tahoma, Arial}`

#### Propiedad **background** (fondo)

background-color || background-image || background-repeat || background-attachment || background-position

**Ejemplo:** `body {background: #fff url(../images/ejemplo.gif) no-repeat fixed center}`

#### **margin** (Margen)

longitud | porcentaje | auto

##### Ejemplo:

`body {margin: 5px} /* todos los márgenes a 5px */` `p {margin: 2px 4px} /* márgenes superior e inferior a 2px, márgenes izquierdo y derecho a 4px */` `DIV {margin: 1px 2px 3px 4px} /* margen superior a 1px, right margin a 2px, bottom margin a 3px, left margin a 4px */`

#### **padding** (Relleno)

longitud | porcentaje | auto

##### Ejemplo:

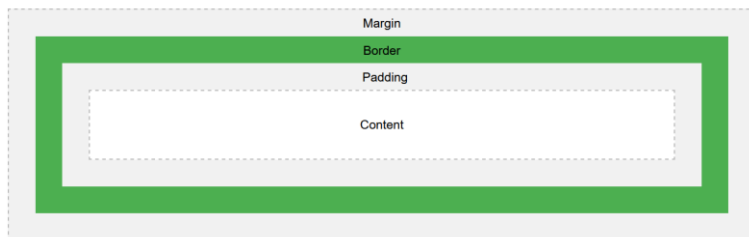
`body {padding: 2em 3em 4em 5em} /* Si definimos cuatro valores estamos aplicando el padding superior, derecho, inferior e izquierdo */` `body {padding: 2em 4em} /* Si definimos dos o tres valores, los valores faltantes se toman del lado opuesto: superior e inferior a 2em, derecho e izquierdo a 4em */` `Body {padding: 5em} /* Si definimos un solo valor se aplican a todos los lados */`

#### **border** (Borde)

border-width || border-style || color

**Ejemplo:** `h3 {border: thick dotted blue}`

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:



Explanation of the different parts:

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent

### Pseudo-element

Los **pseudo-element** sirven para aplicar estilos a partes más específicas dentro de una etiqueta. Es decir, para el ejemplo concreto de la etiqueta párrafo, con los pseudo-elementos podemos definir el estilo para la primera letra del párrafo y para la primera línea. Es decir, con CSS podemos definir el estilo de una etiqueta, pero con los pseudo-elementos no nos limitamos a definir el estilo para todo el contenido de esa etiqueta, sino que indicamos el estilo para una parte restringida de esa etiqueta. Existen diversos tipos de pseudo-elementos.

**Pseudo-element *first-letter***

Hacer más grande la primera letra de una página y decorarla de alguna manera.

Se utiliza de esta manera:

```
p::first-letter {  
  font-size: 200%;  
  color: #933; font-weight: bold;}
```

Así estamos asignando un tamaño de letra 200% más grande del propio del párrafo. También estamos cambiando el color de esa primera letra y poniéndola en negrita.

De entre todas las propiedades de estilos, sólo algunas se pueden aplicar a los pseudo-elementos *first-letter*. Son las siguientes, según la especificación del W3C: font properties, color properties, background properties, 'text-decoration', 'vertical-align' (sólo si 'float' está asignado a 'none'), 'text-transform', 'line-height', margin properties, padding properties, border properties, 'float', 'text-shadow' y 'clear'.

**Pseudo-element *first-line***

Asignar un estilo propio a la primera línea del texto. Un ejemplo de su uso sería el siguiente:

```
p::first-line {  
  text-decoration: underline;  
  font-weight: bold;}
```

Las propiedades de estilos que se pueden aplicar al pseudo-element *first-line* son las siguientes: font properties, color properties, background properties, 'word-spacing', 'letter-spacing', 'text-decoration', 'vertical-align', 'text-transform', 'line-height', 'text-shadow' y 'clear'.

**Uso de clases con los pseudo-elementos**

En determinadas ocasiones podemos necesitar crear una clase (class) de CSS a la que asignar los pseudo-elementos, de modo que estos no se apliquen a todas las etiquetas de la página. Por ejemplo, podemos desear que solamente se modifique el estilo de la primera línea del texto en algunos párrafos y no en todos los de la página.

Una clase se define con el nombre de la etiqueta seguido de un punto y el nombre de la clase. Si además deseamos definir un pseudo-elemento, deberíamos indicarlo a continuación, con esta sintaxis:

```
p.nombreclase:first-line {font-weight: bold;}
```

**Pseudo-elementos en CSS2**

Aparte de *first-line* y *first-letter*, en las especificaciones de CSS 2 existen otros pseudo elementos, se tratan de **before** y **after** y sirven para insertar "contenidos generados" antes y después del elemento al que asignemos estos pseudo-element.

Un ejemplo de ello es: `p.nota::before {content: "Nota: "}`

Así se ha definido una clase de párrafo llamada "nota" en la que se indica que antes de la propia nota se debe incluir el texto indicado, es decir, "Nota: ".

**Definir estilos utilizando clases**

Las clases nos sirven para crear definiciones de estilos que se pueden utilizar repetidas veces.

Para definir las utilizamos la siguiente sintaxis, un punto seguido del nombre de la clase y entre llaves los atributos de estilos deseados. De esta manera:

```
.nombredelaclase {atributo1: valor1; atributo2: valor2; ...}
```

Una vez tenemos una clase, podemos utilizarla en cualquier etiqueta HTML. Para ello utilizaremos el atributo class, poniéndole como valor el nombre de la clase, de esta forma:

```
<ETIQUETA class="nombredelaclase">
```

**Estilo en los enlaces**

Una técnica muy habitual, que se puede realizar utilizando las hojas de estilo en cascada y no se podía en HTML, es la definición de estilos en los enlaces, quitándoles el subrayado o hacer enlaces en la misma página con distintos colores.

**Enlaces normales**

```
a:link {atributos}
```

**Enlaces visitados**

```
a:visited {atributos}
```

**Enlaces activos** (Los enlaces están activos en el preciso momento en que se pincha sobre ellos)

```
a:active {atributos}
```

**Enlaces hover** (Cuando el ratón está encima de ellos)

```
a:hover {atributos}
```

El atributo para definir enlaces sin subrayado es **text-decoration:none**, y para darles color es **color:tu\_color**.

**URL como valor de un atributo:**

Determinados atributos de estilos, como **background-image** necesitan una URL como valor, para indicarnos las podemos definir tanto caminos relativos como absolutos. Así pues, podemos indicar la URL de la imagen de fondo de estas dos maneras:

```
background-image: url(fondo.gif) En caso de que la imagen esté en el mismo directorio que la página.  
background-image: url(http://www.desarrolloweb.com/images/fondo.gif)
```



**Vinculación entre html y css con el atributo id:**

#valor del atributo id

`<p id="prueba">Texto</p>`En CSS: `#prueba {color: #fa9;}`**Vinculación entre html y css con cualquier atributo:**`<p name="miparrafo">Texto</p>`En CSS: `p[name] {color: #abc;} o bien p[name="miparrafo"] {color: #abc;}`

**Pseudo Clase :nth-child()** Esta pseudo clase encuentra un hijo específico siguiendo la estructura de árbol de HTML. Por ejemplo, con el estilo **span:nth-child(2)** estamos referenciando el elemento **<span>** que tiene otros elementos **<span>** como hermanos y está localizado en la posición **2**. Este número es considerado el índice. En lugar de un número podemos usar las palabras clave **odd** y **even** para referenciar elementos con un índice impar o par respectivamente (por ejemplo, **span:nthchild(odd)**).

**Pseudo Clase :first-child** Esta pseudo clase es usada para referenciar el primer hijo, similar a **:nth-child(1)**.

**Pseudo Clase :last-child** Esta pseudo clase es usada para referenciar el último hijo.

**Pseudo Clase :only-child** Esta pseudo clase es usada para referenciar un elemento que es el único hijo disponible de un mismo elemento padre.

**Pseudo Clase :not()** Esta pseudo clase es usada para referenciar todo elemento excepto el declarado entre paréntesis.

## Selectores

CSS3 también incorpora nuevos selectores que ayudan a llegar a elementos difíciles de referenciar utilizando otras técnicas.

**Selector >** Este selector referencia al elemento de la derecha cuando tiene el elemento de la izquierda como padre. Por ejemplo, **div > p** referenciará cada elemento **<p>** que es hijo de un elemento **<div>**.

**Selector +** Este selector referencia elementos que son hermanos. La referencia apuntará al elemento de la derecha cuando es inmediatamente precedido por el de la izquierda. Por ejemplo, **span + p** afectará a los elementos **<p>** que son hermanos y están ubicados tras un elemento **<span>**.

**Selector ~** Este selector es similar al anterior, pero en este caso el elemento de la derecha no tiene que estar ubicado inmediatamente después del de la izquierda.

### Más potencia en la selección

`p[name^="mi"] {color: #abc;}` Quiere decir que el valor en nombre comienza por mi

`p[name$="mi"] {color: #abc;}` Quiere decir que el valor en nombre finaliza por mi

`p[name*="mi"] {color: #abc;}` Quiere decir que el valor en nombre contiene a mi

### Introducción a las capas, la etiqueta <div> utilizada para construirla y los atributos CSS con los que podemos aplicar estilos.

Con **<div>** podemos aplicar estilo a partes de la página HTML.

La diferencia entre **<span>** y **<div>** es que con esta última podemos aplicar estilo a una parte más amplia de la página, por ejemplo, a tres párrafos. Además, que la etiqueta **<div>** tiene un uso adicional que es el de crear divisiones en la página a las que podremos aplicar una cantidad adicional de atributos para modificar sus comportamientos. Su uso más destacado es el de convertir esa división en una capa.

**Una capa es una división**, una parte de la página, **que tiene un comportamiento muy independiente** dentro de la ventana del navegador, ya que la podemos colocar en cualquier parte de la misma y la podremos mover por ella independientemente.

Una capa puede tener cualquier atributo de estilos de los que hemos visto. Así, el atributo **color** indica el color del texto de la capa, el atributo **font-size** indica el tamaño del texto y así con todos los atributos ya vistos.

### Los atributos para que la división sea una capa:

#### position, left, top, width, height, z-index

El primero, **position**, indica que se posicione de manera absoluta en la página y los segundos, **left** y **top**, son la distancia desde el borde izquierdo de la página y el borde superior.

Hay otros atributos especiales para capas como **width** y **height** para indicar la anchura y altura de la capa, **z-index** que sirve para indicar qué capas se ven encima de qué otras, **clip** que sirve para recortar una capa y hacer que partes de ella no sean visibles, o **visibility** para definir si la capa es visible o no.

### Atributo position

Indica el tipo de posicionamiento de la capa. Puede tener dos valores, **relative** o **absolute**.

- **relative** indica que la posición de la capa es relativa al lugar donde se estaba escribiendo en la página en el momento de escribir la capa con su etiqueta
- **absolute** indica que la posición de la capa se calcula con respecto al punto superior izquierdo de la página

### Atributo top

Indica la distancia en vertical donde se colocará la capa. Si el atributo position es **absolute**, top indica la distancia del borde superior de la capa con respecto al borde superior de la página. Si el atributo position era **relative**, top indica la distancia desde donde se estaba escribiendo en ese momento en la página hasta el borde superior de la capa.

### Atributo left

Básicamente funciona igual que el atributo top, con la diferencia que el atributo left indica la distancia en horizontal a la que estará situada la capa.

### Atributo height

Sirve para indicar el tamaño de la capa en vertical, es decir, su altura.

### Atributo width

Indica la anchura de la capa

### Atributo visibility

Sirve para indicar si la capa se puede ver en la página o permanece oculta al usuario. Este atributo puede tener tres valores.

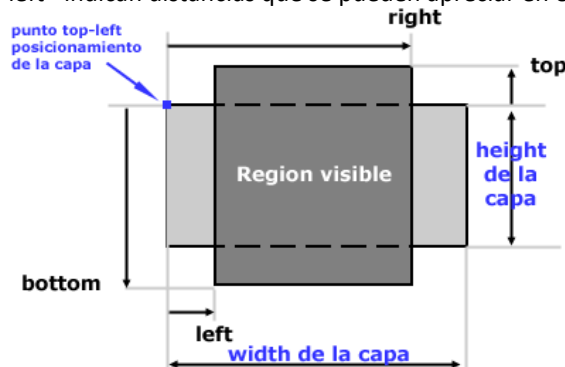
- **visible** sirve para indicar que la capa se podrá ver.
- **hidden** indicará que la capa está oculta.
- **inherit** es el valor por defecto, que quiere decir que hereda la visibilidad de la capa donde está metida la capa en cuestión. Si la capa no está metida dentro de ninguna otra se supone que está metida en la capa documento, que es toda la página y que siempre está visible.

### Atributo z-index

Sirve para indicar la posición sobre el eje z que tendrán las distintas capas de la página. Dicho de otra forma, con z-index podemos decir qué capas se verán encima o debajo de otras, en caso de que estén superpuestas. El atributo z-index toma valores numéricos y a mayor z-index, más arriba se verá la página.

### Atributo clip

Sirve para recortar determinadas áreas de la capa. Se verá la zona indicada en el clip, la otra parte de la capa no se visualiza. El clipping se indica por medio de 4 valores, con esta sintaxis: `rect (<top>px, <right>px, <bottom>px, <left>px)`. Los valores <top>, <right>, <bottom> y <left> indican distancias que se pueden apreciar en este esquema.



Este es un ejemplo de capa que utiliza todos los atributos que hemos visto en este artículo y alguno más para aplicar estilo a la capa.

```
<div>{clip: rect(0,158,148,15); height: 250px; width: 170px; left: 10px; top: 220px; position: absolute; visibility: visible; z-index:10; font-size: 14pt; font-family: verdana; text-align: center; background-color: #bbbbbb";}
```

## Modelo de Caja de CSS

### Una primera aproximación visual

Es uno de los elementos básicos de las Hojas de Estilo en Cascada y por lo tanto su correcta interpretación resulta fundamental a la hora de lograr los resultados deseados en un diseño, por más simple que éste resulte.

Para entrar en tema, vamos a construir un sencillo ejemplo utilizando un único elemento `<div>` al que aplicaremos un estilo. El resultado producido será analizado con la ayuda de una figura en la que hemos modelado el orden de apilado de los elementos del `<div>` en una disposición tridimensional que nos ayudará a comprenderlo.

Supongamos el siguiente código:

#### El elemento `<div>`

```
<div>
<p>Este es el contenido de nuestra caja.</p>
<p>Este es el contenido de nuestra caja.</p>
<p>Este es el contenido de nuestra caja.</p>
<p>Este es el contenido de nuestra caja.</p>
</div>
```

#### El estilo

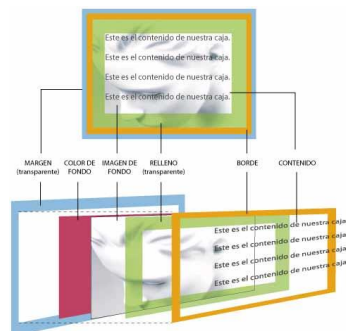
```
div {
background-color: #be4061; /*color bordó para el fondo*/
background-image: url('cabeza.jpg');
border: 10px solid #e7a219; /*color naranja para el borde*/
margin: 10px;
padding: 20px;
}

p {
margin: 0 0 20px 0; /*margen inferior de 20 px para el párrafo*/
padding: 0;
}
```

El código anterior generará una caja como la que muestra la figura siguiente, en la que adicionalmente se ha dado color a los elementos transparentes (margen y padding) solo para hacerlos visibles.

Un detalle interesante que puede apreciarse en la representación tridimensional en que la capa superior del apilamiento no es el borde, como podría suponerse intuitivamente sino el margen.

La capa situada encima de todas las demás es la de Contenido.



#### Áreas y propiedades

Cada caja en CSS posee, además de su área de Contenido, otras tres áreas opcionales:

- Área de Margen - **Margin**
- Área de Relleno - **Padding**
- Área de Borde - **Border**

Cada área, a su vez, puede dividirse en cuatro segmentos según su posición: left, right, top y bottom.

Por ejemplo, la siguiente sentencia asignará un margen homogéneo de 20 píxeles alrededor de la caja:

```
div { margin: 20px }
```

Si en cambio se desea asignar valores distintos a cada uno de los segmentos, pueden reflejarse los cuatro valores numéricos siguiendo el orden top - right - bottom - left.

El siguiente ejemplo asigna 10 píxeles arriba, 5 a la derecha, 20 abajo y nada a la izquierda:

```
div { margin: 10px 5px 20px 0 }
```

Pueden especificarse valores también con la siguiente notación, en la que ya no es necesario mantener el orden:

```
div {
margin-top: 10px ;
margin-right: 5px ;
margin-bottom: 20px ;
} El margin-left:0px no se pone.
```

La lista completa de propiedades es la siguiente:

#### Propiedades del Margen

"margin-top", "margin-right", "margin-bottom", "margin-left" y "margin"

#### Propiedades del Padding (relleno)

"padding-top", "padding-right", "padding-bottom", "padding-left" y "padding"

#### Propiedades del Borde

##### 1) Ancho (width)

"border-top-width", "border-right-width", "border-bottom-width", "border-left-width" y "border-width". Pueden ser valores específicos o los valores "thin" (fino), "medium" (medio) y "thick" (grueso)

##### 2) Color (color)

"border-top-color", "border-right-color", "border-bottom-color", "border-left-color" y "border-color"

##### 3) Estilo (style)

"border-top-style", "border-right-style", "border-bottom-style", "border-left-style" and "border-style". Toma una serie de posibles valores, tales como: none, hidden, dotted, dashed, solid, double, groove, ridge, inset y outset. Es una propiedad algo conflictiva ya que no todos los navegadores interpretan sus valores de la misma manera.

Como corolario de esta aproximación al modelo de caja resta analizar qué es lo que se verá en cada área cuando la página se muestre en un navegador:

- En el área de Contenido y en la de Relleno se verá aquello que se determine en la propiedad "background" del elemento (un color o una imagen, según el orden de apilado).
- En el área de Borde se verá aquello que se determine en las propiedades del Borde (ancho, color y estilo).
- El área de Margen es siempre transparente.

Básicamente, la "caja" en CSS puede ser asimilada a una tabla con una sola celda.

De forma obviamente rectangular, esa caja puede tener bordes (border), márgenes transparentes por fuera de los bordes (margin) y relleno transparente por dentro de los bordes (padding). Cualquiera de estos atributos puede ser asignado para cada uno de los cuatro lados de la caja separadamente.

El contenido de la caja se ubicará dentro del área de padding (relleno).

En general, podemos hablar de dos tipos de diseño:

- Los diseños "líquidos" no asignan un valor específico al ancho (width) y alto (height) de la caja, por lo que resulta que esas dimensiones surgirán del contenedor de la caja y de la extensión del contenido. Concretamente, el ancho de esa caja será todo el permitido por la estructura que la contenga (utilizará todo el ancho disponible) y el alto será el necesario para mostrar todo el contenido ("crecerá" hacia abajo todo lo necesario).
- Los diseños "estáticos" surgen cuando se decide asignar a la caja un ancho específico mediante la asignación de un valor concreto a la propiedad "width".

#### Comportamientos inline y block y cómo afectan al flujo de la página

Cuando tratamos con etiquetas, existen dos modos principales de comportamiento. Etiquetas como una imagen, o una negrita, que funcionan en línea ("inline"), es decir, que se colocan en la línea donde se está escribiendo y donde los elementos siguientes, siempre que también sean "inline", se posicionan todo seguido en la misma línea. Tenemos por otra parte los elementos que funcionan como bloque ("block") que implican saltos de línea antes y después del elemento. Por ejemplo, los párrafos o encabezamientos son elementos con comportamiento predeterminado tipo "block".

#### Atributo CSS Float y el flujo

Otro atributo que afecta al flujo de los elementos en la página es el atributo float de CSS, que se utiliza bastante en la maquetación web. Este atributo podemos utilizarlo sobre elementos de la página de tipo "block" y lo que hace es convertirlos, en "flotantes". Con

float podemos indicar tanto left como right y conseguiremos que los elementos se posicionen a la izquierda o la derecha, con el contenido que se coloque a continuación rodeando al elemento flotante. La diferencia es que los elementos continúan siendo tipo "block" y aceptan atributos como el margen (atributo CSS margin), para indicar que haya un espacio en blanco a los lados y arriba y abajo del elemento.

Por ejemplo, los elementos de las listas (etiqueta li) son por defecto de tipo "block", por eso aparecen siempre uno abajo de otro, en líneas consecutivas. Pero nosotros podríamos cambiar ese comportamiento con:

```
li{
  float: right;
}
```

Así, una lista como esta:

```
<ul>
<li>Elemento1</li>
```

```
</li>Elemento2</li>  
</li>Elemento3</li>  
</ul>
```

Veríamos como el primer elemento aparece a la derecha del todo y los otros elementos van colocándose en la misma línea en el siguiente espacio libre que haya. Así, el segundo elemento se colocaría en la misma línea, todo a la derecha que se puede, conforme al espacio que se tenga en el contenedor donde estén colocados.

#### Flujo y el atributo position

Con el atributo CSS position, podemos indicar otros valores que hacen que los elementos salgan del flujo HTML y se posicionen en lugares fijos, que no tienen que ver con la posición en la que aparezcan en el código HTML. Por ejemplo:

```
div {position: absolute; top: 10px; left: 100px;}
```

Este elemento tiene posicionamiento absoluto, hace que ese elemento quede fuera del flujo de elementos en la página y entonces aparecería en el lugar que se indica con los atributos top y left (top indica la distancia desde la parte de arriba y left la distancia desde el borde izquierdo). Los otros elementos que formen parte del flujo de la página no quedan afectados por los elementos con posicionamiento absoluto.

Otro valor para el atributo **position** que hace que los elementos queden posicionados fuera del flujo normal de elementos en la página es "**fixed**".

#### Atributos para posicionamiento CSS

##### Atributo position:

Este atributo es el principal para definir el tipo de posicionamiento de un elemento. Sus posibles valores son **absolute**, **fixed**, **relative**, **static** e **inherit**.

##### Atributos top, left, right, bottom:

Sirven para indicar la posición de un elemento, cuando éste tiene los valores de position "absolute", "relative" o "fixed" (en otros valores del atributo position estos atributos son ignorados). El atributo top indica la distancia desde el borde superior de la página y left desde el borde de la izquierda. También se puede indicar opcionalmente la posición con bottom, que es la distancia desde abajo y right, que es la distancia desde la derecha.

#### Atributos float y clear:

**float** sirve para establecer que un elemento tiene que "flotar", colocándose los valores "right" o "left" para que floten a izquierda o derecha. Con el atributo **clear** hacemos que el elemento se coloque en el primer área libre que tenga al lugar donde se indique. Por ejemplo el valor de clear "right" hace que el elemento se coloque en el primer lugar donde no tenga ningún elemento flotando a la derecha. El valor de clear "both" hace que el elemento se coloque donde no tenga elementos flotando, tanto a la derecha como a la izquierda.

#### Atributo clip:

Establece un área de recorte de la porción visible de un elemento.

#### Atributo display:

Especifica el tipo de caja que debe tener un elemento, que puede ser de diversas formas. Este atributo también tiene bastante utilización y entre los valores más corrientes podríamos destacar: "**none**", que hace que esa caja o elemento no aparezca en la página ni se reserve espacio para ella. "**block**", que sirve para que la caja sea un bloque y se muestre en una línea o líneas independientes de otros elementos de la página. "**inline**", que indica que esa caja tiene que mostrarse en la misma línea que otros elementos escritos antes o después.

#### Atributo overflow:

Este atributo sirve para decir qué es lo que pasa con los elementos que no caben en una caja debido a las dimensiones de la misma y del contenido que tenga.

#### Atributo visibility:

Atributo para definir la visibilidad de un elemento. Con este atributo podemos decir que ciertos elementos de la página sean visibles o invisibles, pero atención, aunque un elemento sea invisible, continúa ocupando espacio en la página. Si queremos que no sea visible y no se le reserve espacio en la página, hay que utilizar el atributo display con el valor "**none**". Los valores más corrientes de visibility son: "**visible**", que hace que el elemento se vea (valor por defecto) y "**hidden**", que hace que el elemento sea invisible, aunque continúe ocupando espacio.

#### Atributo z-index:

Este atributo tiene como valor cualquier número entero. Sirve para indicar qué capa se tiene que ver por encima o por debajo de otra u otras, en caso que varias capas estén superpuestas. A mayores valores de z-index, la capa se coloca más al frente, tapando otras capas que tengan valores menores de z-index.

#### position: static

Es el valor predeterminado del atributo y el posicionamiento normal de los elementos en la página. Quiere decir que los elementos se colocarán según el flujo normal del HTML, es decir, según estén escritos en el propio código HTML. Static no provoca ningún posicionamiento especial de los elementos y por tanto, los atributos top, left, right y bottom no se tendrán en cuenta.

Podemos ver un ejemplo de posicionamiento static:

```
div {position: static; background-color: #ff9; padding: 10px; width: 300px;}
```

Esto es una capa con posicionamiento estático. No haría falta indicarlo porque es el valor por omisión de **position**.

#### position: absolute

El valor absolute en el atributo position permite posicionar elementos de manera absoluta, esto es de manera definida por valores de los atributos top, left, bottom y right, que indican la distancia con respecto a un punto. Las capas o elementos con posicionamiento absoluto quedan aparte del flujo normal del HTML, quiere decir esto que no se afectan por el lugar donde aparezcan en el código HTML y tampoco afectan ellas a otros elementos que sí que formen parte del flujo normal del HTML.

Los valores top, left, bottom y right se expresan con *unidades CSS* y son una distancia con respecto al primer elemento contenedor que tenga un valor de *position* distinto de **static**. Si todos los contenedores donde esté la capa posicionada con position absolute (todos sus padres hasta llegar a body) son static, simplemente se posiciona con respecto al lado superior de la página, para el caso de top, el inferior para bottom, del lado izquierdo para left o el derecho, en el caso de utilizar right.

Veamos el siguiente código HTML en el que hemos preparado varias capas con position absolute, pero con distintas características:

```
div {position: absolute; width: 300px; height: 140px; top: 100px; left: 30px; background-color: #ff8800; color: #fff; padding: 15px; z-index: 2;}>
```

Esta capa tiene posicionamiento absoluto. Me permite especificar *top* y *left* para colocarla con respecto a la esquina superior izquierda.

```
div {position: absolute; width: 820px; height: 30px; padding: 10px; background-color: #ddf; top: 150px; left: 10px; z-index: 1;}
```

Posicionamiento absoluto con **z-index** menor (la capa aparece por debajo de otras que se superponen con z-index mayor).

Veamos un ejemplo donde vamos a colocar una capa con posicionamiento absoluto y dentro varias capas también posicionadas con absolute.

```
<div style="position: absolute; top: 100px; left: 200px; background-color: #ff9966; width: 400px; height: 100px;">
  <div style="position: absolute; top: 10px; left: 10px;">
    Uno
  </div>
  <div style="position: absolute; top: 10px; left: 100px;">
    Dos
  </div>
  <div style="position: absolute; top: 10px; left: 200px;">
    Tres
  </div>
</div>
```

En este caso la primera capa, que no está dentro de ninguna otra, se posiciona con top y left con respecto a la esquina superior izquierda del espacio disponible en el navegador para el cuerpo de la página. Las capas anidadas están también con position: absolute, pero al estar dentro de otra capa que tiene posicionamiento distinto de static, sus valores top y left son relativos a la esquina superior izquierda de la capa que las contiene.

#### **position: relative**

El valor relative en el atributo position indica que la capa sí forma parte del flujo normal de elementos de la página, por lo que su posición dependerá del lugar donde esté en el código y el flujo HTML. Además, las capas con posicionamiento relative, admiten los valores top y left para definir la distancia a la que se colocan con respecto al punto donde esté en ese momento el flujo normal del HTML. Como afectan al mencionado flujo del HTML, los elementos colocados tras las capas relative, tendrán en cuenta sus dimensiones para continuar el flujo y saber dónde colocarse. Sin embargo, no se tendrá en cuenta los top y left configurados.

Veamos un ejemplo que quizás aclare las cosas.

```
<h1>Hola</h1>
<div style="background-color: #606; color: #ffc; padding: 10px; text-align: center; width: 300px;">Hola esto es una prueba</div>
<div style="position: relative; width: 300px; padding: 10px; background-color: #066; color: #ffc; top: 100px; left: 30px;">capa de
posicionamiento relative<br>Se tiene en cuenta esta capa para posicionar las siguientes.</div>
<h2>hola de nuevo!</h2>
```

Las etiquetas h1 y h2 respetan el flujo HTML y también tenemos un elemento *div* donde no se ha especificado nada en position, luego es static y por tanto también es afectada por el flujo. Hay una capa *relative*, en el segundo elemento div, que también se posiciona con respecto al flujo normal. Como tiene un top y left, aparece un poco desplazada del lugar que le tocaría con respecto al flujo.

El último h2 que aparece se coloca teniendo en cuenta el flujo y tiene en cuenta la capa *relative*, por eso deja un espacio en blanco arriba, pero no atiende a la posición real de ésta, que se marcó con los atributos top y left.

#### **position: fixed**

Este atributo sirve para posicionar una capa con posicionamiento absoluto y su posición final será siempre fija, es decir, aunque se desplace el documento con las barras de desplazamiento del navegador, siempre aparecerá en la misma posición.

El lugar donde se "anclará" la capa siempre es relativo al cuerpo (el espacio disponible del navegador para la página). Si utilizamos top y left, estaremos marcando su posición con respecto a la esquina superior izquierda y si utilizamos bottom y right su posición será relativa a la esquina inferior derecha.

Veamos un ejemplo.

```
div {position: fixed; width: 300px; height: 140px; top: 100px; left: 30px;
background-color: #ff8800; color: #fff; padding: 15px; z-index: 1;}
```

Esta capa tiene posicionamiento fixed.

#### **position: inherit**

El valor inherit indica que el valor de position tiene que heredarse del elemento padre.

#### **Overflow**

sirve en el modelado de cajas para indicar al navegador qué es lo que debe hacer con el contenido que no cabe dentro de una capa, según las dimensiones que se le han asignado.

Como sabemos, a las capas (elementos div) podemos asignarles un tamaño, en anchura y altura. Pero muchas veces el contenido que colocamos en la capa sobrepasa el espacio que hemos destinado a ella. Entonces lo que suele ocurrir es que la capa crece lo suficiente para que el contenido colocado dentro quepa. Habitualmente las capas crecen en altura, por lo que a más contenido más tamaño tendrá en altura. Este es un comportamiento que podemos alterar con el uso del atributo *overflow*.

*overflow* permite que se recorte el contenido de una capa, para mostrar únicamente el contenido que quepa, según sus dimensiones. Para acceder al contenido que no se muestra, porque no cabe en la capa, se puede configurar overflow para que aparezcan unas barras de desplazamiento.

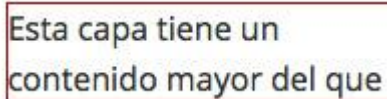
Así pues, pasemos directamente a ver cuáles son los atributos posibles con el atributo *overflow*:

- **visible:** Este valor indica que se debe mostrar todo el contenido de la capa, aunque no quepa en el tamaño configurado.
- **hidden:** Este valor indica que los contenidos que, por el tamaño de la capa, no quepan en la misma, se deben recortar. Por ello, la capa tendrá siempre el tamaño configurado, pero los contenidos en ocasiones podrán no verse por completo.
- **scroll:** Este valor indica que la capa debe tener el tamaño que se haya configurado inicialmente y que además se deben mostrar unas barras de desplazamiento, para mover el contenido de la capa dentro del espacio de la misma. Las barras de desplazamiento siempre salen, se requieran o no.
- **auto:** Con este valor también se respetarán las dimensiones asignadas a una caja. El contenido será recortado, pero aparecerán las barras de desplazamiento para moverlo. Sin embargo, en este caso las barras de desplazamiento podrán salir o no, depende de si son necesarias o no para ver todo el contenido de la capa.

Es muy utilizado para mostrar textos largos, que se desean integrar dentro de otro texto o una interfaz donde no tenemos espacio disponible para colocarlos o no deseamos que crezcan más de la cuenta. Ejemplo:

```
div {overflow: auto; width: 300px; height: 100px; background-color: #ededed; border: 1px solid #990000;}
```

Ahora veamos otro ejemplo, en el que simplemente se recorta el texto que no cabe en la capa. Hemos indicado `overflow: hidden`, por lo que el texto que sobra no se va a visualizar.



Esta capa tiene un  
contenido mayor del que

En este caso vemos como el texto aparece recortado, porque no cabe en el espacio asignado de la capa. El código sería como el que sigue:

```
div {overflow: hidden; width: 200px; height: 50px; border: 1px solid #990000;}
```

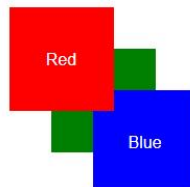


**El problema**

En el siguiente HTML tienes tres elementos `<div>`, y cada `<div>` contiene un único elemento `<span>`. Cada `<span>` tiene un color de fondo — rojo, verde y azul respectivamente. Cada `<span>` está también posicionado absolutamente cerca de la posición superior izquierda del documento, escasamente solapando a los otros elementos `<span>` así que puedes ver cuáles están apilados en frente de cada cual. El primer `<span>` tiene un valor `z-index` de 1, mientras que los otros dos no tienen ningún valor `z-index` fijado.

Aquí tenemos el HTML y CSS básicos para entenderlo. También he incluido una *demo* visual (vía [Codepen](#)) con el código que se muestra a continuación:

```
<div>
<span class="rojo">Rojo</span>
</div>
<div>
<span class="verde">Verde</span>
</div>
<div>
<span class="azul">Azul</span>
</div>
.rojo, .verde, .azul {
position: absolute;
}
.rojo {
background: red;
z-index: 1;
}
.verde {
background: green;
}
.azul {
background: blue;
}
```



### Ordenar el apilamiento

Z-index parece sencillo: elementos con un z-index mayor son apilados delante de los elementos con un menor z-index, ¿no? Bien, de hecho, **no**. Esto es parte del problema del z-index. Parece sencillo, así que los desarrolladores no dedican tiempo a leer sus reglas. Cada elemento en un documento HTML puede estar delante o detrás de otro elemento en el documento. Esto se conoce como el orden de apilamiento. Cuando el z-index y las propiedades de posición no están declaradas, las reglas son muy simples: básicamente, el orden de apilamiento es el mismo que el orden de aparición en el HTML. (DE ACUERDO, es de hecho un [poco más complicado](#) que eso, pero si no estás usando márgenes negativos para solapar elementos en la misma línea de posición, probablemente no te encontrarás con elementos problemáticos).

Cuando introduces la propiedad de posición en la mezcla, cualesquiera elementos posicionados (y sus hijos) se muestran delante de elementos no posicionados (decimos que un elemento está “posicionado” cuando tiene un valor de posición distinto de “static”, como “relative”, “absolute”, etc.)

Lo primero de todo, z-index solo funciona con elementos posicionados. Si intentas asignarle un z-index a un elemento sin posición especificada, no hará nada. En segundo lugar, los valores z-index pueden crear contextos de apilamiento.

### Contextos de apilamiento

Grupos de elementos con un padre común que se mueven hacia adelante o hacia atrás juntos en el orden de apilamiento, se integran en lo que es conocido como contexto de apilamiento. Cada contexto de apilamiento tiene un elemento HTML como su elemento raíz. Cuando un nuevo contexto de apilamiento se forma en un elemento, éste contexto confina todos sus elementos hijos en un lugar particular del orden. Eso significa que, si un elemento es contenido en un contexto de apilamiento en lo más bajo del orden, no hay manera de que aparezca delante de otro elemento en un contexto de apilamiento que está en un nivel superior en el orden, ¡incluso con un z-index de mil millones!

Los nuevos contextos de apilamiento pueden ser formados en un elemento de tres maneras:

- Cuando un elemento es el elemento raíz de un documento (el elemento <html>).
- Cuando un elemento tiene un valor de posición que no sea “static” y valor de z-index distinto a “auto”.
- Cuando un elemento tiene un valor de opacidad menor que uno.

### Determinando una posición de elemento en el orden de apilamiento

Vamos a dividir el orden en individuales contextos de apilamiento:

#### Orden de apilamiento dentro del mismo contexto de apilamiento

Aquí están las reglas básicas de orden de apilamiento dentro de un contexto simple (desde el fondo hacia delante):

1. El contexto de apilamiento del elemento raíz.
2. Elementos posicionados (y sus hijos) con valores z-index negativos (los valores más altos son apilados delante de valores menores; elementos con el mismo valor son apilados de acuerdo a su aparición en el HTML).
3. Elementos no posicionados (ordenados por aparición en el HTML).
4. Elementos posicionados (y sus hijos) con un valor z-index de auto (ordenados por aparición en el HTML).
5. Elementos posicionados (y sus hijos) con valores z-index positivos (los valores mayores son apilados delante de valores menores; elementos con el mismo valor son apilados de acuerdo a su aparición en el HTML).

**Nota:** elementos posicionados con z-indexes negativos se ordenan primero dentro de un contexto de apilamiento, lo cual significa que aparecen detrás de otros elementos. A causa de esto, llega a ser posible para un elemento aparecer detrás de su propio padre, lo cual normalmente no es posible. Esto funcionará solamente si el elemento parental está en el mismo contexto de apilamiento y no es el elemento raíz de ese contexto de apilamiento

### Orden de apilamiento global

Con una firme comprensión de cómo funcionan los contextos de apilamiento, y de cómo funciona el orden de apilamiento dentro de un contexto de apilamiento, imaginar dónde un elemento particular podrá aparecer en el orden de apilamiento global se convierte en una tarea más sencilla.

La clave es evitar la confusión generada cuando se forman nuevos contextos de apilamiento. Si pones un z-index con un valor de mil millones en un elemento y no se mueve hacia delante en el orden de apilamiento, echa un vistazo a su ancestro y mira si alguno de sus padres está formando contextos de apilamiento. Si lo hacen, tu z-index es inútil.

### box-sizing

Valores: content-box y border-box. Con content-box (valor por omisión, a la anchura de la caja –width- se añaden los bordes y los paddings), con border-box, en el width están incluidos los bordes y los paddings.

box-sizing: content-box | border-box;

### border-radius

Redondea los vértices de la caja. A mayor número más redondeo

border-radius: 20px;

**box-shadow**

Proporciona sombra a la caja. Son necesarios 6 parámetros: color, coordenada x, coordenada y, distancia de difuminación, desparramar la sombra, interna/externa. Las coordenadas pueden ser valores positivos, negativos o cero.

```
Box-shadow: rgb(120,200,90) 5px 5px 10px 5px inset;
```

**text-shadow**

Da sombra a las letras no a la caja.

```
text-shadow: rgb(0,0,150) 3px 3px 5px;
```

Son necesarios 4 parámetros: color, coordenada x, coordenada y, distancia de difuminación

**@font-face**

La propiedad **@font-face** permite a los diseñadores proveer un archivo conteniendo una fuente específica para mostrar sus textos en la página. Ahora podemos incluir cualquier fuente que necesitemos con solo proveer el archivo adecuado

```
#titulo
{
    font: bold 36px MiNuevaFuente, verdana, sans-serif;
    text-shadow: rgb(0,0,150) 3px 3px 5px;
}
@font-face
{
    font-family: 'MiNuevaFuente';
    src: url('font.ttf');
```

**linear-gradient**

Los gradientes son configurados como fondos, por lo que podemos usar las propiedades background o background-image para declararlos. La sintaxis para los valores declarados en estas propiedades es *linear-gradient*(posición inicio, color inicial, color final).

Los atributos de la función *linear-gradient()* indican el punto de comienzo y los colores usados para crear el gradiente. El primer valor puede ser especificado en pixeles, porcentaje o usando las palabras clave top, bottom, left y right (como hicimos en nuestro ejemplo). El punto de comienzo puede ser reemplazado por un ángulo para declarar una dirección específica del gradiente

```
background: linear-gradient(30deg, #FFFFFF, #006699);
background: linear-gradient(top, #FFFFFF 50%, #006699 90%);
```

**radial-gradient**

```
background: radial-gradient(center, circle, #FFFFFF 0%, #006699 200%);
```

La posición de comienzo es el origen y puede ser declarada en pixeles, porcentaje o una combinación de las palabras clave center, top, bottom, left y right. Existen dos posibles valores para la forma (circle y ellipse) y la terminación para el color indica el color y la posición donde las transiciones comienzan.

**rgba**

La función rgba() tiene cuatro atributos. Los primeros tres son similares a los usados en rgb() y simplemente declaran los valores para los colores rojo, verde y azul en números decimales del 0 al 255. El último, en cambio, corresponde a la nueva capacidad de opacidad. Este valor se debe encontrar dentro de un rango que va de 0 a 1, con 0 como totalmente transparente y 1 como totalmente opaco.

```
rgba(50,100,20,0.5)
```

**hsla**

La función hsla() es simplemente un función diferente para generar colores. La sintaxis de esta función es: **hsla(tono, saturación, luminosidad, opacidad)**.

```
color: hsla(120, 100%, 50%, 0.5);
```

**tono** representa el color extraído de una rueda imaginaria y es expresado en grados desde 0 a 360. Cerca de 0 y 360 están los colores rojos, cerca de 120 los verdes y cerca de 240 los azules. El valor **saturación** es representado en porcentaje, desde 0% (escala de grises) a 100% (todo color o completamente saturado). La **luminosidad** es también un valor en porcentaje desde 0% (completamente oscuro) a 100% (completamente iluminado). El valor 50% representa luminosidad normal o promedio. El último valor, representa la opacidad.

**outline**

Esta propiedad era usada para crear un segundo borde, y ahora ese borde puede ser mostrado alejado del borde real del elemento con outline-offset.

```
outline: 2px dashed #009;
outline-offset: 15px;
```

### border-image

La propiedad **border-image** toma una imagen y la utiliza como patrón. De acuerdo a los valores otorgados, la imagen es cortada como un pastel, las partes obtenidas son, posteriormente, ubicadas alrededor del objeto para construir el borde. Necesitamos especificar **tres atributos**: el nombre del archivo de la imagen, el tamaño de las piezas que queremos obtener del patrón y algunas palabras clave para declarar cómo las piezas serán distribuidas alrededor del objeto. La **palabra clave round** considerará qué tan largo es el lado a ser cubierto y ajustará el tamaño de las piezas para asegurarse que cubren todo el lado y ninguna pieza es cortada. Finalmente, la **palabra clave stretch** estira solo una pieza para cubrir el lado completo.

```
border-image: url("diamonds.png") 29 stretch;
```

estamos definiendo un borde de 29 píxeles para la caja de nuestra cabecera y luego cargando la imagen diamonds.png para construir ese borde. El valor 29 en la propiedad border-image declara el tamaño de las piezas y stretch es uno de los métodos disponibles para distribuir estas piezas alrededor de la caja. Existen tres valores posibles para el último atributo. La **palabra clave repeat** repetirá las piezas tomadas de la imagen todas las veces que sea necesario para cubrir el lado del elemento. En este caso, el tamaño de las piezas es preservado y la imagen será cortada. Se puede también usar border-with para especificar diferentes tamaños para cada lado del elemento (la propiedad border-with usa cuatro parámetros, con una sintaxis similar a margin y padding).

### Transform

La propiedad **transform** puede operar cuatro transformaciones básicas en un elemento: **scale** (escalar), **rotate** (rotar), **skew** (inclinarse) y **translate** (trasladar o mover). Si se usan individualmente, solo el efecto de la última prevalece.

#### scale

La función **scale** recibe dos parámetros: el valor **X** para la escala horizontal y el valor **Y** para la escala vertical. Si solo un valor es provisto el mismo valor es aplicado a ambos parámetros.

Los valores entre 0 y 1 reducirán el elemento, un valor de 1 mantendrá las proporciones originales y valores mayores que 1 aumentarán las dimensiones del elemento de manera incremental.

```
transform: scale(1,-1); /*Existen también scaleX y scaleY*/
```

#### rotate

La función **rotate** rota el elemento en la dirección de las agujas de un reloj. El valor debe ser especificado en grados usando la unidad **"deg"**.

```
transform: rotate(30deg);
```

Si un valor negativo es declarado, solo cambiará la dirección en la cual el elemento es rotado.

#### skew

Esta función cambia la simetría del elemento en grados y en ambas dimensiones. Existen también skewX, skewY.

```
transform: skew(20deg);
```

```
transform: skew(20deg,10deg);
```

#### translate

La función **translate** mueve o desplaza el elemento en la pantalla a una nueva posición. Existen también translateX, translateY.

```
translate(100px,200px)
```

#### Todas las transformaciones juntas

Para hacer varias transformaciones al mismo tiempo hay que tener en cuenta que el orden es importante:

```
transform: scale(2,0.5) translate(500px,300px) skew(15deg) rotate(-45deg);
```

#### TRANSFORMACIONES DINÁMICAS

Combinación de transformaciones y pseudo clases para convertir nuestra página en una aplicación dinámica

```
#principal:hover{transform: rotate(5deg);}
```

### Transition

La propiedad **transition** fue incluida para suavizar los cambios, creando mágicamente el resto de los pasos que se encuentran implícitos en el movimiento. Solo agregando esta propiedad forzamos al navegador a tomar cartas en el asunto, crear para nosotros todos esos pasos invisibles, y generar una transición suave desde un estado al otro.

```
transition: transform 1s ease-in-out 0.5s;
```

La propiedad **transition** puede tomar hasta cuatro parámetros separados por un espacio. El primer valor es la propiedad que será considerada para hacer la transición (en nuestro ejemplo elegimos **transform**). Esto es necesario debido a que varias propiedades pueden cambiar al mismo tiempo y probablemente necesitemos crear los pasos del proceso de transición solo para una de ellas. El segundo parámetro especifica el tiempo que la transición se tomará para ir de la posición inicial a la final. El tercer parámetro puede ser cualquiera de las siguientes palabras clave: **ease**, **linear**, **ease-in**, **ease-out** o **ease-in-out**. Estas palabras clave

determinan cómo se realizará el proceso de transición basado en una curva Bézier. Cada una de ellas representa diferentes tipos de curva Bézier. El último parámetro para la propiedad **transition** es el retardo. Éste indica cuánto tiempo tardará la transición en comenzar.

Curva de Bézier, llamada así en honor al matemático francés que la creó hacia 1962. Se utilizó en aeronáutica y automoción.

Forma general:

$$B(t) = \sum_{i=0}^n \binom{n}{i} P_i (1-t)^{n-i} t^i \quad t \in [0,1] \text{ y } P_i \text{ Puntos inicial, final y puntos intermedios.}$$

## 10 ERRORES COMUNES

### 1. Uso innecesario del valor 0

El código siguiente no necesita la unidad especificada si el valor es cero.

```
padding:0px 0px 5px 0px;
```

En su lugar puede ser escrito de esta manera:

```
padding:0 0 5px 0;
```

De la misma manera es igual para otros estilos. Ej.:

```
margin:0;
```

No malgastes espacios agregando unidades tales como px, pt, em, etc, cuando el valor es cero. La única razón de hacer esto es si necesitas cambiar estos valores más tarde. Si no declarar estas unidades no tiene sentido. Los pixeles cero son iguales que los puntos cero.

Sin embargo, line-height puede no tener unidad. Por eso es válido lo siguiente:

```
line-height:1;
```

De cualquier manera, puedes utilizar una unidad en concreto como em si lo deseas.

### 2. Los colores en formato hexadecimal necesitan una almohadilla

Esto está mal:

```
color: ea6bc2;
```

Debe ser:

```
color: #ea6bc2;
```

O esto otro:

```
color: rgb(234,107,194);
```

### 3. Valores duplicados en los códigos de colores

No escribir el código de esta manera:

```
color: #ffffff;
```

```
background-color:#000000;
```

```
border:1px solid #ee66aa;
```

Los valores duplicados pueden ser omitidos. Escribiendo los códigos de esta manera:

```
color:#fff;
```

```
background-color:#000;
```

```
border:1px solid #e6a;
```

¡Por supuesto esto no debes hacerlo con códigos como este!

```
color: #fe69b2;
```

### 4. Evitar repeticiones de código innecesaria

Evita usar varias líneas cuando lo puedes conseguir con una sola. Por ejemplo, al fijar los bordes, algunas veces se debe hacer por separado, pero en casos como el siguiente no es necesario:

```
border-top:1px solid #00f;
```

```
border-right:1px solid #00f;
```

```
border-bottom:1px solid #00f;
```

```
border-left:1px solid #00f;
```

Podríamos resumirlo en una única línea esta:

```
border:1px solid #00f;
```

### 5. La duplicación es necesario con los estilos en cascada

En los estilos en cascada es aceptable repetir el mismo código para un elemento dos veces, si significa evitar la repetición mencionada en el punto arriba. Por ejemplo, digamos que tenemos un elemento donde solamente es diferente el "border" izquierda. En vez de poner cada "border" escrito usando cuatro líneas, uso sólo dos:

```
border:1px solid #00f;
```

```
border-left:1px solid #f00;
```

En este caso primero definimos todos los "borders" con el mismo color, pero más tarde para ahorrarnos dos líneas de código redefinimos el "border" izquierdo a otro color, de esta manera hemos ahorrado dos líneas de código.

El ejemplo malgastando espacio quedaría así:

```
border-top:1px solid #00f;
border-right:1px solid #00f;
border-bottom:1px solid #00f;
border-left:1px solid #f00;
```

#### **6.Los estilos inválidos no hacen nada**

Un ejemplo es suficiente para explicar este error:

```
padding:auto
```

Este estilo solo puede ser aplicado a width y height pero no a padding.

#### **7.Código específico para cada navegador**

Obviamente este tipo de código solo funcionará en el navegador al que va destinado, pero hay que pensar si es rentable puesto que solo algunos usuarios podrán apreciar esos cambios.

#### **8.Espacio perdido**

No estoy seguro del porqué, pero muchos diseñadores están empeñados en desaprovechar el espacio en su código, usando un montón de innecesarios saltos de línea. Recuerda que eso sólo lo verás tú y estás haciendo un uso excesivo de ancho de banda. También tu código será más fácil de leer puesto que tendrá menos "boquetes".

Por supuesto es aconsejable dejar un cierto espacio para mantenerlo legible, aunque a algunos les encanta condensar todo, no dejando ningún espacio.

#### **9.Especificar los colores sin usar palabras**

Definir los colores usando las palabras que lo definen no es una buena idea puesto que estaríamos confiando en el navegador para que él, interprete, que color y código debe aplicar. Las tonalidades para un mismo nombre de color cambian mucho de un navegador a otro.

Es una buena práctica especificar siempre el color por su código hexadecimal.

Ej.: utilizar "#fff" en lugar de white.

#### **10.Agrupar estilos idénticos**

Es común ver los estilos escritos una y otra vez con el mismo código, aunque el estilo sea igual.

Sería conveniente agruparlos y así optimizaríamos espacio:

```
h1, p, #footer, .intro {font-family:Arial,Helvetica,sans-serif;}
```

Las **reglas de estilo de usuario** (user stylesheet rules) las define cada persona en su navegador, a modo de configuración global, para todas las páginas que visita.

Las **reglas de estilo de autor** (author stylesheet rules) son las que definen los autores de las páginas en las hojas de estilo css

#### **Dónde se colocan las reglas de estilo de usuario**

Cada navegador se configura de una manera distinta.

En principio, se trata de un archivo de texto que debe contener el código CSS que queremos que se utilice de manera predeterminada al ver una página web. Podremos alterar el estilo de cualquier elemento, igual que lo hacemos con las hojas de estilo de autor, con la diferencia que esos estilos se aplicarán a todas las páginas que visitemos.

En el navegador Firefox se pueden definir las reglas de estilo de usuario en un archivo llamado "userContent.css" que se encuentra en la carpeta "chrome", del perfil de usuario que estemos utilizando. El directorio donde están los perfiles de Firefox depende del sistema operativo que estemos trabajando, en el caso de Windows Vista, y para mi usuario en particular, está en:

```
C:\Users\Miguel\AppData\Roaming\Mozilla\Firefox\Profiles\j6bik046.default\
```

Así que simplemente podrás crear el archivo con los estilos que quieras, que deberías colocarlo en una ruta como esta.

```
C:\Users\Miguel\AppData\Roaming\Mozilla\Firefox\Profiles\j6bik046.default\chrome\userContent.css
```

#### **Alterar la precedencia para que las reglas de usuario dominen sobre las de autor**

Como hemos dicho, en caso que una regla de estilo de usuario se defina también como regla de estilo de autor, se tiene en cuenta lo que se haya definido por el autor o diseñador de la web. Pero esto podemos cambiarlo en las reglas de estilo que queramos.

Imaginemos el caso del usuario que decidió que quería ver siempre las fuentes con un tamaño mayor, para leer mejor el contenido de las webs en su ordenador. Esta persona definió en su archivo de reglas de estilo de usuario un tamaño de letra mayor para determinados elementos de la página. Pero si un desarrollador luego ha definido un tamaño de letra distinto, el tamaño definido por el usuario se pierde y con ello quizás ahora no pueda leer la web tan cómodamente.

Podemos utilizar entonces la directriz de CSS !important, que cuando se coloca en las reglas de estilo de usuario, hace que siempre se tenga en cuenta lo que se haya definido allí.

Así pues, esta persona puede obligar a que en el cuerpo de la página siempre se utilice el tamaño de fuente que había determinado, de la siguiente manera.

```
body
{
```

```
font-size: 16pt !important;
font-family: verdana, arial;
}
```

Si vemos el anterior código de ejemplo, al atributo font-size le hemos aplicado la declaración !important, luego siempre se tendrá en cuenta antes que los estilos declarados en las reglas de estilo de autor y por tanto, aparecerán todos los textos del cuerpo de la página con tamaño de 16pt. Ahora bien, se había definido una tipografía como Verdana, Arial, pero no era !important, luego sólo se utilizará esta regla si el diseñador no llegó a especificar la familia tipográfica con sus CSS para el cuerpo de la página.

## MEDIA QUERY

### @MEDIA

Una página web se puede mostrar en diferentes dispositivos. Por tanto, podemos dejar de pensar que nuestra web se va a acceder simplemente desde un ordenador y para ayudarnos a que se vea bien en cualquier dispositivo, podemos definir estilos distintos para cada tipo de medio con la regla @media. Se dice que las páginas que se adaptan son: **Responsive**

### Sintaxis y usos de la regla @media de CSS 2

La regla @media permite especificar estilos para distintos tipos de medios en la misma hoja de estilos. En ella podemos informar sobre los tipos de medio sobre los que queremos definir estilos CSS. Por ejemplo, podríamos escribir estilos para tipos de medios como la impresión o estilos para el medio pantalla del ordenador.

Para definir un estilo para un tipo de medio, o medios, específicos podemos escribir la regla @media seguida de los tipos de medios sobre los que queremos aplicar los estilos, separados por comas.

Así definiríamos estilos que funcionarían sólo en la impresión en papel:

```
@media print
{
    table
    {
        width: 90%;
        border: 2px solid #ff000;
    }
    .miclase
    {
        display: none;
    }
}
```

Como decíamos, podemos indicar estilos CSS para varios medios a la vez:

```
@media tv, handheld
{
    body
    {
        font-size: 0.5 em;
    }
}
```

Además, si lo deseamos, podemos especificar estilos para todos los medios, con el tipo de medio "all".

```
@media all
{
    div.imprimir
    {
        display: hidden;
    }
}
```

**Nota:** Aparte de la regla @media que estamos explicando existe una manera de especificar estilos definidos en archivos externos, que sólo se apliquen para determinados tipos de medios. Esto se hace con la directiva "media" que se aplica en la etiqueta LINK para enlazar con una hoja de estilos externa, en el atributo "media".

```
<link media="print" href="css_solo_para_impresion.css" rel="stylesheet" type="text/css">
```

### Tipos de medios en CSS 2

Ahora podemos ver un listado de los tipos de medios que se definen en las especificaciones del lenguaje CSS 2.

- All: Cualquier tipo de medio.
- Braille: medio relacionado con dispositivos táctiles braille.

- Embossed: Para impresoras braille.
- Handheld: para dispositivos de bolsillo o de mano que normalmente tienen una pantalla pequeña.
- Print: medio específico para cuando se imprimen documentos en la impresora. Desde la vista previa para imprimir que tienen los navegadores, generalmente en el menú de Archivo, también podemos ver el resultado de la página para impresión que utiliza los estilos CSS del tipo de media "print".
- Projection: tipo de medio que se aplica para las presentaciones que se muestran con proyector.
- Screen: medio que se utiliza para pantallas grandes, generalmente las pantallas de los ordenadores personales.
- Speech: medio para sintetizadores de voz.
- Tty: tipo de medio que se utiliza en dispositivos que tienen un tamaño fijo de carácter, como un teletipo, terminal, consola de comandos etc. En este tipo de media no se puede usar la unidad de medida de píxeles (px) porque todo lo que se puede mostrar es a nivel de carácter.
- Tv: para cuando se accede a una web desde un dispositivo de televisión.
- Estos tipos de medios son los que eran válidos con las especificaciones de CSS 2. Es obvio que con el paso del tiempo se crearán otros tipos de medios que se irán incorporando al lenguaje. Si se utiliza un tipo de medio que no existe o que no es reconocido, el sistema simplemente lo ignora. Por ejemplo:
  - @media tv, nevera
  - {
  - p
  - {
  - background-color: #ccc;
  - }
  - }
  - Esta declaración de estilos sólo se aplica en las televisiones y en los monitores acoplados en las neveras de la cocina. Como en estos momentos no existe el tipo de medio "nevera", pues simplemente se ignora y en la práctica ese estilo sólo servirá para cuando se muestre la página en un televisor.

More examples:

If the browser window is 600px or smaller, the background color will be lightblue:

@media only screen and (max-width: 600px)

```
{  
  body {background-color: lightblue;}  
}
```

When the screen (browser window) gets smaller than 768px, each column should have a width of 100%:

```
/* For desktop: */  
.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}
```

@media only screen and (max-width: 768px)

```
{  
  /* For mobile phones: */  
  [class*="col-"]{width: 100%;}  
}
```

### Always Design for Mobile First

Mobile First means designing for mobile before designing for desktop or any other device (This will make the page display faster on smaller devices).

This means that we must make some changes in our CSS.



Instead of changing styles when the width gets *smaller* than 768px, we should change the design when the width gets *larger* than 768px. This will make our design Mobile First:

Example

```
/* For mobile phones: */
```

```
[class*="col-"]{width: 100%;}
```

```
@media only screen and (min-width: 768px) and (max-width: 900px)
```

```
@media screen and (max-device-width:640px), screen and (max-width:640px)
```

```
@media (min-resolution: 1dpi)
```

```
@media print
```

```
@media screen and (orientation: landscape)
```

```
{  
/* For desktop: */  
.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}  
}
```