

## **TEMA 2**

# CSS3 Avanzado

---

<b>Capítulo 1. Técnicas imprescindibles</b>	<b>8</b>
1.1. Propiedades shorthand	8
1.2. Limpiar floats	10
1.3. Elementos de la misma altura	12
1.4. Sombras	13
1.5. Transparencias	14
1.6. Sustitución de texto	16
1.6.1 La directiva @font-face	16
1.6.2 Optimiza @font-face con google fonts	17
1.6.3 Qué son los Icon Fonts	17
1.7. Esquinas redondeadas	19
1.8 Rollovers y sprites	20
1.8.2 CSS Sprites Generator de Toptal	27
1.9. Texto	28
1.9.1. Tamaño de letra	28
1.9.2. Efectos gráficos	29
1.9.2.1. Texto con sombra	29
1.9.2.2. Efecto brillo	29
1.9.2.3 Textos en 3D	30
1.9.2.4. Texto con relleno gradiente o degradado e imágenes	31
<b>Capítulo 2. Buenas prácticas</b>	<b>33</b>
2.1. Inicializar los estilos	33
2.2. Comprobar el diseño en varios navegadores	36
2.2.1. Principales navegadores	36
2.2.2. Probar el diseño en todos los navegadores	36
2.3. Mejora progresiva	37
2.4. Depuración	39
Chrome Developer Tools	39
Herramienta del W3C para validación de hojas de estilo	39
2.5. Hojas de estilos	40
2.5.1. Llaves	40
2.5.2. Tabulaciones	41
2.5.3. Propiedades	42
2.5.4. Selectores	43
2.5.5. Organización	43

---

2.5.6. Comentarios	44
2.6 Consistencia y Legibilidad	45
2.7 Mantenibilidad	45
2.8 Herramientas y metodologías para optimizar CSS.	46
2.9. Rendimiento	49
<b>Capítulo 3. Selectores</b>	<b>53</b>
3.1. Selector de hijos	53
3.2.1 Selector adyacente	54
3.2.2 Selector general de elementos hermanos,	55
3.3. Selector de atributos	56
3.4. Pseudo-clases	58
3.4.1. La pseudo-clase :first-child	58
3.4.2 La pseudo-clase : nth-child	59
3.4.3. Las pseudo-clases :link y :visited	60
3.4.4. Las pseudo-clases :hover, :active y :focus	60
3.5. Pseudo-elementos	64
3.5.1. El pseudo-elemento :first-line	64
3.5.2. El pseudo-elemento :first-letter	65
3.5.3. Los pseudo-elementos :before y :after	65
3.5.4 Pseudo-elemento ::selecion	65
3.5.5 El selector :target	65
<b>Capítulo 4. Propiedades avanzadas</b>	<b>67</b>
4.1. Propiedad white-space	67
4.1.2 Cambio de línea (word-break)	70
4.2. Propiedad display	71
4.2.2 Columnas múltiples	73
4.3. Propiedad outline	76
outline-offset	78
<b>Capítulo 5. Técnicas avanzadas</b>	<b>81</b>
5.1. Imágenes embebidas	81
5.2. Mapas de imagen	84
5.3 La propiedad shape-outside	89
5.4 Control del Scroll	92
5.4.1 Parallax	92
5.4.2 Scroll Snap	93
5.4.3 scroll-behavior	95

CSS avanzado	Capítulo 1. Técnicas imprescindibles
5.5 Variables CSS	95
5.6 La función calc()	96
<b>Capítulo 6. Transformación y Animación</b>	<b>98</b>
6.1.Transform.	98
6.1.2 Transform-origin.	100
6.2 Propiedades 3D.	101
6.2.1 La creación de un punto de vista (o perspectiva).	101
6.3.2 Transform-style	103
6.3.3 Funciones de transformación 3D	104
6.4. Animaciones	105
6.4.1 Módulo Animations	105
6.4.1.1 Fotogramas claves de las animaciones	105
6.4.1.2 @keyframes	106
6.4.1.3 Aplicando la animación a un elemento	107
6.4.1.4 animation-timing-function	108
6.4.2 Módulo Transition	108
6.4.2.1 Propiedades "animables"	109
6.4.2.2 Propiedad transition-property	109
6.4.2.3 Propiedad transition-duration	109
6.4.2.4 Propiedad transition-timing-function	110
6.4.2.5 Propiedad transition-delay	110
6.4.2.6 Propiedad transition	111
6.4.2.7 Listas de valores	111
6.4.2.8 Finalización de una transición	112
6.4.2.9 Transiciones y JavaScript	112
<b>Capítulo 7 Degradados</b>	<b>114</b>
7.1 Degradados lineares:	114
7.2 Degradados circulares:	115
7.3 Explicación detallada de los degradados lineales.	115
7.4. Atributo repeating-linear-gradient, degradados lineales con múltiples repeticiones del mismo gradiente de color.	118
7.5. Degradados en un gradiente de color de forma radial, círculos y elipses.	120
7.6. degradados radiales, con repeticiones, fondos con gradientes de color.	123
7.7 Generadores de gradientes CSS.	124
7.8 Patrones creados con CSS.	125
Patternizer (generador de patrones)	126

<b>Capítulo 8 Flexbox</b>	<b>129</b>
8.1 Introducción	129
8.2. La propiedad flex-direction	131
8.3 La propiedad flex-wrap	133
8.4 La propiedad align-items	136
8.5 La propiedad justify-content	140
8.6 La propiedad align-content	142
8.7 La propiedad align-self	147
align-items: flex-start	147
align-items: flex-end	148
align-items: center	149
align-items: stretch	149
align-items: baseline	150
8.8 La propiedad flex	150
La propiedad flex-grow	151
La propiedad flex-shrink	153
La propiedad flex-basis	154
8.9 La propiedad order	156
8.10 Modernizr	158
Descargar modernizr	158
Cómo funciona	159
¿Qué quiere decir esto?	160
<b>Capítulo 9. GRID LAYOUT</b>	<b>161</b>
Que es el grid layout	161
Utilizar prefijos	161
9.1 Grid layout – un ejemplo básico	161
Propiedades del contenedor grid	163
9.2 Las propiedades grid-template-columns y grid-template-rows.	164
9.4 La propiedad grid-template	167
none	167
Enlace	169
Enlace	169
Enlace	170
9.6 Justificar y alinear dentro del contenedor grid	170
Las propiedades justify-items & align-items ( dos propiedades del contenedor grid )	170
9.8 Propiedades de los ítems grid	173

---

9.10 Las propiedades grid-column y grid-row	174
La propiedad grid-area	176
9.11 Las propiedades justify-self y align-self	177
La propiedad order	179
9.9 Algunas palabras clave	180
El método repeat()	181
Las palabras clave max-content y min-content	183
Las palabras clave start y end	184
Líneas y carriles	185
<b>Capítulo 10 Responsive WEB</b>	<b>187</b>
1.1 Ventajas	187
1.2 Los ingredientes	188
10.2 Estructura de rejilla flexible	188
10.2.1 VIEWPORT	189
10.3 Crear una rejilla flexible	192
10.3.1 Fuentes flexibles	193
10.3.2 Contenedores flexibles	194
10.3.3 Márgenes flexibles	196
10.3.4 Rellenos flexibles	197
10.3.5 Imágenes flexibles	197
10.4 Imágenes flexibles	197
10.4.1 Imágenes fluidas	198
10.4.2 Optimización de Imágenes en el Diseño Responsivo.	199
10.4.3 Optimización. Imágenes SVG	200
10.4.4 Optimización. Imágenes PNG/GIF/JPEG/WEBP	200
10.4.5 Diseño “ART-DIRECTOR”	201
10.5.1 Esconder columnas	202
10.5.2 Convertir Listas a filas	203
10.5.3 Scroll controlado	204
10.6 Media Queries	205
10.6.1 Sintaxis	206
10.6.1.1 Operadores lógicos o logical operators	207
and	207
comma-separated lists	207
not	208
only	208

---

10.6.1.2 Características de los medios	208
10.7 Patrones Responsive	209
10.7.1 La importancia del contexto	210
10.7.4 Patrones Responsivos	212
Column Drop	213
Mostly Fluid	213
Layout Shifter	214
Off Canvas	214
<b>Capítulo 11 Preprocesador SASS</b>	<b>215</b>
Introducción	215
Capítulo 12. Frameworks	216

# Capítulo 1. Técnicas imprescindibles

El estándar CSS3 incluye un gran número de propiedades de todo tipo para diseñar el aspecto de las páginas HTML.

En las próximas secciones se muestran las siguientes técnicas imprescindibles:

- Propiedades *shorthand* para crear hojas de estilos concisas.
- *Limpiar floats*, para trabajar correctamente con los elementos posicionados de forma flotante.
- Cómo crear elementos de la misma altura, imprescindible para el *layout* o estructura de las páginas.
- Sombras, transparencias y esquinas redondeadas.
- Rollovers y *sprites CSS* para mejorar el tiempo de respuesta de las páginas.
- Técnicas para trabajar con el texto y la tipografía...

## 1.1. Propiedades shorthand

Algunas propiedades del estándar CSS son especiales, ya que permiten establecer simultáneamente el valor de varias propiedades diferentes. Este tipo de propiedades se denominan "*propiedades shorthand*" y todos los diseñadores web profesionales las utilizan.

font	Tipografía
Valores	( ( <font-style>    <font-variant>    <font-weight> )? <font-size> ( / <line-height> )? <font-family> )   caption   icon   menu   message-box   small-caption   status-bar   inherit
Se aplica a	Todos los elementos
Valor inicial	-
Descripción	Permite indicar de forma directa todas las propiedades de la tipografía de un texto

La gran ventaja de las *propiedades shorthand* es que permiten crear hojas de estilos mucho más concisas y por tanto, mucho más fáciles de leer. A continuación se incluye a modo de referencia la definición formal de las seis *propiedades shorthand* disponibles en el estándar CSS .

<b>margin</b>	Margen
<b>Valores</b>	( <medida>   <porcentaje>   auto ) {1, 4}   inherit
<b>Se aplica a</b>	Todos los elementos salvo algunos casos especiales de elementos mostrados como tablas
<b>Valor inicial</b>	-
<b>Descripción</b>	Establece de forma directa todos los márgenes de un elemento
<b>padding</b>	Relleno
<b>Valores</b>	( <medida>   <porcentaje> ){1, 4}   inherit
<b>Se aplica a</b>	Todos los elementos excepto algunos elementos de tablas como grupos de cabeceras y grupos de pies de tabla
<b>Valor inicial</b>	-
<b>Descripción</b>	Establece de forma directa todos los rellenos de los elementos
<b>border</b>	Estilo completo de todos los bordes
<b>Valores</b>	( <border-width>    <border-color>    <border-style> )   inherit
<b>Se aplica a</b>	Todos los elementos
<b>Valor inicial</b>	-
<b>Descripción</b>	Establece el estilo completo de todos los bordes de los elementos
<b>background</b>	Fondo de un elemento
<b>Valores</b>	( <background-color>    <background-image>    <background-repeat>    <background-attachment>    <background-position> )   inherit
<b>Se aplica a</b>	Todos los elementos
<b>Valor inicial</b>	-
<b>Descripción</b>	Establece todas las propiedades del fondo de un elemento
<b>list-style</b>	Estilo de una lista
<b>Valores</b>	( <list-style-type>    <list-style-position>    <list-style-image> )   inherit
<b>Se aplica a</b>	Elementos de una lista
<b>Valor inicial</b>	-
<b>Descripción</b>	Propiedad que permite establecer de forma simultánea todas las opciones de una lista

```
p {  
    font-style: normal;  
    font-variant: small-caps;  
    font-weight: bold;  
    font-size: 1.5em;  
    line-height: 1.5;  
    font-family: Arial, sans-serif;  
}  
  
div {  
    margin-top: 5px;  
    margin-right: 10px;  
    margin-bottom: 5px;  
    margin-left: 10px;  
    padding-top: 3px;  
    padding-right: 5px;  
    padding-bottom: 10px;  
    padding-left: 7px;  
}  
  
h1 {  
    background-color: #FFFFFF;  
    background-image: url("imagenes/icono.png");  
    background-repeat: no-repeat;  
    background-position: 10px 5px;  
}
```

Utilizando las *propiedades shorthand* es posible convertir las 24 líneas que ocupa la hoja de estilos anterior en sólo 10 líneas, manteniendo los mismos estilos:

```
p { font: normal small-caps bold 1.5em/1.5 Arial, sans-serif; }  
  
div {  
    margin: 5px 10px;  
    padding: 3px 5px 10px 7px; }  
  
h1 {  
    background: #FFF url("imagenes/icono.png") no-repeat 10px 5px; }
```

## 1.2. Limpiar floats

La principal característica de los elementos posicionados de forma flotante mediante la propiedad **float** es que desaparecen del flujo normal del documento. De esta forma, es posible que algunos o todos los elementos flotantes se salgan de su elemento contenedor.

La siguiente imagen muestra un elemento contenedor que encierra a dos elementos de texto. Como los elementos interiores están posicionados de forma flotante y el elemento contenedor no dispone de más contenidos, el resultado es el siguiente:

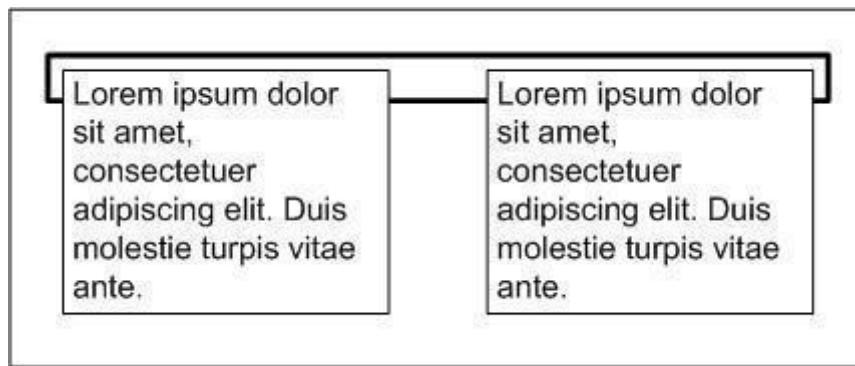


Figura 1.1. Los elementos posicionados de forma flotante se salen de su elemento contenedor

El código HTML y CSS del ejemplo anterior se muestra a continuación:

```
<div id="contenedor">
    <div id="izquierda">
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis
        molestie turpis vitae ante.
    </div>
    <div id="derecha">
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla
        bibendum mi non lacus.
    </div>
</div>
#contenedor {
    border: thick solid #000;
}

#izquierda {
    float: left; width: 40%;
}

#derecha {
    float: right;
    width: 40%;
}

</div>
```

La técnica de corregir los problemas ocasionados por los elementos posicionados de forma flotante se suele denominar "*limpiar los float*".

Existe una solución para *limpiar los float* que no obliga a añadir nuevos elementos HTML y que además es elegante y muy sencilla. La solución consiste en utilizar la propiedad `overflow` de CSS sobre el elemento contenedor.

Si se modifica el código CSS anterior y se incluye la siguiente regla:

```
#contenedor {
    border: thick solid #000;
    overflow: hidden;
}
```

Ahora, el contenedor encierra correctamente a los dos elementos `<div>` interiores y no es necesario añadir ningún elemento adicional en el código HTML. Además del valor `hidden`, también es posible utilizar el valor `auto` obteniendo el mismo resultado.

### 1.3. Elementos de la misma altura

Existe una solución para asegurar que dos elementos tengan la misma altura sin utilizar flex-box. Se basa en el uso de la propiedad `display` de CSS.

En primer lugar, es necesario añadir un elemento adicional (`<div id="contenidos">`) en el código HTML de la página:

```
<div id="contenedor">
    <div id="contenidos">
        <div id="columna1"></div>
        <div id="columna2"></div>
        <div id="columna3"></div>
    </div>
</div>
```

A continuación, se utiliza la propiedad `display` de CSS para mostrar los elementos `<div>` anteriores como si fueran celdas de una tabla de datos:

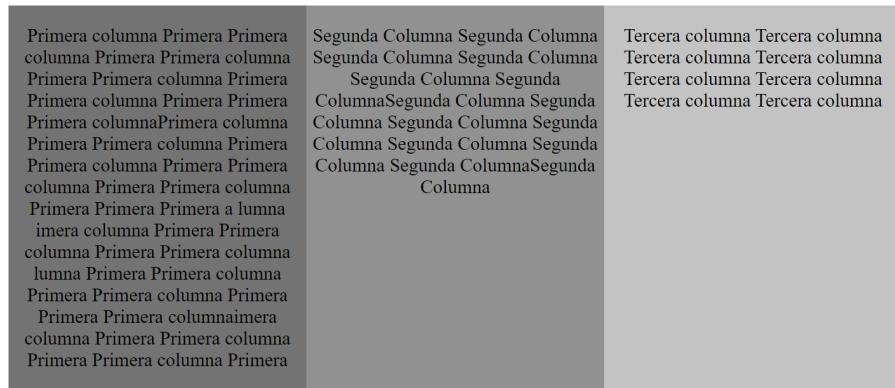
```
#contenedor {
    display: table;
}

#contenidos {
    display: table-row;
}

#columna1, #columna2, #columna3 {
    display: table-cell;
}
```

Gracias a la propiedad `display` de CSS, cualquier elemento se puede comportar como una tabla, una fila de tabla o una celda de tabla, independientemente del tipo de elemento que se trate.

De esta forma, los elementos `<div>` que forman las columnas de la página en realidad se comportan como celdas de tabla, lo que permite que el navegador las muestre con la misma altura.



## 1.4. Sombras

```
.elemento {  
    box-shadow: 2px 2px 5px 999;  
}
```

La sintaxis completa de la propiedad `box-shadow` es muy compleja y se define en el [borrador de trabajo del módulo de fondos y bordes de CSS3](#) (<http://dev.w3.org/csswg/css3-background/#the-box-shadow>) .

A continuación, se muestra su sintaxis simplificada habitual:

`box-shadow: <medida> <medida> <medida>? <medida>? <color>`

- La primera medida es obligatoria e indica el desplazamiento horizontal de la sombra. Si el valor es positivo, la sombra se desplaza hacia la derecha y si es negativo, se desplaza hacia la izquierda.
  - La segunda medida también es obligatoria e indica el desplazamiento vertical de la sombra. Si el valor es positivo, la sombra se desplaza hacia abajo y si es negativo, se desplaza hacia arriba.
  - La tercera medida es opcional e indica el radio utilizado para difuminar la sombra. Cuanto más grande sea su valor, más borrosa aparece la sombra. Si se utiliza el valor 0, la sombra se muestra como un color sólido.
  - La cuarta medida también es opcional e indica el radio con el que se expande la sombra. Si se establece un valor positivo, la sombra se expande en todas direcciones. Si se utiliza un valor negativo, la sombra se comprime.

La siguiente regla CSS muestra una sombra en los navegadores Firefox y Safari:

```
.elemento {
    -webkit-box-shadow: 2px 2px 5px #999;
    -moz-box-shadow: 2px 2px 5px #999;
}
```

Si incluimos como último parámetro **inset**, las propiedades de las sombras se aplicarán dentro del elemento del “box-model”, eso quiere decir que los parámetros partirán del border hasta el centro:

```
box-shadow: 20px 20px 4px 5px #555 inset;
```

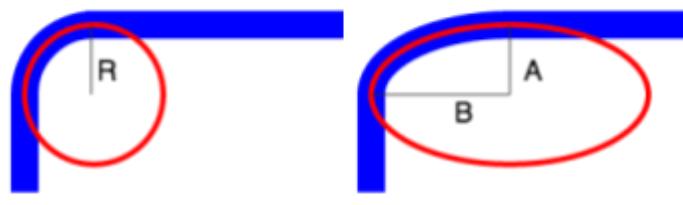


## 1.5. Esquinas redondeadas

CSS 3 incluye varias propiedades para definir bordes redondeados. La propiedad **border-radius** establece la misma curvatura en todas las esquinas y también se definen las propiedades **border-top-right-radius**, **border-bottom-right-radius**, **border-bottom-left-radius**, **border-top-left-radius** para establecer la curvatura de cada esquina.

En cada propiedad se debe indicar obligatoriamente una medida y se puede indicar opcionalmente una segunda medida. Cuando se indica una sola medida, la esquina es circular y su radio es igual a la medida indicada. Cuando se indican dos medidas, la esquina es elíptica, siendo la primera medida el radio horizontal de la elipse y la segunda medida su radio vertical.

```
div {
    -webkit-border-radius: 5px 10px; /* Safari */
    -moz-border-radius: 5px 10px; /* Firefox */
}
```



`border-radius: R;`

`border-radius: A/B;`

Se podrían combinar las propiedades como en el ejemplo siguiente:



```
#combinado{  
    height:6em;  
    width:6em;  
    background-color:#8AC007;  
    border-top-left-radius: 3em 1em;  
    border-top-right-radius: 1em 3em;  
    border-bottom-right-radius: 3em 1em;  
    border-bottom-left-radius:1em 3em;}
```

### 1.5.2 Neumorfismo

Neumorfismo es un estilo de diseño que combina elementos minimalistas. Se caracteriza por el uso de sombras suaves y colores pastel para crear un aspecto tridimensional y atractivo.



```
border-radius: 50px;  
background: linear-gradient(145deg, #cacaca, #f0f0f0);  
box-shadow: 20px 20px 60px #bebebe,  
-20px -20px 60px #ffffff;
```



Generador: <https://neumorphism.io/>

## 1.6. Transparencias

CSS3 incluye una propiedad llamada **opacity**, definida en [el módulo de colores de CSS 3](#) (<http://www.w3.org/TR/css3-color/>) y que permite incluir transparencias en el diseño de la página.

El valor de la propiedad **opacity** se establece mediante un número decimal comprendido entre **0.0** y **1.0**. La interpretación del valor numérico es tal que el valor **0.0** es la máxima transparencia (el elemento es invisible) y el valor **1.0** se corresponde con la máxima opacidad (el elemento es completamente visible). De esta forma, el valor **0.5** corresponde a un elemento semitransparente y así sucesivamente.

En el siguiente ejemplo, se establece la propiedad **opacity** con un valor de **0.5** para conseguir una transparencia del 50% sobre dos de los elementos **<div>**:

```
#segundo, #tercero {  
    opacity: 0.5;  
}  
  
#primero {  
    background-color: blue;  
}  
  
#segundo {  
    background-color: red;  
}  
  
#tercero {  
    background-color: green;  
}
```

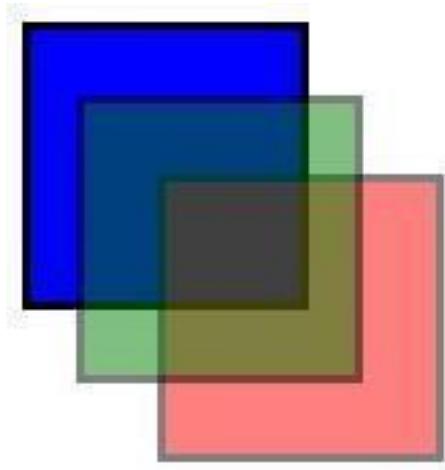


Figura 1.5. Creando elementos semitransparentes con la propiedad opacity

## RGBA

En CSS3 podemos utilizar `rgba` para aplicar color. El último parámetro **a**, representa el valor alpha ( como en *transparencia alpha* ). De hecho el parámetro alpha representa la opacidad y es un número decimal de 0.0 ( *completamente transparente* ) a 1.0 ( *totalmente opaco* ).

La ventaja que ofrece `rgba` con respecto a `opacity` es que los elementos que estén contenidos no tendrán transparencias a diferencia de `opacity`.

```
background-color: rgba(83,83,83,0.8);
```

## 1.7. Sustitución de texto

La limitación más frustrante para los diseñadores web que cuidan especialmente la tipografía de sus páginas es la imposibilidad de utilizar cualquier tipo de letra para mostrar los contenidos de texto. Como se sabe, las fuentes que utiliza una página deben estar instaladas en el ordenador o dispositivo del usuario para que el navegador pueda mostrarlas.

Por lo tanto, si el diseñador utiliza en la página una fuente especial poco común entre los usuarios normales, el navegador no encuentra esa fuente y la sustituye por una fuente por defecto.

El resultado es que la página que ve el usuario y la que ve el diseñador se diferencian completamente en su tipografía.

Con la directiva **@font-face** podemos incluir fuentes que no estén en nuestro sistema operativo.

### 1.7.1 La directiva @font-face

[Web Fonts](http://www.w3.org/TR/css3webfonts/) (<http://www.w3.org/TR/css3webfonts/>).

La directiva **@font-face** permite describir las fuentes que utiliza una página web. Como parte de la descripción se puede indicar la dirección o URL desde la que el navegador se puede descargar la fuente utilizada si el usuario no dispone de ella. A continuación se muestra un ejemplo de uso de la directiva **@font-face**:

```
@font-face {
    font-family: "Fuente muy rara";
    src: url("http://www.ejemplo.com/fuentes/fuente_muy_rara.ttf");
}

h1 { font-family: "Fuente muy rara", sans-serif; }
```

La directiva **@font-face** también permite definir otras propiedades de la fuente, como su formato, grosor y estilo. A continuación, se muestran otros ejemplos:

```
@font-face {
    font-family: Fontinsans;
    src: url("fonts/Fontin_Sans_SC_45b.otf") format("opentype");
    fontweight: bold; font-style: italic;
}

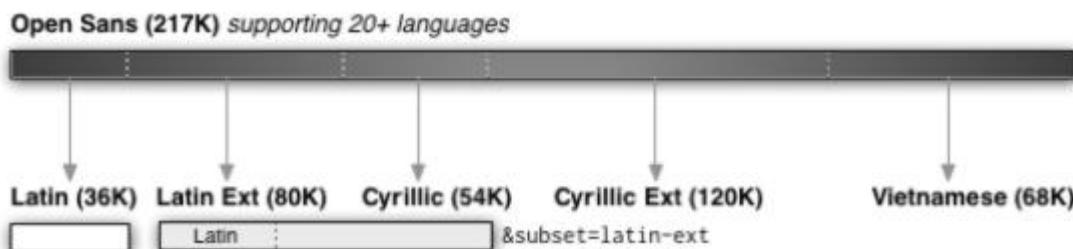
@font-face {
    font-family: Tagesschrift;
    src: url("fonts/YanoneTagesschrift.ttf") format("truetype");
}
```

### 1.7.2 Optimiza @font-face con google fonts

Si se utiliza este servicio de google, puedes ir un paso más allá a la hora de optimizar la tipografía elegida.

Cada archivo incluye los juegos de caracteres necesarios para dar soporte a un montón de idiomas (character subset): latín, cirílico, japonés... en sus versiones "normal" y extendida.

La popular *Open Sans* da soporte a más de 20 idiomas con un peso de 217kb.



Para lograr aligerar el peso del archivo y por lo tanto el tiempo de carga, sólo necesitas indicar el juego de caracteres que deseas cargar.

```
<!-- El juego de caracteres latín extendido -->
<link href="http://fonts.googleapis.com/css?family=Open+Sans&subset=latin-ext"
      rel="stylesheet" />
```

Lo único que hay que hacer en el enlace facilitado por Google, es añadir a continuación del nombre de la fuente la cadena de texto `&subset=latin` o el juego que necesites.

### 1.7.3 Qué son los Icon Fonts

Como sugiere el nombre los Icon Fonts son iconos utilizados mediante fuentes tipográficas, ya sean letras y números o caracteres Unicode. Un ejemplo en este caso puede ser muy ilustrativo. Primero tenemos que definir la familia de fuentes a utilizar, los Icon Fonts en este caso. Para esto utilizamos la declaración `@font-face`.

A continuación, designamos las clases que utilizarán esta fuente. En este caso serán todas las clases que empiezan por "icon".

```
[class^="icon"] {font-family: 'miFuente';}
```

Hay varios métodos para utilizar los icon-fonts. Los podemos utilizar directamente en el HTML:

```
<span class="icon-lapiz">#xe905;</span>
```

O en el CSS como el **content** del pseudo-elemento **:before**.

```
.icon-lapiz:before{content: "\e905;"}
```

O todavía mejor: utilizando el valor de un atributo **data-icon**

```
<span class="icon" data-icon=""></span>  
  
.icon:before{ content:attr(data-icon);}
```



### ¿Por qué utilizar Icon Fonts?

Utilizar iconos de fuentes tipográficas tiene muchas ventajas:

- Podemos redimensionar fácilmente, exactamente cómo cambiamos el tamaño de las letras: basta con cambiar el tamaño de fuente ( font-size: 24px; ).
- Podemos cambiar fácilmente el color de los iconos utilizando la propiedad color ( color:#abcdef; ).
- Podemos aplicar sombras con text-shadow, y generalmente, podemos hacer casi todo lo que podemos hacer con fuentes.

Probablemente las más conocidas Font Icons son las de Bootstrap que podemos descargar aquí:



<https://fontawesome.com/>

## 1.8 Rollovers y sprites

Hasta el 80% de la mejora en el rendimiento de la descarga de páginas web depende de la parte del cliente. Generar dinámicamente el código HTML de la página y servirla ocupa el 20%

del tiempo total de descarga de la página. El 80% del tiempo restante los navegadores descargan las imágenes, archivos JavaScript, hojas de estilos y cualquier otro tipo de archivo enlazado.

Además, en la mayoría de páginas web *normales*, la mayor parte de ese 80% del tiempo se dedica a la descarga de las imágenes. Por tanto, aunque los mayores esfuerzos siempre se centran en reducir el tiempo de generación dinámica de las páginas, se consigue más y con menos esfuerzo mejorando la descarga de las imágenes.

La idea para mejorar el rendimiento de una página que descarga por ejemplo 15 imágenes consiste en crear una única imagen grande que incluya las 15 imágenes individuales y utilizar las propiedades CSS de las imágenes de fondo para mostrar cada imagen. Esta técnica se presentó en el artículo *CSS Sprites: Image Slicing's Kiss of Death* (<http://www.alistapart.com/articles/sprites>) y desde entonces se conoce con el nombre de *sprites CSS*.

El siguiente ejemplo explica el uso de los *sprites CSS* en un sitio web que muestra la previsión meteorológica de varias localidades utilizando iconos:

## Previsiones meteorológicas



Localidad 1: soleado, máx: 35°, min: 23°



Localidad 2: nublado, máx: 25°, min: 13°



Localidad 3: muy nublado, máx: 22°, min: 10°



Localidad 4: tormentas, máx: 23°, min: 11°

Figura 1.8. Aspecto de la previsión meteorológica mostrada con iconos

La solución tradicional para crear la página anterior consiste en utilizar cuatro elementos <img> en el código HTML y disponer de cuatro imágenes correspondientes a los cuatro iconos:

```
<h3>Previsiones meteorológicas</h3>

<p id="localidad1">
    
    Localidad 1: soleado, máx: 35°, min: 23°
</p>

<p id="localidad2">
    
    Localidad 2: nublado, máx: 25°, min: 13°
</p>

<p id="localidad3">
```

```
  
Localidad 3: muy nublado, máx:22º, mín: 10º  
</p>  
  
<p id="localidad4">  
  
  
  
Localidad 4: tormentas, máx: 23º, mín: 11º  
  
</p>
```

Aunque es una solución sencilla y que funciona muy bien, se trata de una solución completamente ineficiente. El navegador debe descargar cuatro imágenes diferentes para mostrar la página, por lo que debe realizar cuatro peticiones al servidor.

Después del tamaño de los archivos descargados, el número de peticiones realizadas al servidor es el factor que más penaliza el rendimiento en la descarga de páginas web. Utilizando la técnica de los *sprites CSS* se va a rehacer el ejemplo anterior para conseguir el mismo efecto con una sola imagen y por tanto, una sola petición al servidor.

El primer paso consiste en crear una imagen grande que incluya las cuatro imágenes individuales. Como los iconos son cuadrados de tamaño 32px, se crea una imagen de 32px x 128px:

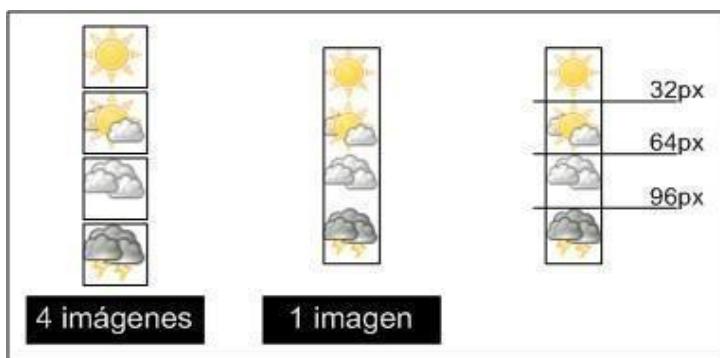


Figura 1.9. Creando un sprite de CSS a partir de varias imágenes individuales

Para facilitar el uso de esta técnica, todas las imágenes individuales ocupan el mismo sitio dentro de la imagen grande. De esta forma, los cálculos necesarios para desplazar la imagen de fondo se simplifican al máximo.

El siguiente paso consiste en simplificar el código HTML:

```
<h3>Previsiones meteorológicas</h3>  
<p id="localidad1">Localidad 1: soleado, máx: 35º, mín: 23º</p>  
<p id="localidad2">Localidad 2: nublado, máx: 25º, mín: 13º</p>  
<p id="localidad3">Localidad 3: muy nublado, máx: 22º, mín: 10º</p>  
<p id="localidad4">Localidad 4: tormentas, máx: 23º, mín: 11º</p>
```

La clave de la técnica de los *sprites* CSS consiste en mostrar las imágenes mediante la propiedad `background-image`. Para mostrar cada vez una imagen diferente, se utiliza la propiedad `background-position` que desplaza la imagen de fondo teniendo en cuenta la posición de cada imagen individual dentro de la imagen grande:

```
#localidad1, #localidad2, #localidad3, #localidad4{
    padding-left: 38px;
    height: 32px;
    line-height: 32px;
    background-image: url("imagenes/sprite.png");
    background-repeat: no-repeat;
}

#localidad1 {
    background-position: 0 0;
}
#localidad2 {
    background-position: 0 -32px;
}
#localidad3 {
    background-position: 0 -64px;
}
#localidad4 {
    background-position: 0 -96px;
}
```

La siguiente imagen muestra lo que sucede con el segundo párrafo:

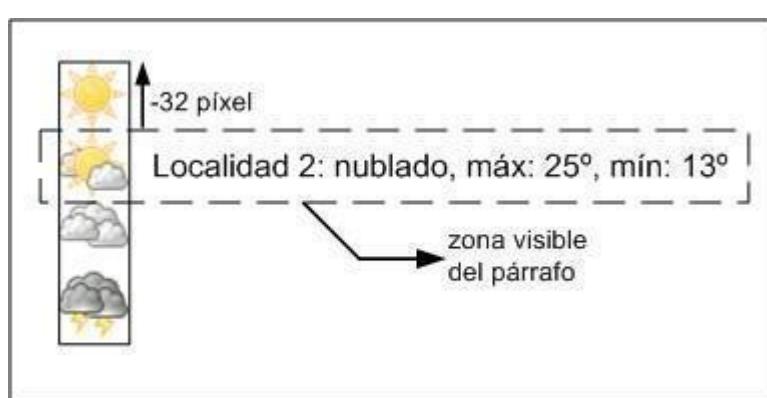


Figura 1.10. Funcionamiento de la técnica de los sprites CSS

El párrafo tiene establecida una altura de 32px, idéntica al tamaño de los iconos. Después de añadir un `padding-left` al párrafo, se añade la imagen de fondo mediante `background-image`. Para cambiar de una imagen a otra, sólo es necesario desplazar de forma ascendente o descendente la imagen grande.

Si se quiere mostrar la segunda imagen, se desplaza de forma ascendente la imagen grande. Para desplazarla en ese sentido, se utilizan valores negativos en el valor indicado en la propiedad `background-position`. Por último, como la imagen grande ha sido especialmente

preparada, se sabe que el desplazamiento necesario son 32 píxel, por lo que la regla CSS de este segundo elemento resulta en:

```
#localidad2 {
    padding-left: 38px;
    height: 32px;
    line-height: 32px;
    background-image:url("imagenes/sprite.png");
    background-repeat: no-repeat;
    background-position: 0 -32px;
}
```

La solución original utilizaba cuatro imágenes y realizaba cuatro peticiones al servidor. La solución basada en *sprites CSS* sólo realiza una conexión para descargar una sola imagen. Además, los iconos que se han utilizado en este ejemplo ocupan 7.441 bytes cuando se suman los tamaños de los cuatro iconos por separado. Por su parte, la imagen grande que contiene los cuatro iconos sólo ocupa 2.238 bytes.

Los *sprites* que incluyen todas sus imágenes verticalmente son los más fáciles de manejar. Si en el ejemplo anterior se emplea un *sprite* con las imágenes dispuestas también horizontalmente:

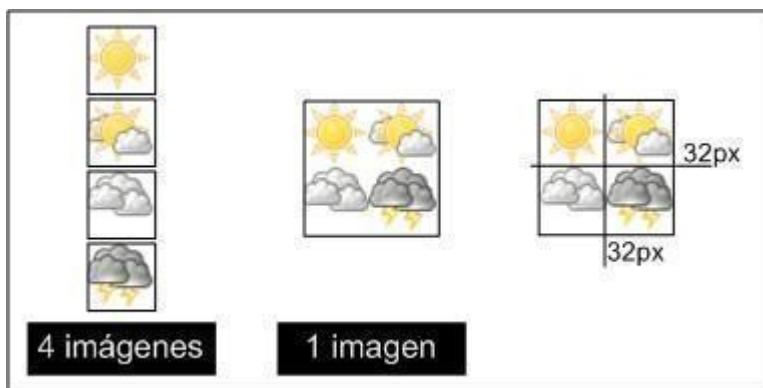


Figura 1.11. Sprite complejo que dispone las imágenes de forma vertical y horizontal

Aparentemente, utilizar este nuevo *sprite* sólo implica que la imagen de fondo se debe desplazar también horizontalmente:

```
#localidad1, #localidad2, #localidad3, #localidad4 {

    padding-left: 38px;
    height: 32px;
    line-height: 32px;
    background-image:url("imagenes/sprite.png");
    background-repeat: no-repeat;
}

#localidad1 {
    backgroundposition: 0 0;}
```

```
#localidad2 {
    backgroundposition: -32px 0; }

#localidad3 {
    backgroundposition: 0 -32px; }

#localidad4 {
    backgroundposition: -32px -32px; }
```

El problema del *sprite* anterior es que cuando una imagen tiene a su derecha o a su izquierda otras imágenes, estas también se ven:

## Previsiones meteorológicas

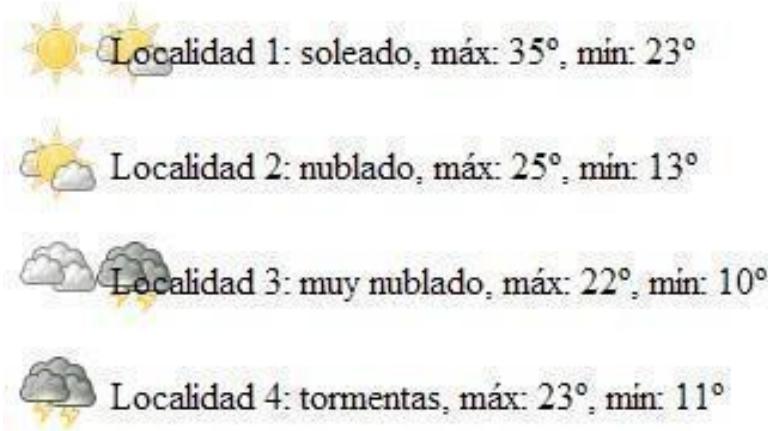


Figura 1.12. Errores producidos por utilizar un sprite complejo

La solución de este problema es sencilla, aunque requiere algún cambio en el código HTML:

Primero pondremos un SPAN en el párrafo:

```
<li><span id="localidad1"></span> Localidad 1: soleado, máx: 35°, mín: 23° </li>
```

En la declaración de css pondremos las siguientes propiedades:

```
#localidad1, #localidad2, #localidad3, #localidad4 {

    display: inline-block;
    height: 65px;
    vertical-align: middle;
    padding-left: 85px;
    padding-top: 8px;
    background-image: url("imagenes/el_tiempo.png");
    background-repeat: no-repeat;

}
```

Utilizar *sprites CSS* es una de las técnicas más eficaces para mejorar los tiempos de descarga de las páginas web complejas. La siguiente imagen muestra un *sprite* complejo que incluye 241 iconos y sólo ocupa 42 KB:



Figura 1.13. Sprite complejo que incluye 210 iconos en una sola imagen

La mayoría de sitios web profesionales utilizan *sprites* para mostrar sus imágenes de adorno. La siguiente imagen muestra el *sprite* del sitio web YouTube:

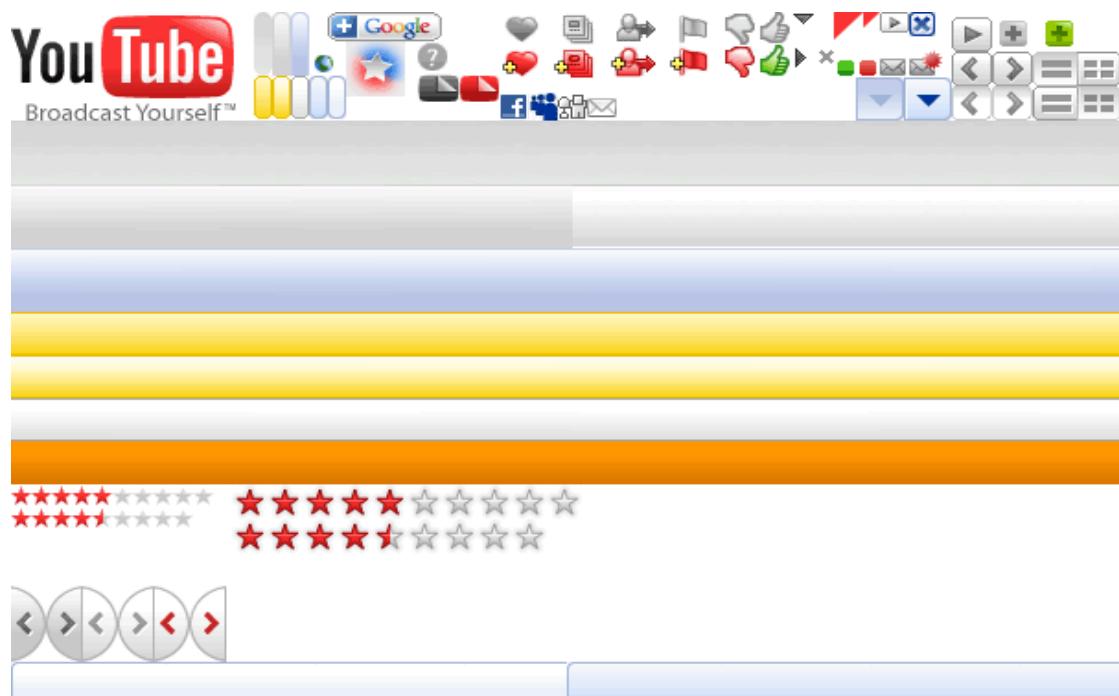
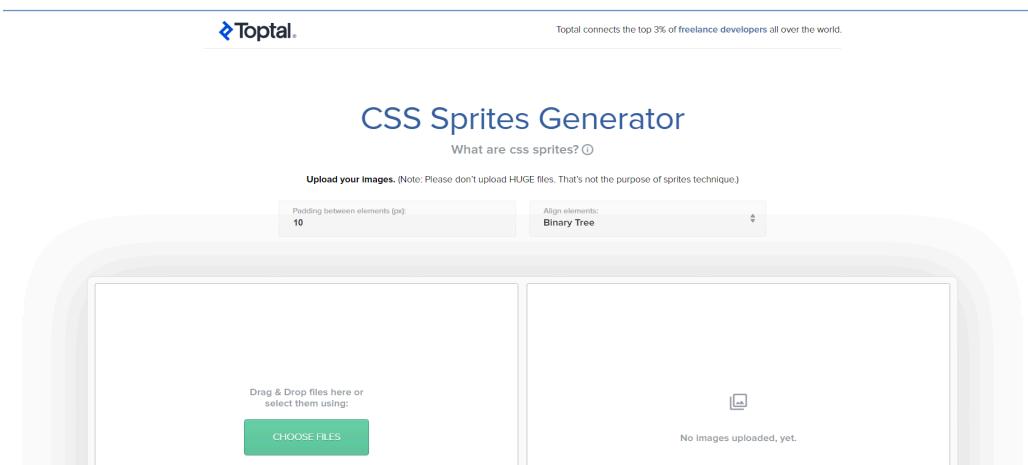


Figura 1.14. Sprite utilizado por el sitio web de YouTube

Los principales inconvenientes de los *sprites CSS* son la poca flexibilidad que ofrece (añadir o modificar una imagen individual no es inmediato) y el esfuerzo necesario para crear el *sprite*.

Afortunadamente, existen aplicaciones web como [CSS Sprite Generator](http://spritegen.website-performance.org/) (<http://spritegen.website-performance.org/>) que generan el *sprite* a partir de un archivo comprimido en formato ZIP con todas las imágenes individuales.

## 1.8.2 CSS Sprites Generator de Toptal [Link](http://spritegen.website-performance.org/)



Nos ofrece el sprite de imagen para su descarga directa y el CSS para cada elemento de forma individual.

## 1.8.3 Animar sprites



Pero los sprites nos permiten hacer aún más cosas, como animaciones. En las que iremos recorriendo el sprite, a modo de animación stop-motion, consiguiendo un efecto similar al de un GIF animado.

```
.zombi-sprite {  
    width: 155px;  
    height: 194px;  
    background: url('zombi-sprite.png');  
    animation: andarzombi 0.7s steps(5) infinite; }  
@keyframes andarzombi {  
    0% {background-position: 0 0;}  
    20% {background-position: 0 -194px;}  
    40% {background-position: 0 -388px;}  
    60% {background-position: 0 -582px;}  
    80% {background-position: 0 -776px;}  
    100% {background-position: 0 -970px;}}
```

```
@keyframes andarzombi{
    from{background-position:10px;
    }
    to{background-position:-800px;}
}

<div class="zombi-sprite"></div>
```

## 1.9. Texto

### 1.9.1. Tamaño de letra

La recomendación más importante cuando se trabaja con las propiedades tipográficas de CSS está relacionada con el tamaño de la letra. La propiedad `font-size` permite utilizar cualquiera de las nueve unidades de medida definidas por CSS para establecer el tamaño de la letra. Sin embargo, la recomendación es utilizar únicamente las unidades relativas `%`, `em` y `rem`.

Además de mejorar la accesibilidad de los contenidos de texto, las unidades de medida relativas `%`, `em` y `rem` hacen que las páginas sean más flexibles y se adapten a cualquier medio y dispositivo sin apenas tener que realizar modificaciones. Además, si se utilizan las unidades de medida relativas es posible modificar todos los tamaños de letra del sitio de forma consistente e inmediata.

Aunque todos los diseñadores web profesionales conocen esta recomendación y utilizan sólo las unidades `%`, `em` y `rem` para establecer todos sus tamaños de letra, los diseñadores que comienzan a trabajar con CSS encuentran dificultades para comprender estas unidades y prefieren utilizar la unidad `px`.

Si tienes dificultades para comprender la unidad `em` y prefieres establecer los tamaños de letra mediante píxeles, puedes utilizar el siguiente truco. Como todos los navegadores establecen un tamaño de letra por defecto equivalente a `16px`, si se utiliza la siguiente regla CSS:

```
body {
    font-size: 62.5%;
}
```

El tamaño de letra del elemento `<body>`, y por tanto el tamaño de letra base del resto de elementos de la página, se establece como el `62.5%` del tamaño por defecto. Si se calcula el resultado de `16px x 62.5%` se obtienen `10px`.

La ventaja de establecer el tamaño de letra del <body> de esa forma es que ahora se pueden utilizar **em** mientras se piensa en px. En efecto, las siguientes reglas muestran el truco en la práctica:

```
body { font-size: 62.5%; }

h1 { font-size: 2em; /* 2em = 2 x 10px = 20px */ }

p { font-size: 1.4em; /* 1.4em x 10px = 14px */ }
```

Como el tamaño base son 10px, cualquier valor de **em** cuya referencia sea el elemento <body> debe multiplicarse por 10, por lo que se puede trabajar con **em** mientras se piensa en px.

La unidad de medida **rem** es muy similar a em, con la única diferencia de que no es escalable, esto quiere decir que no depende del elemento padre, sino del elemento raíz del documento, el elemento HTML. Rem significa «Root Em», o sea, es un em basado en la raíz.

## 1.9.2. Efectos gráficos

### 1.9.2.1. Texto con sombra

La propiedad llamada **text-shadow** se utiliza para mostrar textos con sombra.

```
h1 {
    color: #000;
    text-shadow: 2px 2px 3px #555; }
```

La sintaxis de la propiedad **text-shadow** obliga a indicar dos medidas y permite establecer de forma opcional una tercera medida y un color. Las dos medidas obligatorias son respectivamente el desplazamiento horizontal y vertical de la sombra respecto del texto. La tercera medida opcional indica lo nítido o borroso que se ve la sombra y el color establece directamente el color de la sombra.

### 1.11.2.2. Efecto brillo

Si queremos crear un resplandor en vez de una sombra, tenemos que darle valor 0 a **text-shadow** en ambas distancias y aplicar un color claro.

```
text-shadow: 0px 0px 20px rgb(0,255,0);
```

Si queremos aumentar la intensidad del brillo, le añadimos más valores de sombra.

```
text-shadow: 0px 0px 20px rgb(0,255,0), 0px 0px 20px rgb(0,255,0), 0px 0px 20px
rgb(0,255,0);
```



### 1.9.2.3 Textos en 3D

Veamos una aplicación práctica de text-shadow que dará como resultado un efecto distinto a una simple sombra paralela.

Sabemos que podemos aplicar diferentes sombras a un texto, y que podemos decidir que el difuminado de la sombra sea 0. Partiendo de esta base, podemos crear un efecto de pseudo 3D a cualquier texto.

Primero tenemos que aplicar el efecto sombra a nuestro texto. Creamos un texto blanco y una sombra a una distancia x e y de un color gris claro y con difuminado 0:

```
text-shadow: 1px 1px 0px rgb(230,230,230);
```



Ahora mismo tiene poco de 3D. Probemos a ir añadiéndole unas cuantas sombras más, aumentando en cada una 1 pixel las distancias x e y y haciendo el gris más oscuro:

```
text-shadow:  
    1px 1px 0px rgb(230,230,230),  
    2px 2px 0px rgba(200,200,200),  
    3px 3px 0px rgb(180,180,180),  
    4px 4px 0px rgb(160,160,160);
```



Eso ya tiene algo más de aspecto 3D, pero aún le podemos dar un aspecto más realista. Aplicaremos una nueva sombra siguiendo el método anterior, con la diferencia que esta vez le daremos color negro. Por último añadimos una sombra mayor que todas las demás, de color gris claro y con difuminado:

```
text-shadow:  
    1px 1px 0px rgb(230,230,230),  
    2px 2px 0px rgb(200,200,200),  
    3px 3px 0px rgb(180,180,180),  
    4px 4px 0px rgb(160,160,160),  
    /*Añadimos*/ 5px 5px 0px rgb(0,0,0), 8px 8px 20px rgb(0,0,0);
```



#### 1.9.2.4. Texto con relleno gradiente o degradado e imágenes

Combinando el texto con imágenes y degradados, se pueden lograr fácilmente efectos gráficos propios de los programas de diseño. A continuación, se detalla cómo crear un texto que muestra su color en forma de degradado o gradiente.

Para ello utilizaremos las siguientes propiedades:

- **background-clip: text;** Con esta propiedad adoptaremos el background a la forma del texto.
- **text-fill-color: transparent;** Con esta propiedad ponemos el color del texto transparente para poder visualizar su background-image.

Utilizando estas propiedades podemos declarar lo siguiente:

```
h1 {  
  
    background-image:url("imagenes/sprite.png"); /* Imagen fondo */  
    -webkit-background-clip: text;           /* adaptamos el degradado al texto */  
    -webkit-text-fill-color: transparent;   /* Ocultamos el color del texto */  
    font-size:6em;  
  
}
```



Si utilizamos la propiedad **gradient** (que estudiaremos más adelante) podremos realizar un degradado en el fondo del texto:

```
h1 {  
    background-image:-webkit-gradient(linear,left top,left bottom,from(blue),to(black));  
    /* Degradado */  
    -webkit-background-clip: text;  
    /* Con esto adaptamos el degradado al texto */  
    -webkit-text-fill-color: transparent;  
    /* Ocultamos el color del texto para no afectar el degradado */  
    font-size:4em;      }  
  
Texto degradado
```

# Capítulo 2. Buenas prácticas

## 2.1. Inicializar los estilos

Cuando los navegadores muestran una página web, además de aplicar las hojas de estilo de los diseñadores, siempre aplican otras dos hojas de estilos: la del navegador y la del usuario.

La hoja de estilos del navegador se utiliza para establecer el estilo inicial por defecto a todos los elementos HTML: tamaños de letra, decoración del texto, márgenes, etc. Esta hoja de estilos siempre se aplica a todas las páginas web, por lo que cuando una página no incluye ninguna hoja de estilos propia, el aspecto con el que se muestra en el navegador se debe a esta hoja de estilos del navegador.

Por su parte, la hoja de estilos del usuario es la que puede aplicar el usuario mediante su navegador. Aunque la inmensa mayoría de usuarios no utiliza esta característica, en teoría es posible que los usuarios establezcan el tipo de letra, color y tamaño de los textos y cualquier otra propiedad CSS de los elementos de la página que muestra el navegador.

El orden en el que se aplican las hojas de estilo es el siguiente:



Figura 2.1. Orden en el que se aplican las diferentes hojas de estilos

Por tanto, las reglas que menos prioridad tienen son las del CSS de los navegadores, ya que son las primeras que se aplican. A continuación, se aplican las reglas definidas por los usuarios y por último se aplican las reglas CSS definidas por el diseñador, que por tanto son las que más prioridad tienen.

**Nota**

CSS define la palabra reservada `!important` para controlar la prioridad de las declaraciones.

El principal problema de las hojas de estilo de los navegadores es que los valores que aplican por defecto son diferentes en cada navegador. Aunque todos los navegadores coinciden en algunos valores importantes (tipo de letra serif, color de letra negro, etc.) presentan diferencias en valores tan importantes como los márgenes verticales (`margin-bottom` y `margin-top`) de los títulos de sección (`<h1>`, ... `<h6>`), la tabulación izquierda de los elementos de las listas (`margin-left` o `padding-left` según el navegador) y el tamaño de línea del texto (`line-height`).

Como todas las hojas de estilo de los navegadores son diferentes, cuando un diseñador prueba sus estilos sobre diferentes navegadores, es común encontrarse con diferencias visuales apreciables. La solución a este problema es muy sencilla y consiste en borrar o *resetear* los estilos que aplican por defecto los navegadores.

Una de las formas más sencillas de neutralizar algunos de los estilos de los navegadores consiste en *eliminar el margen y relleno a todos los elementos* de la página para establecerlos posteriormente de forma individual:

```
* { margin: 0;
  padding: 0; }
```

Aunque la regla CSS anterior se ha utilizado desde hace muchos años, se trata de una solución muy rudimentaria y limitada. La solución completa consiste en crear una hoja de estilos CSS que neutralice todos los estilos que aplican por defecto los navegadores y que pueden afectar al aspecto visual de las páginas. Este tipo de hojas de estilos se suelen llamar "*reset CSS*".

A continuación se muestra la hoja de estilos `reset.css` propuesta por el diseñador [Eric Meyer](http://meyerweb.com/eric/tools/css/reset/) (<http://meyerweb.com/eric/tools/css/reset/>)

```
/* v1.0 / 20080212 */

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre, a,
abbr, acronym, address, big, cite, code, del,
dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var, b, u,
i, center, dl, dt, dd, ol, ul, li, fieldset,
form, label, legend, table, caption, tbody,
tfoot, thead, tr, th, td {

    margin: 0;
    padding: 0;
    border: 0;
    outline: 0;
    font-size: 100%;
    vertical-align: baseline;
    background: transparent;
}
```

```

body
{
    line-height: 1;
}
ol,ul
{
    list-style: none;
}
blockquote, q
{
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after
{
    content: '';
    content: none;
}

/* No olvides definir estilos para focus */
:focus { outline: 0;
}

/* No olvides resaltar de alguna manera el texto insertado/borrado
*/
ins {
    text-decoration: none;
}
del
{
    text-decoration: line-through;
}

/* En el código HTML es necesario añadir cellspacing="0" */
table { border-collapse: collapse; border-spacing: 0;
}

```

El propio Eric Meyer recuerda que la hoja de estilos anterior es sólo un punto de partida que debe ser adaptado por cada diseñador hasta obtener los resultados deseados. Utilizar una hoja de estilos de tipo *reset* es una de las buenas prácticas imprescindibles para los diseñadores web profesionales.

## 2.2. Comprobar el diseño en varios navegadores

### 2.2.1. Principales navegadores

Una de las prácticas imprescindibles de los diseñadores web profesionales consiste en **probar su trabajo en varios navegadores diferentes**. De esta forma, el diseñador puede descubrir los errores de su trabajo y también los errores causados por los propios navegadores.

El número de navegadores y versiones diferentes que se deben probar depende de cada diseñador. En el caso ideal, el diseñador conoce estadísticas de uso de los navegadores que utilizan los usuarios para acceder al sitio o aplicación web que está diseñando. Una buena práctica consiste en probar los diseños en aquellos navegadores y versiones que sumen un 90% de cuota de mercado.

Cuando no se dispone de estadísticas de uso o el diseño es para un sitio web completamente nuevo, se debe probar el diseño en los navegadores más utilizados por los usuarios en general. Aunque no existe ninguna estadística completamente fiable y los resultados varían mucho de un país a otro, en general los siguientes navegadores y versiones suman más del 90% de cuota de mercado: Microsoft Edge, Firefox, Safari, Chrome y Opera .

En primer lugar, los diseñadores web profesionales disponen de todos los navegadores principales instalados en sus equipos de trabajo.

### **2.2.2. Probar el diseño en todos los navegadores**

En algunas ocasiones no es suficiente con probar los diseños en los navegadores más utilizados, ya que el cliente quiere que su sitio o aplicación web se vea correctamente en muchos otros navegadores. Además, en otras ocasiones el diseñador ni siquiera dispone de los navegadores más utilizados por los usuarios, de forma que no puede probar correctamente sus diseños.

Afortunadamente, existe una aplicación web gratuita que permite solucionar todos estos problemas.

Aplicaciones web que realizan testing de navegadores:

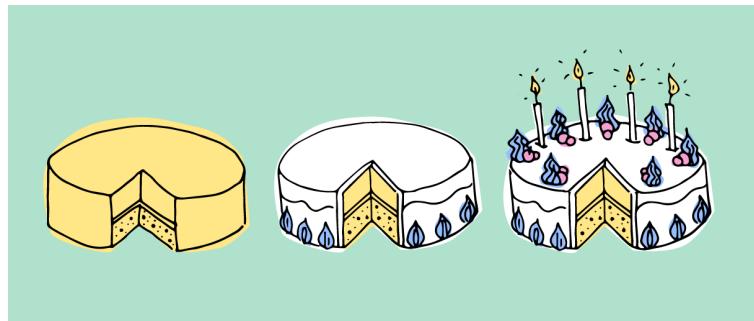
<http://crossbrowsertesting.com/>

<https://www.browserling.com/>

<https://www.browserstack.com>

<https://blisk.io/> (visualizar la web en diferentes dispositivos y navegadores, responsive)

## 2.3. Mejora progresiva



La mejora progresiva ("*progressive enhancement*") es uno de los conceptos más importantes del diseño web y a la vez uno de los más desconocidos. Su origen proviene de su concepto contrario, la degradación útil o "*graceful degradation*".

La degradación útil es un concepto propuesto hace décadas por el psicólogo inglés David Courtenay Marr. Aplicada al diseño web, la degradación útil significa que un sitio web sigue funcionando correctamente cuando el usuario accede con un navegador limitado o antiguo en el que no funcionan las características más avanzadas.

La mejora progresiva toma ese concepto y lo aplica de forma inversa. De esta forma, aplicada al diseño web la mejora progresiva significa que el sitio web dispone de características más avanzadas cuanto más avanzado sea el navegador con el que accede el usuario.

La idea propuesta por la técnica de la mejora progresiva consiste en que el diseño web permita el acceso completo y correcto a toda la información de la página independientemente del tipo de navegador utilizado por el usuario.

Usando la regla **@supports** se puede hacer una consulta de características, verificando en CSS si una función de CSS es compatible con el navegador. Se pueden utilizar las palabras clave **and** y **or**.

```
header h1 {
    font-size: 70px;
    color: green;
    padding: 0;
    margin: 0;
    text-align: center;
}

@supports (mix-blend-mode: screen) {
    header {
        padding: 0;
        background-image: url(https://justmarkup.com/iceland/img/l/DSCF3732.JPG);
        background-size: cover;
        background-repeat: no-repeat;
        width: 100%;
        min-height: 56vw;
        display: flex;
    }
}
```

```
    align-items: center;
    justify-content: center;
}
header h1 {
    color: rgba(55,255,255,0.8);
    mix-blend-mode: screen;
    line-height: 0.5;
}
}
```



## 2.4. Depuración

Inevitablemente, todos los diseñadores web cometan errores en los trabajos que realizan. En la mayoría de las ocasiones, los errores se descubren al probar el diseño en diferentes navegadores. Además de mostrar los errores, los principales navegadores disponen de herramientas que permiten descubrir de forma sencilla la causa concreta del error.

## 🎬 Chrome Developer Tools

Chrome Developer Tools te permite profundizar en las entrañas de una página web y revisar casi todo, desde su estructura hasta los recursos utilizados. En general puedes:

- Obtener una visión general de los estilos utilizados en una página y qué estilo se aplica a cada elemento.
- Visualizar y modificar el código correspondiente a los elementos HTML o Modificar un estilo y ver los cambios en el navegador en tiempo real o Ejecutar código JavaScript en el contexto de la página o Ver las peticiones HTTP realizadas por el navegador o Identificar cuellos de botella que afecten al rendimiento
- Consultar métricas de rendimiento.
- Investigar los recursos offline para averiguar qué datos de la página se almacenan localmente.



## 🎬 Herramienta del W3C para validación de hojas de estilo

El servicio de validación de CSS es un software libre creado para ayudar a diseñadores y desarrolladores web a validar Hojas de Estilo en Cascada (CSS) o

<https://jigsaw.w3.org/css-validator/>

## 2.5. Hojas de estilos

Las hojas de estilos reales de los sitios web profesionales suelen contener cientos de reglas y ocupan miles de líneas. Por este motivo, es imprescindible seguir unas **buenas prácticas al crear las hojas de estilos para mejorar la productividad y reducir los posibles errores**. A continuación, se muestran algunas de las recomendaciones más útiles para crear hojas de estilos profesionales.

## 2.5.1. Llaves

Uno de los elementos básicos que los diseñadores web deben acordar es el tipo de llaves que se utilizan para encerrar la declaración de cada regla CSS. Aunque se utilizan muchos modelos diferentes, a continuación, se muestran los más populares.

Llave de apertura en la misma línea del selector y llave de cierre en una nueva línea para separar unas reglas de otras:

```
selector {
    propiedad1: valor1;
    propiedad2: valor2;
}
```

Una variante del modelo anterior consiste en mostrar cada llave en su propia línea, para separar aún más el selector de su declaración. Este modelo lo utilizan normalmente los programadores web:

```
selector
{
    propiedad1: valor1;
    propiedad2: valor2;
}
```

Por último, existe un **modelo compacto** que no crea nuevas líneas para las llaves. Aunque es mucho más compacto, normalmente es **más difícil de leer**:

```
selector {
    propiedad1: valor1;
    propiedad2: valor2; }
```

## 2.5.2. Tabulaciones

Tabular el código **facilita significativamente su lectura**, por lo que tabular las propiedades CSS es una de las mejores prácticas de los diseñadores web profesionales. Como conocen la mayoría de programadores, no es recomendable insertar tabuladores en el código, sino que se deben emplear 2 o 4 espacios en blanco.

```
/* Propiedades sin tabular */
selector {
propiedad1: valor1;
propiedad2: valor2;
}

/* Propiedades tabuladas con 2 espacios */

selector {

    propiedad1: valor1;
    propiedad2: valor2;
}
```

Extendiendo la práctica de tabular las propiedades, algunos diseñadores recomiendan tabular también las reglas CSS relacionadas. El siguiente ejemplo muestra tres reglas CSS relacionadas entre sí:

```
ul {
    margin: 1em 0;
    padding: 0;
}
ul li {
    list-style-type: square;
}
ul li ul {
    font-style: italic;
    list-style-type: disc;
}
```

La recomendación consiste en tabular las reglas CSS que están relacionadas porque sus selectores están anidados. Por tanto, la regla cuyo selector es `ul li` debería estar tabulada respecto de la regla `ul` y de la misma forma la regla cuyo selector es `ul li ul` debería estar tabulada respecto de `ul li`:

```
ul {
    margin: 1em 0;
    padding: 0;
}
ul li {
    list-style-type: square;
}
ul li ul {
    font-style: italic;
    list-style-type: disc;}
```

### 2.5.3. Propiedades

Cuando se establece la misma propiedad en varios selectores diferentes pero relacionados, se recomienda escribir las reglas CSS en una única línea, para facilitar su lectura:

```
#contenedor #principal h1 { font-size: 2em; }
#contenedor #principal h2 { font-size: 1.8em; }
#contenedor #principal h3 { font-size: 1.6em; }
```

Respecto al orden en el que se indican las propiedades en la declaración, algunos diseñadores recomiendan agruparlas por su funcionalidad:

```
selector {

    position: absolute; /* propiedades de posicionamiento */
```

```

    right: 0;
    bottom: 10px;
    width: 300px;          /* propiedades de tamaño */
    height: 250px;

    color: #000;           /* propiedades tipográficas */
    font-size: 2em;
}

```

Otros diseñadores recomiendan ordenar alfabéticamente las propiedades para facilitar su búsqueda y evitar duplicidades:

```

selector {
    bottom: 10px;
    color: #000;
    font-size: 2em;
    height: 250px;
    position: absolute;
    right: 0;
    width: 300px;
}

```

Según la “[Google HTML/CSS Style Guide](#)” el orden de las propiedades se recomienda alfabéticamente.

Existe una extensión de VSC que nos permite ordenar alfabéticamente **css sorter**.



(Una vez instalado, en la paleta de comandos (ctrl + shift+p) ponemos: **sort css** y buscamos “sort css in alphabetical order”)

#### 2.5.4. Selectores

Los selectores deben ser descriptivos para facilitar la lectura de la hoja de estilos, por lo que hay que poner especial cuidado en elegir el nombre de los atributos **id** y **class**. Además, aunque aumenta el tamaño total de la hoja de estilos, se recomienda utilizar selectores lo más específicos posible para facilitar el mantenimiento de la hoja de estilos:

```

p.especial { ... }                      /* poco específico */
#contenedor #principal p.especial { ... } /* muy específico */

```

Por otra parte, cuando una regla CSS tiene varios selectores largos, es mejor colocar cada selector en su propia línea:

```
#contenedor h1,
#contenedor #principal h2,
#contenedor #lateral blockquote {
    color: #000;
    font-family: arial, sans-serif;
}
```

## 2.5.5. Organización

Cuando una hoja de estilos tiene cientos de reglas, es recomendable dividirla en secciones para facilitar su mantenimiento. Aunque no existe ninguna organización seguida por todos los diseñadores, en general se utilizan las siguientes secciones:

- Estilos básicos (estilos de <body>, tipo de letra por defecto, márgenes de <ul>, <ol> y <li>, estilos de los enlaces, etc.)
- Estilos de la estructura o *layout* (anchura, altura y posición de la cabecera, pie de página, zonas de contenidos, menús de navegación, etc.)
- Estilos del menú de navegación.
- Estilos de cada una de las zonas (elementos de la cabecera, titulares y texto de la zona de contenidos, enlaces, listas e imágenes de las zonas laterales, etc.)

Si la hoja de estilos tiene muchas secciones, algunos diseñadores incluyen un índice o tabla de contenidos al principio del todo.

```
/* TOC:
Random HTML Styles
Forms
General Structure
Navigation
Quotations
Comments and Other Asides
Emphasis
Computers - General
Code
Examples and Figures
Q and A (FAQ)
Tables
Headers
Meta
Specific to Products Pages
*/
```

## 2.5.6. Comentarios

Separar las secciones de la hoja de estilos y otros bloques importantes es mucho más fácil cuando se incluyen comentarios. Por este motivo se han definido decenas de tipos diferentes de comentarios separadores.

El modelo básico sólo añade un **comentario destacado** para el inicio de cada sección importante:

```
/* -----*/
/* ----->>> CONTENEDOR <<<-----*/
/* -----*/
```

Otros diseñadores emplean diferentes comentarios para indicar el comienzo de las secciones y el de las partes importantes dentro de una sección:

```
/* ----- CABECERA ----- */
/* Logotipo ----- */
/* Buscador ----- */
```

## 2.6 Consistencia y Legibilidad

Dos conceptos fundamentales que nos ayudarán a hacer un CSS de calidad / óptimo.

- Consistencia.
- Legibilidad.

### 2.6.1 Consistencia

Una vez establecemos las reglas, las que hemos establecido anteriormente u otras, la consistencia consiste en mantener esas reglas a lo largo del tiempo.

Podemos establecer 3 niveles de consistencia:

- Con nosotros mismos.
- Con el entorno (equipo, librerías de terceros etc..).
- Con estándares.

### 2.6.2 Legibilidad

Un CSS es legible cuando es fácil de leer y entender tanto por el autor como por los demás. Así conseguimos que cualquier cambio posterior, sea por el motivo que sea, sea más sencillo de generar.

Algunos de los aspectos que debemos considerar para mejorar la legibilidad son los siguientes:

- La simplicidad (+ simple, + fácil de leer y entender).

- El uso de IDs y CLASES. (deben de ser significativos, cortos y largos cuando sea necesario).
- El uso de abreviaturas. (equilibrio entre tamaño y legibilidad).
- El uso de !IMPORTANT hace perder legibilidad, ya que se hace más difícil saber qué reglas se están aplicando.

## 2.7 Mantenibilidad

Un CSS es mantenible cuando es fácil de:

- modificar
- corregir
- extender

Y por lo tanto es más rápido hacer cambios y corregir errores.

La mantenibilidad es un concepto multi-factor y debemos de considerar:

- Consistencia y la legibilidad (evidentemente).
- DRY (don't repeat yourself, repetición de selectores o repetición de declaraciones).
- Separación de intereses (Separation of concerns), concepto relacionado con la programación, separación de tipos de ficheros (CSS/JS/HTML), modularidad en los ficheros CSS y separación estilos por funcionalidad.

## 2.8 Herramientas y metodologías para optimizar CSS.

Conforme nuestros proyectos sean más grandes vamos necesitar más ayuda para mantener esa CONSISTENCIA, LEGIBILIDAD y MANTENIBILIDAD.

Para ello necesitaremos herramientas como los preprocesadores que veremos más adelante:



### 2.8.1 METODOLOGÍAS

Además de las herramientas comentadas, existen una serie de metodologías que nos dan una serie de reglas para escribir CSS de manera que sea más mantenible y legible.

- OOCSS
- BEM
- SMACSS

### 2.8.1.2 BEM

La metodología BEM (<http://getbem.com/>) divide la interfaz de usuario en bloques independientes para crear componentes escalables y reutilizables.



BEM (Block, Element, Modifier o Bloque, Elemento, Modificador) es una metodología ágil de desarrollo basada en componentes.

Propone un estilo descriptivo para nombrar cada una de las clases a utilizar, permitiendo así crear una estructura de código consistente.

#### 2.8.1.2.1 Bloques, Elementos y Modificadores (BEM)

Para lograr su objetivo, la metodología se centra en la **reutilización de código**, crea una documentación explícita en cada nueva declaración.

BEM tiene como horizonte **modularizar lo máximo posible cada uno de los bloques de código dispuesto**. Se centra en tres parámetros o variables posibles: **Bloques** (div, section, article, ul, ol, etc.), **elementos** (a, button, li, span, etc.) y **modificadores**. Estos últimos se definen de acuerdo a la posterior utilización que haga el desarrollador a cargo.

##### **Bloques**

Es un contenedor donde se encontrarán los diferentes elementos. Por ejemplo, un encabezado (header), una barra lateral (sidebar/aside) , un área de contenido principal (main) y un pie de página (footer), se consideraría cada uno como un bloque.

El bloque de elementos **corresponde a la raíz de la clase y deberá ir siempre primero**. Cuando ya está definido es posible nombrar los elementos que compondrán la estructura final y su contenido.

**HTML**

```

11   <section>
12     |   <article class="noticia">
13     |     |   <!--Bloque contenedor-->
14     |     |   </article>
15   </section>

```

**CSS**

```

1 .noticia{
2   background: lightgray;
3 }

```

**Elementos**

El elemento es una de las variadas piezas que componen la estructura de un bloque. El bloque es el todo y los elementos son las piezas de este bloque.

De acuerdo a la metodología BEM, cada elemento se escribe después del bloque padre, usando **dos guiones bajos**:

**HTML**

```

11   <section>
12     |   <article class="noticia">
13     |     |   <h1 class="noticia__titulo">Título de la noticia</h1>
14     |     |   </article>
15   </section>

```

**CSS**

```

5 .noticia__titulo{
6   font-family: 'Times New Roman', serif;
7 }

```

Al convertirse en el centro de los nombres de las clases, los elementos ayudan a entender cómo estructurar y manejar las hojas de estilos una vez que se empiece a diseñar el proyecto. Por esto, es importante usar nombres claros y precisos.

**Modificadores**

El ejercicio de nombrar las clases sirve para que los elementos puedan repetirse. Si los elementos son los mismos, no será necesario escribir nuevas clases en otras áreas del sitio.

Para modificar el estilo de un elemento específico, existen los modificadores. Estos se usan agregando **un doble guión** justo después del elemento o bloque que se quiere modificar:

**HTML**

```

11  <section>
12    <article class="noticia--destacada">
13      <h1 class="noticia__titulo--uppercase">Título de la noticia</h1>
14    </article>
15  </section>

```

**CSS**

```

9   .noticia--destacada{
10  background: #dimgrey;
11 }
12  .noticia__titulo--uppercase{
13  text-transform: uppercase;
14 }

```

**Ventajas y desventajas de BEM**

La metodología BEM se encuentra en constante evolución, pero ha demostrado su eficiencia al establecer una buena jerarquía en el desarrollo de plataformas.

- **Añade especificidad:** Usa un selector único para cada regla, lo que permite reducirla y hacer menos repeticiones.
- **Aumenta la independencia:** Los bloques se pueden mover a cualquier parte del documento, sin afectar los estilos.
- **Mejora la herencia múltiple:** Se puede cambiar cualquiera de las tres partidas sin afectar a las demás.
- **Permite la reutilización:** Es posible reciclar ciertas áreas de código desde un proyecto hacia otro, esto debido a la no existencia de dependencias mayores en cuanto a la implementación de cada uno de los bloques estructurados.
- **Entrega simplicidad:** Permite un fácil entendimiento una vez conocido el proceso lógico sobre el cual se basa. A su vez, las convenciones a la hora de nombrar las clases permiten tener un control absoluto al saber a qué, quién y hacia dónde hacemos referencia dentro de una estructura.

Sin embargo, existen algunas **opiniones en contra de esta metodología**. Las más comunes son:

- Las convenciones pueden ser muy largas.
- A algunas personas les puede tomar tiempo aprender la metodología.
- El sistema de organización puede ser difícil de implementar en proyectos pequeños.

## 2.9. Rendimiento

Antes de su explicación detallada, se muestra a continuación la lista de estrategias útiles para mejorar el rendimiento de la parte de CSS de las páginas web:

- Utilizar *sprites CSS* para reducir el número de imágenes de adorno a una única imagen.
- Enlazar hojas de estilos externas en vez de incluir los estilos en la propia página.
- Enlazar las hojas de estilos mediante `<link>` en vez de `@import`. También es importante donde esté declarado el `<link>`.
- Reducir el número de archivos CSS de la página.
- Combinar si es posible todos los archivos CSS individuales en un único archivo CSS.

Del tiempo total que el usuario espera hasta que la página solicitada se carga completamente, el 20% del tiempo corresponde a la parte del servidor (normalmente generar la página HTML de forma dinámica utilizando información de una base de datos) y el 80% restante corresponde a la parte del cliente (normalmente descargar hojas de estilos CSS, archivos JavaScript e imágenes).

De esta forma, es mucho más fácil mejorar el tiempo de respuesta de la página mejorando el rendimiento de la parte del cliente. Como uno de los principales elementos de la parte del cliente está formado por las hojas de estilos CSS, a continuación, se indican algunas de las técnicas para mejorar su rendimiento.

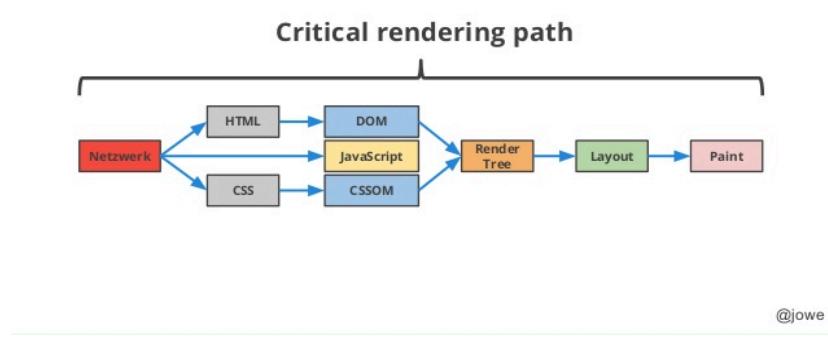
Aproximadamente el 20% de las páginas vistas de un sitio web no disponen de sus elementos guardados en la caché del navegador del usuario. Esto supone que en el 20% de las páginas vistas, los navegadores de los usuarios se descargan todos sus elementos: imágenes, archivos JavaScript y hojas de estilos CSS.

**La conclusión de lo anterior es que resulta mucho mejor reducir el número de archivos diferentes y no reducir el tamaño de cada archivo individual. El objetivo es reducir el número de peticiones HTTP que la página realiza al servidor para descargar todos los contenidos.**

Los factores que influyen en rendimiento y las posibles soluciones serán:

- **El uso del CSS.** ¿Voy a usar todas las reglas del CSS?, ¿Construyo mi propio CSS? ¿Uso frameworks CSS?, el estudio de la **cobertura** CSS nos indicará qué partes de nuestro CSS son utilizadas por nuestra web. (ver “coverage” dev tools (console, Ctrl+ Shift+p, coverage)).

- **Como escribo CSS.** ¿Importan los selectores en el rendimiento? ¿Debo usar inline css para mejorar el rendimiento? ¿Da igual donde incluya el CSS dentro del HTML?.
- **El tamaño de mi CSS.** Al final siempre debo de tener en cuenta que cuanto más pequeño es mi CSS: Más rápido se va a transmitir a través de Internet y más rápido se va a procesar en nuestro navegador.
- **Número de ficheros:** cada petición para obtener un CSS tiene un coste: A nivel de tiempo, el tiempo en realizar la petición y obtener el CSS de vuelta. Para tener el 100% de cobertura en cada página tengo que partir los CSS. Esto requiere más peticiones, un solo CSS con todo , una sola petición, hace disminuir la cobertura.
- **Minimizar el fichero:** No incluir ";" tras la última propiedad. Colores de 3 dígitos en caso de que sea posible. Eliminar los 0's innecesarios en los valores. No usar abreviaturas para propiedades...
- **Comprimir ficheros CSS:** La compresión de los ficheros CSS antes de mandarlos desde el servidor, reduce aún más el tamaño del fichero a transferir entre el cliente y el servidor (hasta 70%...).
- **Cachear el fichero css:** Cuando hacemos CACHING lo que pretendemos es minimizar el tráfico y que el usuario final, al usar nuestra aplicación/web, solo tenga que descargar los archivos estáticos (css/js/img) una única vez.



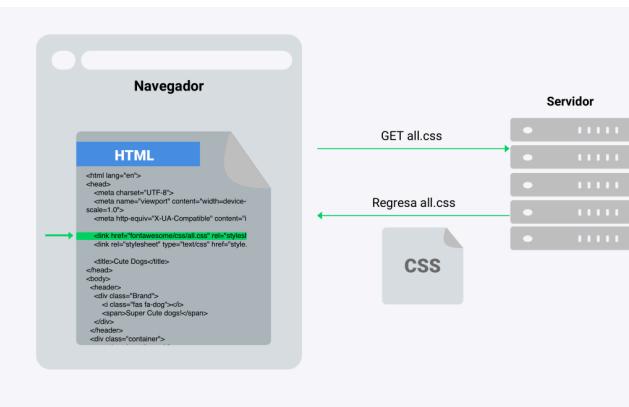
@jowe

## 2.9.2 Critical Render Path

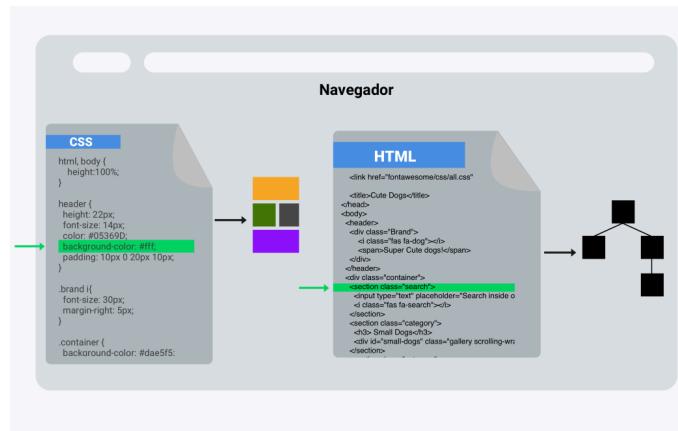
Se refiere a la secuencia de pasos que el navegador realiza para recibir código en HTML, CSS y JavaScript y convertirlos en la vista inicial de una página web.

Cuando un usuario ingresa a este sitio, el navegador hace un request al servidor pidiendo el contenido, y el servidor envía al navegador un archivo HTML.

Una vez que el navegador tiene el archivo HTML, empieza a leerlo y **va creando el DOM** (Document Object Model). Antes de poder comenzar a crear el DOM si encuentra un archivo CSS, lo pide al servidor.

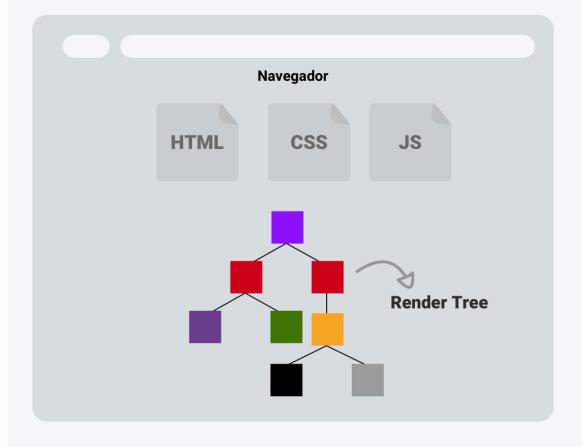


Cuando llega el archivo CSS, el navegador empieza a leerlo y a generar otro árbol llamado **CSSOM**. Este objeto contiene toda la información sobre cómo deberían ser los estilos del DOM.



El navegador seguirá trabajando mientras siga encontrando código HTML o código CSS, hasta que encuentre un archivo JavaScript y nuevamente el navegador le pedirá al servidor el archivo.

Cuando llega el archivo JavaScript, el navegador comienza a leerlo y a realizar modificaciones tanto en el DOM como en el CSSOM. Una vez que termina de leer los 3 archivos, el navegador **combina el DOM y el CSSOM dentro de un nuevo árbol llamado Render Tree**.



Ahora el navegador puede utilizar el Render Tree para pintar el layout y crear la estructura de la página web.

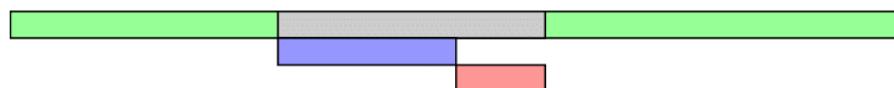
**JavaScript** también puede bloquear *el renderizado del navegador*, pero el procesamiento asíncrono es posible con:

- El atributo **async** para descargar scripts en paralelo, que se ejecutan inmediatamente cuando están listos.
- El atributo **defer** para descargar en paralelo, y luego ejecutar en orden cuando el DOM está listo.
- El atributo **type="module"** para cargar un módulo ES (que se comporta como defer).



### <script>

Let's start by defining what `<script>` without any attributes does. The HTML file will be parsed until the script file is hit, at that point parsing will stop and a request will be made to fetch the file (if it's external). The script will then be executed before parsing is resumed.



### <script async>

`async` downloads the file during HTML parsing and will pause the HTML parser to execute it when it has finished downloading.



### <script defer>

`defer` downloads the file during HTML parsing and will only execute it after the parser has completed. `defer` scripts are also guaranteed to execute in the order that they appear in the document.



Las **imágenes**. Cada vez que el navegador encuentre una imagen, ya sea en el archivo HTML, CSS o JavaScript, las pedirá al servidor, pero las **imágenes no entrarán dentro del proceso anterior**.

Todas las imágenes son **descargadas en segundo plano** y una vez que estén listas aparecerán en la pantalla.

Las imágenes pueden **cargarse de forma diferida** (atributo `loading="lazy"`) sin bloquear la renderización del navegador, el navegador carga las imágenes y el resto de datos solo cuando estos aparecen en la ventana gráfica o viewport (el área visible) del usuario, por ejemplo, al desplazarse por la ventana del navegador o maximizarla.

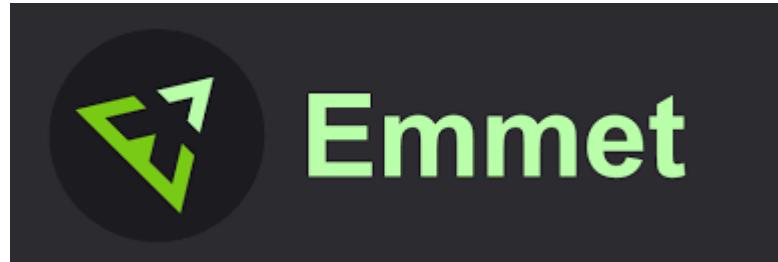
```

```

Nada de esto es posible con CSS. El archivo se almacena en la caché, por lo que las siguientes cargas de la página deberían ser más rápidas

[Más info](#)

## 2.10. Emmet, en Visual Studio.



Emmet en Visual Studio es una herramienta de trabajo que nos permite autocompletar código de una forma muy rápida y muy eficiente.

Más información: [Link](#)

# Capítulo 3. Selectores

Conocer y dominar todos los selectores de CSS es imprescindible para crear diseños web profesionales.

Utilizando solamente los cinco selectores básicos de CSS (universal, de tipo, descendente, de clase y de id) es posible diseñar cualquier página web. No obstante, los selectores avanzados de CSS3 permiten simplificar las reglas CSS y también el código HTML.

## 3.1. Selector de hijos

Se trata de un selector similar al selector descendente, pero muy diferente en su funcionamiento. **Se utiliza para seleccionar un elemento que es hijo de otro elemento** y se indica mediante el "*signo de mayor que*" (>).

Mientras que en el selector descendente sólo importa que un elemento esté dentro de otro, independientemente de lo profundo que se encuentre, **en el selector de hijos el elemento debe ser hijo directo de otro elemento**.

```
p > span { color: blue; }

<p>
    <span>Texto1</span>
</p>

<p>
    <a href="#">
        <span>Texto2</span>
    </a>
</p>
```

En el ejemplo anterior, el selector p > span se interpreta como "*cualquier elemento <span> que sea hijo directo de un elemento <p>*", por lo que el primer elemento <span> cumple la condición del selector. Sin embargo, **el segundo elemento <span> no la cumple porque es descendiente pero no es hijo directo de un elemento <p>**.

Utilizando el mismo ejemplo anterior se pueden comparar las diferencias entre el selector descendente y el selector de hijos:

```

p  a { color: red; }
p > a { color: blue; }

<p>
    <a href="#">Enlace1</a>
</p>

<p>
    <span>
        <a href="#">Enlace2</a>
    </span>
</p>

```

El primer selector es de tipo descendente (`p a`) y por tanto se aplica a todos los elementos `<a>` que se encuentran dentro de elementos `<p>`. En este caso, los estilos de este selector se aplican a los dos enlaces.

El segundo selector es de hijos (`p > a`) por lo que obliga a que el elemento `<a>` sea hijo directo de un elemento `<p>`. Por tanto, los estilos del selector `p > a` no se aplican al segundo enlace del ejemplo anterior.

### 3.2.1 Selector adyacente

El selector adyacente se emplea para seleccionar elementos que son **hermanos** (su elemento padre es el mismo) **y están seguidos en el código HTML**. Este selector emplea en su **sintaxis el símbolo +**. Si se considera el siguiente ejemplo:

```

h1 + h2 { color: red }

<body>
    <h1>Título1</h1>

    <h2>Subtítulo</h2>

    <h2>Otro subtítulo</h2>
</body>

```

Los estilos del selector `h1 + h2` se aplican al primer elemento `<h2>` de la página, pero no al segundo `<h2>`, ya que:

- El elemento padre de `<h1>` es `<body>`, el mismo padre que el de los dos elementos `<h2>`. Así, los dos elementos `<h2>` cumplen la primera condición del selector adyacente.
- El primer elemento `<h2>` aparece en el código HTML justo después del elemento `<h1>`, por lo que este elemento `<h2>` también cumple la segunda condición del selector adyacente.

- Por el contrario, el segundo elemento `<h2>` no aparece justo después del elemento `<h1>`, por lo que no cumple la segunda condición del selector adyacente y por tanto no se le aplican los estilos de `h1 + h2`.

El siguiente ejemplo puede ser útil para los textos que se muestran como libros:

```
p + p { text-indent: 1.5em; }
```

En muchos libros es habitual que la primera línea de todos los párrafos esté indentada, salvo la primera línea del primer párrafo. El selector `p + p` selecciona todos los párrafos que están dentro de un mismo elemento padre y que estén precedidos por otro párrafo. En otras palabras, el selector `p + p` selecciona todos los párrafos de un elemento salvo el primer párrafo.

El selector adyacente requiere que los dos elementos sean *hermanos*, por lo que su elemento *padre* debe ser el mismo. Si se considera el siguiente ejemplo:

```
p + p { color: red; }

<p>Lorem ipsum dolor sit amet...</p>
<p>Lorem ipsum dolor sit amet...</p>
<div>
    <p>Lorem ipsum dolor sit amet...</p>
</div>
```

En el ejemplo anterior, solamente el segundo párrafo se ve de color rojo, ya que:

- El primer párrafo no va precedido de ningún otro párrafo, por lo que no cumple una de las condiciones de `p + p`
- El segundo párrafo va precedido de otro párrafo y los dos comparten el mismo parente, por lo que se cumplen las dos condiciones del selector `p + p` y el párrafo muestra su texto de color rojo.
- El tercer párrafo se encuentra dentro de un elemento `<div>`, por lo que no se cumple ninguna condición del selector `p + p` ya que ni va precedido de un párrafo ni comparte parente con ningún otro párrafo.

### **3.2.2 Selector general de elementos hermanos,**

Generaliza el selector adyacente. Su sintaxis es `elemento1 ~ elemento2` y selecciona el `elemento2` que es *hermano de elemento1* y se encuentra detrás en el código HTML. En el selector adyacente la condición adicional era que los dos elementos debían estar uno detrás de otro en el código HTML, mientras que ahora la única condición es que uno esté detrás de otro.

Si se considera el siguiente ejemplo:

```

h1 + h2 { ... } /* selector adyacente */

h1 ~ h2 { ... } /* selector general de hermanos */

<h1>...</h1>
<h2>...</h2>
<p>...</p>
<div>
    <h2>...</h2>
</div>
<h2>...</h2>

```

El primer selector (`h1 + h2`) sólo selecciona el primer elemento `<h2>` de la página, ya que es el único que cumple que es hermano de `<h1>` y se encuentra justo detrás en el código HTML. Por su parte, el segundo selector (`h1 ~ h2`) selecciona todos los elementos `<h2>` de la página salvo el segundo. Aunque el segundo `<h2>` se encuentra detrás de `<h1>` en el código HTML, no son elementos hermanos porque no tienen el mismo elemento padre.

### 3.3. Selector de atributos

El último tipo de selectores avanzados lo forman los selectores de atributos, que permiten seleccionar elementos HTML en función de sus atributos y/o valores de esos atributos.

**Los cuatro tipos de selectores** de atributos son:

- `[nombre_atributo]`, selecciona los elementos que tienen establecido el atributo llamado `nombre_atributo`, independientemente de su valor.
- `[nombre_atributo=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` con un valor igual a `valor`.
- `[nombre_atributo~=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` y cuyo valor es una lista de palabras separadas por espacios en blanco en la que al menos una de ellas es exactamente igual a `valor`.
- `[nombre_atributo|=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` y cuyo valor es una serie de palabras separadas con guiones, pero que comienza con `valor`. Este tipo de selector sólo es útil para los atributos de tipo `lang` que indican el idioma del contenido del elemento.
- `elemento[atributo^="valor"]`, selecciona todos los elementos que disponen de ese atributo y cuyo valor comienza exactamente por la cadena de texto indicada. (Acento circunflejo, Alt +94).

- **elemento[atributo\$="valor"]**, selecciona todos los elementos que disponen de ese atributo y cuyo valor termina exactamente por la cadena de texto indicada.
- **elemento[atributo\*="valor"]**, selecciona todos los elementos que disponen de ese atributo y cuyo valor contiene la cadena de texto indicada.

A continuación, se muestran algunos ejemplos de estos tipos de selectores:

```

/* Se muestran de color azul todos los enlaces que tengan
un atributo "class", independientemente de su valor */
a[class] { color: blue; }

/* Se muestran de color azul todos los enlaces que tengan
un atributo "class" con el valor "externo" */
a[class="externo"] { color: blue; }

/* Se muestran de color azul todos los enlaces que apunten
al sitio "http://www.ejemplo.com" */
a[href="http://www.ejemplo.com"] { color: blue; }

/* Se muestran de color azul todos los enlaces que tengan
un atributo "class" en el que al menos uno de sus valores
sea "externo" */
a[class~="externo"] { color: blue; }

/* Selecciona todos los elementos de la página cuyo atributo
"Lang" sea igual a "en", es decir, todos los elementos en inglés */
*[lang=en] { ... }

/* Selecciona todos los elementos de la página cuyo atributo
"Lang" empiece por "es", es decir, "es", "es-ES", "es-AR", etc. */
*[lang|=es] { color : red }

/* Selecciona todos los enlaces que apuntan a una dirección de correo electrónico */
a[href^="mailto:"] { ... }

/* Selecciona todos los enlaces que apuntan a una página HTML */
a[href$=".html"] { ... }

/* Selecciona todos los títulos h1 cuyo atributo title contenga la palabra "capítulo" */
h1[title*=capítulo] { ... }

```

## 3.4. Pseudo-clases

### 3.4.1. La pseudo-clase :first-child

La pseudo-clase `:first-child` selecciona el primer elemento hijo de un elemento. Si se considera el siguiente ejemplo:

```
p em:first-child {
    color: red;
}

<p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim. Praesent nulla
ante, <em>ultricies</em> id, porttitor ut, pulvinar quis, dui.</p>
```

El selector `p em:first-child` selecciona el primer elemento `<em>` que sea hijo de un elemento y que se encuentre dentro de un elemento `<p>`. Por tanto, en el ejemplo anterior sólo el primer `<em>` se ve de color rojo.

La pseudo-clase `:first-child` también se puede utilizar en los selectores simples, como se muestra a continuación: | `p:first-child { ... }`

La regla CSS anterior aplica sus estilos al primer párrafo de cualquier elemento. Si se modifica el ejemplo anterior y se utiliza un selector compuesto:

```
p:first-child em {
    color: red;
}

<body> <p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing
elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim.</p>

<p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing elit.
Praesent odio sem, tempor quis, <em>auctor eu</em>, tempus at, enim.</p>

<div> <p>Lorem <span><em>ipsum dolor</em></span> sit amet, consectetuer adipiscing
elit.
Praesent odio sem, tempor quis, <span><em>auctor eu</em> </span>, tempus at,
enim.</p>
</div>
</body>
```

El selector `p:first-child em` selecciona todos aquellos elementos `<em>` que se encuentren dentro de un elemento `<p>` que sea el primer hijo de cualquier otro elemento.

El primer párrafo del ejemplo anterior es el primer hijo de `<body>`, por lo que sus `<em>` se ven de color rojo. El segundo párrafo de la página no es el primer hijo de ningún elemento,

por lo que sus elementos `<em>` interiores no se ven afectados. Por último, el tercer párrafo de la página es el primer hijo del elemento `<div>`, por lo que sus elementos `<em>` se ven de la misma forma que los del primer párrafo.

### 3.4.2 La pseudo-clase :nth-child

- `elemento:nth-child(Número)`, selecciona el elemento indicado pero con la condición de que sea el hijo enésimo de su padre. Este selector es útil para seleccionar el segundo párrafo de un elemento, el quinto elemento de una lista, etc.

Podemos seleccionar los hijos impares (`odd`) o pares (`even`) utilizando las palabras clave `odd` o `even`.

- `elemento:nth-last-child(Número)`, idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.
- `elemento:empty`, selecciona el elemento indicado pero con la condición de que no tenga ningún hijo. La condición implica que tampoco puede tener ningún contenido de texto.

Ej: Se podría utilizar para aplicar tramas a celdas vacías en una tabla:

```
Table td:empty{ background-image:url(images/pattern.png);}
```

- `elemento:first-child` y `elemento:last-child`, seleccionan los elementos indicados pero con la condición de que sean respectivamente los primeros o últimos hijos de su elemento padre.
- `elemento:nth-of-type(Número)`, selecciona el elemento indicado pero con la condición de que sea el enésimo elemento hermano de ese tipo.
- `elemento:nth-last-of-type(Número)`, idéntico al anterior pero el número indicado se empieza a contar desde el último hijo.

Algunas pseudo-clases como `:nth-child(Número)` permiten el uso de expresiones complejas para realizar selecciones avanzadas:

```
li:nth-child(2n+1) { ... } /* selecciona todos Los elementos impares de una Lista */
li:nth-child(2n) { ... } /* selecciona todos los elementos pares de una Lista */

/* Las siguientes reglas alternan cuatro estilos diferentes para Los párrafos
*/ p:nth-child(4n+1) { ... } p:nth-child(4n+2) { ... } p:nth-child(4n+3) { ... }
p:nth-child(4n+4) { ... }
```

Empleando la pseudo-clase `:nth-of-type(número)` se pueden crear reglas CSS que alternan la posición de las imágenes en función de la posición de la imagen anterior:

```
img:nth-of-type(2n+1) { float: right; } img:nth-of-
type(2n) { float: left; }
```

`:not()`, se puede utilizar para seleccionar todos los elementos que no cumplen con la condición de un selector:

```
:not(p) { ... } /* selecciona todos los elementos de la página que no sean párrafos */
*/
:not(#especial) { ... } /* selecciona cualquier elemento cuyo atributo id no sea "especial" */
```

### 3.4.3. Las pseudo-clases `:link` y `:visited`

Las pseudo-clases `:link` y `:visited` se pueden utilizar para aplicar diferentes estilos a los enlaces de una misma página:

- La pseudo-clase `:link` se aplica a todos los enlaces que todavía no han sido visitados por el usuario.
- La pseudo-clase `:visited` se aplica a todos los enlaces que han sido visitados al menos una vez por el usuario.

El navegador gestiona de forma automática el cambio de enlace no visitado a enlace visitado. Aunque el usuario puede borrar la caché y el historial de navegación de forma explícita, los navegadores también borran de forma periódica la lista de enlaces visitados.

Por su propia definición, las pseudo-clases `:link` y `:visited` son mutuamente excluyentes, de forma que un mismo enlace no puede estar en los dos estados de forma simultánea.

Como los navegadores muestran por defecto los enlaces de color azul y los enlaces visitados de color morado, es habitual modificar los estilos para adaptarlos a la guía de estilo del sitio

```
web: a:link { color: red; }
a:visited { color: green; }
```

### 3.4.4. Las pseudo-clases `:hover`, `:active` y `:focus`

Las pseudo-clases `:hover`, `:active` y `:focus` permiten al diseñador web variar los estilos de un elemento en respuesta a las acciones del usuario. Al contrario que las pseudo-clases `:link` y `:visited` que sólo se pueden aplicar a los enlaces, estas pseudo-clases se pueden aplicar a cualquier elemento.

A continuación se indican las acciones del usuario que activan cada pseudo-clase:

- `:hover`, se activa cuando el usuario pasa el ratón o cualquier otro elemento apuntador por encima de un elemento.

- **:active**, se activa cuando el usuario activa un elemento, por ejemplo cuando pulsa con el ratón sobre un elemento. El estilo se aplica durante un espacio de tiempo prácticamente imperceptible, ya que sólo dura desde que el usuario pulsa el botón del ratón hasta que lo suelta.
- **:focus**, se activa cuando el elemento tiene el foco del navegador, es decir, cuando el elemento está seleccionado. Normalmente se aplica a los elementos `<input>` de los formularios cuando están activados y por tanto, se puede escribir directamente en esos campos.

De las definiciones anteriores se desprende que **un mismo elemento puede verse afectado por varias pseudo-clases diferentes** de forma simultánea. Cuando se pulsa por ejemplo un enlace que fue visitado previamente, al enlace le afectan las pseudo-clases `:visited`, `:hover` y `:active`.

Debido a esta característica y al comportamiento en cascada de los estilos CSS, **es importante cuidar el orden en el que se establecen** las diferentes pseudo-clases. El siguiente ejemplo muestra el único orden correcto para establecer las cuatro pseudo-clases principales en un enlace:

```
a:link { ... }
a:visited { ... }
a:hover { ... }
a:active { ... }
```

Por último, también es posible aplicar estilos combinando varias pseudo-clases compatibles entre sí. La siguiente regla CSS por ejemplo sólo se aplica a aquellos enlaces que están seleccionados y en los que el usuario pasa el ratón por encima:  
`a:focus:hover { ... }`

### 3.4.5 Pseudoclases de validación

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

- :required** Cuando el campo es obligatorio, o sea, tiene el atributo `required`.
- :optional** Cuando el campo es opcional (por defecto, todos los campos).
- :invalid** Cuando los campos no cumplen la validación HTML5.
- :valid** Cuando los campos cumplen la validación HTML5.
- :out-of-range** Cuando los campos numéricos están fuera del rango.
- :in-range** Cuando los campos numéricos están dentro del rango.
- :placeholder-shown** Se aplica al `input` cuando el texto del placeholder se está mostrando

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



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

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

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

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

HTML5 brinda un excelente soporte de validaciones desde el lado del cliente, pudiendo comprobar si los datos especificados son correctos o no antes de realizar las validaciones en el lado del servidor, y evitando la latencia de enviar la información al servidor y recibirla de vuelta.

**Ojo:** Ten en cuenta que la validación de cliente es apropiada sólo para reducir la latencia de envío/recepción al servidor, pero nunca como estrategia para evitar problemas de seguridad o similares, para la cuál se debe tener validación en el servidor siempre.

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

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

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

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

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

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

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

Lo ideal sería establecer un rango, algo que se suele hacer muy a menudo si tenemos campos numéricos de formulario:

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

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

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

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

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

Ejemplo de uso con :placeholder-shown

```
// Posibles estados de placeholder

:reqired:placeholder-shown {
    border: 1px solid #888;
```

```

}

:required:not(:placeholder-shown):invalid {
    border: 1px solid red;
    box-shadow: 0 0 0 1px;
}

:required:not(:placeholder-shown):valid {
    border: 1px solid green;
    box-shadow: 0 0 0 1px;
}

:required:focus:invalid {
    border: 1px solid red;
    box-shadow: 0 0 0 1px;
}

```

## 3.5. Pseudo-elementos

Los selectores de CSS, las pseudo-clases y todos los elementos HTML no son suficientes para poder aplicar estilos a algunos elementos especiales. Si se desea por ejemplo cambiar el estilo de la primera línea de texto de un elemento, no es posible hacerlo con las utilidades anteriores.

La primera línea del texto normalmente es variable porque el usuario puede aumentar y disminuir la ventana del navegador, puede disponer de más o menos resolución en su monitor y también puede aumentar o disminuir el tamaño de letra del texto.

La única forma de poder seleccionar estos elementos especiales es mediante los pseudo-elementos definidos por CSS para este propósito.

### 3.5.1. El pseudo-elemento :first-line

El pseudo-elemento `:first-line` permite seleccionar la primera línea de texto de un elemento. Así, la siguiente regla CSS muestra en mayúsculas la primera línea de cada párrafo:

```
p:firstline { text-transform: uppercase; }
```

Este pseudo-elemento sólo se puede utilizar con los elementos de bloque y las celdas de datos de las tablas.

Se pueden combinar varios pseudo-elementos de tipo `:first-line` para crear efectos avanzados:

```

div:first-line { color: red; }
p:first-line { text-transform: uppercase; }

<div>
```

```
<p>Lorem ipsum dolor sit amet...</p>
<p>Lorem ipsum dolor sit amet...</p>
<p>Lorem ipsum dolor sit amet...</p>
</div>
```

En el ejemplo anterior, la primera línea del primer párrafo también es la primera línea del elemento `<div>`, por lo que se le aplican las dos reglas CSS y su texto se ve en mayúsculas y de color rojo.

### 3.5.2. El pseudo-elemento :first-letter

El pseudo-elemento `:first-letter` permite seleccionar la primera letra de la primera línea de texto de un elemento. De esta forma, la siguiente regla CSS muestra en mayúsculas la primera letra del texto de cada párrafo:

```
p:first-letter { text-transform: uppercase; }
```

Los signos de puntuación y los caracteres como las comillas que se encuentran antes y después de la primera letra también se ven afectados por este pseudo-elemento.

Este pseudo-elemento sólo se puede utilizar con los elementos de bloque y las celdas de datos de las tablas.

### 3.5.3. Los pseudo-elementos :before y :after

Los pseudo-elementos `:before` y `:after` se utilizan en combinación con la propiedad `content` de CSS para añadir contenidos antes o después del contenido original de un elemento.

Las siguientes reglas CSS añaden el texto `Capítulo -` delante de cada título de sección `<h1>` y el carácter `.` detrás de cada párrafo de la página:

```
h1:before { content: "Capítulo - ";
```

```
p:after { content: "."; }
```

El contenido insertado mediante los pseudo-elementos `:before` y `:after` se tiene en cuenta en los otros pseudo-elementos `:first-line` y `:first-letter`.

### 3.5.4 Pseudo-elemento ::selection

- `::selecion`, selecciona el texto que ha seleccionado un usuario con su ratón o teclado.

```
h1::selection { background-color: yellow; }
```

### 3.5.5 El selector :target

Todos sabemos cómo enlazar a un elemento con un id especificado dentro de una página:

```
<p id="arriba">Arriba</p>
. . .
<a href="#arriba">Ir arriba</a>
```

El elemento enlazado es un elemento de destino, o un elemento "target". Podemos dar formato al elemento de destino utilizando el selector `:target` de CSS3.

```
:target {
    border: 1px solid #d9d9d9;
    background-color: #FFC;
}
```

**:target** representa el único elemento, si existe alguno, con un id coincidente con el identificador de fragmentos de la URL del documento.

Considera una URL como ésta: <http://ejemplo.com/ejemplo.html#arriba>

Si un usuario accede a la url anterior, el elemento que tiene la referencia de Id="arriba" será el target, por lo que los estilos definidos anteriormente se aplican a ese elemento.

Ejemplo creado con grid y :target [link](#)

### 3.5.6 Agrupación de selectores:

- Combinador :is()

Reescribir de forma más compacta y sencilla los selectores múltiples combinados

```
.container .item,
.container .parent,
.container .element {
    /* ... */
}
```

Con el combinador :is() quedaría de la siguiente manera:

```
.container :is(.item, .parent, .element) {
    /* ... */
}
```

- Nesting CSS

Anidar código CSS, crear componentes CSS nativos autocontenido dentro de otros

```
.container {
    width: 800px;
    height: 300px;
    background: grey;

    .item {
        height: 150px;
        background: orangered;
    }
}
```

Sería equivalente a decir: .container .item {...}

Dentro del CSS Nesting tenemos el **operador &**, que nos permite hacer referencia al selector inmediatamente padre dentro del anidamiento. Esto puede resultarnos muy útil en algunos casos

diferentes al anterior. Observa el siguiente ejemplo, donde en lugar de la clase .item seleccionamos cualquier elemento div que esté dentro de .container:

```
.container {
    background: grey;
}

& div {
    background: indigo;
}
```

## Capítulo 4. Control del espacio

### 4.1. Propiedad white-space

<b>Definición</b>	Establece el tratamiento de los espacios en blanco
<b>Valores permitidos</b>	Uno y sólo uno de los siguientes valores: <ul style="list-style-type: none"> <li>▪ normal</li> <li>▪ pre</li> <li>▪ nowrap</li> <li>▪ pre-wrap</li> <li>▪ pre-line</li> <li>▪ <u>inherit</u></li> </ul>
<b>Valor inicial</b>	normal
<b>Se aplica a</b>	Todos los elementos
<b>Válida en</b>	<u>medios visuales</u>
<b>Se hereda</b>	si
<b>Definición en el estándar</b>	<a href="http://www.w3.org/TR/CSS21/text.html#propdef-white-space">http://www.w3.org/TR/CSS21/text.html#propdef-white-space</a>

El tratamiento de los espacios en blanco en el código HTML es una de las características más desconcertantes para los diseñadores web que comienzan a crear páginas. A continuación se muestra cómo visualizan los navegadores dos párrafos de ejemplo:

*Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum*

*tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.*

*Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.*

Aunque los dos párrafos anteriores se visualizan de la misma forma, en realidad su código HTML es completamente diferente:

```
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.</p>
```

```
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.</p>
```

El segundo párrafo contiene numerosos espacios en blanco y saltos de línea. Sin embargo, los navegadores eliminan automáticamente todos los espacios en blanco sobrantes salvo el espacio en blanco que separa las palabras del texto.

Los dos párrafos se ven exactamente igual.

La única excepción de este comportamiento es la etiqueta `<pre>` de HTML, utilizada para mostrar texto que ya tiene formato (su nombre viene de *preformatado*) y que por tanto respeta todos los espacios en blanco y todos los saltos de línea:

```
  Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.
```

El código HTML del ejemplo anterior es:

```
<pre>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non sem quis tellus vulputate lobortis. Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede. Nulla cursus porta sem. Donec mollis nunc in leo.</pre>
```

La propiedad `white-space` permite variar el comportamiento de los espacios en blanco. El estándar CSS define cinco modelos diferentes de tratamiento de espacios en blanco:

- **normal**: los espacios en blanco sobrantes y los saltos de línea se eliminan. No obstante, el texto se muestra en tantas líneas como sea necesario para que sus contenidos no se salgan del elemento contenedor.
- **pre**: no se eliminan los espacios en blanco sobrantes y sólo se muestran los saltos de línea incluidos en el texto original. Este comportamiento puede provocar que los contenidos de texto se salgan de su elemento contenedor.
- **nowrap**: se comporta igual que **normal** en los espacios en blanco, pero no añade saltos de línea en el texto original, por lo que los contenidos se pueden salir de su elemento contenedor.
- **pre-wrap**: se comporta igual que **pre**, pero se introducen los saltos de línea que sean necesarios para que los contenidos de texto nunca se salgan de su elemento contenedor.
- **pre-line**: se eliminan los espacios en blanco sobrantes, pero se respetan los saltos de línea originales y se crean tantos saltos de línea como sean necesarios para que el contenido de texto no se salga de su elemento contenedor.

La siguiente tabla resume el comportamiento de cada valor:

Valor	Respetá espacios en blanco	Respetá saltos de línea	Ajusta las líneas
<b>normal</b>	no	no	sí
<b>pre</b>	sí	sí	no
<b>nowrap</b>	no	no	no
<b>pre-wrap</b>	sí	sí	sí
<b>pre-line</b>	no	sí	sí

A continuación se muestra el efecto de cada modelo de tratamiento de espacios en blanco sobre un mismo párrafo que contiene espacios en blanco y saltos de línea y que se encuentra dentro de un elemento contenedor de anchura limitada:

```
[white-space: normal] Lorem
ipsum dolor sit amet, consectetuer
adipiscing elit. Vestibulum dictum.
Class aptent taciti sociosqu ad litora
torquent per conubia nostra, per
inceptos hymenaeos.
```

```
[white-space: pre] Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Vestibulum dictum. Class aptent taciti sociosqu ad litora
torquent per conubia nostra, per inceptos hymenaeos.
```

**[white-space: pre-wrap]** Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vestibulum dictum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

**[white-space: nowrap]** Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vestibulum dictum. Class

**[white-space: pre-line]** Lorem ipsum dolor sit amet, consectetuer adipiscing elit

Vestibulum dictum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

### 4.1.2 Cambio de línea (word-break)

La propiedad word-break decide si las palabras tienen que romperse o no al final de línea. La propiedad word-break puede tomar dos valores:

word-break: *break-all* (*rompe las palabras* )

word-break: *keep-all* (*no las rompe* )

A mi me gusta  
Marry Poppins.  
**Supercalifragilístic  
oexpialidoso** es el  
título de una  
canción muy  
llamativa de la  
película de Disney  
Mary Poppins.

A mi me gusta  
Marry Poppins.  
**Supercalifragilísticoexpialidoso**  
es el título de una  
canción muy  
llamativa de la  
película de Disney  
Mary Poppins.

---

```
p.rompe_palabras{ word-break: break-all; }
```

```
p.palabras_enteras{ word-break: keep-all; }
```

## 4.2. Propiedad **display**

<b>Definición</b>	Establece el tipo de caja generada por un elemento		
<b>Valores permitidos</b>	Uno y sólo uno de los siguientes valores: <ul style="list-style-type: none"> <li>▪ inline</li> <li>▪ run-in</li> <li>▪ inline-table</li> <li>▪ table-footer-group</li> <li>▪ table-column</li> <li>▪ none</li> <li>▪ block</li> <li>▪ inline-block</li> <li>▪ table-row-group</li> <li>▪ table-row</li> <li>▪ table-cell</li> <li>▪ inherit</li> <li>▪ list-item</li> <li>▪ table</li> <li>▪ table-header-group</li> <li>▪ table-column-group</li> <li>▪ table-caption</li> </ul>		
<b>Valor inicial</b>	inline		
<b>Se aplica a</b>	Todos los elementos		
<b>Válida en</b>	all		
<b>Se hereda</b>	no		
<b>Definición en el estándar</b>	<a href="http://www.w3.org/TR/CSS21/visuren.html#propdef-display">http://www.w3.org/TR/CSS21/visuren.html#propdef-display</a>		

La propiedad **display** es una de las propiedades CSS más infráutilizadas. Aunque todos los diseñadores conocen esta propiedad y utilizan sus valores **inline**, **block** y **none**, las posibilidades de **display** son mucho más avanzadas.

De hecho, la propiedad **display** es una de las más complejas de CSS, ya que establece el tipo de la caja que genera cada elemento.

El valor más sencillo de **display** es **none** que hace que el elemento no genere ninguna caja. El resultado es que el elemento desaparece por completo de la página y no ocupa sitio, por lo que los elementos adyacentes ocupan su lugar. Si se utiliza la propiedad **display: none** sobre un elemento, todos sus descendientes también desaparecen por completo de la página.

Si se quiere hacer un elemento invisible, es decir, que no se vea pero que siga ocupando el mismo sitio, se debe utilizar la propiedad **visibility**. La propiedad **display: none** se utiliza habitualmente en aplicaciones web dinámicas creadas con

JavaScript y que muestran/ocultan contenidos cuando el usuario realiza alguna acción como pulsar un botón o un enlace.

Los otros dos valores más utilizados son `block` e `inline` que hacen que la caja de un elemento sea de bloque o en línea respectivamente.

Si se aplica la propiedad `display: inline` a un párrafo, su caja se convierte en un elemento en línea y por tanto sólo ocupa el espacio necesario para mostrar sus contenidos.

De la misma forma, si a un enlace se emplea la propiedad `display: block` se transforma en elementos de bloque, por lo que siempre empiezan en una nueva línea y siempre ocupan todo el espacio disponible en la línea, aunque sus contenidos no ocupen todo el sitio.

Uno de los valores más curiosos de `display` es `inline-block`, que crea cajas que son de bloque y en línea de forma simultánea. Una caja de tipo `inline-block` se comporta como si fuera de bloque, pero respecto a los elementos que la rodean es una caja en línea.

El enlace del siguiente ejemplo es de tipo `inline-block`, lo que permite por ejemplo establecer un tamaño mediante la propiedad `width`:

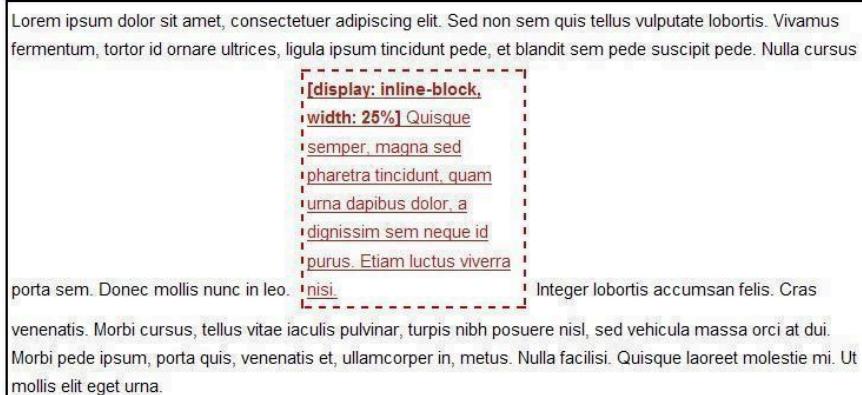


Figura 4.1. Ejemplo del valor `inline-block` de la propiedad `display`

Otro de los valores definidos por la propiedad `display` es `list-item`, que hace que cualquier elemento de cualquier tipo se muestre como si fuera un elemento de una lista (elemento `<li>`). El siguiente ejemplo muestra tres párrafos que utilizan la propiedad `display: list-item` para simular que son una lista de elementos de tipo `<ul>`:

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.

- Sed non sem quis tellus vulputate lobortis.
- Vivamus fermentum, tortor id ornare ultrices, ligula ipsum tincidunt pede, et blandit sem pede suscipit pede.

El resto de valores de la propiedad `display` están relacionados con las tablas y hacen que un elemento se muestre como si fuera una parte de una tabla: fila, columna, celda o grupos de filas/ columnas. Los valores definidos por la propiedad `display` son `inline-table`, `table-row-group`, `table-header-group`, `table-footer-group`, `table-row`, `table-column-group`, `table-column`, `table-cell`, `table-caption`.

Por último tenemos el valor **FLEX** para la propiedad `display`. Flexbox representa un modelo básico de maquetación que supone la existencia de una caja padre llamada contenedor flexible o caja flex. Los elementos hijos situados dentro del contenedor flexible llevan el nombre de elementos o ítems flex, pero eso lo veremos en el siguiente tema.

## 4.2.2 Columnas múltiples

En CSS3 podemos crear fácilmente varias columnas de texto, como las de los periódicos. ( Dicen los expertos que las columnas mejoran la legibilidad del texto. ) Para crear columnas múltiples en CSS3 podemos utilizar las siguientes propiedades:

Propiedad	Sintaxis	Descripción
<code>column-count</code>	<code>column-count: número   auto</code>	Especifica el número de columnas
<code>column-gap</code>	<code>column-gap: tamaño   normal</code>	Especifica el tamaño ( px, em, etc... ) del espacio entre columnas.
<code>column-rule</code>	<code>column-rule: column-rule-width column-rule-style column-rule-color</code>	Método abreviado para establecer las propiedades del separador entre columnas.
<code>column-rule-color</code>	<code>column-rule-color: color</code>	Establece el color del separador entre columnas.
<code>column-rule-style</code>	<code>column-rule-style: none   hidden   dotted   dashed   solid   double   groove   ridge   inset   outset;</code>	Establece el estilo del separador entre columnas.
<code>column-rule-width</code>	<code>column-rule-width: medium   thin   thick   anchura;</code>	Especifica la anchura del separador entre columnas.
<code>column-span</code>	<code>column-span: 1   all;</code>	Establece si un elemento puede extenderse a lo largo de varias columnas.

<b>column-width</b>	column-width: <i>anchura</i>   auto;	Sugiere la anchura óptima de las columnas.
<b>columns</b>	columns: <i>column-width</i> <i>column-count</i>	Método abreviado para establecer las propiedades column-width y column-count

Hay que preceder todas estas propiedades con los prefijos ( vendor prefixes ) correspondientes ( -webkit-, -moz- ... ).

### La propiedad column-count ( número de columnas )

La propiedad column-count (*número de columnas*) especifica el número de columnas de un elemento. En este caso queremos que el elemento .múltiple tenga 2 columnas.

```
.multiple {
    -webkit-column-count :2;
    -webkit-column-gap : 3em;
    width:100%;

}
<div class="multiple">
    <p>En un lugar...
    </p>
</div>
```

#### DEMO:

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de

añadidura los domingos, consumían las tres partes de su hacienda. El resto della concluían sayo de velarte, calzas de velludo para las fiestas, con sus pantuflos de lo mismo, y los días de entresemana se honraba con su vellori de lo más fino.

### La propiedad column-gap ( espaciado de columnas )

La propiedad column-gap (*espaciado de columnas*) especifica el tamaño del espacio entre columnas. En el ejemplo anterior hemos establecido una distancia entre columnas de 3em.

Si no especificamos el valor de column-gap el navegador decidirá el valor de column-gap: normal. La especificación recomienda que el valor para la palabra clave normal sea de 1em.

### La propiedad column-rule ( borde entre columnas )

Podemos separar las columnas con una raya vertical utilizando la propiedad column-rule (*borde o separador entre columnas*). Esta propiedad toma los mismos valores que border.

Exactamente como border, la propiedad column-rule especifica de manera abreviada las características del borde entre columnas. Si queremos, podemos desglosar utilizando column-rule-color ( para especificar el color ), column-rule-style ( para especificar el estilo de línea ) y column-rule-width ( para especificar la anchura del borde ).

```
.borde {
    -webkit-column-rule: 1px solid #d9d9d9;
}

.multiple {
    -webkit-column-count :2;
    column-gap : 3em;
    width:100%;
}

<div class="multiple borde">
    <p>En un lugar de la Mancha...
    </p>
</div>
```

### La propiedad column-width ( ancho de columnas )

La propiedad column-width ( *ancho de columnas* ) **sugiere** la anchura óptima de las columnas. El navegador volverá a calcular el ancho de columna teniendo en cuenta el valor de column-width. Si utilizamos esta propiedad sin indicar el número de columnas, el CSS generará un número variable de columnas dependiendo de la anchura del elemento padre.

```
.anchura {
    -webkit-column-width : 300px;
    ...
}
```

### La propiedad column-span ( extensión de columna )

La propiedad column-span hace que el texto se expanda en múltiples columnas ( column-span :all ) o en una sola ( column-span:1 )

```
.spanAll {
    -webkit-column-span: all;
    text-align: center;
}
```

### 4.3. Propiedad outline

<b>Definición</b>	Establece algunas o todas las propiedades de todos los perfiles de los elementos
<b>Valores permitidos</b>	Alguno o todos los siguientes valores y en cualquier orden: <ul style="list-style-type: none"> <li>▪ Color de perfil (<a href="#">./outline-color.html</a>)</li> <li>▪ Estilo de perfil (<a href="#">./outline-style.html</a>)</li> <li>▪ Anchura de perfil (<a href="#">./outline-width.html</a>)</li> </ul>
<b>Valor inicial</b>	Cada propiedad define su propio valor por defecto
<b>Se aplica a</b>	Todos los elementos
<b>Válida en</b>	medios visuales, medios interactivos ( <a href="#">./medios.html</a> )
<b>Se hereda</b>	no
<b>Definición en el estándar</b>	<a href="http://www.w3.org/TR/CSS21/ui.html#propdef-outline">http://www.w3.org/TR/CSS21/ui.html#propdef-outline</a>

La propiedad **outline** es una de las "*propiedades shorthand*" que define CSS y que se utilizan para establecer de forma abreviada el valor de una o más propiedades individuales. En este caso, se utiliza para establecer el mismo grosor, estilo y/o anchura de todos los perfiles de un elemento.

Aunque es cierto que guarda muchas similitudes con la propiedad **border**, en realidad se diferencian en algunos aspectos muy importantes:

1. Los perfiles no ocupan espacio, mientras que los bordes normales sí.
2. Los perfiles pueden tener formas no rectangulares.

Desde el punto de vista del diseño, la primera característica es la más importante. Los perfiles u **outline** siempre se dibujan "por encima del elemento", por lo que no modifican la posición o el tamaño total de los elementos.

En el siguiente ejemplo se muestran dos cajas; la primera muestra un borde muy grueso y la segunda muestra un perfil de la misma anchura:

```
div { border: 5px solid #369; }
```

```
div { outline: 5px solid #369; }
```

El perfil de la segunda caja se dibuja justo por el exterior de su borde. Aunque visualmente no lo parezca, el perfil y el borde no se encuentran en el mismo plano. De esta forma, el perfil no se tiene en cuenta para calcular la anchura total de un elemento y no influye en el resto de elementos cercanos.

La segunda característica importante de los perfiles pueden tener formas no rectangulares. En el siguiente ejemplo se muestra un texto muy largo encerrado en una caja muy estrecha, lo que provoca que el texto se muestre en varias líneas:

```
span { border: 2px solid ;  
outline: 2px solid #369;
```

La propiedad `outline` perfila los contenidos del elemento y los encierra con una forma no rectangular. No obstante, si visualizas esta misma página en diferentes navegadores, verás que todos los navegadores dibujan el perfil de forma diferente, al contrario de lo que sucede con el borde.

Aunque la propiedad `outline` es infinitamente menos utilizada que la propiedad `border`, la has visto muchas más veces de las que crees. Si pulsas repetidamente la tecla del tabulador en una página web, el navegador va seleccionando de forma secuencial todos los elementos *pinchables* o *seleccionables*: enlaces, botones, controles de formulario, etc.

Para indicar el elemento que está seleccionado, el navegador muestra un perfil muy fino de 1px de ancho, de estilo punteado y del color que más contrasta con el color de fondo de la página. En la mayoría de las páginas web, el perfil que se muestra por defecto es `outline: 1px dotted #000`.

`pseudo-clase:focus` que permite establecer el estilo de los elementos seleccionados.

Utilizando la propiedad `outline` junto con `:focus` se puede modificar el estilo por defecto del navegador:

```
<style type="text/css">  
:focus { outline: 2px solid red; }  
</style>
```

## outline-offset

Por defecto, outline se dibuja a partir y hacia el exterior del border del elemento ( o donde acabe su anchura/altura si es 0). Sin embargo con la propiedad outline-offset esto se puede alterar.

outline-offset define la distancia al borde donde comenzará a dibujarse el contorno formado por outline.

Particularidades y características de outline-offset:

- El valor por defecto es 0 (cero).
- Valores negativos están permitidos. Eso significa que el outline se dibuja hacia el interior del elemento.
- En según qué circunstancias, la conjunción de 'outline-width' y 'outline-offset' puede suponer que tape al propio elemento por la característica de que siempre se dibuja sobre las cajas. Observa lo que ocurre con el siguiente código y su resultado según el navegador con el que lo visualices.

```
.el {
    border: 2px solid #000;
    outline: 12px solid rgba(255,0,0,.5);
    outline-offset: -12px;
}
```

**Nota:** Tener en cuenta que para quitar el border de un input cuando utilizamos una pseudoclase :focus ó :placeholder-shown, necesitamos poner el outline: 0 para poder visualizar el border. ej:

```
input:not(:placeholder-shown){
    background-color: azure;
    border-color: red;
    outline: 0;
}
```

## 4.4 BOX-SIZING

La **box-sizing** propiedad CSS nos permite incluir el relleno y el borde en el ancho y alto total de un elemento.

Por defecto, el ancho y el alto de un elemento se calcula así:

- ancho + relleno + borde = ancho real de un elemento
- alto + relleno + borde = altura real de un elemento

Esto significa: cuando establece el ancho / alto de un elemento, el elemento a menudo parece más grande de lo que ha establecido (porque el borde y el relleno del elemento se agregan al ancho / alto especificado del elemento).

La siguiente ilustración muestra dos elementos <div> con el mismo ancho y alto especificados:

Este div es más pequeño (el ancho es de 300 px y la altura es de 100 px).

Este div es más grande (el ancho también es de 300 px y la altura es de 100 px).

Los dos elementos <div> anteriores terminan con diferentes tamaños en el resultado (porque div2 tiene un relleno especificado):

```
.div1 {
    width: 300px;
    height: 100px;
    border: 1px solid blue;
}

.div2 {
    width: 300px;
    height: 100px;
    padding: 50px;
    border: 1px solid red;
}
```

La **box-sizing** propiedad nos permite incluir el relleno y el borde en el ancho y alto total de un elemento.

Si establece **box-sizing: border-box;** en un elemento, el relleno y el borde se incluyen en el ancho y la altura: ¡Ambos divs son del mismo tamaño ahora!

```
.div1 {
    width: 300px;
    height: 100px;
    border: 1px solid blue;
    box-sizing: border-box;
}

.div2 {
    width: 300px;
    height: 100px;
    padding: 50px;
    border: 1px solid red;
    box-sizing: border-box;
}
```

## 4.5 Espacio adyacente: La propiedad shape-outside

Ya sabemos que el texto situado al lado de una caja flotante adapta su contenido para que fluya alrededor del elemento posicionado.

La propiedad **shape-outside** que define cómo se ajusta el texto alrededor del elemento flotante exige algunas condiciones básicas:

- Que el elemento sea flotante ( `elemento{float:left;}` - por ejemplo ).
- Que tenga dimensiones claramente definidas ( `width, height` ).

```
elemento{
    width: 10em;
    height: 10em;
    float: left;
    -webkit-shape-outside: circle();
    shape-outside: circle();
}
```

La propiedad `shape-outside` puede tomar una de estas valores: **circle()**, **ellipse()**, **polygon()**, **inset()** y **url()**.

### **circle()**:

Es una función utilizada para especificar una forma geométrica básica: un círculo.

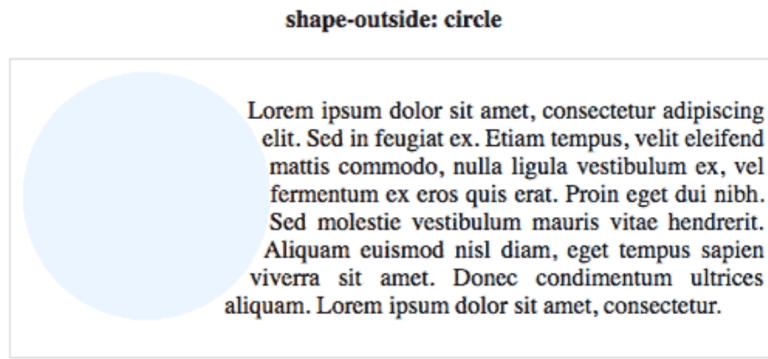
```
elemento{
    shape-outside: circle(radio at posición);
}
```

El primer parámetro representa el radio del círculo y es opcional. El valor por defecto es `closest-side` o sea la distancia hasta el lado más cercano.

El segundo parámetro representa la posición del centro (x,y) y es también opcional. Si no especificamos la posición, el CSS considera que el centro del círculo se encuentra en el centro del elemento.

Para especificar el valor del radio o las coordenadas del centro podemos utilizar palabras clave ( `closest-side` o `farthest-side` ) unidades de longitud ( `px, em` etc... ) o porcentajes.

```
shape-outside: circle(closest-side at 50% 50%); // palabras clave
shape-outside: circle(5em at 5em 5em); // unidades de longitud
shape-outside: circle(50% at 50% 50%); // porcentajes
shape-outside: circle(); // los parámetros son opcionales
shape-outside: circle()
```



Ejemplo:

```
.forma-circulo{
  width: 10em;
  height: 10em;
  float: left;
  opacity: .25;
  background-color: #abcdef;
  shape-outside: circle(50%);
  clip-path: circle(); /* crear la forma círculo, sería equivalente
  a poner border-radius:50%; */
}
```

## url():

También podemos utilizar una imagen con transparencias y dejar que el texto se ajuste alrededor de esta.

En este caso utilizamos la ya conocida función url() para referenciar la imagen.

```
.elemento {
  shape-outside: url('imagen.png');
}
```

Para crear un margen alrededor de la imagen utilizar la propiedad **shape-image-threshold**, esta propiedad define el umbral del canal alfa utilizado para extraer una forma CSS aplicada a un elemento mediante una imagen. Cuando se utiliza una imagen, debe ser compatible con CORS (El Intercambio de Recursos de Origen Cruzado), se genera una petición CORS cuando se solicita un recurso desde un dominio distinto, un protocolo o un puerto diferente al del documento que lo generó.

Por ello, deberemos probar este ejemplo con un servidor local (Xampp).

```
.forma {
  background: url("imagen.png")no-repeat;
  float: right;
  height: 350px;
  justify-content: center;
  shape-outside: url("imagen.png");
  width: 350px;
```

```
    shape-image-threshold: 0.15;
}
```

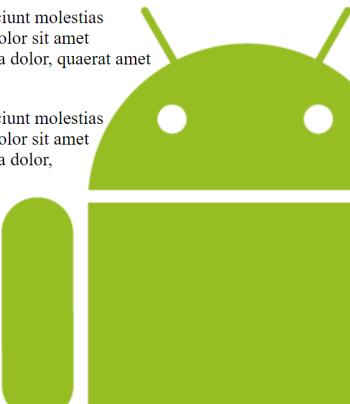
esse vel? Dolorem cupiditate sunt dolor nesciunt molestias , id magnam praesentium sit. Lorem ipsum dolor sit amet r eaque fugit commodi doloremque aut soluta dolor, quaerat amet nmodi reiciendis.

esse vel? Dolorem cupiditate sunt dolor nesciunt molestias , id magnam praesentium sit. Lorem ipsum dolor sit amet r eaque fugit commodi doloremque aut soluta dolor, ssimos ad commodi reiciendis.

esse vel? Dolorem cupiditate sunt dolor a error, explicabo, id magnam praesentium

esse vel? Dolorem cupiditate sunt dolor a error, explicabo, id magnam praesentium

esse vel? Dolorem cupiditate sunt dolor a error, explicabo, id magnam praesentium



## Capítulo 5. Técnicas avanzadas

### 5.1. Imágenes embebidas

El tiempo de carga de una página media depende en un 80% de la parte del cliente y en un 20% de la parte del servidor. Los navegadores de los usuarios dedican la mayor parte del tiempo a descargar imágenes, archivos JavaScript, hojas de estilos CSS y otros recursos externos.

Por este motivo, las mejoras en la parte del cliente generan muchos más beneficios que las mejoras en la parte del servidor. De todos los elementos de la parte del cliente, las imágenes son normalmente las que más penalizan el rendimiento. Además del peso de cada imagen, el rendimiento se resiente porque cada imagen requiere realizar una petición al servidor. Como los navegadores tienen un límite de conexiones simultáneas con el servidor (entre 2 y 8), la descarga de una página con muchas imágenes se bloquea continuamente.

Como ya se explicó en las secciones anteriores, la solución para mejorar el rendimiento de las imágenes consiste en combinarlas en una imagen grande llamada *sprites CSS* y utilizar la propiedad `background-image` en cada elemento HTML.

Además de los *sprites CSS* existe otra técnica que permite *embeber* o incluir las imágenes en la propia página HTML u hoja de estilos CSS. La técnica se suele denominar "*imágenes embebidas*" o "*imágenes en línea*".

Normalmente, en las hojas de estilos y páginas HTML sólo se indica la URL de la imagen que debe descargar el navegador. En la técnica de las imágenes embebidas no se indica la URL, sino que **se incluyen directamente los bytes de la imagen**, para que el navegador pueda mostrarla de forma inmediata.

Los datos de la imagen se incluyen mediante un *esquema* especial llamado **data:**, de la misma forma que las URL se indican mediante el esquema **http:**, las direcciones de correo electrónico mediante el esquema **mailto:**, etc. El esquema **data:** se define en el estándar [RFC 2397](http://www.ietf.org/rfc/rfc2397.txt) (<http://www.ietf.org/rfc/rfc2397.txt>) y su **sintaxis** es la siguiente:

**data:[<mediatype>][;base64],<data>**

El atributo **<mediatype>** corresponde al tipo de contenido de los bytes que se incluyen. Los tipos de contenido están estandarizados y los que utilizan habitualmente las imágenes son: **image/gif, image/jpeg y image/png**. Si no se indica el tipo de forma explícita, el navegador supone que es **text/plain**, es decir, texto normal y corriente.

El valor **base64** es opcional e indica que los datos de la imagen se han codificado según el formato **base64**. Si no se indica este valor, el navegador supone que los bytes de la imagen no están codificados.

A continuación se muestra el ejemplo de una imagen HTML (**<img>**) que no se indica mediante una URL sino que se incluyen sus bytes directamente en la página:

```
<!-- Imagen externa que el navegador debe descargar desde el servidor -->
![Icono de un libro](/imagenes/icono_libro.png)
/>

<!-- Imagen embebida que el navegador puede mostrar directamente porque ya dispone
de sus bytes -->
![Icono de un libro](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAQCAYAAQF8/9hAAAABGdBTUEAAK/INwWK6QAAABl0RVh0U29mdHd
hcmUAQWRvYmUgSW1hZ2VSZWFkeXHJZTwAAAHjSURBVDjLdZO/a1VBEMZ/5+TemxAfFUUskqAoSOJNp4KC
4As0PoGFIHY+gA+jijXaKIiChbETtBYLUbSMRf6Aydnfmfs9kRjvhDhGVh2fvN9uz0NJK7fe7Ai6algA
3FZCAmQqEF/dnihpK1v7x7dPw0woF64Izg3X15s1n9uIe01QYUFCTjc+sVuEqHBKfpVAXB1vLzQXFtdYP
HKGFUCoahVo1Y/fnie+bkBV27c5R8A0pHxyhKvPn5hY2MHRQAQeyokFGJze4cuZfav3gLNYDTg7Pk1zpw
4ijtIQYRwFx6BhdjtCk+erU0CCPfg+/o2o3ZI13WU1LGo58YNg+GIY4dmCwkJCAAgPzAspJW5ePFP1V3VI
4uHbz5S5IQfy/yooHngxzFser30iFcNcuAVGw3A0I1t91IkAsyCXQg5Q00szHEIrogkiguwN2acCoJhn
ZGKYx4Ujz5W0A2YD1BMU+BBSYVUVNpxkXuIuWgbsoxTHrG3UHIFWIhsgXtQQpTizNBS5jXZQkhkcywZqQ
Q1Ajdrwiml7wU5xWLal1AvZa8WIjALzIRZ7YVWDW5CiIj48Z8F2pYL11ZR0+AuZEX0UX035mxIkLq0dhD
w5vXL97fr503rfwQHjhPx4uuH57f2AL8BfPrV1rs6xwsAAAAASUVORK5CYII=)

```

El atributo **src** de la imagen del ejemplo anterior utiliza el esquema **data:** para incluir en la página los bytes de la imagen. El valor **image/png** indica que los datos corresponden a

una imagen en formato PNG. El valor `base64` indica que los datos incluidos están codificados según el formato `base64`.

Aunque el ejemplo anterior funciona correctamente, como todos los datos se incluyen en el propio código HTML, el navegador no puede hacer uso de la caché para reutilizarlos posteriormente. El resultado es que el número de peticiones HTTP se reduce drásticamente, pero aumenta significativamente el tamaño de todas las páginas HTML.

El siguiente paso consiste en utilizar las imágenes embebidas en las hojas de estilos CSS, de forma que se mantengan las ventajas del ejemplo anterior y se solucionen todas sus desventajas.

Para embeber las imágenes en CSS se sigue la misma estrategia que en HTML, ya que sólo es necesario sustituir la URL por el esquema `data:`, tal y como muestra el siguiente ejemplo:

```
/* Imagen externa que el navegador debe descargar desde el servidor */

ul#menu li { background: #FFF no-repeat center center
url("/imagenes/icono_libro.png"); }

/* Imagen embebida que el navegador puede mostrar directamente porque ya dispone
de sus bytes */

ul#menu li { background: #FFF no-repeat center center
url("data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAf8/9hAAAABGdBTUEAAK/INwWK6
QAAAB10RVh0U29mdHdhcmUAQ_W
RvYmUgSW1hZ2VSZWFKeXHJZTwAAAHjSURBVDjLdZ0/a1VBEMZ/5+TemxAbFUUskqAoSOJNp4KC4Asop
oGFIHY+gA+jiJXaKIIChbETtBYLUbSMRF6Ayndmfks9kRjvHdhGVh2fvN9uzONJK7fe7Ai6algA3FZ
CAmQqEF/dnihpK1v7x7dPw0woF64Izg3X15s1n9uIe01QYUFCTjc+sVuEqHBKfpVAXB1vLzQXFtdYPH
kGFUCoahVo1Y/fnie+bkBV27c5R8A0pHxyhKvPh5hY2MHRQAQeyokFGJze4cuZfav3gLNYDTg7Pk1zp
w4ijtIQYRwFx6BhdjtCk+erU0CCPfg+/o2o3ZI13WU1LGo58YMg+GIY4dmCwkJAgPzAspJW5ePFP1V
3VI4uHbz5S5IQfy/yooHngxzFser30iFcNcuAVGw3A0Ilt91IkAsyCXQg5Q00szHEIrogkiguwN2acC
oJhjnZGKYx4Ujz5W0A2YD1BMU+BBSYVUvNpxkXuIuWgbsoxTThrG3UHIFWIhsgXtQQpTizNBS5jXZQkh

kcywZqQQ1AjdRwiml7wU5xwLaL1AvZa8WIjALzIRZ7YVWDW5CiIj48Z8F2pYL11ZR0+AuzEX0UX035m
xIkLq0dhDw5vXL97fr503rfwQHjhPx4uuH57f2AL8BfPrV1rs6xwsAAAAASUVORK5CYII="); }
```

La ventaja de utilizar las imágenes embebidas en CSS es que sólo aumenta el tamaño de las hojas de estilos y no el de todas las páginas HTML del sitio. Además, los navegadores guardan en su caché las hojas de estilos completas, por lo que el aumento en su tamaño no penaliza en exceso el rendimiento global de la descarga de páginas.

Puedes utilizar alguna de las herramientas online que codifican directamente los contenidos del archivo indicado:

- [Base64 encoder/decoder](http://www.motobit.com/util/base64-decoder-encoder.asp) (<http://www.motobit.com/util/base64-decoder-encoder.asp>)
- [Binary File to Base64 Encoder / Translator](http://www.greywyvern.com/code/php/binary2base64) (<http://www.greywyvern.com/code/php/binary2base64>)

Las principales ventajas de la técnica de las imágenes embebidas son las siguientes:

- Reduce drásticamente el número de peticiones HTTP, lo que mejora notablemente el rendimiento.
- Permite guardar una página HTML completa en un único archivo HTML (embebiendo todas sus imágenes en su hoja de estilos o en el propio código HTML).
- Mejora el rendimiento de las peticiones HTTPS en las que las imágenes no se guardan en la caché.

Por contra, sus desventajas son considerables:

- El esquema `data:`: sólo funciona en los navegadores modernos que se preocupan de los estándares (Firefox, Safari y Opera). Internet Explorer 6 y 7 no son capaces de procesar el esquema `data:`. Internet Explorer 8 asegura que permitirá utilizar `data:`, pero solamente para embeber imágenes en CSS.
- El proceso completo es lento y poco flexible, ya que es necesario codificar las imágenes en base64 y codificarlas cada vez que se quieren modificar.
- Las imágenes embebidas aumentan considerablemente el tamaño de la hoja de estilos CSS. Además, la codificación base64 también aumenta mucho el tamaño de los datos de la imagen respecto a sus bytes originales.
- Los navegadores limitan el tamaño máximo de los datos que se pueden embeber mediante `data:`. Algunos navegadores como Opera permiten unos 4.000 bytes de datos, mientras que el navegador Firefox permite hasta 100.000 bytes por cada `data:`.

## 5.2. Mapas de imagen

Los mapas de imagen CSS no suelen utilizarse para definir zonas pinchables dentro de una imagen, sino que se emplean para mostrar información adicional y comentarios sobre las diferentes zonas de una imagen. El sitio de fotografía [Flickr](http://www.flickr.com/) (<http://www.flickr.com/>) utiliza los mapas de imagen para mostrar notas y comentarios de los usuarios. Otros sitios web como [Facebook](http://www.facebook.com/) (<http://www.facebook.com/>) utilizan los mapas de imagen para que los

usuarios etiqueten las fotografías indicando el nombre de las personas que aparecen en cada una.

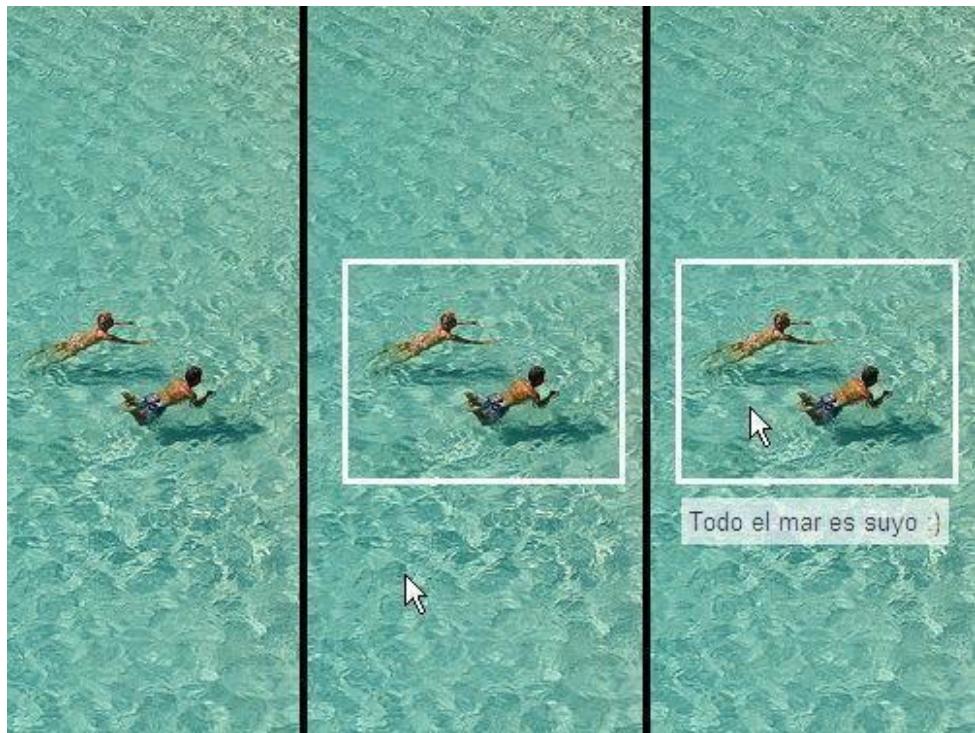
A continuación se explican los pasos necesarios para crear un mapa de imagen exclusivamente con CSS similar a los de Flickr y Facebook.

En primer lugar, selecciona la imagen en la que se van a mostrar las notas. En este ejemplo se utiliza una imagen de la fotógrafa visualpanic (<http://www.flickr.com/photos/visualpanic/>) que se puede utilizar libremente y que está disponible en Flickr (<http://www.flickr.com/photos/visualpanic/233508614/>):



Figura 5.1. Imagen original en la que se va a mostrar el mapa de imagen

El funcionamiento del mapa de imagen terminado es el que muestra la siguiente secuencia de imágenes:



El código HTML del mapa de imagen creado con CSS varía mucho en función de la solución utilizada. Aunque algunas soluciones crean varios `<div>` y tablas por cada nota/comentario, en este ejemplo se simplifica al máximo el código HTML y sólo se utiliza un elemento `<div>` y una lista `<ul>`:

```

<div class="mapa_imagen">
    

    <ul class="notas">
        <li id="nota1"><p>Todo el mar es suyo :)</p></li>
        <li id="nota2"><p>¡Me encanta este color azul!</p></li>
        <li id="nota3"><p>Dan ganas de
            tirarse...</p></li>
    </ul>

</div>

```

El elemento `<div class="mapa_imagen">` encierra todos los elementos que forman el mapa de imagen (la imagen original y las notas/comentarios). Los comentarios se incluyen mediante una lista no ordenada `<ul>`, en la que cada elemento `<li>` es un comentario.

El elemento `<div>` y la lista `<ul>` deben utilizar el atributo `class` y no `id` porque en una misma página puede haber varios mapas de imagen. Por su parte, los elementos `<li>` de cada comentario deben utilizar atributos `id`, ya que cada comentario se muestra en una posición única y tiene unas dimensiones únicas de anchura y altura.

La clave de los mapas de imagen CSS consiste en posicionar cada <li> de forma absoluta respecto de la imagen y asignarles una anchura/altura adecuadas. Posteriormente, se emplea la pseudo-clase :hover para mostrar/ocultar elementos cuando el usuario pasa el ratón por encima.

- 1) Posicionar de forma absoluta cada nota:

```
div.mapa_imagen {
    position: relative;
}
ul.notas li {
    position: absolute;
}
```

- 2) Aplicar los estilos básicos a las notas (borde blanco y sin adornos de lista):

```
ul.notas li {
    border: medium solid #FFF;
    list-style: none;
}
```

- 3) Ocultar por defecto las notas y mostrarlas cuando se pasa el ratón por encima de la imagen:

```
ul.notas li {
    display: none;
}
div.mapa_imagen:hover ul.notas li{
    display: block;
}
```

- 4) Aplicar los estilos básicos al texto de las notas y posicionarlo debajo de cada nota:

```
ul.notas li p {
    position: absolute;
    top: 100%;
    background: #FFF;
    opacity: 0.65;
    margin: 10px 0 0 0;
    padding: 0.3em;
}
```

- 5) Ocultar por defecto el texto de las notas y mostrarlo cuando se pasa el ratón por encima de la nota:

```
ul.notas li p {
    display: none;
}
ul.notas li:hover p{
    display: block;
}
```

- 6) Establecer la posición y dimensiones de cada nota:

```

ul.notas li#nota1 {
    width: 140px;
    height: 110px;
    top: 130px;
    left: 345px;
}
ul.notas li#nota2 {
    width: 30px;
    height: 200px;
    top: 10px;
    left: 10px;
}
ul.notas li#nota3 {

    width: 60px;
    height: 60px;
    top: 200px;
    left: 150px;
}

```

El código CSS completo del mapa de imagen es el siguiente:

```

div.mapa_imagen {
    position: relative;
}
ul.notas li {
    list-style: none;
    display: none;
    position: absolute;
    border: medium solid white;
    background: url("esta_imagen_no_existe");
}

div.mapa_imagen:hover ul.notas li {
    display: block;
}
ul.notas li p {
    margin: 10px 0 0 0;
    padding: .3em;
    display: none;
    background: #FFF;
    opacity: 0.65;
    position: absolute;
    top: 100%;
}
ul.notas li:hover p {
    display: block;
}
ul.notas li#nota1 {

```

```

width: 140px;
height: 110px;
top: 130px;
left: 345px;
}
ul.notas li#nota2 {
width: 30px;
height: 200px;
top: 10px;
left: 10px;
}
ul.notas li#nota3 {
width: 60px;
height: 60px;
top: 200px;
left: 150px;
}
}

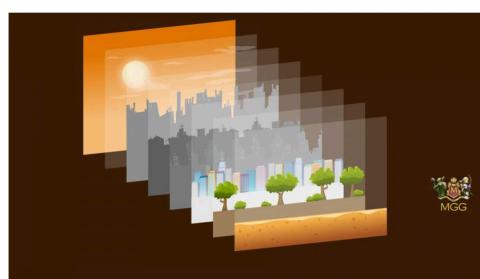
```



## 5.3 Control del Scroll

### 5.3.1 Parallax

El desplazamiento de Parallax es una tendencia del sitio web donde el contenido de fondo (es decir, una imagen) se mueve a una velocidad diferente que el contenido de primer plano durante el desplazamiento.



## Cómo crear un efecto de desplazamiento de paralaje

Use un elemento contenedor y agregue una imagen de fondo al contenedor con una altura específica. Luego use **background-attachment: fixed** para crear el efecto de paralaje real. Las otras propiedades de fondo se utilizan para centrar y escalar la imagen perfectamente:

```
<style>
.parallax{
    background-image:url("img_parallax.jpg");
    height:500px;
    background-attachment:fixed;
    background-position:center;
    background-repeat:no-repeat;
    background-size:cover;
}
</style>

<div class="parallax"></div>
```

Algunos dispositivos móviles tienen un problema con background-attachment: fixed, la solución será desactivarlo para dispositivos móviles:

```
@media only screen and (max-device-width: 1366px){
.parallax{ background-attachment:scroll; }
}
```

### 5.3.2 Scroll Snap

Con la propiedad **scroll-snap-type** y **scroll-snap-align** podemos controlar el funcionamiento del desplazamiento del scroll.

El Scroll Snap nos va a permitir que, al hacer scroll, ya sea con el ratón desde un ordenador o portátil, o desde la pantalla de un smartphone, este desplazamiento se detenga en un punto en concreto de nuestra web, normalmente el comienzo de una nueva sección.

Esto es muy conveniente en sitios web que tengan una disposición en diferentes bloques horizontales, y queremos que nuestros usuarios salten de uno a otro de la forma más directa posible. Un buen ejemplo lo tenemos en la página de [Tesla](#).

Si navegamos por ella, vemos cómo saltamos directamente de una imagen a otra de forma directa.

```
<div class="snap-container">
  <div class="snap-section">
    <h3>Section 01</h3>
    <p> Lorem ipsum...</p>
  </div>
  <div class="snap-section">
    <h3>Section 02</h3>
    <p> Lorem ipsum...</p>
  </div>
  <div class="snap-section">
    <h3>Section 03</h3>
    <p> Lorem ipsum...</p>
  </div>
</div>
```

CSS que le daremos al div padre:

```
.snap-container {
  height: 100vh;
  overflow: auto;
  scroll-snap-type: y mandatory; }
```

Con el valor **y** estamos diciendo que el scroll que estamos marcando es el vertical. Y con el valor **mandatory**, que la página se mueva hasta el principio de la sección siguiente o anterior, según hagamos scroll hacia abajo o hacia arriba. En vez de el valor **mandatory** podemos usar **proximity**, que hará que no nos desplazamos hasta la sección siguiente o anterior hasta que no estemos muy próximos al principio de dicha sección.

```
.snap-section {
  scroll-snap-align: start;
  min-height: 60vh; }
```

Con **scroll-snap-align** designaremos en qué punto que el desplazamiento se pare. En nuestro caso, le hemos dado el valor **start**, por lo que se parará justo al inicio de la sección. Aunque también podemos usar los valores **center** para quedarnos a mitad de la sección, o **end** para quedarnos al final. La altura de la sección la hemos marcado en vh o altura del viewport.

También podemos hacerlo en horizontal. Esto nos puede venir si la exploración de nuestra web se hace en esta dirección, o si queremos hacer un slider. Para conseguirlo, solo hay que cambiar un valor en el contenedor padre: **scroll-snap-type: x proximity;**

Referencia: [MDN Web Doc](#)

### 5.3.3 scroll-behavior

Si queremos hacer el mismo efecto scroll-snap pero con enlaces, podemos usar **scroll-behavior**.

Por defecto no se hereda y tendremos que aplicarlo al elemento contenedor que tendrá el scroll. En el caso del viewport del navegador, deberemos aplicarlo al elemento raíz, html.

```
html {
  scroll-behavior: smooth;
}
```

### 5.3.4 Control del scroll por JavaScript.

Podemos utilizar Javascript para controlar la posición del scroll. Cuando se produce el movimiento de la barra de desplazamiento, se produce un evento **onscroll** que podemos utilizar con el manejador de eventos. Ejemplo web Pico4 [Enlace](#).

Por otro lado con **window.scrollY** podemos saber la posición exacta del desplazamiento del scroll.

```
window.onscroll = function () {
  let imagen = document.getElementById("imagen");
  imagen.style.width= parseInt(window.scrollY)/4 + "px" ;
};
```

### 5.3.5 Parallax con el movimiento del ratón por JavaScript.

Para manejar los elementos en el Document Object Model (DOM) con el movimiento del ratón a través de JavaScript, puedes utilizar varios eventos del ratón y métodos del DOM.

Seleccionar el Elemento del DOM:

Primero, debes seleccionar el elemento del DOM que quieras manipular. Esto se puede hacer usando métodos como `document.getElementById()`, `document.querySelector()`, etc.

```
var elemento = document.getElementById('miElemento');
```

#### Agregar Eventos de Ratón:

Luego, necesitas agregar manejadores de eventos al elemento para responder a acciones del ratón. Los eventos comunes del ratón incluyen **mousedown**, **mousemove**, y **mouseup**.

#### Implementar la Lógica del Movimiento:

Dentro de los manejadores de eventos, puedes implementar la lógica para mover el elemento. Esto puede implicar cambiar la posición del elemento en función de las coordenadas del ratón. Ejemplo: [Link](#)

Utilizaremos las variables `clientX` / `clientY` para saber la posición del puntero en el ViewPort.

Utilizaremos las variables `pageX` / `pageY` para saber la posición del puntero en toda la página.

```
document.addEventListener('mousemove', function(e) {
    elemento.style.left = e.clientX + 'px';
    elemento.style.top = e.clientY + 'px';
});
```

## 5.4 Variables CSS

Las variables CSS, llamadas *custom properties*, están actualmente soportadas en todos los navegadores.

Para declarar una variable CSS utilizamos esta sintaxis:

```
elemento{ --variable: valor; }
```

por ejemplo:

```
elemento{ --color: #abcdef; }
```

Los dos guiones ( -- ) que aparecen delante, dicen al CSS que esta propiedad es una variable, y podemos pensar en ello como una propiedad CSS con un prefijo vacío ( -webkit-propiedad, -moz-propiedad, --propiedad ).

Usamos las variables CSS como valor de una propiedad CSS.

Para utilizar una variable CSS ya definida empleamos la función `var()` y esta sintaxis:

```
elemento{ propiedad: var(--variable); }
```

Por ejemplo:

```
body * { --bg: #f49c14 }
.test {
  width: 100px;
  height: 100px;
  margin: .25em auto;
  background-color: var(--bg);
}
```

Es muy habitual declarar las variables CSS en el pseudo-clase `:root` para que estén disponibles de forma global y tenga más prioridad a la hora de aplicar estilos.

```
:root { --color-dark: #333; }
.dark-text {
  color: var(--color-dark); }
```

## 5.5 La función calc()

La función calc() de CSS3 nos permite realizar operaciones matemáticas sencillas como: sumar (+), restar (-), multiplicar (\*) o dividir (/) y puede ser utilizada siempre que se trate de valores numéricos como: longitud, frecuencia, duración, ángulo o número.

```
section { height: calc(100% - 2*10px); }
```

La función calc() hace posible que podamos sumar, restar, o lo que sea, pixeles ( px ) con em, porcentajes ( % ) con pixeles ( px ), ya que una de las habilidades de calc() es mezclar unidades.

En el siguiente ejemplo queremos construir dos cajas ( <div class="calcTest"> ) cuyo tamaño varía en función del tamaño de la caja contenedora ( <div id="contenedorCalc"> )

Para ser más exactos: la altura de .calcTest es igual a la altura del contenedor #contenedorCalc ( 100% ) de la cual restamos el margen ( 2\*10px ), el padding ( 2\*10px ), y el borde ( 2px ).

```
height: calc(100% - 2*10px - 2*20px - 2px);
```

Calculamos la anchura de las cajas .calcTest: la mitad del ancho del contenedor #contenedorCalc ( 50% ) menos el margen, el padding y el borde.

```
width: calc(50% - 2*10px - 2*20px - 2px);
```

Ejemplo:

```
div[id^="contenedorCalc"]{
    width:80%;
    height:200px;
    border:1px solid #d9d9d9;
    margin:10px auto;
    padding:10px;
}

.calcTest{
    float:left;
    border:1px solid #d9d9d9;
    margin:10px;
```

```
padding:20px;  
background-color:#efefef;  
height: calc(100% - 2*10px - 2*20px - 2px);  
width: calc(50% - 2*10px - 2*20px - 2px);  
}
```

Otras funciones: [enlace](#)

attr(), blur(), brightness(), calc(), clamp(), circle(), conic-gradient(), contrast()  
counter(), cubic-bezier(), drop-shadow(), ellipse(), grayscale(), hsl(), hsla(), hue-rotate(),  
inset(), invert(), linear-gradient(), matrix(), matrix3d(), max(), min(), opacity(), polygon()  
radial-gradient(), repeating-linear-gradient(), repeating-radial-gradient(), rgb(), rgba(), rotate()  
, rotate3d(), rotateX(), rotateY(), rotateZ(), saturate(), sepia(), scale(),  
scale3d(), scaleX(), scaleY(), scaleZ(), skew(ax,ay), skewX(ax), skewY(ay), translate(), translate3d()  
, translateX(), translateY(), translateZ(), url().

# Capítulo 6. Transformación y Animación

## 6.1.Transform.

La especificación oficial y el estado actual de desarrollo del módulo Transforms en CSS3 puede consultarse en <http://www.w3.org/TR/css3-transforms/>.

Al modificar el espacio de coordenadas, las transformaciones CSS permite cambiar la posición del contenido afectado sin interrumpir el flujo normal del resto de cajas. Se llevan a cabo mediante un conjunto de propiedades CSS que permiten aplicar transformaciones lineales afines a los elementos HTML. Estas transformaciones incluyen la rotación, inclinación, escala y traslación tanto en el plano como en un espacio tridimensional.

### **Propiedades principales**

Se utilizan dos **propiedades** principales para definir las transformaciones CSS: *transform* y *transform-origin*.

- **transform-origin:** especifica la posición de origen de la transformación. Por defecto se encuentra en el centro del elemento (como si definiéramos el punto en 50% 50%), pero se puede definir cualquier otro punto. Es utilizado por varias transformaciones, como rotación, escala o inclinación, que necesitan un punto inicial como parámetro.
- **transform:** especifica las transformaciones a aplicar al elemento. Se trata de una lista de funciones de transformación separadas por espacios, que se aplican una detrás de otra.

### **TRANSFORM**

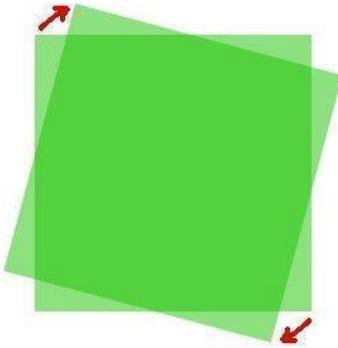
El atributo **transform** nos permite, como su propio nombre indica, transformar un elemento. Su sintaxis es la siguiente:

```
transform: tipo(cantidad);
```

El valor tipo puede tomar cuatro valores y cada uno de ellos realiza una función diferente:

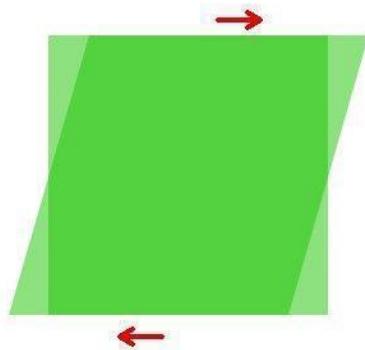
- **Rotate:** Nos permite girar los elementos un número de grados. La sintaxis es:

```
transform:rotate(25deg);
```

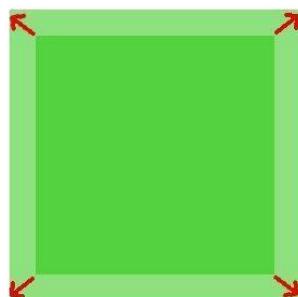


El valor que demos al giro se aplicará en sentido horario

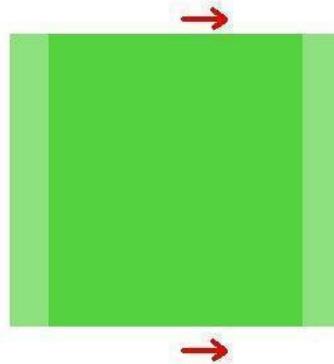
- **Skew:** Podemos inclinar un elemento tanto en coordenadas X como Y. El valor se expresa en grados y la sintaxis es la siguiente: /\*transform: skew(gradosX, gradosY);\*/ transform: skew(15deg, 3deg);



- **Scale:** Con este tipo podremos escalar nuestro elemento tanto en X como en Y en una cantidad expresada en tantos por uno: /\*transform: scale(escalaX,escalaY);\*/ transform: scale(1.5,0.6);



- **Translate:** Podemos desplazar el elemento tanto en X como en Y.  
`/*transform: translate(desplazamientoX, desplazamientoY);*/  
transform:translate(12px, 19px);`



**NOTA:** Se pueden aplicar diferentes transformaciones a un mismo elemento simplemente inscribiéndose de manera consecutiva: `transform: scale(1.6) skew(10deg) translate(5px) rotate(12deg);`

### 6.1.2 Transform-origin.

Especifica la posición de origen de la transformación, puede tomar los parámetros bottom, right ,top y left.

```
div { transform: rotate(90deg);

      transform-origin: bottom left;

}

<div>

  <iframe src="http://www.google.com/" width="500" height="600"></iframe>
</div>
```

## 6.2 Propiedades 3D.

La realización de transformaciones CSS en el espacio es algo más complejo. Hay que empezar configurando el espacio 3D, dándole un punto de vista. Después, hay que configurar cómo se comportan los elementos 2D en ese espacio tridimensional.

Las siguientes propiedades, añaden control adicional a las transformaciones, permitiendo realizar **transformaciones 3D**:

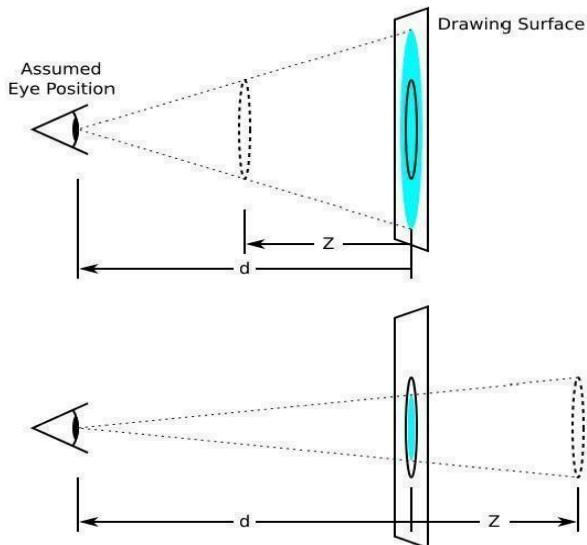
- **perspective:** permite cambiar la perspectiva de los elementos y transmitir la sensación de encontrarse en un entorno en tres dimensiones.

- **perspective-origin:** especifica la posición de origen de la perspectiva.
- **transform-style:** permite a los elementos transformados en 3D y a sus descendientes también transformados en 3D, compartir un espacio 3D común.

### 6.2.1 La creación de un punto de vista (o perspectiva).

El primer elemento a configurar es la perspectiva. La perspectiva es lo que da la impresión de tres dimensiones. Cuanto más lejos del espectador se encuentran los elementos, más pequeños se nos muestran.

Cuán rápido estos elementos reducen su tamaño es definido por la propiedad `perspective`. Cuanto más pequeño es su valor, más profunda es la perspectiva.



La perspectiva se podrá aplicar de dos maneras:

a) Mediante la propiedad **`transform: perspective (valor);`**

```
.box {
    background-color: red;
    transform: perspective( 600px ) rotateY( 45deg );
}
```

b) Mediante la propiedad **perspective**:

```
#blue { perspective: 600px; }

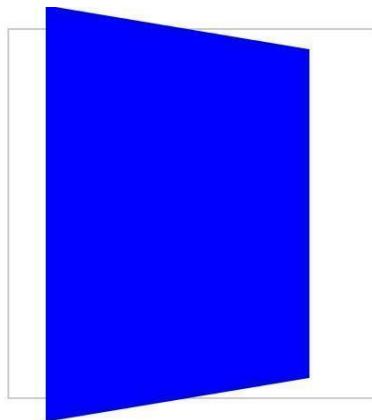
#blue .box {

    background-color: blue;

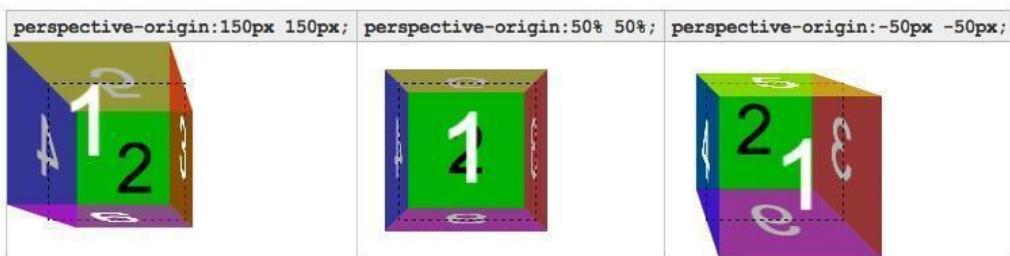
    transform: rotateY( 45deg );

}

}
```



El segundo elemento a configurar es la posición del espectador, con la propiedad **perspective-origin**. Por defecto, la perspectiva se centra en el espectador, lo cual no siempre es lo adecuado. Esta propiedad es equivalente a la propiedad **Transform-origin** en 2d. Una vez realizado esto, se puede trabajar sobre el elemento en el espacio 3D.



### 6.3.2 Transform-style

Esta es otra propiedad importante en el espacio 3D. Se necesitan dos valores: **preserve-3D o flat**. Cuando el valor de estilo de transformación es *preserve-3d* entonces le dice al navegador que el elemento de los hijos también se debe colocar en el espacio 3D. Por otro lado, cuando el valor de esta propiedad es *flat*, indica que los hijos están presentes en el plano del propio elemento. Ejemplo:

```
div { height: 150px;
```

```

width: 150px;

margin: 10px auto 50px; }

.contenedor {

border: 1px solid black;

-webkit-perspective: 500px; }

.transformar {

background-color: blue;

-webkit-transform-style: preserve-3d;

-webkit-transform: rotateY(50deg); }

.hijo {

-webkit-transform-origin: top left;

-webkit-transform: rotateX(40deg);

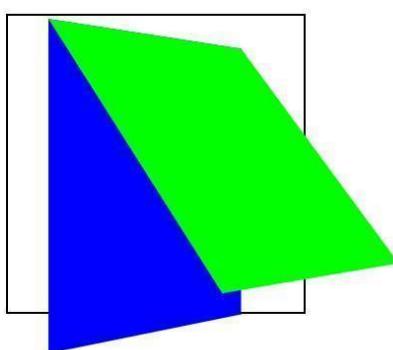
background-color: lime; }

<div class="contenedor">

<div class="transformar">

<div class="hijo"></div>
</div>
</div>

```



### 6.3.3 Funciones de transformación 3D

Funciones que se utilizan en la transformación 3D:

- **Translate3d** (<translation-value>, <translation-value>, <longitud>) : define una translación 3D. Lleva tres parámetros x, y, z valores. El valor Z especifica la translación en el eje Z.
- **TranslateZ** (<longitud>) : Funciona de manera similar a TranslateX () y TranslateY () para el eje Z.
- **Scale3d** (<número>, <número>, <número>) : Esta función hace el escalamiento en las tres dimensiones. Se necesitan tres parámetros como sx, sy y sz. Cada valor define la escala en la dirección correspondiente.
- **ScaleZ** (<número>) : Al igual que el translate () función, igual scaleZ () define la escala en una sola dirección, es decir en la dirección Z. También tenemos scaleX () y scaleY () funciones que también trabajan similar a scaleZ () , pero en sus respectivas direcciones.
- **Rotate3d** (<número>, <número>, <número>, <ángulo>) : Gira un elemento HTML en el ángulo especificado en el último parámetro de [tx, ty, tz] vector especificado por los primeros tres números, ejemplo: transform: rotate3d(1, 1, 1, 45deg);
- **RotateX** (<ángulo >) , **rotateY** (<angle>) y **RotateZ** (<ángulo >) tienen un único valor de ángulo para rotar en el eje correspondiente.

**Nota:** Rotar3D (1,0,0,30 grados) es igual a RotateX (30deg) , Rotar3D (0,1,0,30 grados) es igual a rotateY (30deg) y Rotar3D (0,0,1,30 grados) es igual a RotateZ (30deg).

## 6.4. Animaciones

### 6.4.1 Módulo Animations

Tradicionalmente cuando queríamos incluir cualquier tipo de animación en nuestras páginas webs, teníamos que recurrir a tecnologías como Flash o JavaScript. Si bien la potencia de los dos citados es descomunal en comparación, las nuevas funciones de animación de CSS3 abren una nueva puerta a nuestra imaginación, sin tener que depender de tecnologías extras.

### 6.4.1.1 Fotogramas claves de las animaciones

Un fotograma clave no es más que un punto destacado en el tiempo de nuestra animación. Cualquier animación consta al menos de dos fotogramas claves: el punto inicial y el punto final. Imaginad que nuestra animación es como una carretera:



El primer semáforo actuaría como fotograma clave inicial y el segundo como el final. Entre uno y otro se produciría nuestra animación, que no es más que el desplazamiento del coche hacia la derecha.

### 6.4.1.2 @keyframes

En CSS3 creamos animaciones completas mediante `@keyframes`, que son un conjunto de fotogramas clave. Su sintaxis es la siguiente:

```
@keyframes nombreAnimacion{
    puntoDelKeyframe{ atributosIniciales; }

    puntoDelKeyframe{ nuevosAtributos; }

    .....

    puntoDelKeyframe{ ultimosAtributos; }
}
```

Veamos que tendríamos que hacer para desplazar nuestro coche a la derecha:

```
@keyframes animacionCoche{

    /*Indicamos que salimos de la posición 0*/
    from{ left:0px; }

    /*Indicamos que al final la posición debe ser 350*/
    to{ left:350px; }
}
```

Podemos crear animaciones más complejas estableciendo fotogramas claves intermedios mediante porcentajes:

```
@keyframes animacionCoche{
    from{ left:0px;}
    /*Hasta el 65% de la reproducción sólo queremos que se desplace 10 pixels*/
    65%{ left:10px;}
    to{ left:350px;}
}
```

#### 6.4.1.3 Aplicando la animación a un elemento

Hemos creado nuestra animación. Para aplicarla sobre un elemento de nuestra página, deberemos acudir al mismo y agregarle el atributo de animación:

```
#coche{
    animation-name: animacionCoche;
    animation-duration: 3s;
    animation-iteration-count: 1;
    position:relative;
}
```

Hay tres nuevos atributos que no habíamos visto antes. Estos y otros tantos conforman los atributos destinados a la animación que describo a continuación:

- **Animation-name.** Indica a qué animación corresponde nuestro elemento. Es posible definir más de una animación usando comas como separador.
- **Animation-duration.** Define la duración en segundos de nuestra animación.
- **Animation-iteration-count.** Define cuántas veces se reproduce la animación. Podemos darle el valor *infinite* para que se reproduzca hasta el fin de los tiempos.
- **Animation-direction.** Por defecto nuestra animación se reproducirá hacia delante, como es lógico. Sin embargo si le damos el valor *alternate*, al reproducirse completamente la animación, esta volverá a reproducirse en sentido opuesto, es decir, como si se rebobinara.
- **Animation-delay.** Indica si se produce un retardo en el inicio de la animación.
- **Animation.** El shorthand de todos los anteriores:

```
name duration timing-function delay iteration-count direction;
```

- **Animation-timing-function.** Define cómo progresará la animación entre los keyframes mediante ecuaciones matemáticas.
- **Animation-fill-mode:** Esta propiedad especifica un estilo para el elemento cuando la animación no se está reproduciendo (antes de que comience, después de que termine o ambos). *forwards*, mantendrá los estilos una vez termine la animación.

#### 6.4.1.4 animation-timing-function

Este atributo sirve para aplicar un efecto suavizado a la animación.

Por defecto responde al valor *ease*, pero puede responder a diferentes valores:

- **Ease**
- **Linear**
- **Ease-in**
- **Ease-out**
- **Ease-in-out**

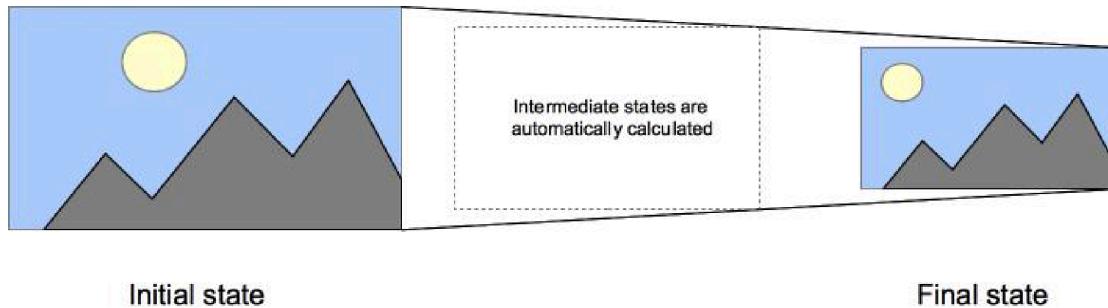
#### 6.4.2 Módulo Transition

La especificación oficial y el estado actual de desarrollo del módulo Transitions en CSS3 puede consultarse en <http://www.w3.org/TR/css3-transitions/>.

Lista de propiedades que se pueden animar: [enlace](#)

Las transiciones CSS, que forman parte del conjunto de especificaciones de CSS3, proporcionan una forma de controlar la velocidad de la animación al cambiar las propiedades CSS. Los cambios en las propiedades no tienen efecto inmediato, sino que se puede establecer un periodo de tiempo para que éstos se ejecuten. Por ejemplo, al cambiar el color de un elemento de blanco a negro, este cambio normalmente se ejecuta de manera inmediata. Sin embargo, con las transiciones CSS, los cambios se producen con intervalos de tiempo que siguen una curva de aceleración, y pueden ser personalizados.

Las animaciones que implican transición entre dos estados se llaman a menudo transiciones implícitas, ya que los estados intermedios entre el inicial y el final son implícitamente definidos por el navegador.



Las transiciones CSS permiten decidir qué propiedades animar (mediante su inclusión explícita), cuándo comenzará esta animación (estableciendo un retraso o delay), cuánto durará (estableciendo una duración), y cómo se ejecutará (definiendo una función de tiempo).

#### 6.4.2.1 Propiedades "animables"

Se puede definir qué propiedad debe ser animada y en qué manera. Esto permite la creación de transiciones complejas. Como la animación de algunas propiedades no tiene sentido, la lista de propiedades "animables" se limita a un conjunto finito.

- [Lista de propiedades animables.](#)

También el valor auto es un caso complejo. La especificación nos dice que no debemos animar desde y hacia dicho valor. Algunos agentes de usuario, como los basados en Gecko o WebKit, implementan este requisito, que al usar animaciones con auto nos puede dar lugar a resultados impredecibles, dependiendo del navegador y su versión; por lo que debemos evitarlo.

#### 6.4.2.2 Propiedad transition-property

El primer paso al crear una transición, es especificar la propiedad o propiedades sobre las que se va a aplicar los efectos de la transición. Podemos decidir que sean todas las propiedades, ninguna o un listado de ellas.

```
transition-property: none | all | [ <property> ] [, <property> ]*
transition-property: all;
transition-property: none;
transition-property: background-color;
transition-property: background-color, height, width;
```

Si se indica la palabra reservada all, todas las propiedades que sean capaces de ser animadas y para las que se ha definido un cambio, serán animadas. Si se especifica none, ninguna propiedad será animada.

#### 6.4.2.3 Propiedad transition-duration

La propiedad transition-duration acepta una lista separada por comas de tiempos, especificadas en segundos o milisegundos, que determinan cuánto tiempo tarda cada propiedad, en completar la transición.

```
transition-duration: <time> [, <time>]*  
transition-duration: 2s;  
transition-duration: 4000ms;  
transition-duration: 4000ms, 8000ms;
```

#### 6.4.2.4 Propiedad transition-timing-function

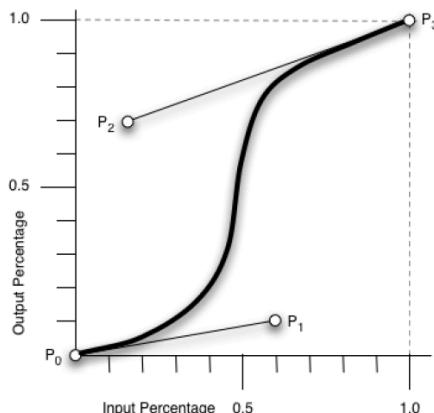
La propiedad transition-timing-function es utilizada para especificar el ritmo en el que se producen los cambios durante la transición. Esto puede realizarse de dos maneras: indicando el nombre de una función de tiempo (ease, linear, ease-in, ease-out o ease-in-out) o definiendo una función de tiempo personalizada (especificando cuatro coordenadas para definir una curva bezier).

```
transition-timing-function: <timing-function> [, <timing-function>]*  
transition-timing-function: ease;  
transition-timing-function: ease, linear;
```

Además de estas funciones de tiempo predefinidas, tenemos la posibilidad de declarar nuestra propia función de tiempo, utilizando una función cubic-bezier.

```
transition-timing-function: cubic-bezier(0.6, 0.1, 0.15, 0.8);
```

Los valores de la curva Bezier serán para los puntos P1 y P2, ( $x_1, y_1, x_2, y_2$ ) donde X debe de estar en el rango [0,1] e Y puede salir de este.



**Figura** Puntos de control en una curva bezier

Si no se especifica ninguna función de tiempo, por defecto se aplica ease.

Ver: [wikipedia.org](https://wikipedia.org) y [cubic-bezier.com](https://cubic-bezier.com)

#### 6.4.2.5 Propiedad transition-delay

El último paso para crear una transición, es especificar un retraso (opcional) en el inicio de la transición. Aquí también podemos especificar una lista de tiempos, en segundos o milisegundos, que determinan el inicio de la transición desde que esta se lanza. El valor por defecto es 0, esto es, se inicia de inmediato.

En este caso los valores negativos sí son aceptados. En este caso, la transición se iniciará tan pronto sea posible, pero dará la impresión que ya lleva tiempo ejecutándose.

```
transition-delay: <time> [, <time>]*
transition-delay: 5s;
transition-delay: 4000ms, 8000ms;
```

#### 6.4.2.6 Propiedad transition

Como de costumbre, disponemos de la propiedad shorthand que nos permite definir todas las propiedades de una sola vez.

```
transition: <transition> [, <transition>]*
<transition> = <transition-property> <transition-duration>
<transition-timing-function> <transition-delay>
transition: background-color 3s linear 1s;
transition: 4s ease-in-out;
transition: 5s;
```

El único valor requerido en esta propiedad es transition-duration.

#### 6.4.2.7 Listas de valores

Si la lista de valores de cualquier propiedad es más corta que otras, sus valores son repetidos hasta hacer que coincidan. Por ejemplo:

```
div {
    transition-property: opacity, left, top, height;
    transition-duration: 3s, 5s;
}
```

Esto es tratado como si fuese:

```
div {
    transition-property: opacity, left, top, height;
    transition-duration: 3s, 5s, 3s, 5s; }
```

De manera similar, si la lista de valores de cualquier propiedad es más larga que la `transition-property`, es acortado, por lo que si tienes este código CSS:

```
div {
    transition-property: opacity, left;
    transition-duration: 3s, 5s, 2s, 1s;
}
```

Se interpreta como

```
div {
    transition-property: opacity, left;
    transition-duration: 3s, 5s;
}
```

#### 6.4.2.8 Finalización de una transición

Existe un sólo evento que se dispara cuando las transiciones se completan. En todos los navegadores que cumplen el estándar, el evento es `transitionend`, en WebKit es `webkitTransitionEnd`. El evento **`transitionend`** ofrece dos propiedades:

- **`propertyName`**: string que indica el nombre de la propiedad CSS cuya transición está completada.
- **`elapsedTime`**: un float que indica el número de segundos que la transición ha estado ejecutándose en el momento en el que se dispara el evento. Este valor no está afectado por el valor de `transition-delay`.

Se puede utilizar el método `element.addEventListener()` para supervisar este evento:

```
el.addEventListener("transitionend", updateTransition, true);
```

#### 6.4.2.9 Transiciones y JavaScript

Las transiciones son una buena herramienta para crear una apariencia mucho más equilibrada sin tener que modificar la funcionalidad JavaScript. Por ejemplo:

```
<p>Hacer click para mover la bola</p>
<div id="mover"></div>
```

Utilizando JavaScript:

```
var f = document.getElementById('mover');
document.addEventListener('click',
```

```
function(ev){  
    f.style.left = (ev.clientX-25)+'px';  
    f.style.top = (ev.clientY-25)+'px';  
},false);
```

Con CSS simplemente es necesario añadir una transición al elemento:

```
p {  
    padding-left: 60px; }  
  
#bola {  
    border-radius: 50px;  
    width: 50px;  
    height: 50px;  
    background: #c00;  
    position: absolute;  
    top: 0;  
    left: 0;  
    transition: all 1s; }
```

# Capítulo 7 Degradados

Los degradados son uno de los recursos que utilizan los diseñadores para decorar las webs y la verdad es que dan mucho juego para mejorar el aspecto de la página. No obstante, hasta la llegada de CSS 3, también tenían una desventaja importante, ya que para implementarlos necesitábamos usar imágenes como fondo de los elementos. Ello tiene algunas desventajas, como una mayor carga de peso en la página y la necesidad de fabricar los archivos gráficos con un programa de diseño, con la molestia adicional que necesitaríamos usar de nuevo el programa de diseño gráfico, para producir una nueva imagen, en el momento que queramos retocar el degradado.

Por suerte, CSS3 dispone de un potente mecanismo para producir degradados que resulta todavía más versátil que el propio uso de imágenes de fondo.

Los degradados implementan un gradiente de color, que pasa de un estado a otro a lo largo del fondo de los elementos HTML, ya sea capas, elementos de listas, botones, etc. Dichos degradados se obtendrán por medio de la especificación de una serie de características, como la posición inicial, la dirección hacia donde se realizará, si es circular o linear, y los colores que se incorporarán en cada uno de los pasos del gradiente.

El navegador es el encargado de renderizar el degradado en función de las características escritas para definirlo. Podemos asignar degradados como fondo en cualquier elemento HTML donde se podía implementar una imagen de fondo.

## 7.1 Degradados lineales:

En los que se crea un gradiente que va de un color a otro de manera lineal. Puede ser de arriba a abajo, de izquierda a derecha y viceversa. Incluso se puede conseguir un degradado en un gradiente de una línea con cualquier ángulo.

Los degradados lineales se consiguen con el atributo `background` asignándole el gradiente con la propiedad "linear-gradient" de CSS 3.

Un ejemplo:

```
div{ height: 130px;  
      width: 630px;  
      background: linear-gradient(orange, pink);}
```



Imagen del renderizado del degradado lineal con CSS3, tal como aparecería en Firefox 4

`linear-gradient(orange, pink);`

## 7.2 Degradados circulares:

En ellos se implementa un gradiente que se distribuye radialmente, desde un punto del elemento hacia fuera, de manera circular, que puede tener el mismo valor de radio (para hacer degradados en círculos perfectos) o con valores de radio variables (lo que generaría elipses).

El valor que asignamos a background en este caso será por medio del atributo "radial-gradient", además de toda la serie de parámetros necesarios para definir el degradado según nuestras intenciones.

```
div.circular{ background: -webkit-radial-gradient(#0f0, #06f);
```



Imagen del renderizado del degradado radial con CSS3, tal como aparecería en Firefox 4

`radial-gradient(#0f0, #06f);`

## 7.3 Explicación detallada de los degradados lineales.

El degradado de colores más sencillo que podemos crear es el degradado lineal. Con CSS3 se pueden especificar con tan solo definir una serie de parámetros en la propiedad linear-gradient, que nos permiten configurar todo tipo de gradientes de dos o más colores, sin la necesidad de usar imágenes.

```
background: linear-gradient(parámetros);
```

Como vemos, para asignar un degradado a un elemento, tenemos que utilizar la propiedad linear-gradient sobre un atributo background, o background-image. Todos los elementos que soportan imágenes de fondo permite también colocar degradados de fondo. Además, tendremos que indicar una serie de parámetros variables para la creación del degradado, que son los que realmente tienen alguna dificultad de entender. Estos parámetros son los siguientes:

**A) Origen-y/o-ángulo del degradado:**

El primer parámetro sería el origen desde donde comenzará el degradado y/o el ángulo de disposición del gradiente de color. Podemos decir que el degradado comience desde arriba, abajo o desde una esquina cualquiera. Por defecto los degradados serán distribuidos en un gradiente en línea recta, pero además podemos indicar un ángulo distinto con el que se vaya produciendo el gradiente de color.

- B) lista-de-colores y opcionalmente, el lugar hasta donde se debe mostrar cada uno:** Luego colocaremos los colores, todos los que queramos, que deben utilizarse en el degradado, separados por comas. Además, si lo deseamos, podemos definir las paradas de color "color stops", que consiste en declarar el lugar desde donde debe empezar el gradiente del color.

```
background: linear-gradient(orange, pink);
```

Esto hace un degradado desde el color naranja hacia el rosa. Todos los demás parámetros quedarían con sus valores predeterminados y el resultado sería que el degradado se realiza en toda la altura del elemento, de arriba a abajo, en un gradiente vertical, comenzando el naranja en la parte de arriba y acabando en rosa en la parte de abajo.

```
background: linear-gradient(top left, #fff, #f66);
```

Este degradado comienza en la esquina superior izquierda y se crea un gradiente que va hacia la esquina opuesta. Por tanto, el degradado formará un gradiente oblicuo, en diagonal desde la esquina superior izquierda, donde estaría el blanco (#fff), hasta la esquina inferior derecha, donde estaría el rosa (#f66).

```
background: -webkit-linear-gradient(180deg, #fff, #f66);
```

Este degradado define su dirección por medio de un ángulo expresado en grados. 0 grados haría que el degradado comenzaría en la parte de la izquierda y 180deg indica que el degradado empezaría justo por el lado contrario, es decir, por la derecha. De modo que en la parte de la derecha tendríamos el color morado y en la izquierda tendríamos el rosado.

```
background: linear-gradient(#00f 50%, #000);
```

Este degradado tiene lo que se llama un "color stop" es decir, una parada de color, que está asignada con el 50% en el primer color. Quiere decir que el primer color estaría homogéneo (sin degradado) hasta el 50% del tamaño del elemento y que luego comenzaría a degradarse hacia el segundo color.

```
background: linear-gradient(45deg, #66f, #f80, #ffc);
```

Este degradado tiene una disposición en diagonal, por los 45 grados que se han definido. Además, podemos ver que hemos definido más de dos colores en el degradado. Podemos poner tantos como queramos, separados por comas. Como no hay "color stops" los tres colores se distribuyen de manera equitativa, desde la esquina inferior izquierda hasta la superior derecha.

```
background: linear-gradient(45deg, #66f 10%, #f80 30%, #ffc 60%);
```

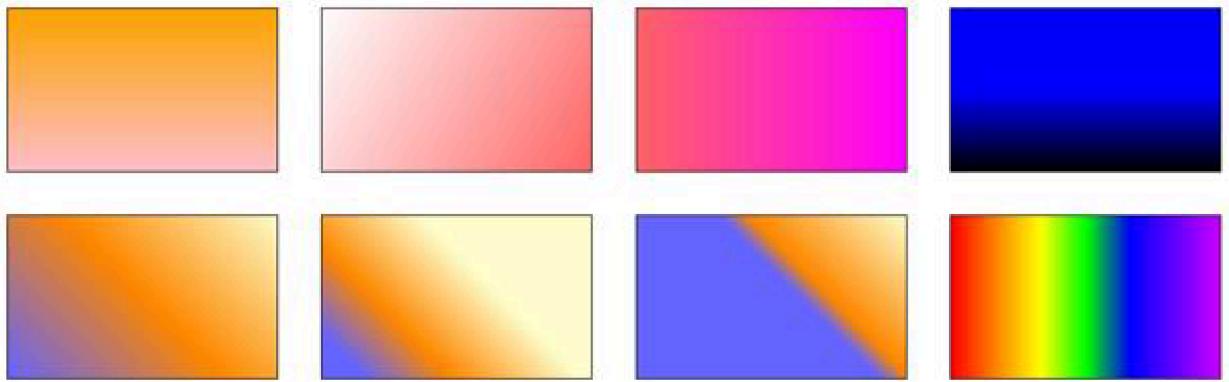
Este degradado se hace también en diagonal, desde la esquina inferior izquierda, igual que el anterior, pero hemos definido una serie de paradas de color (color stops), con lo cual la distribución del gradiente no es homogénea. El primer color empezaría a degradarse hacia el segundo cuando se llega al 10% del tamaño del elemento. El degradado hacia el segundo color se completaría al llegar al 30% y a partir de ese punto empezaría a degradarse hacia el tercer color. El degradado entre el segundo y tercer color se realizaría desde el 30% al 60% del tamaño del elemento y se completaría cuando estamos en el 60%. A partir de ese último color stop (60%) tendríamos el último color de manera homogénea hasta el 100% del tamaño. Por tanto, el color predominante veremos que es el tercero, que tiene un 40% (100% del elemento - 60% del espacio donde veremos degradados) del espacio para mostrarse con su RGB tal cual fue definido.

```
background: linear-gradient(45deg, #66f 40px, #f80 180px, #ffc);
```

Este es el mismo degradado que el anterior, pero con paradas de color distintas. Además, estamos definiendo esos "color stops" con medidas en píxeles en lugar de porcentajes.

```
background: linear-gradient(left, #f00, #f80, #ff0, #0f0, #00f, #60f, #c0f);
```

Este degradado, que empieza en la izquierda y con un gradiente recto hacia la derecha, tiene varios colores, los del arcoiris.



## 7.4. Atributo repeating-linear-gradient, degradados lineales con múltiples repeticiones del mismo gradiente de color.

Los degradados con repetición se realizan de manera similar a los degradados lineales normales, la diferencia es que utilizaremos el nombre de atributo `repeating-linear-gradient` y que, para que se produzcan las repeticiones tendremos que utilizar paradas de color.

La sintaxis es exactamente la misma que ya conocemos de los degradados lineales:

```
repeating-linear-gradient(origen, colores)
```

Siendo que en `origen` podremos colocar, tanto la posición inicial donde comienza el degradado, como el ángulo que debe formar el gradiente. Luego, los colores, como también vimos, se indican separados por comas. Sin embargo, ahora, para que realice la repetición, estamos obligados a señalar una parada de color.

La parada de color la podremos hacer en cualquiera de los colores del degradado, pero claro, tendremos que asegurarnos que el último color tenga una parada de color menor que el tamaño del elemento, o si trabajamos con porcentajes, menor que el 100%. Así, en el espacio del elemento que sobre después de la última parada de color, comenzará la repetición del degradado.

Vamos ahora a mostrar varios ejemplos de degradados con repetición:

```
background: repeating-linear-gradient(#fff, #666 25%);
```

Este degradado comienza en blanco y termina en gris. Como no se indicó nada, irá de arriba a abajo, en un gradiente vertical en línea recta. Pero lo importante en este caso es que el segundo color tiene una parada al 25%. Eso quiere decir que se llegará al gris en el primer 25% del espacio del elemento y que a partir de ese punto comenzará de nuevo el degradado. El segundo degradado ocupará otro 25% de la imagen y entonces se volverá a repetir. Por

tanto, en la práctica veremos que este degradado de blanco a gris se repetirá por 4 veces en el fondo del elemento donde lo coloquemos.

```
background: repeating-linear-gradient(left, #ffc, #f96 30%, #963 40%, #630 51%);
```

En este segundo ejemplo tenemos otro degradado, esta vez con 4 colores. Como se puede ver, se le han asignado varias paradas de color, en lugar de solo una al último elemento. Como el último color stop está al 51% del elemento, el degradado se verá solo dos veces.

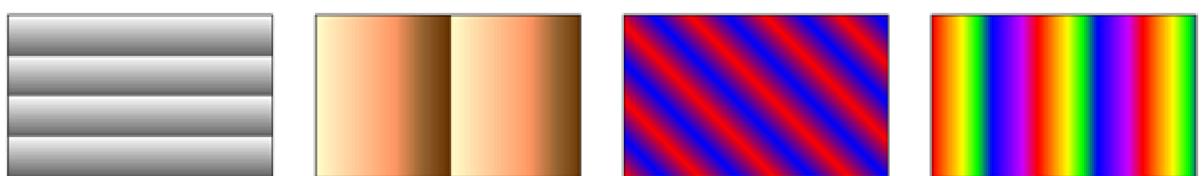
```
background: repeating-linear-gradient(45deg, red, blue, red 50px);
```

Esta es otra declaración de estilos, con un degradado en un gradiente oblicuo. Además, tiene la particularidad que se va de rojo a azul y luego de nuevo a rojo. Con ello conseguimos que las repeticiones del degradado siempre están suavizadas y no se note cuando comienza y acaba una repetición, como nos ocurría en los dos ejemplos anteriores. Además, como se puede ver, la parada de color la hemos colocado en 50px, lo que quiere decir que el degradado se repetirá cada 50 píxeles, de modo que, el número de repeticiones variará dependiendo del tamaño del contenedor donde se asigne este fondo.

```
background: repeating-linear-gradient(left, #f00, #f80, #ff0, #0f0, #00f, #60f, #c0f, #f00 100px);
```

Este último ejemplo tiene los colores del arcoíris repetidos cada 100 píxeles.

Se puede ver los distintos degradados realizados en la siguiente imagen.



## 7.5. Degradados en un gradiente de color de forma radial, círculos y elipses.

Los degradados radiales, que incluyen tanto los que tiene forma circular en general, como los que tienen forma de elipse, se consiguen a través del atributo radial-gradient. De modo que, mediante la aplicación de distintos parámetros, conseguiremos todas las posibilidades. Como veremos a continuación, tienen una forma de definirse muy parecida a la que vimos con los degradados lineales, aunque en este caso tendremos algunos otros parámetros a tener en cuenta, lo que puede dificultar un poquito más su entendimiento.

La sintaxis resumida será la siguiente:

```
background: radial-gradient(parámetros);
```

Los parámetros que podemos indicar en la declaración radial-gradient() es donde realmente radica la dificultad y a la vez la potencia de este tipo de degradados. No obstante, la mayoría de los parámetros son opcionales, por lo que podemos hacer degradados radiales bastante simples, que tomarán parámetros por defecto. En realidad, como veremos, lo único que necesitaremos siempre es definir dos o más colores para realizar el gradiente de color.

El listado de parámetros que podremos indicar es el siguiente:

### A) Posición inicial del gradiente circular:

Los degradados radiales comienzan en un punto cualquiera del fondo de un elemento y se extienden hacia fuera de ese punto con formas circulares o de elipse. Luego, para definirlos, necesitaremos una forma de especificar dicho punto de inicio del degradado. El punto se especifica con una o dos coordenadas, que pueden tener distintas unidades CSS. Si se omite, se entiende que el degradado tiene que comenzar en el punto central del fondo del elemento.

### B) Forma y/o dimensión:

La forma puede ser circular o elipse, para lo cual especificamos las palabras "circle" o "ellipse". El tamaño lo expresamos con otra serie de palabras clave, que indican hasta donde debe crecer el círculo o elipse: closest-side | closest-corner | farthest-side | farthest-corner | contain | cover. Por ejemplo, closest-side indica que el círculo o elipse debe crecer hasta el lado más cercano. La palabra farthest-corner indicaría que debe

crecer hasta la esquina más lejana. Contain sería lo mismo que decir closest-side y cover sinónimo de farthest-corner.

### **Alternativa a B) Tamaño:**

De manera alternativa a especificar la forma y dimensión del degradado -punto B) anterior-, podemos indicar un par de medidas en cualquier unidad CSS o porcentajes. Esas medidas se utilizarían para generar un círculo o una elipse del tamaño deseado para nuestro gradiente. La primera medida sería para la anchura de la elipse y la segunda sería para la altura (si ambas son iguales se mostraría una forma circular en el degradado. Si son distintas, sería una elipse. El tamaño debe ser siempre positivo.

### **C) Colores del degradado:**

Para acabar, se deben indicar cuantos colores se deseen, separados por comas, con la posibilidad de indicar las paradas de color que se deseen.

Ahora, veamos una serie de ejemplos que esperamos aclare las ideas con respecto a la declaración de fondos radiales.

```
background: radial-gradient(#0f0, #06f);
```

Esto hace un degradado desde el verde al azul turquesa, con todos los otros parámetros predeterminados. Haría un gradiente de forma circular, con su punto de inicio en el centro del elemento, en verde, haciendo que se llegase al azul turquesa en los bordes del elemento.

```
background: radial-gradient(top left, #fff, #f66);
```

En este caso hemos definido el punto de inicio del gradiente con "top left". Se trata de la esquina superior izquierda, donde aparecerá el blanco y el degradado tendría forma circular tendiendo hacia rosa, ocupando el 100% del elemento.

```
background: radial-gradient(200px 30px, #f0f, #000);
```

Este degradado también declara la posición inicial del gradiente, pero lo hace mediante las coordenadas definidas con medidas en píxeles. Es circular y ocupa el 100% del espacio disponible en el elemento.

```
background: radial-gradient(center, #00f, #000 50%);
```

En este declaramos la posición inicial con center, el comportamiento predeterminado, que coloca el inicio del degradado en centro, tanto vertical como horizontal. El detalle es que el degradado se realiza desde el centro hasta el 50% del tamaño del elemento, ya que le hemos puesto una parada de color ("color stop") de 50% en el último color.

```
background: radial-gradient(circle, #66f, #f80, #ffc);
```

Este es el primero de los ejemplos en el que definimos la forma del degradado, aunque solo indicamos "circle". Por tanto, el degradado comenzará en el centro y ocupará el 100% del espacio disponible en el elemento, aunque siempre con el mismo radio.

```
background: radial-gradient(ellipse cover, #66f, #f80, #ffc);
```

Este degradado es exactamente igual que el anterior, pero en vez de circular es de elipse, cubriendo el 100% del espacio disponible, y comenzando en el centro. Este es el comportamiento predeterminado del estilo.

```
background: radial-gradient(10%, ellipse closest-side, #fff 60%, #f80 85%, #ffc);
```

Este ejemplo tiene definida la posición del degradado y la forma. Es el primero que especifica esos dos valores al mismo tiempo. En este caso, sobre la posición solo se declara 10%, así que aparecerá centrado en la vertical y en la horizontal aparecerá en el 10% del espacio del contenedor por la parte de la izquierda. Es de forma de elipse y closest-side significa que se expande en forma de elipse hasta completarse en el lado más cercano.

```
background: radial-gradient(10%, ellipse farthest-corner, #66f 60%, #f80 85%, #ffc);
```

Este degradado es igual que el anterior, en la misma posición y de forma de elipse, pero el tamaño se ha definido con farthest-corner, con lo que el degradado será mucho mayor, expandiéndose hasta la esquina más lejana.

```
background: radial-gradient(20px 100px, 30% 80%, #fff, #666, #66f);
```

En este caso hemos definido la posición inicial con las coordenadas en píxeles y, lo que resulta novedad, hemos definido tanto la forma como el tamaño del degradado en porcentaje. La anchura será el 30% del elemento y la altura el 80%.

```
background: radial-gradient(top left, 150px 100px, #ffc, #f96, #963, #630);
```

Hemos definido la posición inicial por medio de los valores top y left y el tamaño por medio de unidades en píxeles.

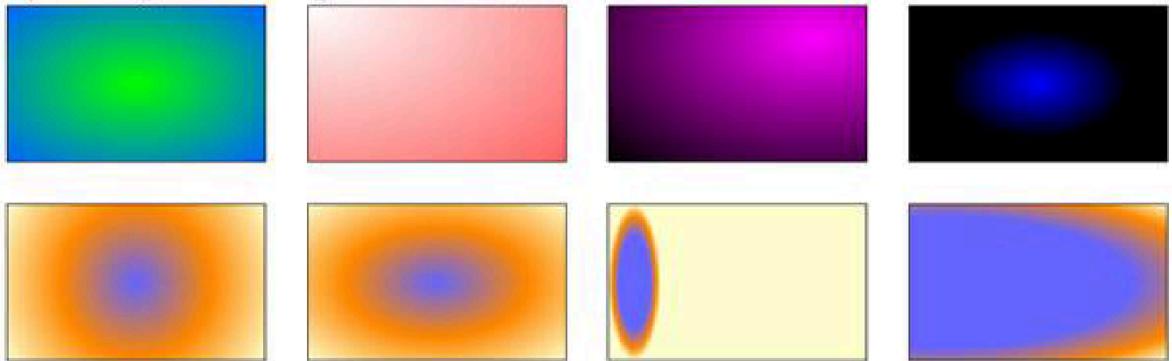
```
background: radial-gradient(20% 80%, 100% 50%, red, blue 50px, red);
```

La posición del centro del degradado está con porcentaje, así como el tamaño de la elipse, también con porcentajes.

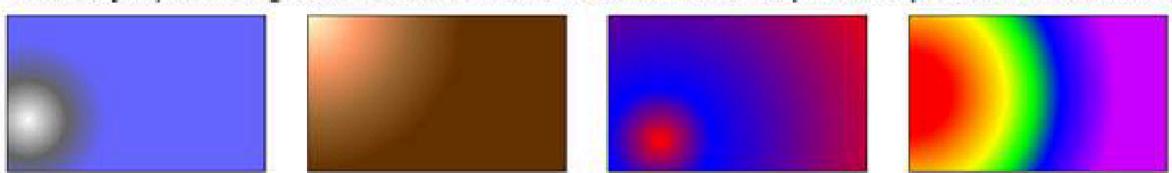
```
background: radial-gradient(left, 200px 200px, #f00 20%, #f80, #ff0, #0f0, #00f, #60f, #c0f);
```

En este hemos definido el tamaño de la anchura y altura con dos valores en píxeles, pero como son iguales, en lugar de una elipse veríamos una forma circular, cuyo radio es siempre igual.

8 primeros ejemplos con degradados radiales, tal como se ven en Firefox:



Últimos ejemplos de degradados radiales. Usan la definición de forma y tamaño que solo se ve en Chrome:



## 7.6. degradados radiales, con repeticiones, fondos con gradientes de color.

Como veníamos diciendo, una de las ventajas de estas nuevas características de las CSS3 es que nos permiten realizar fondos para páginas web con degradados, pero sin tener que usar imágenes. Eso consigue ahorrarnos tiempo de desarrollo y diseño, a la vez que aligera el peso de las páginas web. Pero como curiosidad, en el caso de los degradados radiales con repeticiones, además habría que añadir que serían casi imposibles de realizar usando imágenes de fondo, o por lo menos tendríamos que usar archivos gráficos muy grandes y pesados.

Los degradados radiales con repetición son casi idénticos a los degradados radiales normales.

La sintaxis básica de este tipo de degradados es la siguiente:

`repeating-radial-gradient(parámetros)`

Los parámetros, tales como centro del degradado radial, forma y tamaño del motivo y los colores son exactamente los mismos que para los degradados radiales. Las únicas diferencias es que tenemos que utilizar el atributo `repeating-radial-gradient`. Además, para que se produzca la repetición con el tamaño o intervalo que nosotros deseemos, tendremos que asignar alguna parada de color al último de los colores del degradado, que generalmente tendrá un valor menor del 100% del espacio del elemento.

A continuación podemos ver una serie de ejemplos:

```
background: repeating-radial-gradient(circle, #fff, #666 25%);
```

Este ejemplo hace un degradado entre dos colores, cuyo segundo elemento tiene una parada de color en el 25%. Por ello el resultado producirá el mismo degradado repetido 4 veces, una en cada 25% del espacio del elemento donde se coloque.

```
background: repeating-radial-gradient(left, circle, #ffc, #f96 30%, #963 40%, #630 51%);
```

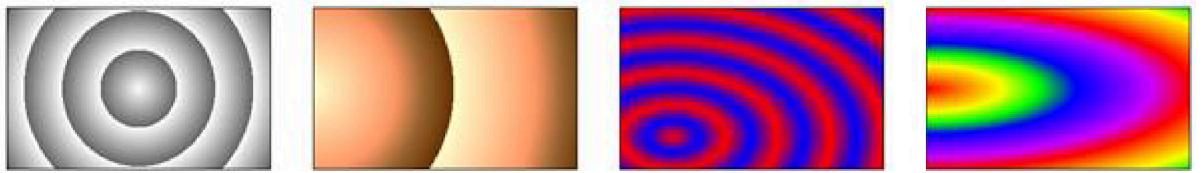
Este degradado hace un gradiente entre varios colores y el último de ellos tiene una parada en el 51%. Esto quiere decir que el degradado se repetirá dos veces.

```
background: repeating-radial-gradient(20% 80%, ellipse closest-side, red, blue, red 50px);
```

En este ejemplo hacemos un degradado con forma de elipse y va desde rojo a azul y luego de nuevo a rojo. Al comenzar y acabar en rojo, se consigue que las repeticiones del degradado no tengan saltos bruscos de un color al otro. Como se puede ver, el último color tiene una parada de color en 50 píxeles, con lo que el degradado se repetirá de nuevo cada 50 puntos en la pantalla. El número de repeticiones dependerá del tamaño del elemento donde se coloque este degradado.

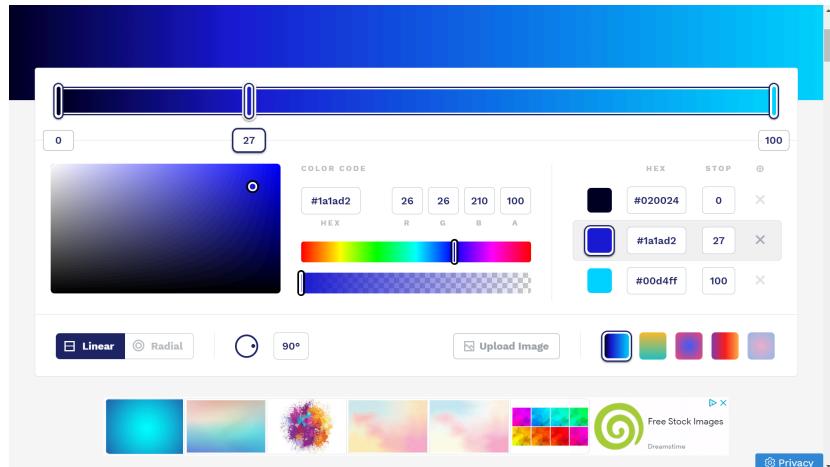
```
background: repeating-radial-gradient(left, ellipse farthest-side, #f00, #f80, #ff0, #0f0, #00f, #60f, #c0f, #f00);
```

Este otro caso produce una repetición del degradado, pero no hemos colocado ninguna parada de color en el último elemento. Esto quiere decir que el degradado se repetirá, pero el intervalo de repetición en esta ocasión dependerá del tamaño y forma radial que se haya configurado. En este caso en concreto tenemos una elipse que se expande, desde la izquierda hasta el lado opuesto, con lo que sólo en una pequeña porción del fondo se verá la repetición de los colores.



## 7.7 Generadores de gradientes CSS.

Afortunadamente contamos con herramientas online que nos facilitan la labor anterior de crear un degradado de colores. Una de estas es: <https://cssgradient.io/>



## 7.8 Patrones creados con CSS.

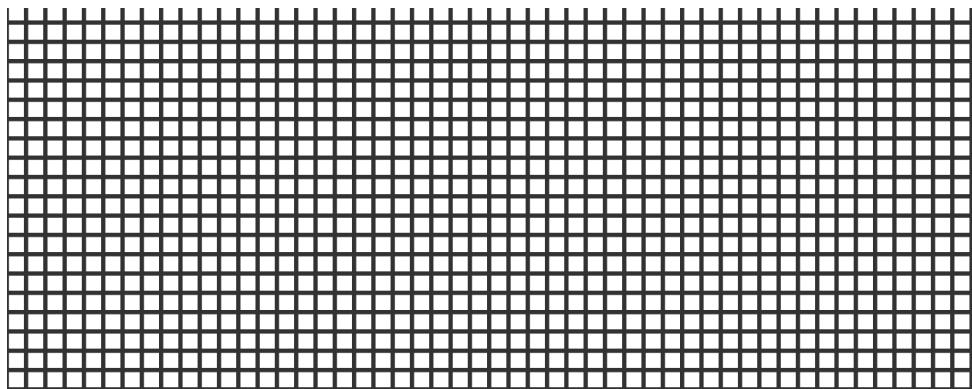
Un patrón no es más que un elemento, que se repite muchas veces. Si hablamos de patrones CSS, en este caso nos referimos a elementos que van a dar estilo a nuestra página web. Una forma bastante habitual de usar estos patrones, es creando fondos visualmente impactantes. Aunque se pueden utilizar en multitud de sitios, como banners, cabeceras, anuncios, etc.

Se pueden crear patrones usando los gradientes CSS, es decir en vez de usar una imagen (en el formato que sea) como background en las páginas, usan un gradiente de CSS.

Básicamente lo que hay que hacer es definir un color de base con la propiedad background-color y después definir por encima el patrón usando reglas como linear-gradient, radial-gradient o repeating-linear-gradient sobre background-image.

```
background-image:linear-gradient(#333 20%, transparent 25%),
    linear-gradient(90deg, #333 20%, transparent 25%);

background-size:20px 20px;
```

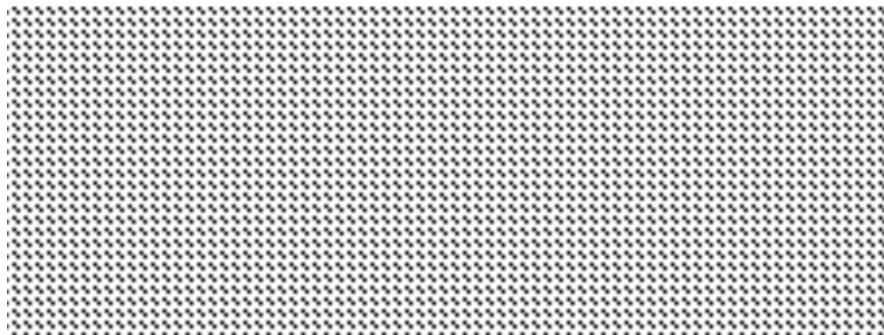


### VELOCIDAD

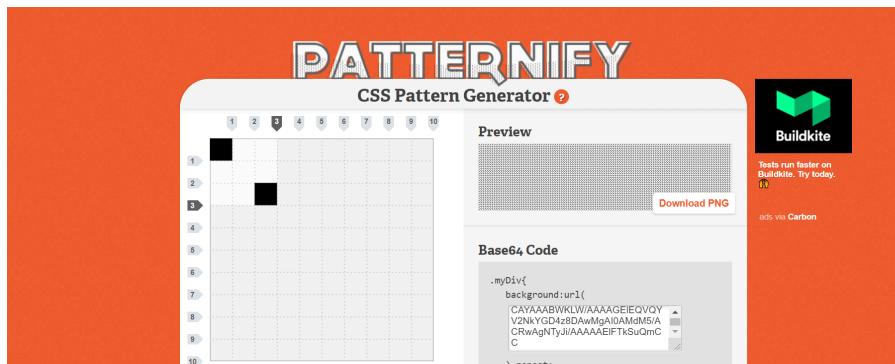
El tener que importar un recurso a nuestra web siempre es más lento, incluso aunque se trate de un archivo pequeño como suelen ser los patrones, ya que el solo hecho de realizar una llamada es lo que nos penaliza.

Por otra parte puedes incrustar la imagen en base64 dentro de la propia hoja de estilos y evitar la llamada pero incluso de esta manera sería más lento y estaríamos subiendo el peso del archivo notablemente.

```
background:url( data:image/png;base64,
    iVBORw0KGgoAAAANSUhEUgAAAAQAAAAECAYAAACp8Z5+AAAAG01EQVQYV2NkYGD4z8DAwMgAB
    XAGNgGwSgwVAFbmAgXQdISfAAAAAE1FTkSuQmCC
) repeat;
```

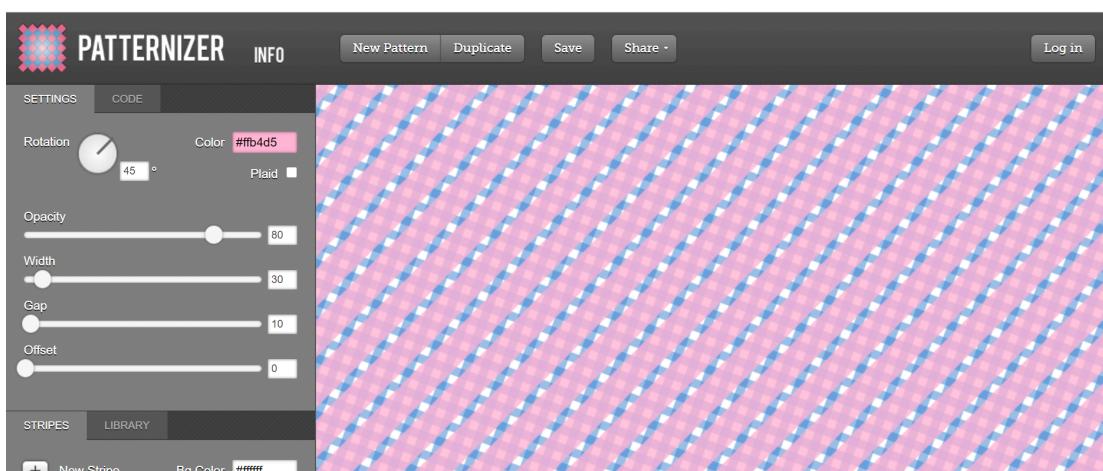


Patternify nos permite crear patrones en base64 embebidos en el código:



Crear patrones con css requiere de un trabajo de diseño y programación avanzado, sobretodo cuando los patrones son ya bastante complejos, en contrapartida tiene grandes ventajas sobre el uso de imágenes de fondo.

### **Patternizer** (generador de patrones)

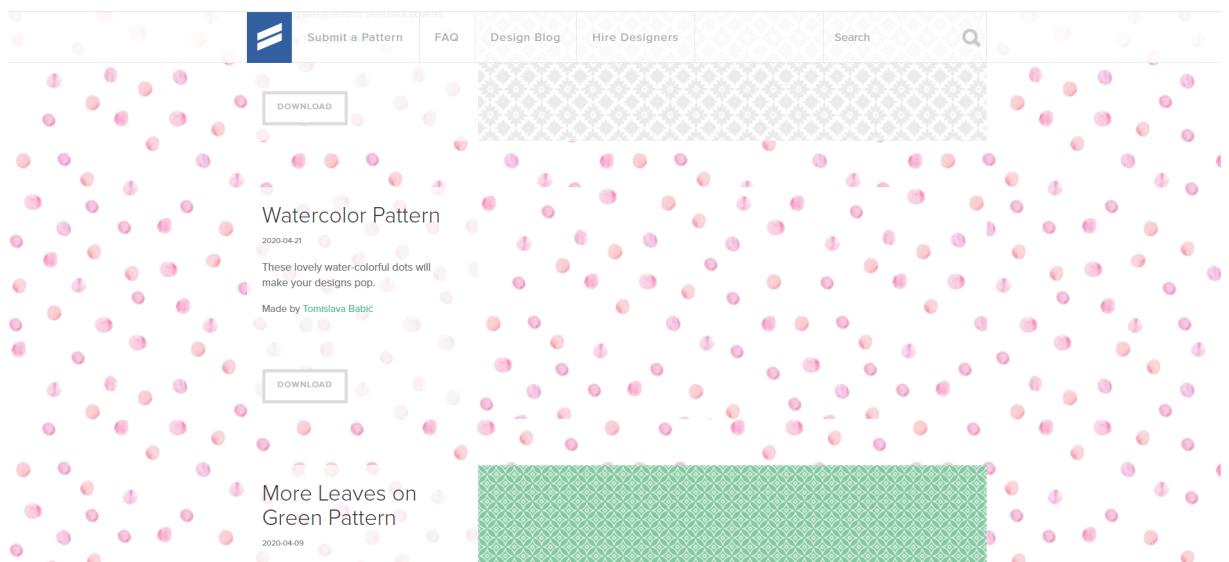


**Galería de Patrones:** (ejemplos de patrones ).



Si necesitamos patrones más complejos que no podemos desarrollar con CSS, nos quedará el último recurso de utilizar imágenes png o svg.

#### Recursos:





Compete y aprende css a través de retos...

# Capítulo 8 Flexbox

## 8.1 Introducción

### ¿Qué es flexbox?

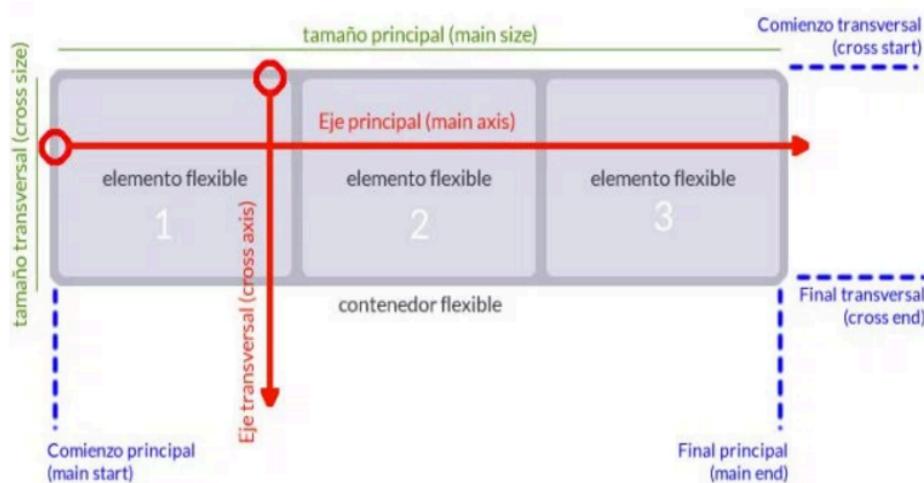
Flexbox ( o el módulo de cajas flexibles ) es probablemente uno de los más completos y eficaces módulos de maquetación. Todo lo que era complicado en versiones anteriores de CSS ( como centrar verticalmente o diseñar estructuras que se redimensionan con elegancia ) con flexbox ( CSS3 ) es ya una tarea muy fácil.

### ¿Cómo crear una caja flex?

Flexbox representa un modelo básico de maquetación que supone la existencia de una caja padre llamada contenedor flexible o caja flex. Los elementos hijos situados dentro del contenedor flexible llevan el nombre de elementos o **ítems** flex.

Los elementos flex tienen la capacidad de redimensionarse y colocarse dentro de la caja flex con facilidad. También tienen la capacidad de alinearse tanto horizontalmente como verticalmente y todo esto puede ser muy interesante a la hora de diseñar páginas web adaptativas.

La propiedad **display** está por ahí desde CSS1, pero es en el CSS3 cuando adquiere la capacidad de transformar una caja cualquiera en un contenedor flex. Para esto tiene que tomar una de estos valores: **flex o flex-inline**.



```
.flex-container{display:flex;}  o
.flex-container{display:flex-inline;}

.flex-container{
  display: flex;
}

.flex-container-inline{
  display: inline-flex;
}
```

## Propiedades del contenedor flex

Propiedad	Sintaxis	Descripción
display	display: flex   inline-flex; display: -ms-flexbox   -ms-inline-flexbox;  .flex-container{ display: -webkit-flex; display: -ms-flexbox; display: flex; }	Si su valor es <b>flex</b> o <b>inline-flex</b> , la propiedad <b>display</b> define un contenedor flex ( flexbox ).
flex-direction	flex-direction: row   row-reverse   column   column-reverse;  .flex-container{ -webkit-flex-direction: row; -ms-flex-direction: row; flex-direction: row; }	Establece la dirección del eje principal y por tanto la dirección de los elementos flex. El valor por defecto es <b>row</b> ( fila ).
flex-wrap	flex-wrap: nowrap   wrap   wrap-reverse;  .flex-container{ -webkit-flex-wrap: wrap; -ms-flex-wrap: wrap; flex-wrap: wrap; }	Especifica si puede haber un cambio de línea ( <b>wrap</b> ) o no ( <b>nowrap</b> ). El valor por defecto es <b>nowrap</b> .
flex-flow	flex-flow: <i>flex-direction</i> [ <i>flex-wrap</i> ]  .flex-container{ -webkit-flex-flow: row nowrap; -ms-flex-flow: row nowrap; flex-flow: row nowrap; }	Es la forma abreviada para las propiedades <b>flex-direction</b> y <b>flex-wrap</b> . El valor por defecto es <b>row nowrap</b>
align-items	align-items: flex-start   flex-end   center   baseline   stretch; -ms-flex-align:start   end   center   baseline   stretch;  .flex-container{ -webkit-align-items: flex-start; -ms-flex-align:start; align-items: flex-start; }	Define como se colocan los elementos dentro de una caja flexible flexbox relativamente al eje transversal. El valor por defecto es <b>stretch</b> .
	justify-content: flex-start   flex-end   center   space-between	

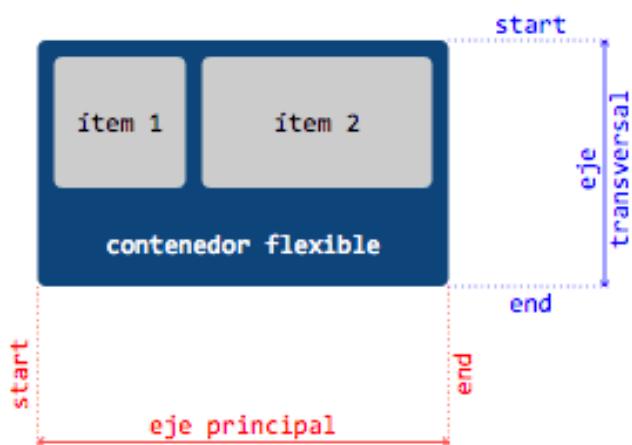
## 8.2. La propiedad flex-direction

La propiedad **flex-direction** es una propiedad del contenedor flex.

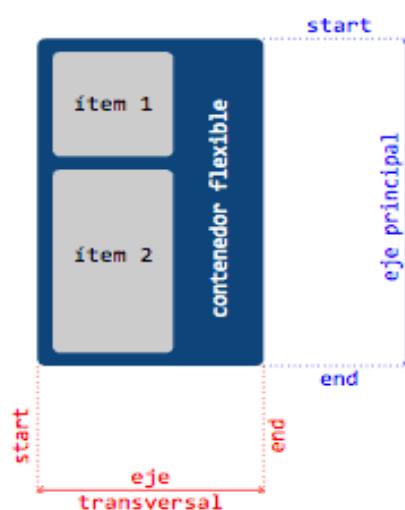
En una caja flex podemos colocar los elementos en cualquier dirección: horizontalmente (**row**) o verticalmente (**column**), en el sentido lógico o en el sentido contrario (**-reverse**).

Para hacerlo, utilizamos la propiedad **flex-direction**, que establece cuál es el eje principal de la caja y por lo tanto la dirección de los elementos hijos.

### **flex-direction:row**



### **flex-direction:column**



**Muy importante:** `flex-direction: row` establece el eje horizontal como eje principal (*main axis*), y el eje vertical como eje transversal (*cross axis*). Si `flex-direction: column` pasa todo lo contrario: el eje vertical es el eje principal (*main axis*) mientras que el eje horizontal es el eje transversal (*cross axis*).

Esto es realmente muy importante ya que las propiedades de flexbox controlan la alineación de los ítems flex a lo largo de estos ejes.

La propiedad `flex-direction` puede tomar una de estos valores:

```
.contenedor {flex-direction: row / row-reverse / column / column-reverse;}
```

`row` (el valor por defecto): coloca los elementos flex horizontalmente (*row = fila*). En idiomas como el castellano, con un sistema de escritura de izquierda a derecha (*ltr - left to right*), los elementos flex se colocan también de izquierda a derecha.

`row-reverse`: coloca los elementos flex horizontalmente pero en sentido contrario (de derecha a izquierda en idiomas como el castellano.)

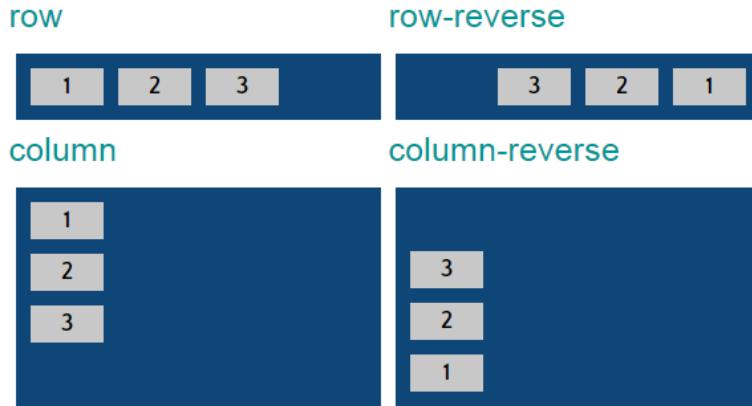
`column`: coloca los elementos flex verticalmente y de arriba abajo (*tb - top to bottom*).

`column-reverse`: coloca los elementos flex verticalmente y de abajo arriba (*bottom to top*).

```
.flex-container.flex-row{
    flex-direction: row;
}

.flex-container.row-reverse{
    flex-direction: row-reverse;
}

.flex-container.flex-column{
    flex-direction: column;
    height: 150px;}
```



## 8.3 La propiedad flex-wrap

La propiedad `flex-wrap` es [una propiedad del contenedor flex](#) y especifica si puede haber un [cambio de línea](#) (`wrap`) o no (`nowrap`). El valor por defecto es `nowrap`. La propiedad `flex-wrap` puede tomar una de estas valores:

```
.contenedor {flex-wrap: nowrap / wrap / wrap-reverse;}
```

[\*\*nowrap\*\*](#) (el valor por defecto): los elementos flex aparecen en una sola línea. En ciertas circunstancias los elementos flex aparecen redimensionados para que puedan acomodarse dentro del [contenedor flex](#). En otras circunstancias el [contenedor flex](#) puede desbordar ([overflow](#)).

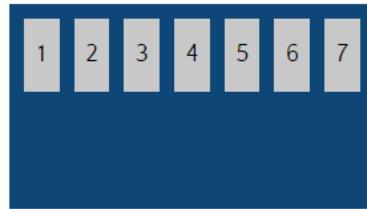
[\*\*wrap\*\*](#): indica al CSS que puede haber [cambio de línea](#). Los elementos flex aparecen colocados en varias líneas.

[\*\*wrap-reverse\*\*](#): igual que [wrap](#) pero las líneas de elementos flex aparecen ordenadas en sentido contrario.



**flex-wrap: nowrap** (*el valor por defecto*)

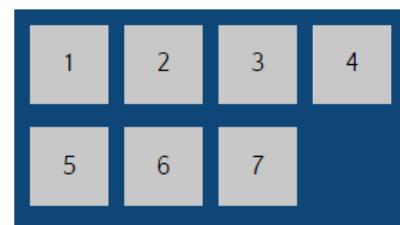
```
.flex-container.nowrap{  
    flex-wrap: nowrap;  
}  
  
.flex-container{  
    width: 250px;  
    height: 140px;  
    padding:5px;  
    margin: 10px auto;  
    background-color:#124678;  
    display: flex;  
}  
  
.flex-item{  
    display: inherit;  
    width:50px;  
    height:50px;  
    background-color:#ccc;  
    margin:5px;  
}  
  
.flex-item p{  
    width:100%;  
    text-align:center;  
    align-self: center;  
    margin:0; }  
  
.flex-container.nowrap{  
    flex-wrap: nowrap;  
}  
  
<div class="flex-container nowrap">  
  
    <div class="flex-item"><p>1</p></div>  
  
    .....  
  
</div>
```



## flex-wrap: wrap

Si `flex-wrap: wrap`, el CSS entiende que puede haber un cambio de línea. Los elementos flex aparecen colocados en varias líneas, tantas como sea necesario.

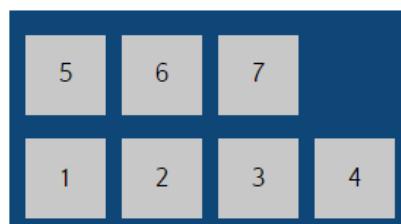
```
.flex-container.wrap{
    flex-wrap: wrap;
}
```



## flex-wrap: wrap-reverse

Si `flex-wrap: wrap-reverse`, el CSS entiende que puede haber un cambio de línea. Los elementos flex aparecen colocados en varias líneas, pero en orden contrario.

```
.flex-container.wrap-reverse{
    flex-wrap: wrap-reverse;
}
```



## La propiedad flex-flow

La propiedad **flex-flow** es una propiedad del contenedor **flex**.

Para que podamos escribir menos código, el CSS nos permite abbreviar las dos propiedades: **flex-direction** y **flex-wrap** ( opcional ) en una sola: **flex-flow**. El valor por defecto es **row nowrap**.

```
.contenedor { flex-flow: flex-direction [flex-wrap]; }
```

```
.flex-container{
    flex-flow: row nowrap;
}
```

## 8.4 La propiedad align-items

Podemos controlar el alineamiento de los elementos de una caja flexible a lo largo de su **eje transversal** con **align-items**.

La propiedad **align-items** es una propiedad del contenedor **flex** y puede tomar una de estas valores:

```
.contenedor { align-items: flex-start | flex-end | center | baseline | stretch; }
```

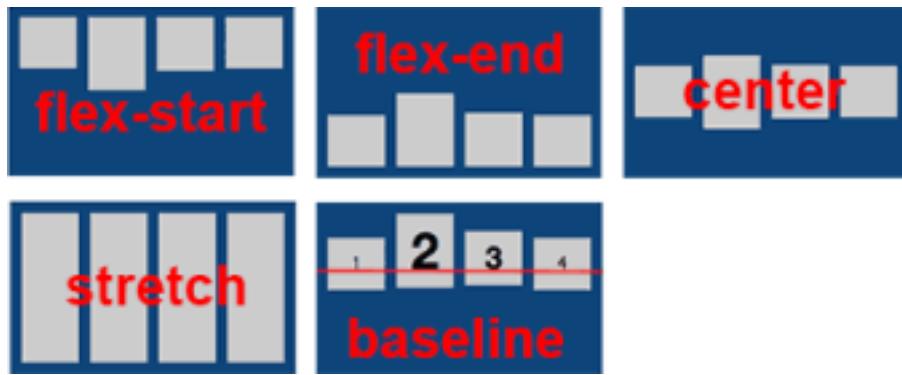
**flex-start**: los elementos aparecen agrupados al principio (*start*) del eje transversal.

**flex-end**: los elementos aparecen agrupados al final (*end*) del eje transversal.

**center**: los elementos aparecen agrupados al centro (*center*) de la caja.

**stretch** ( el valor por defecto ): los elementos aparecen estirados (*stretched*) para ocupar el espacio restante.

**baseline**: los elementos aparecen alineados relativamente a su línea de base (*baseline*).



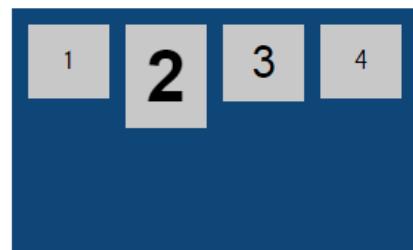
## align-items: flex-start

Si queremos que los elementos (*items*) aparezcan agrupados al principio (*start*) del eje transversal de la caja flex utilizamos `align-items: flex-start;`

```
.flex-container.flex-start{
    align-items: flex-start;
}

.flex-container{
    display: flex;
}

.flex-container.flex-start{
```



## align-items: flex-end

Si queremos que los elementos (*items*) aparezcan agrupados al final (*end*) del eje transversal de la caja flex utilizamos `align-items: flex-end;`

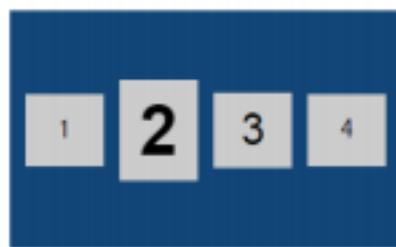
```
.flex-container.flex-end{
    align-items: flex-end;
}
```



## align-items: center

Si queremos que los elementos (*items*) aparezcan agrupados en el centro (*center*) de la caja flex utilizamos `align-items: center;`

```
.flex-container.center{
    align-items: center;
}
```

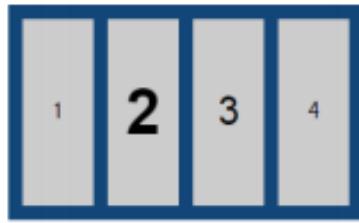


## align-items: stretch

Si queremos que los elementos (*items*) de la caja flex aparezcan estirados (*stretched*) ocupando el espacio restante, utilizamos `align-items: stretch;`

**Nota:** Si los items tienen Altura (height) no se podrán estirar.

```
.flex-container.stretch{
    align-items: stretch;
}
```



### align-items: baseline

Si utilizamos `align-items: baseline;` los elementos (*items*) aparecen distribuidos uniformemente: al principio, en el centro y al final del contenedor flex.

```
.flex-container.baseline{
    align-items: baseline;
}
```



## 8.5 La propiedad justify-content

Podemos controlar el alineamiento de los elementos de una caja flexible (*flexbox*) a lo largo de su eje principal con `justify-content`.

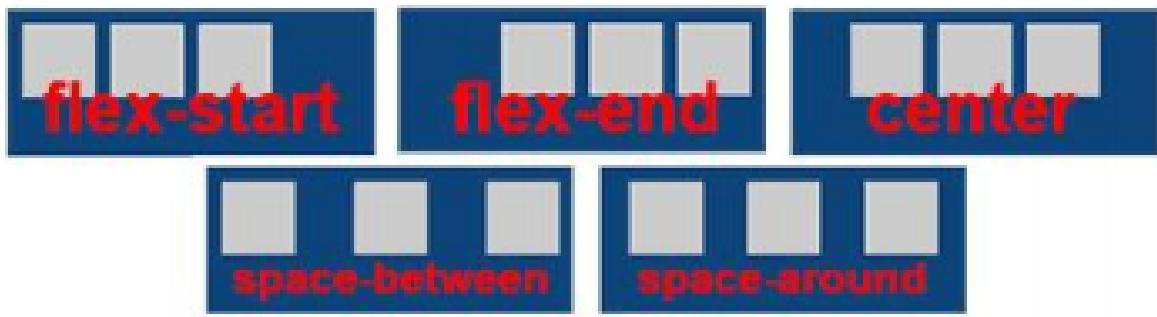
La propiedad `justify-content` es una propiedad del contenedor flex y puede tomar una de estas valores:

`.contenedor { justify-content: flex-start / flex-end / center / space-between / space-around; }`  
`flex-start` (el valor por defecto): los elementos aparecen agrupados al principio (*start*) del eje principal.

`flex-end`: los elementos aparecen agrupados al final (*end*) del eje principal. `center`: los elementos aparecen agrupados al centro (*center*).

`space-between`: los elementos aparecen distribuidos uniformemente: al principio, en el centro y al final del contenedor flex.

`space-around`: los elementos aparecen distribuidos uniformemente, y con un espacio igual entre ellos.



## justify-content: flex-start

Si queremos que los elementos (*items*) aparezcan agrupados al principio (*start*) del eje principal de la caja flex utilizamos `justify-content: flex-start;`

```
.flex-container{
    display: flex;
}

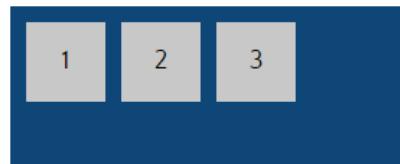
.flex-container.flex-start{

    justify-content: flex-start;
}

<div class="flex-container flex-start">

    <div class="flex-item"><p>1</p></div>
    <div class="flex-item"><p>2</p></div>
    <div class="flex-item"><p>3</p></div>

</div>
```



## justify-content: flex-end

Si queremos que los elementos (*items*) aparezcan agrupados al final (*end*) del eje principal de la caja flex utilizamos `justify-content: flex-end;`

```
.flex-container.flex-end{
    -webkit-justify-content: flex-end;
    -ms-flex-pack: end; justify-content: flex-end;
}
```



## justify-content: center

Si queremos que los elementos (*items*) aparezcan agrupados en el centro (*center*) de la caja flex utilizamos `justify-content: center;`

```
.flex-container.center{
    justify-content: center;
}
```



## justify-content: space-between

Si utilizamos `justify-content: space-between;` los elementos (*items*) aparecen distribuidos uniformemente: al principio, en el centro y al final del contenedor flex.

```
.flex-container.space-between{
    justify-content: space-between;
}
```



## justify-content: space-around

Si utilizamos `justify-content: space-around;` los elementos (*items*) aparecen distribuidos uniformemente, y con un espacio igual entre ellos.

```
.flex-container.space-around{
    justify-content: space-around;
}
```



## 8.6 La propiedad align-content

Podemos controlar el alineamiento de los elementos de una caja flexible (*flexbox*) a lo largo de su eje principal con `justify-content` o a lo largo de su eje transversal con `align-items`.

Pero, a veces, los elementos de la caja flex pueden ocupar varias líneas (*vea flex-wrap*). En este caso podemos controlar el alineamiento de los elementos flex utilizando la propiedad `align-content`.

La propiedad `align-content` es una propiedad del contenedor flex y puede tomar una de estos valores:

```
.contenedor { align-content: flex-start / flex-end / center / space-between / space-around / stretch; }
```

`flex-start`: los elementos aparecen agrupados al principio (*start*) del eje transversal.

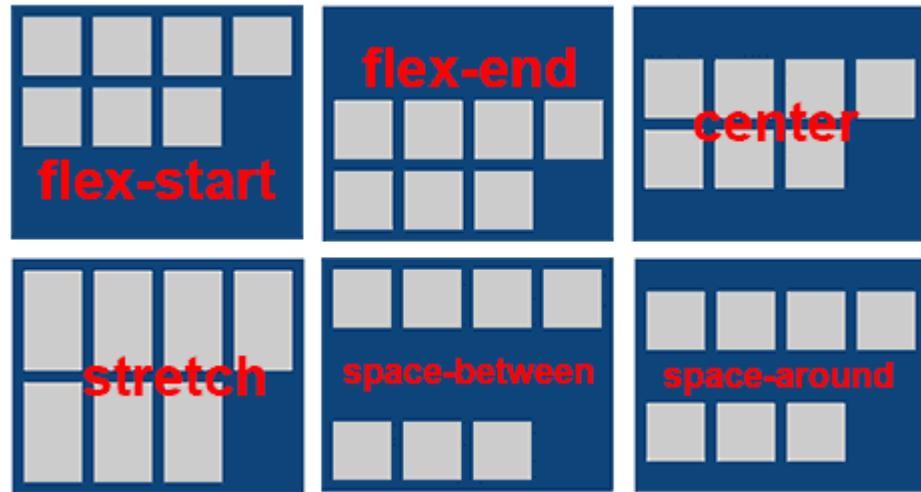
`flex-end`: los elementos aparecen agrupados al final (*end*) del eje transversal. `center`:

los elementos aparecen agrupados al centro (*center*).

`stretch` (el valor por defecto): los elementos aparecen estirados (*stretched*) para ocupar el espacio restante.

`space-between`: los elementos aparecen distribuidos uniformemente: al principio, en el centro y al final del contenedor flex.

`space-around`: los elementos aparecen distribuidos uniformemente, y con un espacio igual entre ellos.



## align-content: flex-start

Si queremos que los elementos (*items*) aparezcan agrupados al principio (*start*) del eje transversal de la caja flex utilizamos `align-content: flex-start;`

```
.flex-container.flex-start{
    align-content: flex-start;
}

.flex-container{
    flex-wrap: wrap;
}

.flex-container.flex-start{

    align-content: flex-start;
}

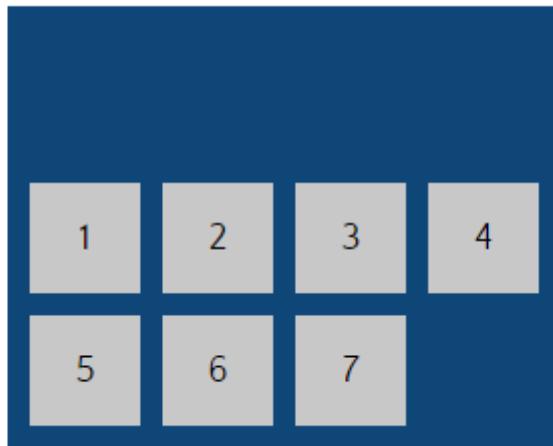
<div class="flex-container flex-start">
    <div class="flex-item"><p>1</p></div>
    ...
</div>
```



## align-content: flex-end

Si queremos que los elementos (*items*) aparezcan agrupados al final (*end*) del eje transversal de la caja flex utilizamos `align-content: flex-end;`:

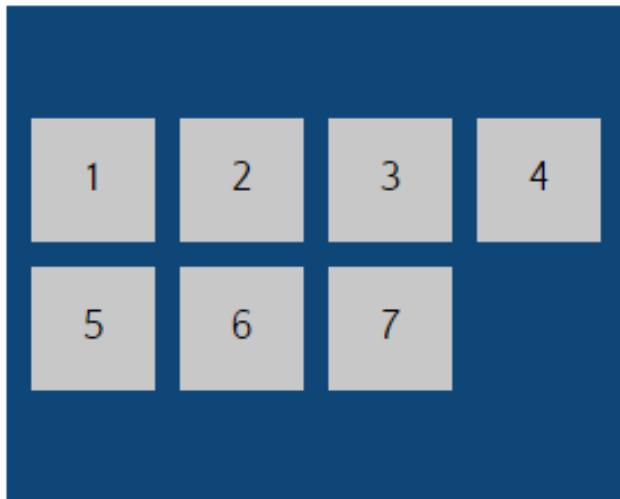
```
.flex-container.flex-end{  
    align-content: flex-end;  
}
```



## align-content: center

Si queremos que los elementos (*items*) aparezcan agrupados en el centro (*center*) de la caja flex utilizamos `align-content: center;`:

```
.flex-container.center{  
    align-content: center;  
}
```



## align-content: stretch

Si queremos que los elementos (*items*) de la caja flex aparezcan estirados (*stretched*) ocupando el espacio restante, utilizamos `align-content: stretch;`

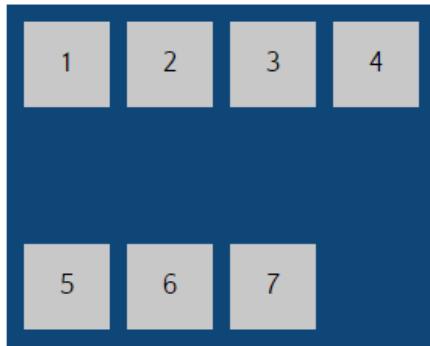
```
.flex-container.stretch{
    align-content: stretch;
}
```



## align-content: space-between

Si utilizamos `align-content: space-between;` los elementos (*items*) aparecen distribuidos uniformemente: al principio, en el centro y al final del contenedor `flex`.

```
.flex-container.space-between{
    align-content: space-between;
}
```



## align-content: space-around

Si utilizamos `align-content: space-around;` los elementos (*items*) aparecen distribuidos uniformemente, y con un espacio igual entre ellos.

```
.flex-container.space-around{
```

```
    align-content: space-around;  
}
```

## 8.7 La propiedad align-self

La propiedad `align-self` reposiciona elementos individuales relativamente al eje transversal de la caja. Generalmente se trata de elementos posicionados con `align-items`.

La propiedad `align-self` es una propiedad de los ítems del contenedor flex y puede tomar una de estas valores:

```
.item { align-self: flex-start / flex-end / center / baseline / stretch; }
```

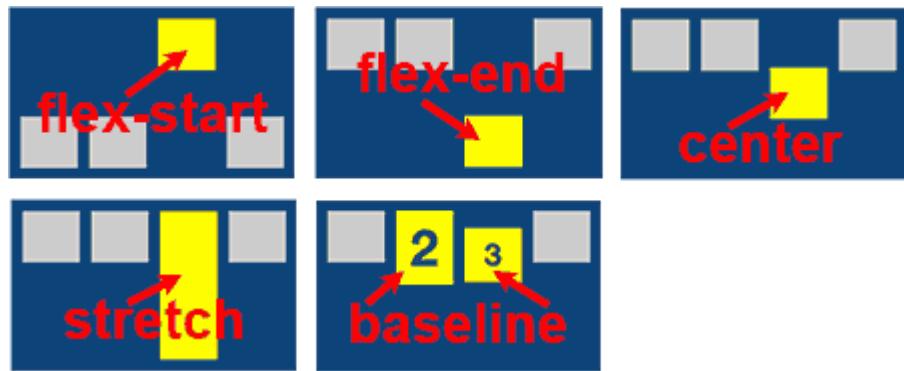
`flex-start`: el elemento aparece al principio (*start*) del eje transversal.

`flex-end`: el elemento aparece al final (*end*) del eje transversal.

`center`: el elemento posicionado aparece en el centro (*center*) de la caja flex.

`stretch` (el valor por defecto): el elemento aparece estirado (*stretched*) para ocupar el espacio restante.

`baseline`: los elementos aparecen alineados relativamente a su línea de base (*baseline*).



## align-self: flex-start

Si queremos que el elemento (*item*) aparezca al principio (*start*) del eje transversal de la caja flex utilizamos `align-self: flex-start;`

```
.flex-container{
    display: flex;
    align-items: flex-start;
}

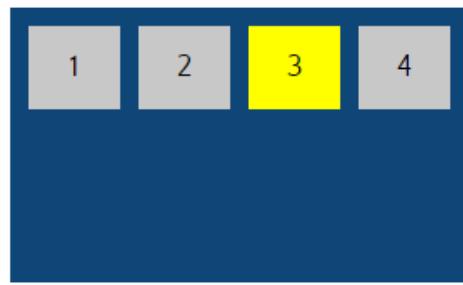
.flex-item:nth-of-type(3){background-color:yellow;}

.flex-container .flex-item *{
    width:100%;
    text-align:center;
    align-self: center;
    margin:10px;}

.flex-container .flex-start{

    align-self: flex-start;
}

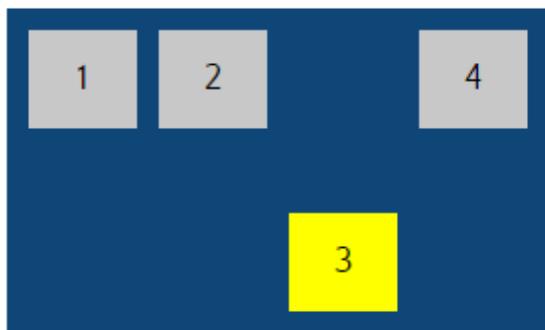
<div class="flex-container">
<div class="flex-item"><p>1</p></div>
    <div class="flex-item"><p>2</p></div>
    <div class="flex-item flex-start"><p>3</p></div>
    <div class="flex-item"><p>4</p></div>
</div>
```



## align-self: flex-end

Si queremos que el elemento (*item*) aparezca al final (*end*) del eje transversal de la caja flex utilizamos `align-self: flex-end;`

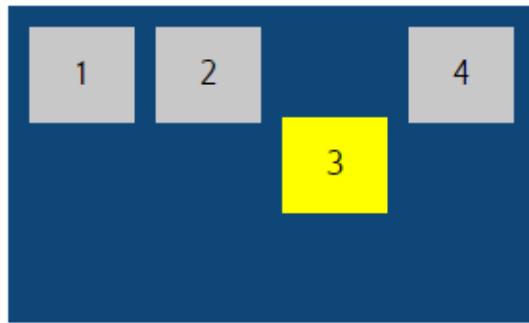
```
.flex-container .flex-end{  
    align-self: flex-end;  
}
```



## align-self: center

Si queremos que el elemento (*item*) aparezca en el centro (*center*) de la caja flex utilizamos `align-self: center;`

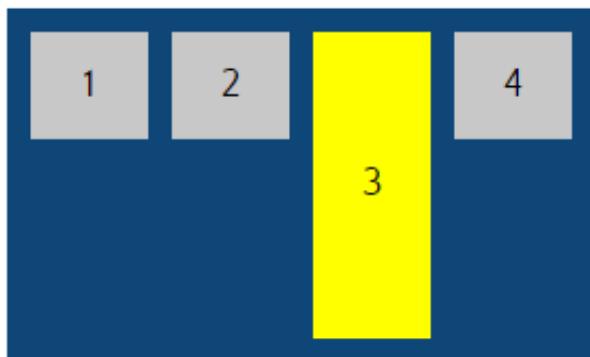
```
.flex-container .center{  
    align-self: center;  
}
```



## align-self: stretch

Si queremos que el elemento (*item*) aparezca estirado (*stretched*) ocupando el espacio restante, utilizamos `align-self: stretch;`

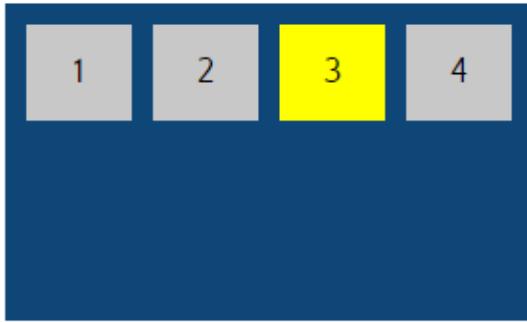
```
.flex-container .stretch{  
    align-self: stretch;  
}
```



## align-self: baseline

Si utilizamos `align-self: baseline;` los elementos (*items*) aparecen alineados relativamente a su línea de base (*baseline*).

```
.flex-container .baseline{  
    align-self: baseline;  
}
```



## 8.8 La propiedad flex

---

La propiedad **flex** es **una propiedad de los ítems** del contenedor flex.

Para que podamos escribir menos código, el CSS nos permite abreviar las tres propiedades: **flex-grow**, **flex-shrink** (opcional) y **flex-basis** (opcional) en una sola: **flex**. El valor por defecto es **flex: 0 1 auto**.

```
.item { flex: flex-grow [flex-shrink] [flex-basis]; }
.item {
    -webkit-flex: 0 1 auto;
    -ms-flex: 0 1 auto;
    flex: 0 1 auto;
}
```

Veamos las tres propiedades:

### La propiedad flex-grow

La propiedad **flex-grow** es **una propiedad de los ítems** del contenedor flex.

La propiedad **flex-grow** establece cuánto puede crecer un elemento flex en relación al resto de elementos de la misma caja flex. Su valor es un número.

El valor por defecto de **flex-grow: 0**, lo que quiere decir que el elemento no puede crecer. Si todos los ítems de una caja tienen el mismo valor de **flex-grow**, por ejemplo **flex-grow:1**; quiere decir que todos los ítems tienen que crecer en igual proporción, hasta ocupar todo el espacio disponible.

```
.flex-item{
    -webkit-flex-grow: 0;
    -ms-flex-grow:0;
    flex-grow:0;
}
```

**Veamos un ejemplo:** los contenedores `.flex-container` tienen una anchura `250px`. Anidados dentro de cada contenedor hay 4 ítems ( elementos flex ) cuya anchura declarada es de `10%`. Esto genera un espacio restante de `60%`.

Queremos que el segundo contenedor `.flex-container` quede completamente ocupado. Decidimos que el segundo ítem crecerá ocupando una parte del espacio restante (`flex-grow:1`), y el cuarto elemento dos partes (`flex-grow:2`). Los demás elementos siguen igual.

```
.flex-container{  
    width: 250px;  
    height: 70px;  
    padding:5px;  
    margin: 10px auto;  
    background-color:#124678;  
    display: flex;  
}  
  
[class^="flex-item"] p{  
    width:100%;  
    text-align:center;  
    align-self: center;  
    margin:0; }  
  
.flex-item{  
    display: inherit;  
    width:10%;  
    height:50px;  
    background-color:#ccc;  
    margin:5px;  
}  
  
.flex-item.flex-grow1{  
    flex-grow:1;  
}  
  
.flex-item.flex-grow2{
```

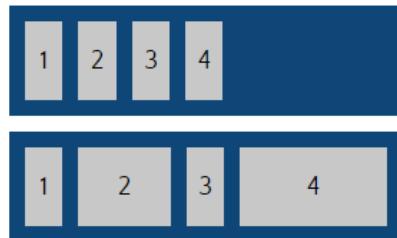
```

flex-grow:2;
}

<div class="flex-container">
  <div class="flex-item"><p>1</p></div>
  <div class="flex-item"><p>2</p></div>
  <div class="flex-item"><p>3</p></div>
  <div class="flex-item"><p>4</p></div>
</div>

<div class="flex-container">
  <div class="flex-item"><p>1</p></div>
  <div class="flex-item flex-grow1"><p>2</p></div>
  <div class="flex-item"><p>3</p></div>
  <div class="flex-item flex-grow2"><p>4</p></div>
</div>

```



## La propiedad flex-shrink

La propiedad **flex-shrink** es una propiedad de los ítems del contenedor flex.

La propiedad **flex-shrink** establece cuánto puede disminuir un elemento flex en relación al resto de elementos de la misma caja flex. Su valor es un número.

El valor por defecto de **flex-shrink: 1**, lo que quiere decir que los elementos de una caja flex disminuirán en igual proporción, por tal de acomodarse dentro de la caja.

```

.flex-item{
  -webkit-flex-shrink: 1;
  -ms-flex-shrink:1;
  flex-shrink:1;
}

```

**Veamos un ejemplo:** los dos contenedores `.flex-container` tienen una anchura `height: 250px`. Anidados dentro de cada contenedor hay 4 ítems ( elementos flex ) cuya anchura declarada es de 40%, y no puede haber cambio de línea ( por defecto `flex-wrap: nowrap` ). Esto genera un exceso de 60%. Si no hacemos nada, todos los ítems disminuirán por igual ( el primer contenedor ).

En el segundo contenedor `.flex-container` decidimos que el tercer ítem disminuirá dos veces más que los demás elementos ( `flex-shrink: 2;` )

```

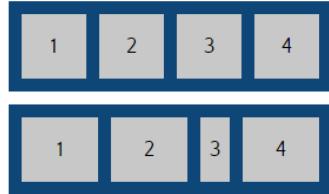
.flex-item1{
    display: inherit;
    width:40%;
    height:50px;
    background-color:#ccc;
    margin:5px;
}

.flex-item1.flex-shrink2{
    flex-shrink:2;
}

<div class="flex-container">
    <div class="flex-item1"><p>1</p></div>
    <div class="flex-item1"><p>2</p></div>
    <div class="flex-item1"><p>3</p></div>
    <div class="flex-item1"><p>4</p></div>
</div>

<div class="flex-container">
    <div class="flex-item1"><p>1</p></div>
    <div class="flex-item1"><p>2</p></div>
    <div class="flex-item1 flex-shrink2"><p>3</p></div>
    <div class="flex-item1"><p>4</p></div>
</div>

```



## La propiedad flex-basis

La propiedad `flex-basis` es una propiedad de los ítems del contenedor flex.

La propiedad `flex-basis` especifica el valor inicial del tamaño principal de un elemento flex, antes de que esté redimensionado con `flex-grow` o `flex-shrink`.

**Recuerde que:** el tamaño principal de un elemento flex es la anchura en contenedores horizontales - donde `flex-direction: row;` o la altura en contenedores verticales - donde `flex-direction: column.`

El valor por defecto es `flex-basis: auto`. En este caso la anchura base del ítem es igual a la anchura declarada (`width`) del elemento, o en su defecto, la anchura calculada por el navegador en base a su contenido.

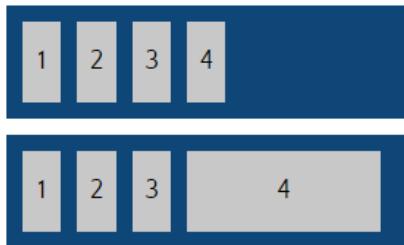
```
.item {
    -webkit-flex-basis: auto;
    -ms-flex-basis: auto;
    flex-basis: auto;
}
```

En el siguiente ejemplo los ítems flex (`.flex-item`) tienen una anchura `width:10%`. Para el cuarto elemento de la segunda caja (`.flex-basis50`) establecemos `flex-basis:60%`. Observamos como `flex-basis` sobrescribe la anchura declarada del elemento.

```
.flex-item {
    width:10%;
}
.flex-basis50 {
    flex-basis: 60%;
}
<div class="flex-container">
    <div class="flex-item"><p>1</p></div>
    <div class="flex-item"><p>2</p></div>
    <div class="flex-item"><p>3</p></div>
    <div class="flex-item flex-basis50"><p>4</p></div>
```

```
</div>

<div class="flex-container">
  <div class="flex-item"><p>1</p></div>
  <div class="flex-item"><p>2</p></div>
  <div class="flex-item"><p>3</p></div>
  <div class="flex-item"><p>3</p></div>
```



## 8.9 La propiedad order

---

Por defecto los elementos flex, como todos los elementos HTML aparecen en el mismo orden que en el código. En cajas flex podemos alterar este orden utilizando la propiedad `order`.

La propiedad `order` es **una propiedad de los ítems** del contenedor flex y su valor es generalmente un número entero, positivo o negativo.

```
.item { order: número / initial / inherit; }
```

El valor por defecto de `order` es 0. Todos los elementos flex con el mismo valor, aparecen en el mismo orden que en el código. En el siguiente ejemplo el tercer elemento `.flex-item` tiene el orden `order:-1`, lo que hace que se desplace delante de los demás elementos `.flex-item` que tienen el orden `order: 0` (el valor por defecto).

Por otra parte el primer elemento `.flex-item` tiene el orden `order: 1`, y por lo tanto aparece detrás de los demás elementos.

```
.flex-item:nth-of-type(3){ order:-1; background-color:yellow; }
.flex-item:nth-of-type(1){ order: 1; background-color:tomato; }

.flex-container{
```

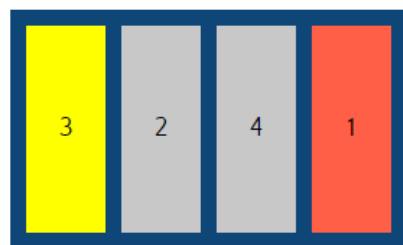
```
width: 250px;
height:150px;
padding:5px;
margin: 10px auto;
background-color:#124678;
display: flex;
}

.flex-item{
    display: inherit;
    width:50px;
    background-color:#ccc;
    margin:5px;
}

.flex-container .flex-item *{
    width:100%;
    text-align:center;
    align-self: center;
    margin:10px;}

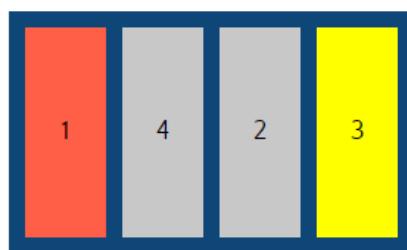
.flex-item:nth-of-type(3){ order:-1; background-color:yellow;}
.flex-item:nth-of-type(1){ order:1; background-color:tomato; }

<div class="flex-container">
    <div class="flex-item"><p>1</p></div>
    <div class="flex-item"><p>2</p></div>
    <div class="flex-item"><p>3</p></div>
    <div class="flex-item"><p>4</p></div>
</div>
```



**advertencia:** el orden depende en última instancia de la dirección de los elementos dentro del contenedor flex ( establecida con flex-direction, flex-wrap o flex-flow ). La única diferencia entre el siguiente ejemplo y el ejemplo anterior es la dirección de los elementos determinada por la propiedad **flex-direction**.

```
.flex-container.row-reverse{
    flex-direction: row-reverse;
}
```



Recursos: Entender FLEX-BOX: [Link](#)

### Entender Flexbox

#### Propiedades del contenedor flex

**FLEX-DIRECTION** ( una propiedad del contenedor flex )

row:  row-reverse:  column:  column-reverse:

**FLEX-WRAP** ( una propiedad del contenedor flex )

nowrap:  wrap:  wrap-reverse:

**ALIGN-ITEMS** ( una propiedad del contenedor flex )

flex-start:  flex-end:  center:  baseline:  stretch:

#### Propiedades de los ítems flex

**ALIGN-SELF** ( una propiedad de los ítems flex )

auto:  flex-start:  flex-end:  center:  baseline:  stretch:

**FLEX-GROW** ( una propiedad de los ítems flex )

elemento 1:  elemento 2:  elemento 3:  elemento 4:  elemento 5:

**FLEX-SHRINK** ( una propiedad de los ítems flex )

Documentación en MDN: [link](#)

The screenshot shows the MDN Web Docs website with the URL [https://developer.mozilla.org/es/docs/Web/CSS/Flexbox](#). The page title is "Flexbox". The main content area contains several code snippets demonstrating flexbox usage. On the left, there's a sidebar with navigation links like "Styling text" and "CSS layout". On the right, there's a sidebar titled "In this article" with a link to "¿Por qué flexbox?". The top of the page features the MDN logo, a "30 day free trial" button, and various navigation links including "References", "Guides", "Plus", "Blog", "Play", "AI Help", "Theme", "Log in", "Sign up for free", and "Español". A banner at the top indicates that the page was translated from English by the community.

# Capítulo 9. GRID LAYOUT

## Que es el grid layout

El grid layout es un sistema bidimensional, que transforma un elemento HTML en una cuadrícula. Los elementos hijos de este pueden ser posicionados dentro de las celdas de esta cuadrícula, o en áreas definidas arbitrariamente con reglas CSS.

## Utilizar prefijos

La buena noticia es que aparte de IE y Edge donde se necesitan prefijos (`-ms-`), en los demás navegadores los prefijos resultan innecesarios.

### 9.1 Grid layout – un ejemplo básico

En el HTML tenemos un `div id="cuadrícula"`, dentro del cual aparecen anidados otros 5 `div class="ítem"`.

```
<div id="cuadricula">
  <div class="ítem"><p>1</p></div>
  <div class="ítem"><p>2</p></div>
  <div class="ítem"><p>3</p></div>
  <div class="ítem"><p>4</p></div>
  <div class="ítem"><p>5</p></div>
</div>
```

En el CSS lo más importante son estas líneas de código:

```
#cuadricula {
  /*transforma la #cuadrícula en un contenedor grid:*/
  display: grid;
  /*esto genera 3 columnas iguales cuya anchura es 1fr */
  grid-template-columns:1fr 1fr 1fr;
  /*establece el espacio entre los ítems del grid*/
  grid-gap:1em;
}
```

La primera línea de código transforma la `#cuadricula` en un contenedor grid:

```
display: grid;
```

La segunda línea de código crea la estructura de columnas:

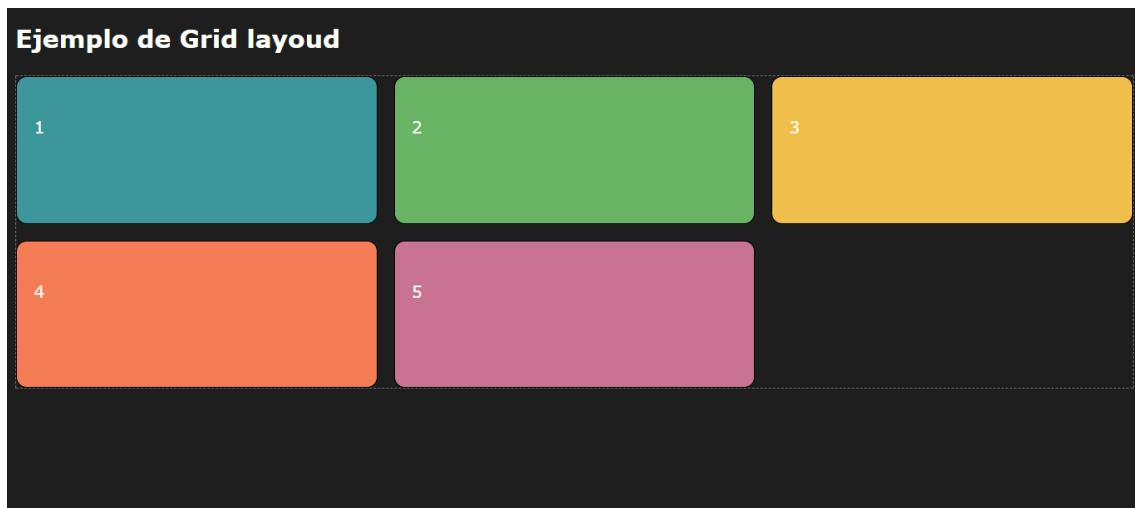
```
grid-template-columns:1fr 1fr 1fr;
```

La propiedad `grid-template-columns` establece:

- La `#cuadricula` tiene tres columnas.
- La anchura de cada columna dentro del contenedor grid es de una fracción de grid ( `1fr` ), o sea, en este caso, la tercera parte del ancho de la `#cuadricula`.

En la tercera línea de código, la propiedad `grid-gap` establece el tamaño del espacio que tiene que haber entre columnas, en este caso una distancia de `1em`:

```
grid-gap:1em;
```



## Propiedades del contenedor grid

Propiedad	Posibles valores	Descripción	Defecto
<b>display</b>	<i>grid   inline-grid   subgrid;</i>	Transforma un elemento HTML en un contenedor grid. Observación: las propiedades <code>column</code> , <code>float</code> , <code>clear</code> , y <code>vertical-align</code> no tienen ningún efecto dentro de un contenedor grid.	
<b>grid-template-columns</b> <b>grid-template-rows</b>	<i>[linea] 1fr [linea] 1fr [linea] 1fr;</i>	Crea la estructura de columnas ( <code>columns</code> ) o filas ( <code>rows</code> ) del contenedor grid. Toma una lista nombres de líneas ( entre corchetes, opcional ) y valores separadas por espacios en blanco.	none
<b>grid-template-areas</b>	<i>grid-template-areas:"a a a" "b c c" "b c c";</i>	Define las áreas del contenedor grid. Toma como valor una lista de nombres para las áreas del contenedor o <code>none</code> ( no define ninguna área )	none
<b>grid-rows-gap</b> <b>grid-column-gap</b>	<i>grid-rows-gap: 1em; grid-columns-gap:.5em;</i>	Definen el espacio que tiene que haber entre las columnas y las filas del contenedor grid.	0
<b>grid-gap</b>	<i>grid-gap: grid-rows-gap / grid-columns-gap</i>	es un método abreviado para <code>grid-row-gap</code> ( espaciado de fila ) y <code>grid-column-gap</code> ( espaciado de columna )	0
<b>grid-auto-flow</b>	<i>row   column   row dense   column dense</i>	Los ítems que no están explícitamente colocados en el contenedor grid, se disponen automáticamente en filas (rows). Para reordenar los ítems del contenedor grid utilizamos la propiedad <code>grid-auto-flow</code> .	row
<b>grid-auto-columns</b> <b>grid-auto-rows</b>	<i>grid-auto-columns:100px; grid-auto-rows:100px;</i>	Establece el tamaño de los ítems situados fuera del grid explícito.	auto
<b>justify-items</b> <b>align-items</b>	<i>justify-items: start   end   center   stretch;</i>	Alinea horizontalmente ( <code>justify-items</code> ) y verticalmente ( <code>align-items</code> ) los ítems dentro de	stretch

		las celdas o de las áreas del contenedor grid.	
<b>justify-content align-content</b>	<i>start   end   center   stretch   space-around   space-between   space-evenly;</i>	Alinea las columnas y las filas del grid relativo al contenedor grid.	start

## 9.2 Las propiedades `grid-template-columns` y `grid-template-rows`.

En el ejemplo anterior hemos visto la propiedad `grid-template-columns` en acción. También hay otra propiedad: `grid-template-rows` (`rows == filas`) utilizada para crear la estructura de filas del contenedor grid.

```
#cuadricula {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    grid-template-rows: 6rem max-content;
    grid-gap: 1em;
}
```

En este caso la propiedad `grid-template-rows` crea una primera fila (`row`) que tiene una altura de `6em`, y una segunda fila cuya altura depende del contenido de los ítems de esta (`max-content`):



## 9.3 La propiedad grid-template-areas

( una propiedad del contenedor grid )

Un área del contenedor grid es un espacio definido entre cuatro líneas: dos líneas verticales y dos horizontales. Dentro de esta área pueden haber, o no, uno o más ítems grid.

Para poder referenciar estas áreas necesitamos definirlas utilizando la propiedad [grid-template-areas](#).

```
.grid {
    /* transforma el elemento en un contenedor grid*/
    display: grid;
    /*define las columnas y las filas del contenedor grid*/
    grid-template-columns:1fr 10em 10em 1fr;
    grid-template-rows:5em 10em 5em;
    /*define las áreas del grid*/
    grid-template-areas:"h h h h"
                        "l . c r"
                        "l f f f";
}
```

Miremos con atención la última regla CSS: el valor de [grid-template-areas](#) es una sucesión de tres cadenas de texto, una para cada fila definida con [grid-template-rows](#). Dentro de cada cadena de texto aparecen exactamente 4 valores: uno para cada columna definida con [grid-template-columns](#). Los nombres utilizados en este caso son **h, l, ., c, r y f**, pero podemos utilizar los nombres que deseamos.

Podemos utilizar estos nombres para definir la posición y la extensión de los ítems grid utilizando la propiedad [grid-area](#):

```
.h{grid-area: h;}
.l{grid-area: l;}
.c{grid-area: c;}
.r{grid-area: r;}
.f{grid-area: f;}
```

Si no utilizamos una de las áreas definidas, esta se queda vacía. En este caso el nombre del área vacía es un punto.

```
.grid {
  display: grid;
  grid-template-columns: 1fr 10em 10em 1fr;
  grid-template-rows: 5em 10em 5em;
  /*****
  grid-template-areas: "h h h h"
    "l . c r"
    "l f f f";
  *****/
  grid-gap: .5em;

  margin: 1em auto;
  border: 1px dashed #777;
  width: 31em;
}
```

#### Ejemplo 3 de Grid layout "grid-template-area "



## 9.4 La propiedad grid-template

La propiedad `grid-template` es el método abreviado (*shorthand*) para declarar las propiedades `grid-template-rows`, `grid-template-columns` y `grid-template-areas` en una sola línea de código. La [especificación](#) nos dice que su valor puede ser:

```
none |
[ <'grid-template-rows'> / <'grid-template-columns'> ] |  
[ <line-names>? <string> <track-size>? <line-names>? ]+ [ /  
<explicit-track-list> ]?
```

Veamos qué quiere decir.

## none

El valor de grid-template puede ser `none`. En este caso no hay filas ni columnas ni líneas. Tampoco hay áreas del grid definidas.

```
grid-template: none;
[<'grid-template-rows'> / <'grid-template-columns'> ]
```

Esto define primero las filas (*rows*) y después las columnas (*columns*) en una sola declaración.

Ejemplos:

```
grid-template: 1fr 2fr / 1fr 1fr 1fr;
```

es equivalente de:

```
grid-template-rows: 1fr 2fr;
grid-template-columns: 1fr 1fr 1fr;
```

Veamos otro ejemplo. Esta vez también definimos las líneas del grid. ( Como ya se sabe el nombre de las líneas que definimos aparece entre corchetes. )

```
grid-template: [r1] 1fr [r2] 2fr [r3]
              / [c1] 1fr [c2] 1fr [c3] 1fr [c4];
```

es equivalente de:

```
grid-template-rows:[r1] 1fr [r2] 2fr [r3];
grid-template-columns: [c1] 1fr [c2] 1fr [c3] 1fr [c4];
```

## 9.4 La propiedad grid-gap

( una propiedad del contenedor grid )

La propiedad `grid-gap` define el espacio que tiene que haber entre las columnas y las filas del contenedor grid, y es un método abreviado para `grid-row-gap` (espaciado de fila) y `grid-column-gap` (espaciado de columna).

Por ejemplo esta regla;

```
.grid{grid-gap:1em;}
```

Establece que la distancia entre las celdas del contenedor grid es de `1em`. O sea: la distancia entre las columnas es igual a la distancia entre las filas del contenedor.

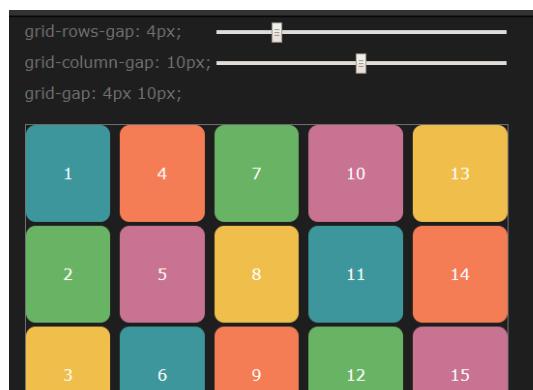
Esta otra regla:

```
.grid{grid-gap:1em .5em;}
```

Establece que la distancia entre las filas del grid es de `1em` mientras que la distancia entre las columnas es de `.5em`.

Los locuaces pueden escribir lo mismo utilizando dos reglas CSS:

```
.grid{
  grid-row-gap: 1em;
  grid-column-gap: .5em;
}
```



## Enlace

### 9.5 La propiedad grid-auto-flow

( una propiedad del contenedor grid )

Podemos reordenar los ítems del contenedor grid utilizando la propiedad `grid-auto-flow`.

La propiedad `grid-auto-flow` puede tomar una de estas valores:

```
.contenedor{
  grid-auto-flow: row | column | row dense | column dense
}
```

El valor por defecto de `grid-auto-flow` es `row`, lo que quiere decir que los elementos se disponen automáticamente en filas.

Si `grid-auto-flow: column` los elementos se disponen en columnas.



## Enlace

Para entender lo que puede hacer la palabra clave `dense`, necesitamos hacer unos pequeños cambios al ejemplo anterior:

```
.item:nth-child(2n+1){grid-row-end:span 2;}
.item:nth-child(2n){grid-column-end:span 2;}
```

Esto hace que algunos ítems sean más anchos o más altos que los demás, y por lo tanto ya no encajan perfectamente y aparecen huecos en la estructura del grid. Y aquí la cosa se pone interesante.

Si añadimos la palabra clave `dense`:

```
.grid{
    grid-auto-flow: row dense;
}
```

el algoritmo auto-flow intenta calcular de nuevo la posición de cada ítem por tal de hacerlos encajar.

## Enlace

## 9.6 Justificar y alinear dentro del contenedor grid

Cuando hablamos de grid, **justificar** (*justify*) actúa horizontalmente mientras que **alinear** (*align*) actúa verticalmente .

### Las propiedades justify-items & align-items ( dos propiedades del contenedor grid )

Utilizamos las propiedades **justify-items** y **align-items** para alinear horizontalmente (**justify-items**) y verticalmente (**align-items**) los ítems dentro de las celdas o de las áreas del contenedor grid.

Estos atributos pueden tomar uno de estos valores:

```
.contenedor{
  justify-items: start | end | center | stretch;
  align-items: start | end | center | stretch;
}
```

En idiomas como el castellano, con un sistema de escritura de izquierda a derecha **direction: ltr;** (*left to right*),

**justify-items: start** alinea los ítems a la izquierda,  
**justify-items: end** alinea los ítems a la derecha.

La otra propiedad

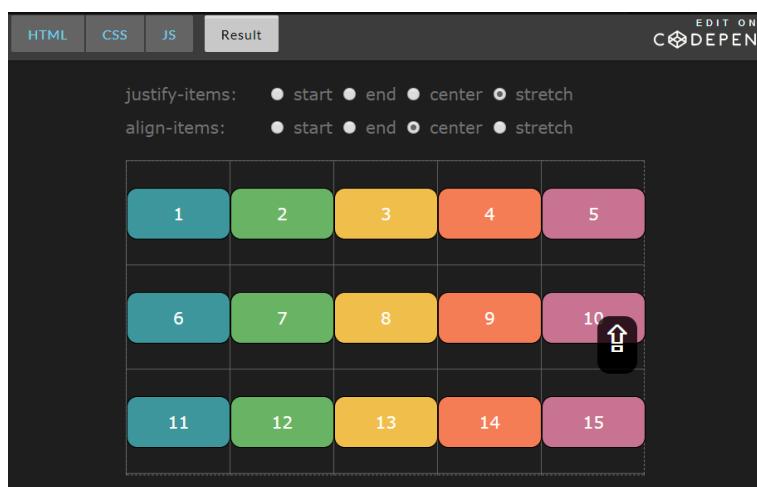
**align-items: start** alinea los ítems a la parte superior de la celda o área  
**align-items: end** alinea los ítems a la parte inferior de la celda o área.

Si utilizamos estas palabras clave:

**center**: el ítem aparece en el centro (*center*) de la celda o área,

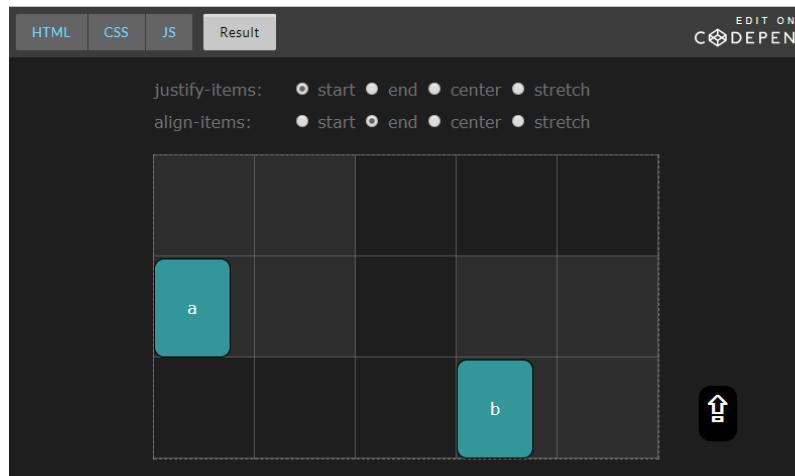
**stretch**: el ítem aparece estirado (*stretched*) para ocupar toda la celda o área.

Alineación de los ítems dentro de las celdas del grid:



## Enlace

Alineación de los ítems dentro de las áreas del grid:



## Enlace

### 9.7 Las propiedades justify-content y align-content ( dos propiedades del contenedor grid )

Hay situaciones en las cuales el contenedor es más grande que el grid. Esto puede pasar si dimensionamos el grid relativo a la ventana, pero las columnas y las filas tienen dimensiones fijas (en px o en em). En estos casos podemos alinear las columnas y las filas del grid relativo al contenedor utilizando las propiedades `justify-content` y `align-content`. Los valores que pueden tomar estas dos propiedades son:

```
.contenedor{
    justify-content: start | end | center | stretch | space-around |
    space-between | space-evenly;

    align-content: start | end | center | stretch | space-around |
    space-between | space-evenly;
}
```

**start** ( el valor por defecto ): los elementos aparecen agrupados al principio ( *start* ) del eje horizontal ( para `justify-content` ) o vertical ( para `align-content` ).

**end**: los elementos aparecen agrupados al final ( *end* ) del eje horizontal ( para `justify-content` ) o vertical ( `align-content` ).

**center**: los elementos aparecen agrupados al centro ( *center* ).

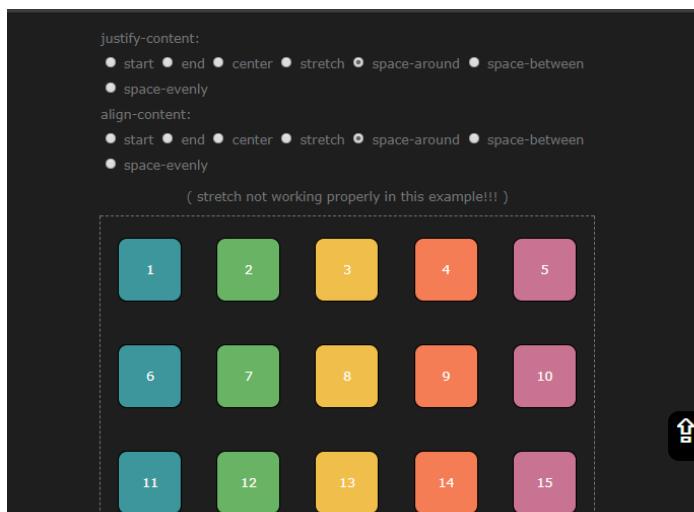
**stretch**: los elementos aparecen estirados ( *stretched* ) para ocupar el espacio restante.

**space-around**: los elementos aparecen distribuidos uniformemente, y con un espacio igual entre ellos. ( medio espacio entre los bordes del contenedor y los ítems ).

**space-between**: los elementos aparecen distribuidos uniformemente: al principio, en el centro y al final del eje ( sin espacio entre los bordes del contenedor y los ítems ).

**space-evenly**: los elementos aparecen distribuidos uniformemente, y con un espacio igual entre ellos y entre ellos y los bordes del contenedor.

Si esto no parece muy claro, pruebe esta demostración:



[Enlace](#)

## 9.8 Propiedades de los ítems grid

Propiedad	Posibles valores	Descripción
<a href="#"><u>grid-column-start</u></a>	<i>el nombre o el número de la línea</i>	Definen la posición y la extensión de un elemento (ítem) dentro del contenedor grid.
<a href="#"><u>grid-column-end</u></a>		
<a href="#"><u>grid-row-start</u></a>		

<u>grid-row-end:</u>		
<u>grid-column</u> <u>grid-row</u>	$grid\text{-}row: grid\text{-}row\text{-}start/grid\text{-}row\text{-}end;$ $grid\text{-}column: grid\text{-}column\text{-}start/grid\text{-}column\text{-}end;$	Declaraciones abreviadas para grid-row-start/grid-row-end, y grid-column-start/grid-column-end respectivamente.
<u>grid-area</u>	$grid\text{-}area: grid\text{-}row\text{-}start / grid\text{-}column\text{-}start /$ $grid\text{-}row\text{-}end / grid\text{-}column\text{-}end;$	Otra manera abreviada de definir la posición y la extensión de un ítem dentro del contenedor grid
<u>justify-self</u> <u>align-self</u>	$start   end   center   stretch;$	se utilizan para alinear los elementos relativo al eje de la fila (justify-self) o de la columna (align-self).
<u>order</u>	<i>número entero</i>	Define el orden para los ítems del grid

## 9.10 Las propiedades grid-column y grid-row

(dos propiedades de los ítems grid)

A veces necesitamos definir la posición y la extensión de un elemento ( ítem ) dentro del contenedor grid. Por ejemplo, para decir que el elemento se sitúa entre la segunda y la tercera línea vertical y entre la segunda y la tercera línea horizontal tenemos que escribir estas reglas:

```
.item{
    grid-column-start: 2;
    grid-column-end: 3;
```

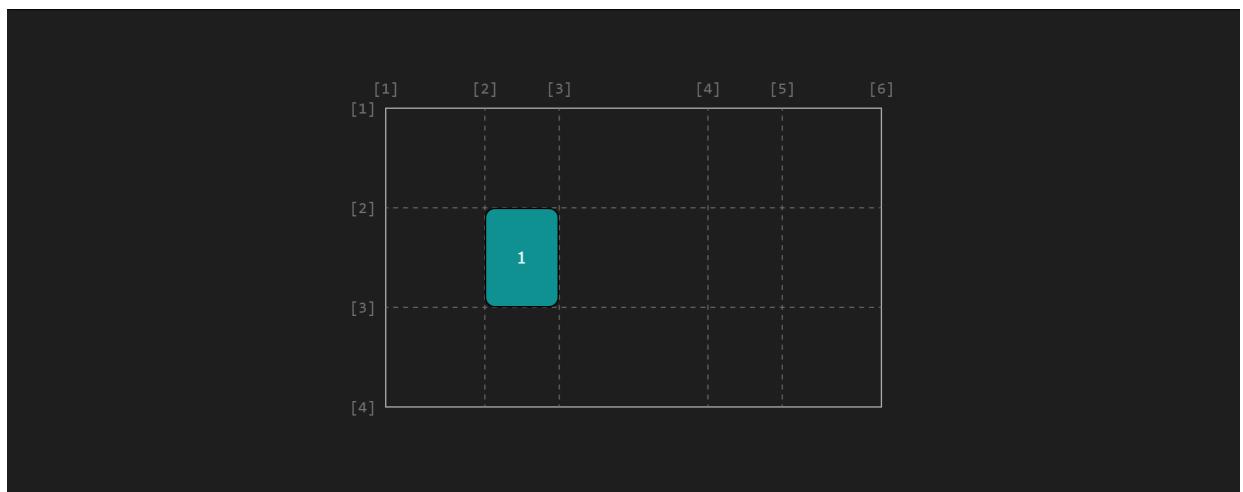
```
grid-row-start: 2;
grid-row-end: 3;
}
```

Podemos escribir lo mismo utilizando solo dos líneas de código:

```
.item{
    grid-column: 2 / 3;
    grid-row: 2 / 3;
}
```

En estos ejemplos los números utilizados son los nombres de las líneas verticales y horizontales definidas con [grid-template-columns](#) y/o [grid-template-rows](#).

### Ejemplo:



También podemos utilizar la palabra clave [span](#) (*se extiende*). Por ejemplo, [span 1](#) quiere decir que el ítem se extiende exactamente una celda, fuera el que fuera el tamaño de esta.

```
.item{
    grid-column: 2 / span 1;
    grid-row: 2 / span 1;
}
```

Todavía más: podemos escribirlo todo utilizando una sola línea de código. Pero para esto necesitamos utilizar otra propiedad: [grid-area](#).

## La propiedad grid-area

(una propiedad de los ítems grid)

La propiedad [grid-area](#) es una manera abreviada para definir la posición y la extensión de un ítem dentro del contenedor grid. La sintaxis es la siguiente:

```
.item{
    grid-area: <grid-row-start> / <grid-column-start> / <grid-row-end>
    / <grid-column-end>
}
```

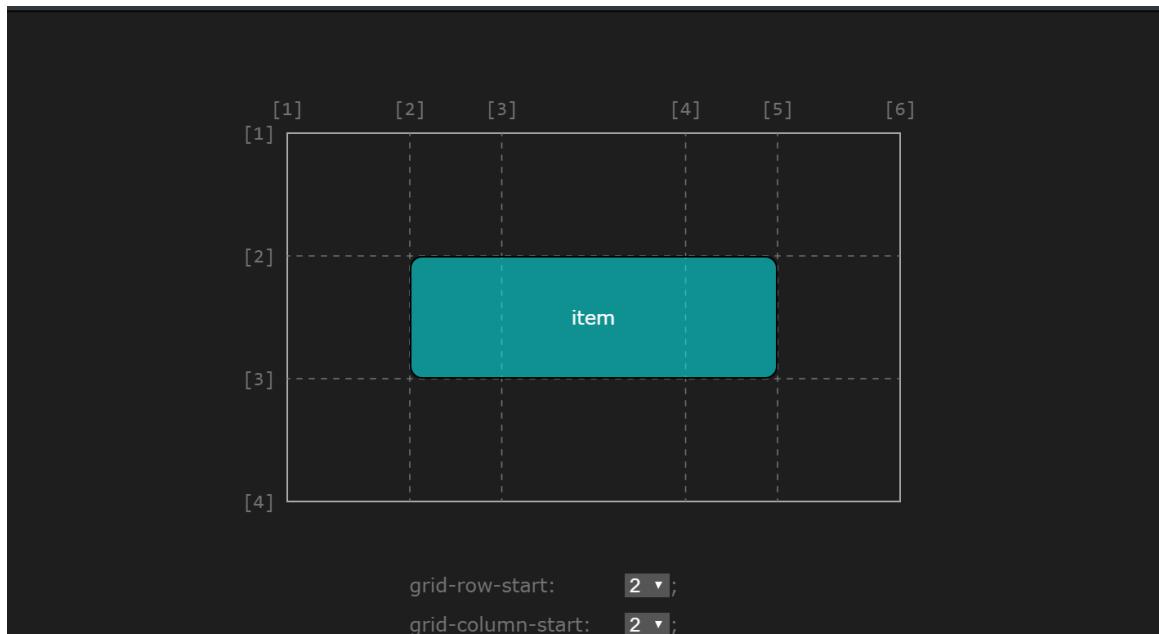
O sea: podemos tomar esto:

```
.item{
    grid-row-start: 2;
    grid-column-start: 2;
    grid-row-end: 3;
    grid-column-end: span 3;
}
```

Y transformarlo en esto:

```
.item{
    grid-area: 2 / 2 / 3 / span 3;
}
```

Ejemplo: [Link](#)



## 9.11 Las propiedades justify-self y align-self

(dos propiedades de los ítems grid)

Tanto **justify-self** como **align-self** son propiedades de los ítems (NO del contenedor) grid, y se utilizan para alinear los elementos relativos al eje de la fila (**justify-self**) o de la columna (**align-self**).

Los valores que pueden tomar estas dos propiedades son:

**align-self: start | end | center | stretch;**

**justify-self: start**: alinea el contenido del elemento a la izquierda.

**align-self: start**: alinea el contenido del elemento a la parte superior.

**justify-self: end**: alinea el contenido del elemento a la derecha.

**align-self: end**: alinea el contenido del elemento a la parte inferior.

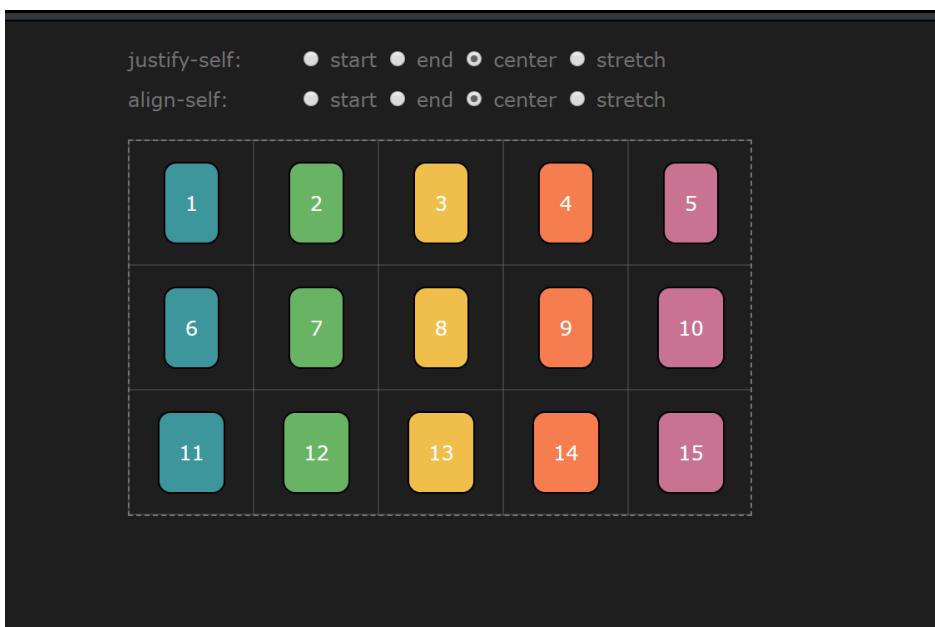
**justify-self: center**

`align-self: center`: alinea el contenido del elemento en el centro.

`justify-self: stretch`

`align-self: stretch`: el contenido aparece estirado (*stretched*) para ocupar todo el espacio.

Ejemplo:



## La propiedad `order`

(una propiedad de los ítems grid)

Los ítems de un contenedor grid, como todos los elementos HTML aparecen en el mismo orden que en el código. Podemos alterar este orden utilizando la propiedad `order`.

Por defecto los elementos no tienen definido un `order` (orden). Si definimos el atributo `order = "1"` para un ítem cualquiera, este aparecerá en la última posición dentro de la caja, ya que 1 es más grande que nada.

En el siguiente ejemplo primero definimos el orden para cada elemento del grid: el primer ítem: `order:1`; el segundo ítem: `order:2` ... etc.

```
var items = document.querySelectorAll("#cuadricula .item");
for(var i = 0; i < items.length; i++){
    items[i].style.order = i+1;
}
```

Cada vez que el `input type="number"` cambia, el valor del atributo `order` del primer ítem cambia también, y por lo tanto cambia también la posición de este.

```
orderInput.addEventListener("input", function(){
    items[0].style.order = orderInput.value;
},false);
```

### Ejemplo:

grid-auto-flow: column;  
 grid-auto-flow: row;

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

The first item has order:1;  
 The second item has order:2; . . .etc  
 You can modify the order of the first item by using the input type number.

## 9.9 Algunas palabras clave

---

### Una fracción de grid

Aunque podemos utilizar cualquier otras unidades para longitud disponibles en CSS ( [px](#), [%](#), [em](#), [rem](#) ...etc ), es importante saber que el grid tiene una unidad específica: [fr](#) que representa una fracción del espacio disponible dentro del contenedor grid.

Veamos un caso muy sencillo. Si queremos crear una estructura de 3 columnas , podemos escribir:

```
grid-template-columns:1fr 1fr 1fr;
```

En este caso cada una de las tres columnas tiene una anchura de 33.33%.

Si queremos que la columna central sea más ancha podemos escribir:

```
grid-template-columns:1fr 2fr 1fr;
```

Esta vez dividimos el espacio disponible por 4 (  $1\text{fr} + 2\text{fr} + 1\text{fr} == 4\text{fr}$  ), y en este caso la primera y la tercera columna tendrán una anchura de 25%, mientras que la columna central tendrá una anchura de 50%.

No es absolutamente necesario utilizar números enteros. También podemos usar números decimales. Por ejemplo, esta es una declaración perfectamente válida, i el resultado es idéntico al de antes:

```
grid-template-columns:0.5fr 1fr 0.5fr;
```

Tampoco es necesario utilizar exclusivamente fracciones de grid ([fr](#)). Podemos utilizar cualquier combinación de unidades de longitud, según sea necesario.

En este caso:

```
grid-template-columns:1fr 100px 10em;
```

la primera columna ocupa todo el espacio disponible dentro del contenedor grid, y es equivalente a:

```
grid-template-columns:calc(100% - 100px - 10em) 100px 10em;
```

En este caso, también podemos utilizar la palabra clave `auto` con el mismo efecto:

```
grid-template-columns:auto 100px 10em;
```

## El método `repeat()`

Para no repetirnos tanto (`grid-template-rows:1fr 1fr 1fr;`) podemos utilizar el método `repeat()`:

```
grid-template-columns: repeat(3, 1fr);
```

donde el primer argumento (`3`) especifica cuantas veces tiene que repetirse la sucesión especificada por el segundo argumento (`1fr`).

```
#cuadricula {
    display: grid;
    /*grid-template-columns:1fr 1fr 1fr;*/
    grid-template-columns: repeat(3, 1fr);
    grid-gap:1em;
}
```

También podemos repetir una sucesión de columnas:

```
grid-template-columns: repeat(3, 1fr .5fr);
```

y esto es equivalente a:

```
grid-template-columns: 1fr .5fr 1fr .5fr 1fr .5fr;
```

O podemos utilizar una combinación, por ejemplo, algo así:

```
grid-template-columns: repeat(3, 1fr .5fr) 1fr;
```

que es equivalente a:

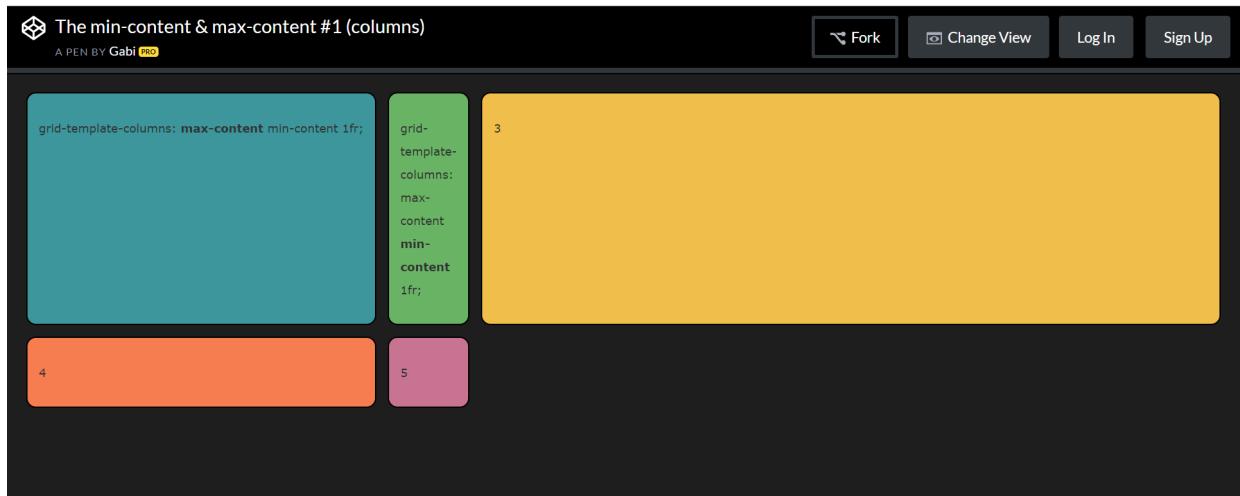
```
grid-template-columns: 1fr .5fr 1fr .5fr 1fr .5fr 1fr;
```

## Las palabras clave max-content y min-content

Si utilizamos **max-content** para definir una columna o una fila, esto quiere decir que esta tendrá el *tamaño* (anchura para columnas o altura para filas) mínimo necesario para encajar el contenido máximo.

Si utilizamos **min-content**, esto quiere decir que el carril (fila o columna ) tendrá el tamaño mínimo necesario para que no haya overflow.

### Ejemplo:



En el siguiente ejemplo la propiedad **grid-template-rows** crea una primera fila ( *row* ) que tiene una altura de 6em, y una segunda fila cuya altura depende del contenido de los ítems de esta ( **max-content** ).

De otra parte, el contenido del div naranja es una lista cuyos elementos <li> pueden ser editados.

```
<li contenteditable="true">
```

En el CSS los elementos li tienen `white-space: nowrap;` para prevenir los saltos de línea.

```
li{white-space: nowrap;}
```

Por favor edite los elementos de lista para ver `min-content` in acción. Empiece por el elemento de lista más largo.

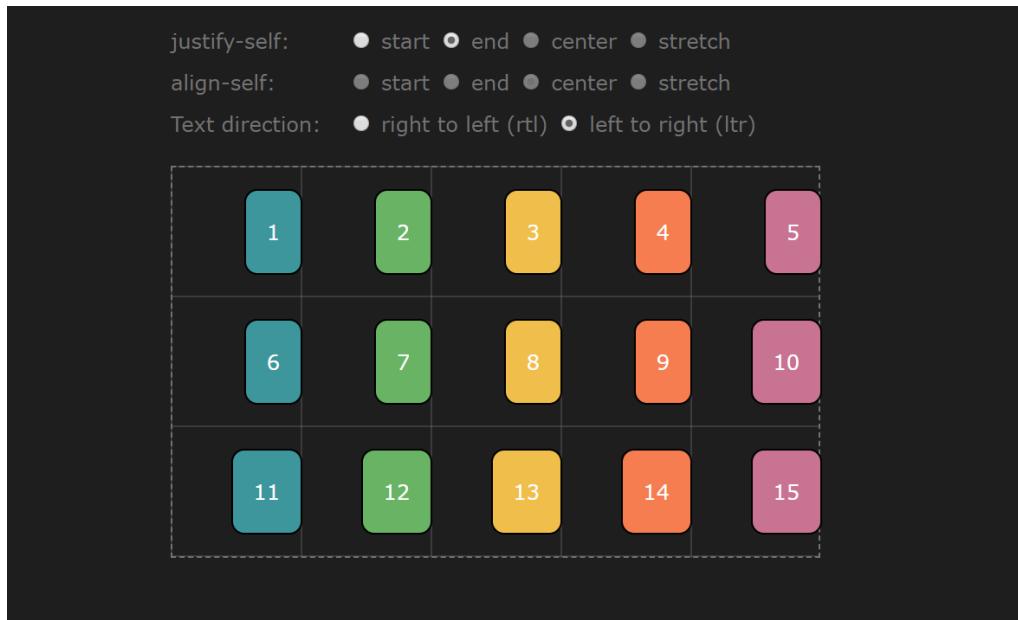
### Ejemplo:

The screenshot shows a CodePen interface with the title "GRID: min-content max-content". The grid layout consists of three columns. Column 1 (orange) contains a list of navigation items: Index, Plantilla básica, Plantilla elegante, Galería de imágenes, Plantilla WordPress básica, and Contact. Column 2 (pink) contains text explaining that the orange column's content is a list where items can be edited because they have `contenteditable="true"`. It also mentions that the CSS rule `li{white-space: nowrap;}` prevents line breaks. Column 3 (teal) contains a long Latin text snippet. The grid uses `min-content` for the first two columns and `max-content` for the third.

## Las palabras clave `start` y `end`

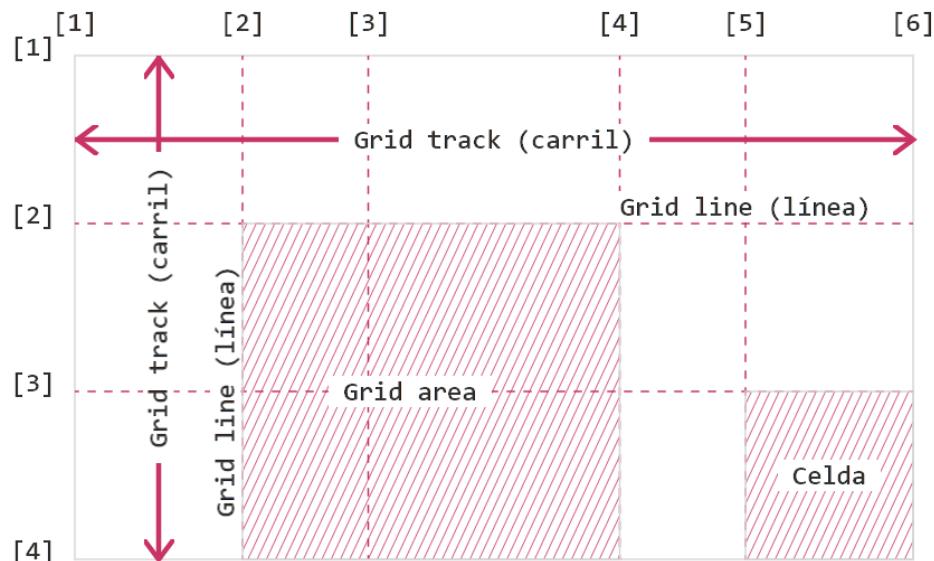
Dos de las palabras clave muy utilizadas: `start` (*inicio*) y `end` (*final*) pueden crear una cierta confusión. Es importante saber que en idiomas como el castellano, con un sistema de escritura de izquierda a derecha `direction: ltr;` (*left to right*), la palabra clave `start` se refiere a la izquierda para columnas y arriba para filas. De la misma manera la palabra clave `end` se refiere a la derecha del contenedor grid para columnas y abajo para filas.

### Ejemplo:



## Líneas y carriles

Cuando definimos las columnas y las filas (*tracks* o *carriles*) de un contenedor grid, automáticamente definimos las líneas (*lines*) de este.



En idiomas como el castellano, con un sistema de escritura de izquierda a derecha (`direction: ltr;`) , la primera línea vertical coincide con el borde izquierdo del contenedor

grid, y la podemos referenciar utilizando el número 1. De la misma manera la primera línea horizontal coincide con el borde superior del contenedor grid, y de nuevo la podemos referenciar utilizando el número 1.

Si no nos gusta utilizar números, o, si por alguna razón consideramos que pueden crear confusión, podemos dar nombres a las líneas utilizando esta sintaxis:

```
grid-template-columns: [col-1-a] 1fr [col-1-z col-2-a] 1fr [col-2-z col-3-a] 1fr  
[col-3-z];
```

En este ejemplo la primera línea se llama `col-1-a`, pero la podemos referenciar también utilizando el número 1. La segunda línea tiene dos nombres: `col-1-z` y `col-2-a` que aparecen en el código separados por un espacio en blanco (`[col-1-z col-2-a]`). Exactamente como antes, también la podemos referenciar por el número de orden, 2 en este caso.

¿Y para qué necesitamos todos estos nombres? Los necesitamos al definir la posición y extensión de los ítems grid, con las propiedades `grid-area`, `grid-row-start`, `grid-column-start`, `grid-row-end` y/o `grid-column-end`.

## Ejemplos

Estos son algunos ejemplos creados con Grid-Layout:

<https://silocreativo.com/labs/escape-from-a-christmas-tale/es/>

<https://labs.jensimmons.com/2017/01-011.html>

<https://codepen.io/julesforrest/full/QaBvPe>

# Capítulo 10 Responsive WEB

Responsive Web Design (o diseño web adaptativo) se trata de una técnica de diseño y desarrollo web por el que se consigue que un único sitio se adapte perfectamente a todos los dispositivos que puedan consumirlo, desde ordenadores de escritorio a netbooks, tablets, teléfonos móviles, televisores, etc. En definitiva, se trata de construir una única web para que se vea correctamente y aproveche las particularidades de todo dispositivo que hoy exista, o pueda existir en el futuro, independientemente de la pantalla en la que se muestre.

## 1.1 Ventajas

Las ventajas de utilizar un diseño web adaptativo son más que evidentes:

- Con **una sola versión** en HTML y CSS se cubren todas las resoluciones de pantalla, es decir, el sitio web creado estará optimizado para todo tipo de dispositivos: PCs, tabletas, teléfonos móviles, etc. Esto mejora la experiencia de usuario a diferencia de lo que ocurre, por ejemplo, con sitios web de ancho fijo cuando se acceden desde dispositivos móviles.
- **Reducción de costos.** Se reducen los costes ya que hasta hoy se debe hacer un portal para la Web y otro para dispositivos móviles. Esto origina mayores costos de creación y mantenimiento de la información.
- **Eficiencia en la actualización.** El sitio solo se debe actualizar una vez y se ve reflejada en todas las plataformas.
- **Mejora el posicionamiento SEO.** Google tiene distintos tipos de robots que acceden y escanean los sitios web indexados en su buscador emulando a los diferentes dispositivos que navegan por internet. Estos robots, valoran la accesibilidad de la web en los diferentes dispositivos y la tienen muy en cuenta a la hora de posicionar la web en los resultados de búsqueda.



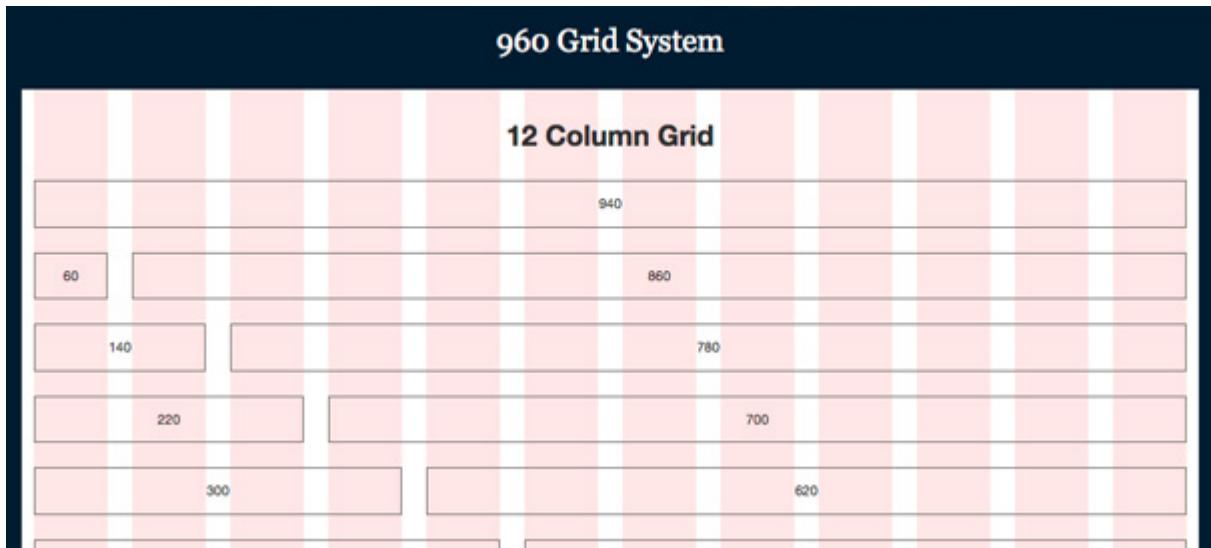
## 1.2 Los ingredientes

Los elementos básicos necesarios para construir un diseño adaptativo, son al menos los tres siguientes:

- Una estructura flexible, basada en rejilla.
- Imágenes, tablas, multimedia... flexible.
- *Media Queries*, como parte de la especificación de CSS 3.

## 10.2 Estructura de rejilla flexible

Los diseños basados en rejilla han sido utilizados ampliamente en sectores como el diseño gráfico, pero tardaron en ser adoptados por la web. Estos diseños consisten en una serie de columnas que permiten ordenar y estructurar los contenidos (texto, imágenes, media...) de una manera ordenada y uniforme. Una estructura clásica de rejilla es la que se muestra en la siguiente imagen:



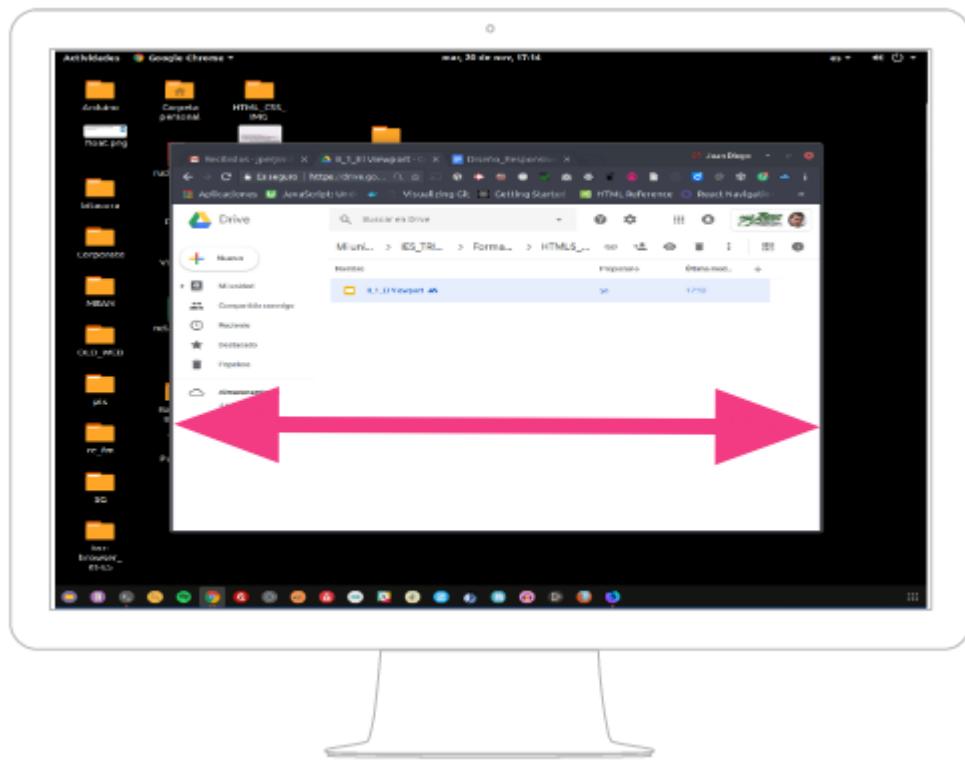
Se pueden observar claramente las líneas verticales y horizontales utilizadas para posicionar los elementos en el lienzo, así como los márgenes que separan los distintos elementos. De esta manera tan simple, es posible crear diseños con un aspecto visual uniforme.

### 10.2.1 VIEWPORT

Uno de los conceptos más importantes que debemos conocer antes de lanzarnos a realizar diseños responsivos es el concepto de **Viewport**.

*“Área de la pantalla en la que el navegador puede renderizar contenido, es decir, el espacio disponible para mostrar mi página web.”*

Este concepto es muy fácil de comprender cuando nos referimos a él en sistemas de escritorios o en laptops o portátiles. En estos casos el Viewport coincide con la pantalla de nuestro navegador.

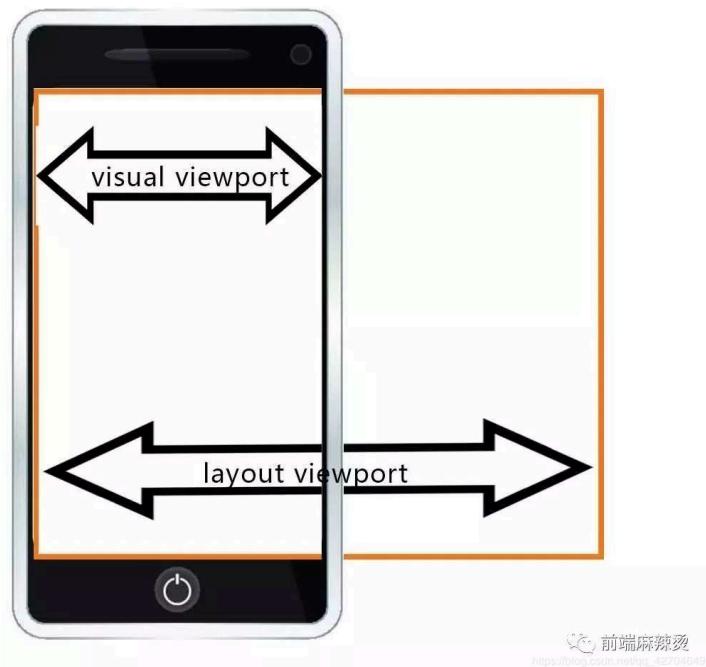


Sin embargo, si nos referimos a móviles y tablets estaremos hablando de otra cosa diferente.

Cuando este tipo de dispositivos irrumpieron, las páginas se maquetan usando normalmente una anchura fija en píxeles que solía variar entre 800px y 1000px que, obviamente, era mayor que la anchura de la pantalla de los móviles.

Ante esta circunstancia los fabricantes hicieron que este tipo de dispositivos tuvieran dos Viewports:

- El *Layout-viewport* que es el viewport que es tenido en cuenta para la aplicación de estilos. Suele ser de aproximadamente 960 px.
- El *Visual-viewport* que es el viewport que realmente ve el usuario.

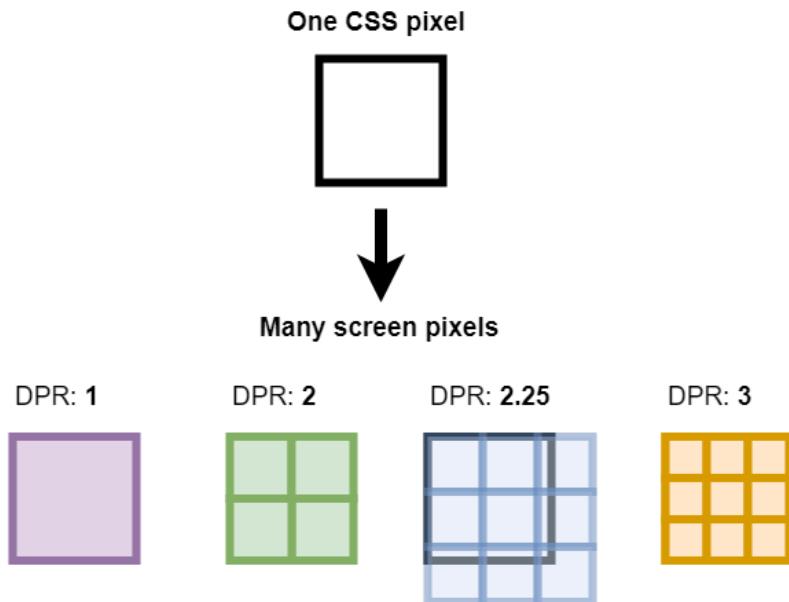


Además, sucede que en este tipo de dispositivos podemos hacer Zoom mediante gestos lo que provoca que:

- No cambie el *Layout-viewport*
- Cambie el *Visual-viewport*.

Y no acaban ahí los problemas. También, en este mundo de miles de dispositivos diferentes no vamos a encontrar con que tenemos que lidiar con diferentes valores para los:

- Hardware Pixels: son los píxeles de resolución que nos da la tarjeta gráfica.
- DPI(Device Independent Pixels): Que es la unidad de medida del navegador. Se relaciona con la distancia real y ocupan lo mismo independientemente de la densidad de píxeles de la pantalla.
- DPR(Device Pixel Ratio): Hw Pixel / DPI (es una dimensión).



Para gestionar toda esta situación, se añadirá una línea en la cabecera de la página Web como esta:

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
</head>
```

Con esa simple línea podremos empezar a maquetar diseños responsivos de manera eficiente e incluso faremos que las páginas que no están preparadas para móviles se muestren correctamente aunque tengamos, eso sí, que hacer continuos zooms para poder usarlas.

### 10.3 Crear una rejilla flexible

Mientras que los media query ofrecen la potencia real para desarrollar un sitio web adaptativo, es posible ahorrar trabajo y código utilizando una aproximación en base a rejillas flexibles. Utilizando este tipo de rejillas, podemos asegurar que el sitio web se redimensiona en función del espacio disponible, sin la necesidad de utilizar media query. Posteriormente es posible utilizar media query si es necesario realizar cambios significativos en la estructura de la web.

Vamos a ver cinco componentes que nos permitirán construir una rejilla flexible, y cómo utilizarlas:

- Fuentes flexibles
- Contenedores flexibles
- Márgenes flexibles
- Rellenos flexibles
- Imágenes flexibles

### **10.3.1 Fuentes flexibles**

Cuando hablamos de diseño responsive solemos dar más importancia al layout, pero el texto que vamos a presentar es igual de importante para conseguir un buen diseño.

Si no tenemos cuidado, nos podemos encontrar con varios problemas:

- Líneas cortas con pocos caracteres, lo que dificulta la lectura.
- Líneas largas con muchos caracteres, lo que también dificulta la lectura.
- Caracteres muy pequeños.

Lo ideal, según estudios, es una letra de un tamaño adecuado, para poder leer sin forzar la vista, y que se disponga formando líneas de entre 60-80 caracteres.

Para intentar solucionar esto con garantías lo más recomendable es utilizar para el texto unidades de tamaño relativas al viewport. Tendremos:

- **vw** en relación a la anchura del viewport.
- **vh** en relación a la altura del viewport.
- **vmin** el valor menor en relación a la dimensión pequeña del viewport (anchura o altura).
- **vmax** el valor máximo en relación a la dimensión más grande del viewport (anchura o altura).

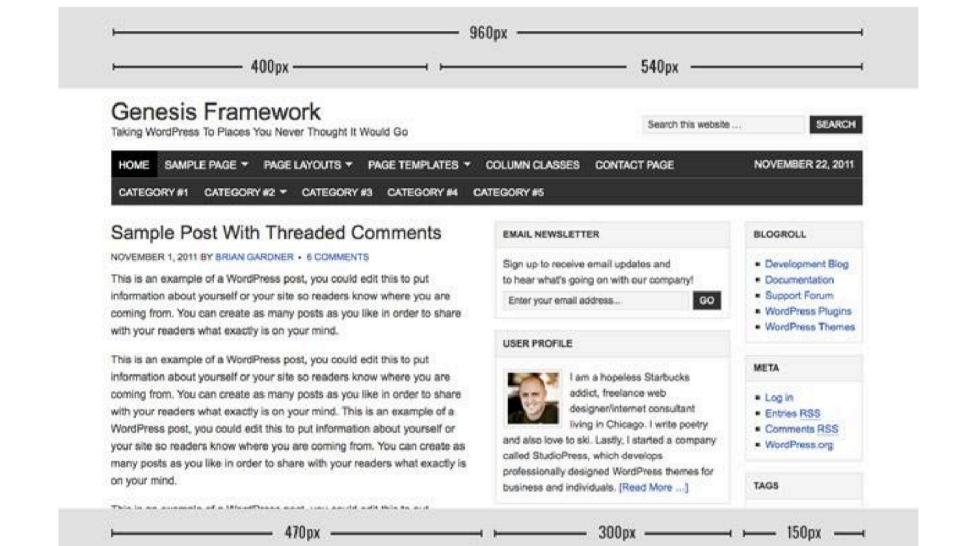
Teniendo en cuenta, por ejemplificar que 1vw es un 1% de la anchura del viewport.

No obstante, encontrar siempre el tamaño perfecto de texto es algo que puede ser complicado sino imposible. Sin embargo allí van unas posibles pautas:

- Calcular el tamaño mínimo y máximo que me puedo permitir usando calc() (función css)
- No importa si el texto hace “reflow” (pasa a la siguiente línea) en determinadas ocasiones.
- Mantener el tamaño perfecto de línea puede que sea imposible.
- Si tenemos que elegir es conveniente priorizar móviles y tablets.
- Algunos elementos, por ejemplo en un menú de navegación, puede tener sentido usar tamaños fijos de letra en vez de unidades relativas al viewport.

### 10.3.2 Contenedores flexibles

En la siguiente imagen se puede ver que se ha optado por un diseño de 960px de ancho, dividido en una cabecera con dos contenedores y un cuerpo con tres contenedores, ambos separados por un espacio de 20px.



Este es el HTML correspondiente a la estructura:

```
<div id="wrap">
  <div id="header">
    <div id="title-area"></div>
    <div class="widget-area"></div>
  </div>
</div>
```

```
<div id="inner">

<div id="content-sidebar-wrap">

<div id="content"></div>

<div id="sidebar"></div>

<div id="sidebar-alt">

</div>

</div>
```

Y la el código CSS base de la estructura:

```
#wrap { width: 960px; }

#header { width: 960px; }

#title-area { width: 400px; }

#header .widget-area { width: 540px; }

#inner { width: 960px; }

#content-sidebar-wrap { width: 790px; }

#content { width: 470px; }

#sidebar { width: 300px; }

#sidebar-alt { width: 150px; }
```

Conociendo los valores objetivo, la primera tarea es establecer nuestro ancho base. Esta medida hace referencia al **ancho** que hemos definido a nuestro layout sin aplicar ningún diseño adaptativo. Como medida general, suele adoptarse como ancho base un 90% del área visible de la pantalla, lo que permite seguir manteniendo toda la estructura y contenidos de la web siempre visibles.

A la hora de aplicar la fórmula, multiplicar el resultado por 100. Si aplicamos la fórmula a la estructura anterior, el resultado sería el siguiente:

```
#wrap {
```

```

width: 90%;

max-width: 960px;

}

#header { width: 100%; /* 960px/960 */ }

#title-area { width: 41.666667% /* 400px/960 */; }

#header .widget-area { width: 56.25% /* 540px/960 */; }

#inner { width: 100% /* 960px/960 */; }

#content-sidebar-wrap { width: 82.291667% /* 790px/960 */; }

#content { width: 48.958333% /* 470px/960 */; }

#sidebar { width: 31.25% /* 300px/960 */; }

#sidebar-alt { width: 15.625% /* 150px/960 */; }

```

Ahora que tenemos nuestros contenedores flexibles, el siguiente paso es obtener el mismo comportamiento para los márgenes y rellenos.

### 10.3.3 Márgenes flexibles

Para determinar la medida de los márgenes, vamos a utilizar la anchura del elemento contenedor. Tomemos como ejemplo la barra lateral. En este caso, el margen lateral izquierdo está definido a 25px, mientras que el ancho del contenedor es de 150px. Por lo tanto, la manera de calcular el nuevo margen es dividir el 25px (target) entre 150px (context), dando como resultado 16.666667%

```

.widget-area ul {

margin: 10px 0 0 25px;

} .widget-area ul {

margin: 10px 0 0 16.666667%;

}

```

### 10.3.4 Rellenos flexibles

En este caso, el comportamiento es exactamente igual al utilizado para calcular la media de los márgenes. Dividimos el relleno actual, 10px, entre la anchura del elemento contenedor 300px, dando como resultado un relleno del 3.33333%.

```
.enews p {  
    padding: 5px 10px 0;  
  
} .enews p {  
    padding: 5px 3.33333% 0;  
  
}
```

### 10.3.5 Imágenes flexibles

El último punto a tener en cuenta es el comportamiento de las imágenes. Por suerte, este es el caso más sencillo. Es suficiente con definir el ancho máximo de las imágenes al 100% (todo el ancho de su contenedor).

```
img { max-width: 100%; }
```

## 10.4 Imágenes flexibles

Teniendo en cuenta este pequeño bloque

```
<div class="figure">  
    <p>  
          
    </p>  
</div>
```

```
.figure {
    float: right;
    margin-bottom: 0.5em;
    margin-left: 2.53164557%; /* 12px / 474px */
    width: 48.7341772%; /* 231px / 474px */
}
```

El código anterior crea un bloque flotante, de ancho variable que contiene una imagen y una descripción de la misma. El problema surge cuando la imagen proporcionada supera las medidas del bloque que la contiene. La estructura no se ve afectada, y las proporciones de la columna siguen intactas, pero la imagen excede el tamaño del contenedor.

#### **10.4.1 Imágenes fluidas**

La solución al problema anterior, es aplicar la siguiente restricción a las imágenes:

```
img { max-width: 100%; }
```

o utilizar:

```
background-image: url("img/sky.jpg");
background-repeat: no-repeat;
background-size: cover;
```

Las imágenes son un elemento fundamental de todas las páginas y representan una gran parte del peso de la misma. Esta situación plantea ciertos retos a la hora de hacer \*diseño responsivo.

Estos retos son principalmente dos:

- Retos a nivel de optimización de la página web.
- Retos a la hora de diseñar la página.

#### **10.4.2 Optimización de Imágenes en el Diseño Responsivo.**

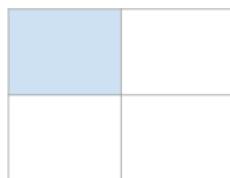
Cuando nos referimos a optimizar el uso de las imágenes nos referimos a:

- Consumir el menor ancho de banda posible.
- Elegir la versión de una misma imagen más adecuada para la resolución.

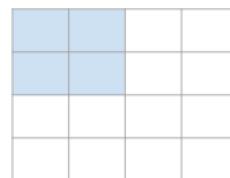
En este proceso de optimización debemos tener en cuenta:

- Ancho del dispositivo.
- Dimensiones de la imagen.
- Resolución de la imagen (en especial en los dispositivos RETINA DISPLAY)

#### **RETINA DISPLAY**



**RESOLUCIÓN  
NORMAL**



**RETINA  
DISPLAY**  
(necesito imágenes de  
tamaño doble...)

Este tipo de dispositivos tienen una densidad de píxeles superior a la normal. En la imagen se muestra la explicación para dispositivos con densidad DPR 2x. En estos casos para que las imágenes se muestren bien tendrán que ser de tamaño doble.

#### **10.4.3 Optimización. Imágenes SVG**

Para solucionar este tipo de problemas lo más fácil es usar imágenes SVG que son gráficos vectoriales que escalan y encogen sin perder resolución.

El problema es que no siempre disponemos de gráficos SVG.

#### 10.4.4 Optimización. Imágenes PNG/GIF/JPEG/WEBP

En estos casos, si no nos importa la optimización es suficiente con usar una imagen de gran resolución y dimensiones y acotarla dentro de un contenedor. Por ejemplo:

```
<div><img src=""....."" /></div>

div {
    /* dimensiones deseadas */
}

img {
    max-width: XXXXXpx;
    width: 100%;
}
```

Pero sin embargo, si nos importa la optimización utilizaremos los atributos `scrset` y/o `sizes` de la imagen que queremos mostrar. Según “retina display” x1 x2 x4...

Por ejemplo:

```
<!-- Considerando resolución: retina
display -->

<div class="container">
    
</div>

<!-- Considerando dimensiones: Unidades del contenedor -->

<div class="container">
```

```

 tamaño real, unidad real */

    sizes="(min-width: 960px) 960px,100vw"

/>

</div>

```

#### 10.4.5 Diseño “ART-DIRECTOR”

La técnica de diseño responsive Art Director consiste en elegir una u otra imagen utilizando la etiqueta source dentro la etiqueta picture y sus atributos srcset para indicar la imagen y media que funciona de manera similar a una media query.

Se ve muy claro con un ejemplo:

```

<div class="art">

    <picture>

        <source media="(min-width: 576px)" srcset="img/big-art.jpg" />

        <source media="(max-width: 575px)" srcset="img/small-art.jpg"
/>

        <!-- En caso de no soportar picture
-->

    </picture>

</div>

```

Normalmente este diseño consiste en fotos que se acercan al objeto importante.

Al final, en casos reales siempre hay que combinar ambas técnicas y probar, en la medida de lo posible, en dispositivos reales, incluidos móviles con Retina Display.

## 10.5 Tablas flexibles

Las tablas son un elemento problemático a la hora de realizar Diseño Responsivo ya que en cuanto tienen un número de columnas considerable pueden provocar la aparición del tan temido scroll horizontal en nuestra página web, sobre todo en pantallas pequeñas.

Para afrontar este tipo de problemas hay tres soluciones que son aceptadas como buenas:

- Esconder columnas.
- Convertir las filas en listas
- Crear un scroll horizontal que solo se aplique a la tabla.

### 10.5.1 Esconder columnas

Esta técnica consiste básicamente en esconder ciertas columnas, que deben de ser las menos importantes, cuando el tamaño de la pantalla es menor que un breakpoint establecido.

Tiene las siguientes ventajas:

- Conseguimos un diseño responsive.
- Priorizo el contenido que quiero mostrar.

Aunque también tiene desventajas:

- Pierdo información en determinadas pantallas.

Un ejemplo para conseguir esto sería una hoja de estilos similar a esta:

```
@media screen and (max-width: 576px) {
    .hidden td.primero, .hidden th.primero {
        display: none;
    }
}

@media screen and (max-width: 768px) {
    .hidden td.segundo, .hidden th.segundo {
        display: none;
    }
}
```

En ese diseño tenemos dos *breakpoints* y ocultaremos unas u otras columnas dependiendo del tamaño (la que tienen la clase *.primero* la clase *.segundo*).

### 10.5.2 Convertir Listas a filas

Esta técnica consiste en hacer desaparecer las cabeceras de la tabla cuando la pantalla es menor que una determinada cantidad y hacer que todas las celdas se conviertan en elementos de bloque para que se muestren una debajo de otra y no al lado.

Tiene las siguientes ventajas:

- Conseguimos un diseño responsive.
- No pierdo información.

Aunque también tiene desventajas:

- No priorizo la información.

- “Desplazar” mucho el resto del layout hacia abajo.

```
@media screen and (max-width: 700px) {

    table.listas,
    table.listas thead,
    table.listas tbody,
    table.listas tr,
    table.listas th,
    table.listas td {

        display: block;
    }
}
```

### 10.5.3 Scroll controlado

Esta técnica consiste en acotar el scroll horizontal para que si ha de aparecer solo afecte a la tabla y no a la página entera.

Tiene las siguiente ventajas:

- Conseguimos un diseño responsivo.
- No pierdo información.

Aunque también tiene desventajas:

- No priorizo la información.
- Aunque local, sigue habiendo un scroll horizontal.

Para conseguir esto debemos “envolver” la tabla en un contenedor y darle las siguientes propiedades:

```
<div class="localscroll">
```

```

<table>    </table>

</div>

div.localscroll {
    overflow-x: auto;
    width: 100%;

}

```

No obstante no hay una solución universal. Debemos experimentar según el Layout que tengamos para ver cuál de estas técnicas (o la mezcla de ellas) se adecúa mejor a nuestro diseño.

## 10.6 Media Queries

Desde la especificación de CSS 2.1, ha sido posible modificar el aspecto de los documentos HTML en función del **tipo de dispositivo** en el que se mostraban. El caso más común es el de crear una hoja de estilos que se aplica al imprimir los documentos:

```

<link rel="stylesheet" type="text/css" href="core.css" media="screen" />

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

```

Como no sólo vamos a formatear nuestros documentos para que se muestren de manera correcta, la especificación CSS ofrece una buena cantidad de tipos de medios (*media type*) para los que podemos aplicar un diseño específico, concretamente los siguientes: all, braille, embossed, handheld, print, projection, screen, speech, tty, y tv.. El problema es que existe una extensa variedad de dispositivos que comparten el mismo tipo de medio, pero son completamente diferentes entre sí.

Por suerte, la W3C introdujo los *Media Query* como parte de la especificación de CSS3, mejorando de manera notable el objetivo de los *Media Type*. Un *Media Query* no sólo nos permite seleccionar el tipo de medio, sino consultar otras características sobre el dispositivo que está mostrando la página.

Un simple ejemplo de *Media Query*:

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px)"
      href="shetland.css" />
```

Esta consulta contiene dos componentes:

- Un tipo de medio (screen).
- Una consulta concreta sobre la característica del medio (max-device-width) y el valor objetivo (480px).

En otras palabras, estamos preguntando al dispositivo si su resolución horizontal (max-device-width) es igual o menor a 480px. Si se cumple la condición, el dispositivo cargará la hoja de estilos shetland.css. De otra manera, el link será ignorado.

Los *Media Query* también pueden ser definidos dentro de la propia hoja de estilos:

```
@media screen and (min-width: 1024px) {
    body {
        font-size: 100%;
    }
}
```

O incluso utilizando la sentencia @import:

```
@import url("wide.css") screen and (min-width: 1024px);
```

No importa la manera en la que se defina el *Media Query*, el resultado debe ser el mismo: si el navegador cumple con el tipo de medio y las condiciones indicadas, se aplicarán las reglas CSS definidas.

### 10.6.1 Sintaxis

Los *Media Queries* pueden contener una o más expresiones, expresadas como funciones multimedia, que se resuelven en *true* o *false*. El resultado del *query* o consulta devuelve *true* si el *media type* especificado en el *Media Query* coincide con el tipo de dispositivo en que el documento está siendo mostrado y todas las expresiones en el *Media Query* devuelven *true*.

Cuando un *Media Query* devuelve *true*, la correspondiente hoja de estilo es aplicada, siguiendo las reglas habituales de CSS. Las hojas de estilo con *Media Queries* adjuntos a los tags <link> seguirán descargando, incluso si sus *Media Queries* resultan *false* (sin embargo, no se aplicarán).

### 10.6.1.1 Operadores lógicos o *logical operators*

Se pueden crear *Media Queries* complejos utilizando *logical operators*, incluyendo *not*, *and* y *only*. El operador *and* es usado para combinar múltiples *media features* en un sólo *Media Query*, requiriendo que cada función devuelva *true* para que el *Query* también lo sea. El operador *not* se utiliza para negar un *Media Query* completo y el operador *only* se usa para aplicar un estilo sólo si el *Query* completo es correcto.

Además, se pueden combinar múltiples *Media Queries* separados por comas en una lista; si alguno de los *Queries* devuelve *true*, todo el *\*media statement* devolverá *true*. Esto es equivalente a un operador *or*.

#### **and**

El keyword *and* se usa para combinar múltiples *media features*, así como combinar éstos con *media types*. Un *Media Query* básico sería:

```
@media (min-width: 700px) { ... }
```

Sin embargo, si quisiéramos que esto sólo se aplicará si la pantalla está en modo *landscape*, se usaría el operador *and*:

```
@media (min-width: 700px) and (orientation: landscape) { ... }
```

Si además, sólo quisiéramos que esto se aplicará si el dispositivo fuera un *media type TV*:

```
@media tv and (min-width: 700px) and (orientation: landscape) { ... }
```

#### **comma-separated lists**

Cuando se utilizan las listas separadas por comas en los *Media Queries*, si alguna de las *Media Queries* devuelve *true*, los estilos se aplican. Cada *Media Query* separado por comas en la lista se trata como un *Query* individual, y cualquier operador aplicado a un *Media Query* no afecta a los demás. Esto significa que los *Media Queries* separados por comas pueden dirigirse a diferentes *media features*, *types* o *states*.

Por ejemplo, si quisiéramos aplicar un conjunto de estilos si el dispositivo de visualización tienen un mínimo de 700px o está en modo *landscape*:

```
@media (min-width: 700px), handheld and (orientation: landscape) { ... }
```

### **not**

La keyword **not** se aplica al *Media Query* en su totalidad y devuelve *true* si el *Media Query* devuelve *false* (como monochrome en una pantalla a color). Este keyword no se puede utilizar para negar un *individual feature query*, solamente un *entire media query*. Por ejemplo:

```
@media not all and (monochrome) { ... }
```

Esto significa que el *Query* es evaluado de esta manera:

```
@media not (all and (monochrome)) { ... }
```

en vez de así:

```
@media (not all) and (monochrome) { ... }
```

Por ejemplo, este otro *Media Query*:

```
@media not screen and (color), print and (color)
```

Se evalúa así:

```
@media (not (screen and (color))), print and (color)
```

### **only**

El keyword **only** previene a los navegadores que no soportan *Media Queries*:

```
<link rel="stylesheet" media="only screen and (color)" href="example.css" />
```

#### **10.6.1.2 Características de los medios**

La especificación de los *Media Query* incluye una larga lista de características que podemos consultar sobre el dispositivo. En este especificación, se hace referencia a dos términos que hay que tener claros:

- *display area*: espacio disponible en la ventana del navegador para

mostrar el contenido de la página web.

- *rendering sourface*: hace referencia al espacio total disponible en el dispositivo.

Característica	Definición	Tiene prefijo min- y max-
width	El ancho del área de visualización (display area)	Sí
height	El alto del área de visualización (display area)	Sí
device-width	El ancho total del dispositivo (rendering sourface)	Sí
device-height	El alto total del dispositivo (rendering sourface)	Sí
orientation	Acepta los valores portrait o landscape 	No
aspect-ratio	Relación de aspecto entre el ancho y alto del área de visualización	Sí
device-aspect-ratio	Relación de aspecto entre el ancho y alto del dispositivo	Sí
color	El número de bits de profundidad de color	Sí
color-index	El número de entradas en la tabla de colores del dispositivo	Sí
monochrome	El número de bits de profundidad de color, en dispositivos monocromáticos	Sí

Una de las ventajas es que podemos encadenar múltiples condiciones en el media query, utilizando la palabra reservada `and`:

```
@media screen and (min-device-width: 480px) and (orientation:landscape)
```

## 10.7 Patrones Responsive

Hasta ahora, hemos definido todas las herramientas necesarias para crear estructuras adaptativas: diseños basados en rejillas flexibles, estrategias para incorporar elementos multimedia y la potencia de los *Media Query* para adaptar el contenido a nuestras necesidades. Vemos cómo incorporar estas técnicas a la hora de desarrollar un sitio web adaptativo.

### 10.7.1 La importancia del contexto

Un diseño adaptativo, implementado de una manera correcta, puede dar a los usuarios de la web, un alto nivel de continuidad entre los diferentes *contextos*. Esto es así, porque de la manera más simple, un diseño adaptativo es capaz de mostrar un documento HTML correctamente en una infinidad de dispositivos, gracias a una estructura flexible y los *Media Query* que aseguran un diseño portable y accesible en la manera de lo posible.

De alguna manera, es posible identificar el *contexto* en el que se visita una web, a partir del dispositivo utilizado. En este contexto, podemos definir un tipo de usuario y sus objetivos. En otras palabras, un usuario móvil quiere un acceso rápido a la información y realizar diferentes tareas a las que realizaría sentado en su sofá con su portátil. En este caso, el tiempo y el ancho de banda están en extremos totalmente opuestos.

Por otra parte, si las prioridades y los objetivos del usuario varían según el *contexto*, entonces puede que disponer de un único documento HTML pueda suponer un problema, dependiendo de la manera en la que se encuentre estructurada la información.

De todas formas, es complicado suponer el *contexto* del usuario únicamente por el tipo de dispositivo, ya que efectivamente, es solamente eso: **una suposición**. ¿Cómo diferenciar, si mi navegación *móvil* se realiza desde la entrada del metro o desde el sofá de mi casa? ¿Es posible por tanto suponer un *contexto*?

Por lo tanto, no podemos inferir el *contexto* del usuario en base a su dispositivo. Así mismo, las palabras *mobile* o *desktop* no definen el comportamiento en la que los usuarios acceden a la web: un portátil puede ser un dispositivo móvil (por ejemplo en un tren), al igual que un *smartphone* o *tablet* puede estar fijo en un lugar. El desarrollo adaptativo no pretende ser un reemplazo de los actuales sitios móviles, sino que forma parte de una estrategia de desarrollo, donde se pretende evaluar si efectivamente es necesario separar totalmente la experiencia móvil o tiene más sentido mostrar la información de una manera adaptativa.

### 10.7.2 Hacia un flujo de trabajo adaptativo

Una de las primeras preguntas (si no la primera) que debemos hacernos, a la hora de plantearnos un diseño adaptativo es la siguiente: **¿En qué medida este contenido o funcionalidad beneficia o aporta valor a nuestros usuarios?**.

Esto es algo que deberíamos preguntar **siempre** para cualquier tipo de proyecto, sea web o no.

Si partimos de un planteamiento *mobile first*, hay que asegurarse que la información que mostramos, y las funcionalidades que implementamos sean importantes para el usuario, ya que no hay espacio suficiente para todo. Hay que darse cuenta de lo que realmente importa. Diseñar partiendo de este paradigma, nos obliga a concentrarnos en lo realmente importante.

*If you design mobile first, you create agreement on what matters most. You can then apply the same rationale to the desktop/laptop version of the web site. We agreed that this was the most important set of features and content for our customers and business; why should that change with more screen space?Luke Wroblewski*

En otras palabras, diseñar desde un inicio pensando en dispositivos móviles puede enriquecer la experiencia de los usuarios, proporcionando un elemento que normalmente se pasa por alto: enfocarnos en lo realmente importante. Esto no quiere decir que los diseños sean simples, faltos de contenidos o funcionalidades, sino que debemos concentrarnos en lo realmente importante.

### 10.7.3 Puntos de ruptura

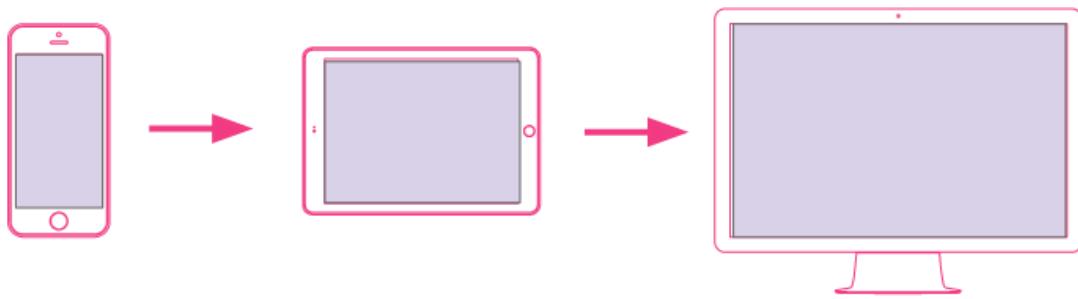
El siguiente paso es identificar el número de diseños diferentes que vamos crear, para acomodarnos a los distintos tamaños de dispositivos. La siguiente tabla muestra los anchos más comunes a la hora de identificar los puntos de ruptura:

#### Punto de ruptura Dispositivo objetivo

- 320 pixels para dispositivos pequeños como teléfonos, en disposición vertical.
- 480 pixels para dispositivos pequeños como teléfonos, en disposición horizontal.
- 600 pixels para tabletas de menor tamaño, como Amazon Kindle (600×800), en disposición vertical.
- 768 pixels para tabletas de unas 10", como el iPad (768x1024), en disposición vertical.
- 1024 pixels para tabletas de unas 10", como el iPad (768x1024), y pequeños portátiles *onetbooks*, en disposición horizontal.
- 1200 pixels para pantallas panorámicas, en portátiles o dispositivos de escritorio.
- 1600 pixels para pantallas panorámicas, en portátiles o dispositivos de escritorio.

### 10.7.3 Patrones responsive

Afrontar un proyecto que requiera el desarrollo de una página web responsiva no es fácil y hay varias estrategias posibles para afrontarlo. No obstante ya es comúnmente aceptado que lo más recomendable es lo siguiente:



Es decir, que el proceso de diseño de nuestra página web debe empezar haciendo que todo quede correctamente en pantallas pequeñas. De este manera:

- Priorizo siempre lo que es importante. Si el proceso fuera al revés es muy fácil que caigamos en el error de quitar cosas que sean realmente importantes.
- Me pregunto en cada diseño si es necesario un diseño nuevo para pantallas más grandes.
- Elijo los breakpoints más adecuados.

### 10.7.4 Patrones Responsivos

En nuestro proceso de diseñar páginas web responsivas, debemos conocer lo que se conoce como *patrones responsivos*.

Los *patrones responsivos* son soluciones que se han dado por buenas para el problema de diseñar páginas web responsivas. Hay muchos pero los más comunes son lo siguientes:

- Column Drop

- Mostly Fluid
- Layout Shifter
- Off Canvas
- Mezclar varios de ellos
- Pequeños ajustes (Tiny Tweaks)

A continuación pasamos a comentar los principales, los cuatro primeros para los que hay un ejemplo en esta misma carpeta de este repositorio:

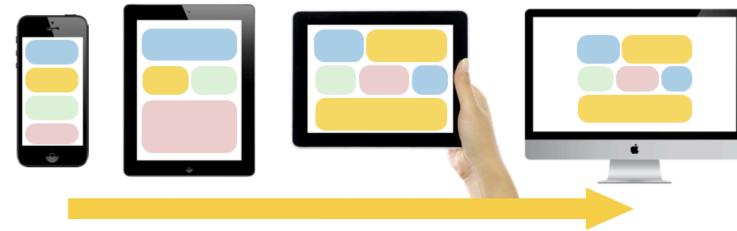
### **Column Drop**

Es el patrón más básico y consiste en que en cada breakpoint se va a apilando un elemento:



### **Mostly Fluid**

Parecido a Column Drop. Es una cuadrícula fluida y en cada breakpoint hay redimensionamiento de varias columnas.



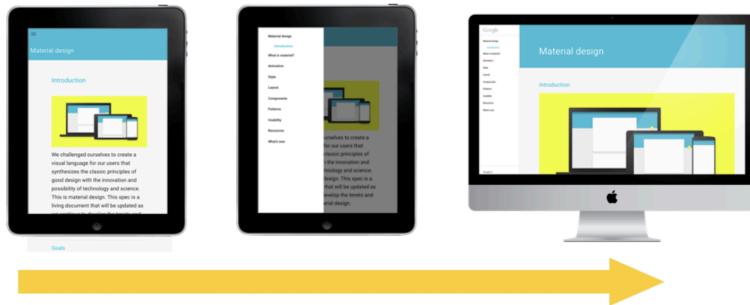
### **Layout Shifter**

Es el patrón más responsivo, en cada breakpoints cambia el diseño del layout, no únicamente el flujo y la anchura de los elementos.



## Off Canvas

En vez de apilar contenidos éstos se colocan fuera de la pantalla cuando el tamaño de pantalla no es lo suficientemente grande.



# Capítulo 11 Preprocesador SASS

## Introducción

Los preprocesadores CSS son herramientas para los desarrolladores de sitios web, que permiten traducir un código de hojas de estilo no estándar, específico del preprocesador en cuestión, a un código CSS estándar, entendible por los navegadores.

Los preprocesadores básicamente nos ofrecen diversas utilidades que a día de hoy no se encuentran en el lenguaje CSS, o bien no son compatibles con todos los navegadores. Ejemplos pueden ser anidación de selectores, funciones (denominadas mixins), etc.

Al desarrollar con un preprocesador se consigue principalmente un ahorro de tiempo, ya que tenemos que escribir menos código para hacer las cosas. Pero también conseguimos una mayor

facilidad de mantenimiento del código, gracias a una mejor organización del código y la posibilidad de editar una vez ciertos valores y que afecten a decenas, o cientos, de lugares del código CSS generado.

### Por qué Sass

Cualquier preprocesador es perfectamente válido. Podríamos sin duda elegir otras alternativas como Less o Stylus y estaría estupendo para nosotros y nuestro proyecto, ya que al final todos ofrecen más o menos las mismas utilidades. Pero sin embargo, Sass se ha convertido en el preprocesador más usado y el más demandado.



## Capítulo 12. Frameworks

Genéricamente, un *framework* es un conjunto de herramientas, librerías, convenciones y buenas prácticas que pretenden encapsular las tareas repetitivas en módulos genéricos fácilmente reutilizables.

De la misma forma, un *framework* CSS es un conjunto de herramientas, hojas de estilos y buenas prácticas que permiten al diseñador web olvidarse de las tareas repetitivas para centrarse en los elementos únicos de cada diseño en los que puede aportar valor.

¿Qué aporta a un diseñador descubrir en cada diseño que debe neutralizar los estilos por defecto que aplican los navegadores? ¿Qué aporta un diseñador que se dedica continuamente a resolver los mismos problemas que se producen al crear *layouts* complejos? ¿Por qué el diseñador se dedica a tareas y problemas que han sido resueltos satisfactoriamente hace mucho tiempo?

Los *frameworks* CSS más completos incluyen utilidades para que el diseñador no tenga que trabajar en ningún aspecto genérico del diseño web. Por este motivo, es habitual que los mejores *frameworks* CSS incluyan herramientas para:

- Neutralizar los estilos por defecto que aplican los navegadores. Se trata de la habitual hoja de estilos *reset.css* que todos los diseñadores profesionales utilizan.
- Manejar correctamente el texto, de forma que todos los contenidos se vean exactamente igual en todos los navegadores y que sean adaptables para mejorar su accesibilidad y permitir su acceso en cualquier medio y/o dispositivo.
- Crear cualquier estructura compleja o *layout* de forma sencilla, con la seguridad de que funciona correctamente en cualquier versión de cualquier navegador.

Actualmente existen decenas de *frameworks* CSS.,

## 12.1. BOOTSTRAP.

