

CONCEPTOS BASICOS DE CSS



Micaela Calvente

INTRODUCCIÓN

Es un lenguaje de diseño que nos permite darle estilos a los componentes de un documento en función de una jerarquía.

CSS es una sigla que proviene de Cascading StyleSheets (Hojas de Estilo en Cascada, en español). La palabra cascada hace referencia a una propiedad muy importante de CSS, y es la forma en que se comporta cuando entran en conflicto dos o más reglas de estilo.

¿Cómo incorporamos CSS?

CSS en Línea: Dentro del atributo `style=""` incorporamos los estilos que se van a aplicar solo en esa misma etiqueta. *Opción no recomendable.*

```
<p style="color: white; background-color: violet">Esto es un párrafo</p>
```

Esto es un párrafo

Utilizando el atributo `style` dentro de la etiqueta le proporcionamos estilo al párrafo. Se pueden utilizar a la vez varias parejas de: **propiedad: valor**

CSS Interno: Incluimos la etiqueta `<style>` dentro del `<head>` en nuestro documento. *Opción menos recomendable.*

```
<!-- Estilo interno -->
<style>
  h1{
    color: white;
    background-color: red;
  }
</style>
```

Título H1

En el ejemplo anterior todas las etiquetas `<h1>` tendrán color de fuente blanco y fondo de color rojo.

CSS Externo: En el *head* del documento HTML tenemos que incluir una referencia al archivo `.css` dentro del elemento `<link>`. *Es la forma más recomendada.*

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

La referencia al archivo externo debe incluir **la ruta completa, el nombre del archivo y su extensión** si se encuentra en alguna subcarpeta dentro del proyecto.

En caso de que el archivo de estilos se encuentre en la misma carpeta que el documento HTML, únicamente se debe incluir el **nombre del archivo y su extensión**. Recordemos que es aconsejable mantener estos archivos (CSS y HTML) en carpetas separadas.

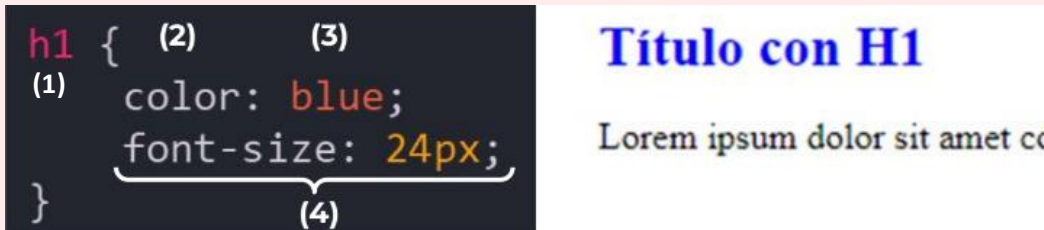
Por ejemplo: `href="css/estilos.css"` o `href="estilos.css"`

Estructura

Selector (1): Indica el elemento al cual vamos a aplicar una regla de estilo.

Propiedad (2) y Valor (3): Especifica qué característica voy a afectar de un elemento y qué valor tomará.

Bloque de declaración (4): Indica el estilo que le daremos al selector.



Los estilos se **heredan** de una etiqueta a otra. Si tenemos declarado en el **<body>** unos estilos, en muchos casos, estas declaraciones también afectarán a etiquetas que estén dentro del **body**.

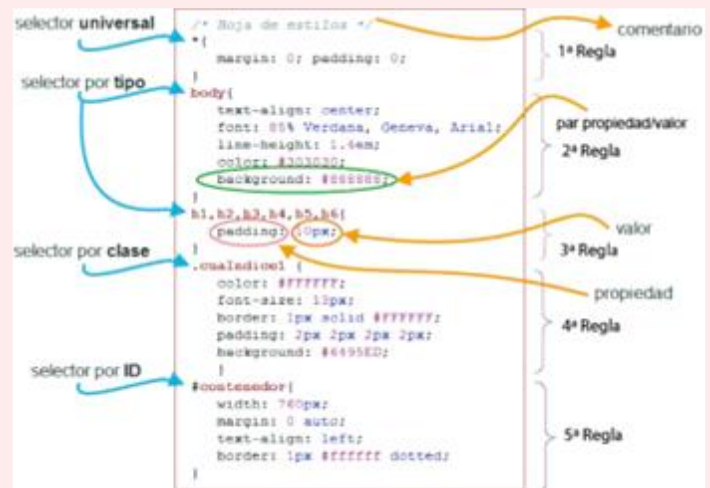
Selectores

Indican el elemento al que se debe aplicar el estilo.

La declaración indica "qué hay que hacer" y el selector indica "a quién hay que aplicarlo".

Hay cuatro selectores básicos:

- **selector universal:** Selecciona todos los elementos de HTML.
- **selector de etiqueta o tipo:** Se utiliza para seleccionar una etiqueta específica.
- **selector de clase:** Se utiliza agregando el atributo class a los elementos que queramos aplicarles estilos.
- **selector de identificador (id):** Similar a .class pero solo se aplica a una etiqueta individual.



Selector universal (*)

Se indica con un * (asterisco) y aplica el estilo a todos los elementos contenidos en el documento HTML.

```
* {
  margin: 10px; /*margin establece el margen para los cuatro lados*/
  padding: 5px; /*padding establece el espacio de relleno*/
  background-color: lightgreen; /*cambia el color de relleno*/
  font-family: Verdana; /*cambia el tipo de letra*/
}
```

Selector de etiqueta o tipo (<tag>)

Este selector afecta a una etiqueta específica. Esto nos permite ser más precisos a la hora de aplicar estilos a elementos particulares. Por ejemplo, a **todas** las etiquetas <h1> o <p> del documento.

```
h1 {
  color: lightblue;
  background-color: blue;
}

p {
  color: black;
  font-style: italic;
  font-size: 130%;
}
```

Selector de clase (.selector)

Se aplica con un punto (.) seguido del nombre que le asignemos al selector. Dentro del documento HTML se lo referencia dentro de la etiqueta usando el atributo *class*, con la siguiente estructura:

class="nombredelselector":

```
.subtitulos {  
  margin-left: 50px;  
  color: yellowgreen;  
  background-color: olivedrab;  
}
```

```
<h3 class="subtitulos">Selectores de clase</h3>
```

Selector de id (#selector)

Se coloca con un numeral (#) en CSS y en el documento HTML se hace referencia al selector con id="nombredelselector", dentro de la etiqueta a la cual se aplica:

```
#texto {  
  color: white;  
  background-color: violet;  
  text-align: center;  
  margin-left: 100px;  
}
```

```
<div id="texto"> Selector de id aplicado a una etiqueta div </div>
```

Span y Div (contenedores de información)

span (abarcar): Es un elemento en línea. Sirve para aplicar estilo al texto o agrupar elementos uno a continuación del otro. Sus etiquetas son `` y `` (ambas obligatorias). Crea una caja que puede contener texto y/u otras etiquetas que se adapten al ancho del contenedor.

div (división): Es un elemento en bloque. Sirve para crear secciones o agrupar contenidos. Sus etiquetas son `<div>` y `</div>` (ambas obligatorias). Crea una caja, que puede contener texto y/u otras etiquetas, que se separa de la caja anterior con un salto a la línea siguiente.

```
<span style="color:red">Un texto en span </span>  
<span style="color:blue">Otro texto en span</span>  
<div style="color:darkgreen">Un texto con div</div>  
<div style="background-color:lightblue">Otro texto con div</div>
```

Un texto en span Otro texto en span
Un texto con div
Otro texto con div

Nota: Con *display: inline* el span no atiende a las propiedades de alto (height) y ancho (width) ya que se adaptará al tamaño del contenido. Cambiando a *display: inline-block* se permiten estas propiedades, porque se comporta como un div.

Se puede hacer que un span se comporte como un div si en CSS agrego *display: block*.

Atributos globales

Estos atributos pueden utilizarse con todos los elementos HTML:

- **style="estilo CSS"**: Especifica un estilo CSS conforme al elemento.
- **class="texto"**: Especifica uno o más nombres de clases para un elemento (haciendo referencia a una clase en una hoja de estilo)
- **id="texto"**: Especifica un id único por cada página.
- **title="texto"**: Especifica información extra sobre un elemento (Tooltip Text).
- **hidden**: Evita que el elemento y sus descendientes se muestran en el navegador. Cualquier control de formulario o de script dentro de la sección hidden será ejecutado, aunque no se muestra al usuario.
- **tabindex="número"**: Especifica la posición del elemento en el orden de tabulación del documento. Se usa para tabular a través de los links de la página (o campos de un formulario).
- **translate="yes|no"**: Indica si el texto del contenido del elemento y los valores del atributo deben ser traducidos o no. La opción por defecto es yes.
- **lang="es"**: Especifica el idioma del contenido del elemento.
- **spellcheck="true|false"**: Especifica si se debe corregir o no la gramática y la ortografía del elemento.
- **draggable="true|false"**: Indica si el elemento es arrastrable; se puede mover haciendo click sin soltar, moviéndolo a una nueva posición en la ventana.

Para mas atributos globales referirse a: [HTML Global attributes \(w3schools.com\)](https://www.w3schools.com/html/html_global_attributes.asp)

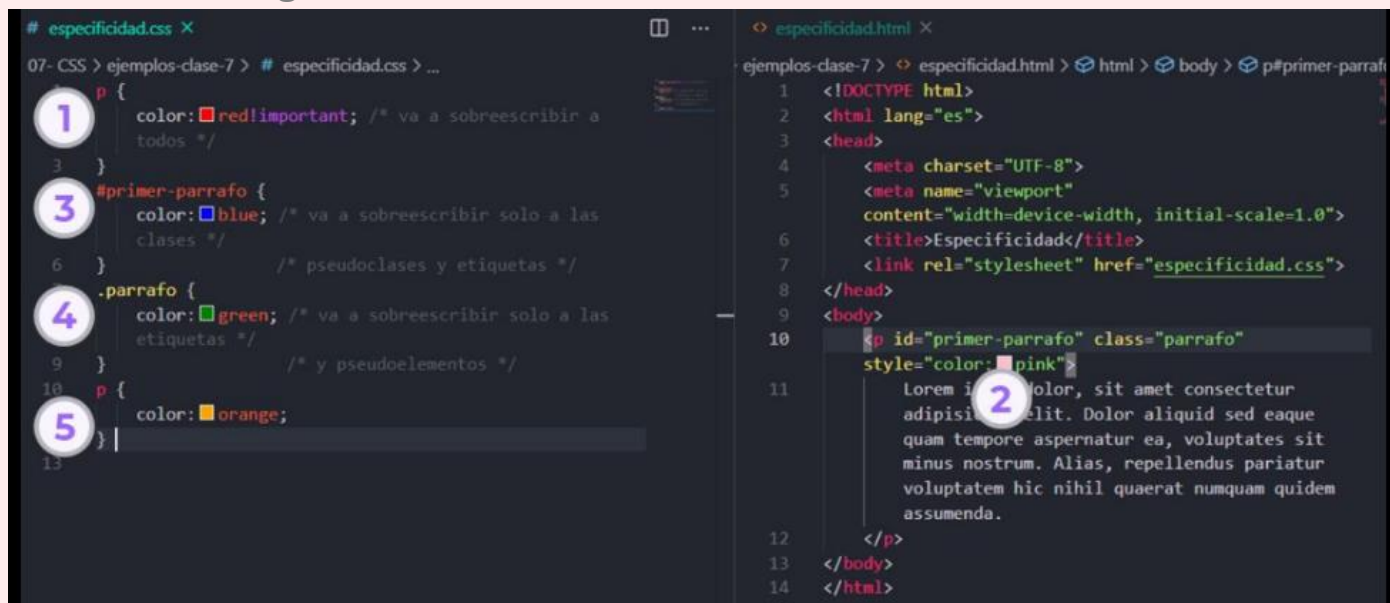
ESPECIFICIDAD

La **especificidad** hace referencia a la relevancia que tiene un estilo sobre un elemento de la página al cual le están afectando varios estilos de CSS al mismo tiempo. Es decir, hace referencia al grado de importancia de un estilo sobre otro.

La siguiente tabla muestra los niveles de especificidad, desde los más específicos a los más generales:

Estilo	Ejemplo	Nivel (peso)
!important	cualquier-selector { color: #FF0000!important; }	1, 0, 0, 0, 0
inline	<p style="color:#FF0000;">Lorem Ipsum</p>	0, 1, 0, 0, 0
ID	#parrafo { color: #FF0000; }	0, 0, 1, 0, 0
Clases, atributos y pseudoclases	.parrafo { color: #FF0000; }	0, 0, 0, 1, 0
Etiquetas y pseudoelementos	p { color: #FF0000; }	0, 0, 0, 0, 1

Orden de las reglas en CSS



Unidades de medida relativas

No están completamente definidas, su valor siempre está referenciado respecto a otro valor (resolución, densidad de pantalla, etc.). Son las más utilizadas por la flexibilidad con la que se adaptan a los diferentes medios y su potencia.

Unidad	Significado	Medida aproximada
em	<<M>>	1em = tamaño de fuente establecida en navegador.
ex	<<x>> (0,5 em apróx)	1ex = mitad del tamaño de la fuente del navegador aprox.
ch	<<zero width>>	1ch = tamaño de ancho del cero (0).
rem	<<root M>>	1rem = tamaño fuente raíz.
%	Porcentaje	Relativa a herencia (contenedor padre)

La unidad **em** equivale a 16px, salvo que se modifique por el usuario.

1em equivaldría a **16px**, mientras que **2em** serían justo el doble: **32px**. Por otro lado, **0.5em** equivalen justo la mitad: **8px**.



em es relativa al tamaño de letra. Si empleamos un font-size de 10px en el body, 1em equivale a 10px. Su tamaño **varía en función del tamaño del elemento padre**. 1.2em sería un 20% más grande que el tamaño de su elemento padre.

Las unidades **ex** o **ch**, menos utilizadas. La unidad **ex** es **aproximadamente la mitad del tamaño** de la fuente establecida por el navegador del usuario, por lo que **1ex** es igual a **0.5em**.



La unidad **ex** se basa en la **altura de la x minúscula**. Su tamaño exacto depende de la tipografía utilizada, y puede ser algo mayor a 0.5em.

La unidad **ch**, equivale al tamaño de ancho del **0** de la fuente actual.

El **porcentaje (%)** define una unidad en función de otra. Por ejemplo, si estamos trabajando en 12px y definimos una unidad como 150% obtendremos 18px.

La unidad **rem (root em)** toma la idea de la unidad em, pero permite establecer un **tamaño base** personalizado (*generalmente para el documento en general, utilizando **html** o la pseudoclase **:root***). De esta forma, podemos trabajar con múltiplos del tamaño base:

```
:root {font-size: 22px; /* Tamaño base */}

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

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

Si queremos cambiar el tamaño del texto en general, sólo tenemos que cambiar el **font-size** de la pseudoclase **:root**, y el resto se modificará en consecuencia.

Unidades de medida flexibles

Las unidades **flexibles (vw y vh o vmin y vmax)** son relativas al tamaño del ancho o alto del viewport (ventana gráfica, región visible de la página Web en el navegador, no el body).

- **vw**: viewport width. Por ejemplo, si decimos que un div debe medir 50vw, es equivalente al 50% del ancho total del viewport.
- **vh**: viewport height. Por ejemplo, si definimos que un div mide 50vh y el alto del viewport es 800px, nuestro div medirá 400px.

Colores y tipografías

Colores

La propiedad **color** se puede usar en cualquier elemento, aunque principalmente se usa para modificar el color del texto y el del *background* de un elemento. Existen diferentes formas de especificar el color:

- **Por palabra clave**: red, blue, lightblue, etc.
- **Valor hexadecimal**: **#31078C** o **#FF0000**. Cada par de letras simboliza el valor del RGB.
- **Valor RGB (Red, Green, Blue)**: **rgb(250, 0, 250)**, **rgb(0, 0, 0)** es el color negro y por el contrario **rgb(255, 255, 255)** es blanco. Valores de 0 a 255.
- **Valor RGBA (RGB + Alpha)**: **rgba(5, 173, 213, 1)** or **rgba(100%, 62.5%, 100%, 1)**.

El valor Alpha tiene que estar comprendido en [0-1] y hace referencia a la transparencia del elemento, siendo 1 = opaco y 0 = transparente.

Color de fondo

El fondo de un elemento, por ejemplo <div> puede ser un color o una imagen:

```
<div style="background-color: #334466;">
  Este div tiene un color de fondo.
</div>

<div style='background-image: url("imagenes/foto.jpg");'>
  Este div tiene una imagen color de fondo.
</div>
```

En el caso de utilizar una imagen, se aplican las reglas vistas a la hora de definir rutas absolutas o relativas.

Tipografías

Las tipografías o fuentes son uno de los pilares del diseño web. La elección de una tipografía adecuada, su tamaño, color, espacio entre letras, interlineado y otras características pueden variar mucho, de forma consciente o inconsciente, la percepción en la que una persona interpreta o accede a los contenidos de una página.

Propiedades básicas:

- **font-size**: tamaño de la fuente (px, em, rem).
- **font-style**: estilo de fuente (normal, italic, oblique).
- **font-family**: lista de fuentes (arial, helvetica, sans-serif, etc).
- **font-weight**: grosor (peso) de la fuente (bold, 400, 600, 800).

Propiedades font-size y font-style

Recordemos que utilizamos <h1>...<h6> para los títulos y <p> para encabezados. **Font-size** (tamaño de la fuente) puede ser absoluto o relativo. El valor predeterminado es 16px

```
<p style='font-size: 16px;'>Fuente en 16px</p>
<p style='font-size: 24px;'>Fuente en 24px</p>
<p style='font-size: 32px;'>Fuente en 32px</p>
```

Fuente en 16px

Fuente en 24px

Fuente en 32px

En cuanto a **font-style** determina el estilo del texto. Posee tres atributos:

```
<p style='font-style: normal;'>Párrafo con estilo normal.</p>
<p style='font-style: italic;'>Párrafo con estilo cursiva.</p>
<p style='font-style: oblique;'>Párrafo con estilo oblicuo.</p>
```

Párrafo con estilo normal.

Párrafo con estilo cursiva.

Párrafo con estilo oblicuo.

Propiedades font-family y font-weight

Font-family establece la familia tipográfica. Los nombres compuestos se colocan entre comillas. Las fuentes sólo se visualizarán si el usuario las tiene instaladas en su dispositivo. Se recomienda agregar más de una fuente, separadas entre comas.

```
div {
  font-family: Georgia, 'Times New Roman', Verdana, serif;
}
```

font-weight: Establece qué tan gruesos o delgados deben mostrarse los caracteres en el texto.

Google Fonts

Si deseamos utilizar alguna fuente que no sea estándar, podemos utilizar la API de **Google Fonts** que dispone de cientos de tipografías para utilizar en nuestra página. Simplemente debemos agregar un enlace en la hoja de estilo.

1. Ingresar a <https://fonts.google.com/>
2. Ingresar en la fuente deseada y hacer clic en "Select this style"
3. Pegar el código en el head de nuestro HTML
4. Copiar y pegar la regla CSS.

```
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Fuentes de Google</title>
9   <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia">
10  <style>
11    body {
12      font-family: "Sofia", sans-serif;
13      font-size: 22px;
14    }
15  </style>
16 </head>
17
18 <body>
19   <h1>Sofia Font</h1>
20   <p>Lorem ipsum dolor sit amet.</p>
21   <p>123456790</p>
22 </body>
23 </html>
```

Sofia Font

Lorem ipsum dolor sit amet.

123456790

Íconos (fontawesome)

Hay varias formas de agregar iconos a tu sitio web, en <https://fontawesome.com/>, hay iconos gratuitos y pagos, te registras en el sitio y te envían un mail con una etiqueta que podés agregar al <head> de tu HTML. Luego podés elegir los iconos a utilizar y agregar a tu página.

Íconos (flaticon)

Otros iconos interesantes se pueden conseguir en <https://www.flaticon.es/> Se pueden descargar o utilizar directamente vinculando las imágenes desde su web.

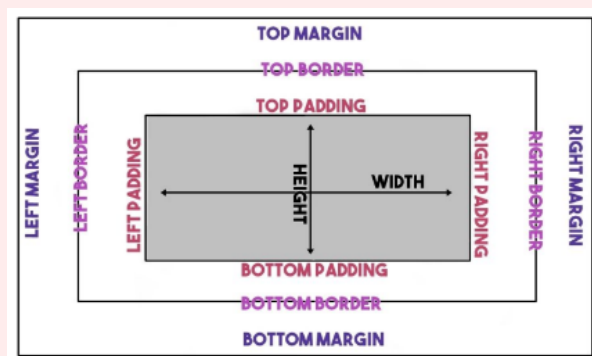
MODELO DE CAJA

El **modelo de caja** o “*box model*” es seguramente la característica más importante del lenguaje de hojas de estilos CSS, ya que condiciona el diseño de todas las páginas web. El Modelo de caja es un mecanismo mediante el cual CSS hace que todos los elementos de las páginas se representen mediante cajas rectangulares.

Los navegadores crean y colocan las cajas de forma automática, pero CSS permite modificar todas sus características.

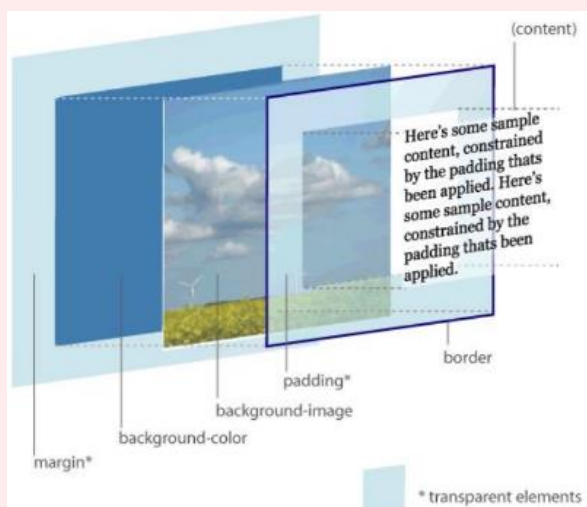
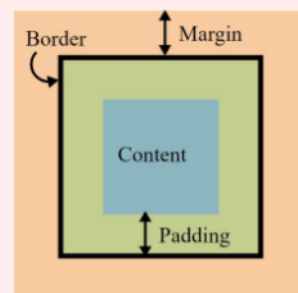
Las cajas de las páginas no son visibles a simple vista porque inicialmente no muestran ningún color de fondo ni ningún borde.

Cada elemento incluido en el documento HTML genera una caja que tiene varios atributos modificables. El comportamiento de esa caja depende de su clasificación, es decir, si se trata de un elemento de línea o de bloque.



Representación básica



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



El relleno y el margen son transparentes, por lo que en el espacio ocupado por el relleno se muestra el color o imagen de fondo (si están definidos) y en el espacio ocupado por el margen se muestra el color o imagen de fondo de su elemento padre (si están definidos). Si ningún elemento padre tiene definido un color o imagen de fondo, se muestra el color o imagen de fondo de la propia página (si están definidos).

DIMENSIONES

Proporcionamos tamaños específicos a los diferentes elementos de un documento HTML asignando valores a las propiedades **width** (ancho) y **height** (alto).

Propiedad	Valor	Significado
<code>width</code>	<code>auto</code> 	Tamaño de ancho de un elemento.
<code>height</code>	<code>auto</code> 	Tamaño de alto de un elemento.

En el caso de utilizar el valor **auto** en las propiedades anteriores (valor por defecto), el navegador se encarga de calcular el ancho o alto necesario, dependiendo del contenido del elemento. El tamaño automático dado a un elemento depende del tipo de elemento que se trate (en bloque o en línea).

Dimensiones en modelo de caja

Si en lugar de utilizar la opción **auto**, o simplemente no indicamos valores para **ancho** y **alto**, el tamaño de la caja suele acomodarse al contenido sin problemas. Pero cuando asignamos valores a estos atributos, forzamos al elemento a tener un aspecto concreto, obteniendo resultados inesperados si su contenido es más grande que el tamaño que hemos definido.



min-width, max-width, min-height y max-height

Una forma de mitigar el problema mencionado consiste en utilizar las propiedades hermanas de width: min-width y max-width; y las propiedades hermanas de height: min-height y max-height. Con estas propiedades, en lugar de establecer un tamaño fijo, establecemos límites máximos y mínimos, donde el ancho o alto de la caja queda comprendido entre esos valores.

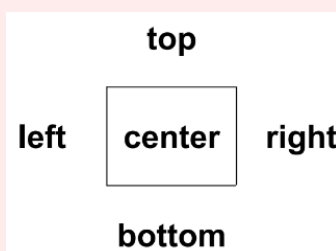
```
div{  
  width: 800px;  
  height: 800px;  
  background: red;  
  max-width: 500px;  
}
```

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

Las propiedades de mínimos (min-width y min-height) por defecto tienen valor 0, mientras que las propiedades de máximos (max-width y max-height) tienen por defecto valor none.

Zonas de un elemento

En CSS existen ciertas palabras clave para hacer referencia a una zona u orientación concreta sobre un elemento. Son conceptos muy sencillos y lógicos:



- **Top:** Parte superior
- **Left:** Parte izquierda
- **Right:** Parte derecha
- **Bottom:** Parte inferior
- **Center:** Se refiere a la posición central entre los extremos horizontales y verticales

Desbordamiento

Puede ocurrir que apliquemos un tamaño de alto y ancho a un elemento HTML, pero su contenido de texto sea tan grande que no quepa dentro de la caja.

En ese caso lo que ocurre es que el contenido se desborda. Podemos modificar este comportamiento con la propiedad de CSS **overflow** o con alguna de sus propiedades específicas **overflow-x** u **overflow-y**

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

Dichas propiedades pueden tomar varios valores, donde **visible** es el valor que tiene por defecto, permitiendo el desbordamiento. Otras opciones son las siguientes:

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

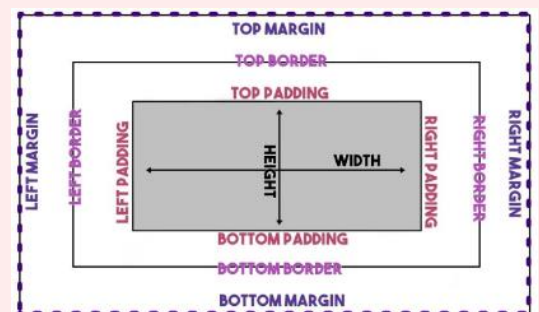
overflow-x y **overflow-y** permiten ocultar alguna barra de desplazamiento (habitualmente la barra de desplazamiento horizontal).

MÁRGENES Y RELLENO

Márgenes (margin)

Se utilizan para crear espacio alrededor de los elementos, fuera de los bordes definidos. Margin especifica el espacio que existe entre el borde de un elemento y el borde de otros elementos adyacentes. Las opciones son:

- margin-top
- margin-right
- margin-bottom
- margin-left

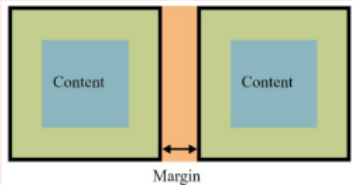


Se puede aplicar en conjunto o de forma concreta a cada una de las zonas del elemento. Estas son las propiedades específicas de cada zona:

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

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

Para centrar un elemento podemos aplicar un ancho fijo al contenedor. Por ejemplo: width:500px con margin: auto. El navegador, al conocer su tamaño, se encarga de repartirlo equitativamente entre los márgenes, dado que conoce el resto del tamaño de la ventana.

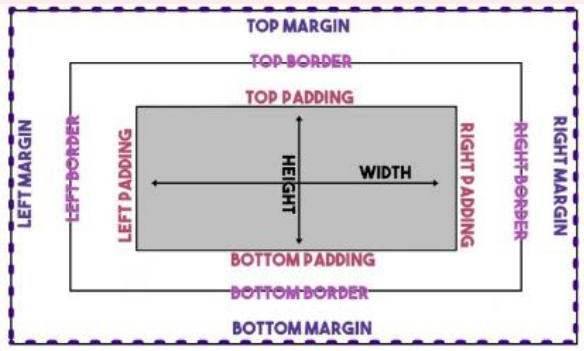


En el siguiente ejemplo nos encontramos con un solapamiento de márgenes. Por defecto, si tenemos dos elementos adyacentes con, por ejemplo, margin: 20px cada uno, ese espacio de margen se solapará y tendremos 20px en total, y no 40px (la suma de cada uno) como podríamos pensar en un principio. +info

Relleno (padding)

Los rellenos (padding) son los espacios que hay entre los bordes del elemento en cuestión y el contenido (por la parte interior). Las opciones son:

- padding-top
- padding-right
- padding-bottom
- padding-left



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

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

Como se puede ver en la tabla, por defecto no hay relleno (el relleno está en cero), aunque puede modificarse tanto con las propiedades anteriores como la propiedad de atajo: Modelo de caja CSS proporciona atajos para los márgenes y los rellenos:

Propiedad	Valores	Significado
margin o padding	SIZE	1 parámetro. Aplica el mismo margen a todos los lados.
	SIZE SIZE	2 parámetros. Aplica margen top/bottom y left/right .
	SIZE SIZE SIZE	3 parámetros. Aplica margen top , left/right y bottom .
	SIZE SIZE SIZE SIZE	4 parámetros. Aplica margen top , right , bottom e left .

margin o padding:

10px; top/right/bottom/left

top/bottom

right/left

10px

20px;

top

right/left

bottom

10px

20px

10px;

top

right

bottom

left

10px

20px

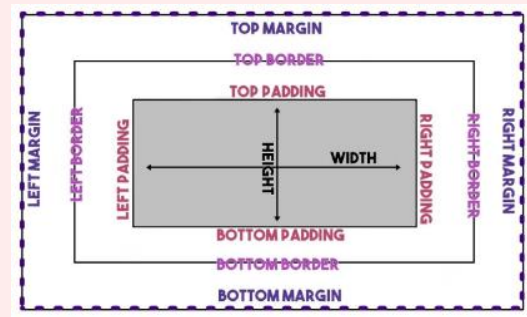
10px

20px;

BORDES

Establece un límite entre la parte interior y la parte exterior de la caja. Se pueden especificar estilo, ancho y color. Las opciones son:

- border-top
- border-right
- border-bottom
- border-left

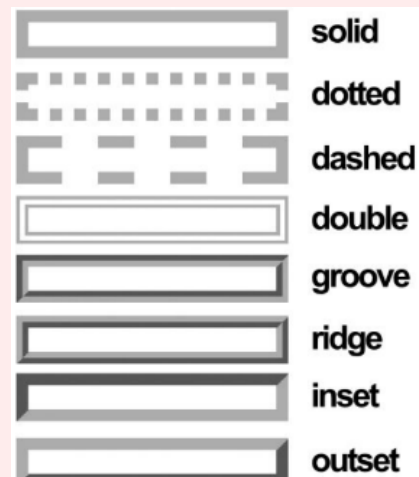


Las propiedades básicas y específicas de los bordes en CSS son las siguientes:

Propiedad	Valor	Significado
<code>border-color</code> <small>1234</small>	<code>black</code> <code>COLOR</code>	Especifica el color que se utilizará en el borde.
<code>border-width</code> <small>1234</small>	<code>thin</code> <code>medium</code> <code>thick</code> <code>SIZE</code>	Especifica un tamaño predefinido para el grosor del borde.
<code>border-style</code> <small>1234</small>	<code>none</code> <code>STYLE</code>	Define el estilo para el borde a utilizar

El estilo de borde más frecuente es `solid` (borde liso y continuo), y que además es la opción por defecto. Pueden utilizarse cualquiera de los estilos indicados en la tabla anterior e incluso combinar con otras propiedades.

- **hidden**: Oculto. Idéntico a `none`, salvo para conflictos con tablas.
- **dotted**: Borde basado en puntos.
- **dashed**: Borde basado en rayas (línea discontinua).
- **solid**: Borde sólido (línea continua).
- **double**: Borde doble (dos líneas continuas).
- **groove**: Borde biselado con luz desde arriba.
- **ridge**: Borde biselado con luz desde abajo. Opuesto a `groove`.
- **inset**: Borde con profundidad «hacia dentro».
- **outset**: Borde con profundidad «hacia fuera». Opuesto a `inset`.



Bordes múltiples

Sólo hemos utilizado un parámetro en cada propiedad, aplicando el mismo valor a cada borde de un elemento. Sin embargo, podemos especificar de uno a cuatro parámetros, dependiendo de lo que queramos hacer:

Propiedad	Valor	Significado
<code>border-color</code> <small>1234</small>	<code>COLOR</code>	1 parámetro. Aplica el mismo color a todos los bordes.
	<code>COLOR</code> <code>COLOR</code>	2 parámetros. Aplica al borde top/bottom y al left/right .
	<code>COLOR</code> <code>COLOR</code> <code>COLOR</code>	3 parámetros. Aplica al top , al left/right y al bottom .
	<code>COLOR</code> <code>COLOR</code> <code>COLOR</code> <code>COLOR</code>	4 parámetros. Aplica al top , right , bottom y left .

Podemos hacer lo mismo con las propiedades `border-width` y `border-style`. Teniendo en cuenta esto, disponemos de una gran flexibilidad a la hora de especificar esquemas de bordes más complejos.

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

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

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Con la propiedad **border-width** pasa exactamente lo mismo que con **margin** y **padding**, actuando en este caso en relación al grosor del borde de un elemento. Se pueden utilizar de 1 a 4 parámetros.

Bordes - Atajos

Con la propiedad de atajo **border**, podemos hacer un resumen sin necesidad de indicar múltiples propiedades individuales por separado, realizando el proceso de forma más corta:

Propiedad	Valor	Significado
<code>border</code>	<code>SIZE</code> <code>STYLE</code> <code>COLOR</code>	Propiedad de atajo para simplificar valores.
<pre>div { border: 1px solid #000000; }</pre>		<p>Lorem ipsum dolor sit amet, con labore et dolore magna aliqua.</p>

Bordes específicos

Una forma más intuitiva, es utilizar las propiedades de bordes específicos (por zonas) y aplicar estilos combinándolos junto a la herencia de CSS.

Para utilizarlas bastaría con indicar la zona justo después de **border**:

<pre>div { border-bottom-width: 2px; border-bottom-style: dotted; border-bottom-color: black; }</pre>	<pre>div { border: 5px solid red; border-top-width: 15px; border-top-color: orange; border-top-style: solid; /* se hereda */ }</pre>
<p>Lorem ipsum dolo labore et dolore m</p>	<p>Lorem ipsum dolor sit ame labore et dolore magna alic</p>

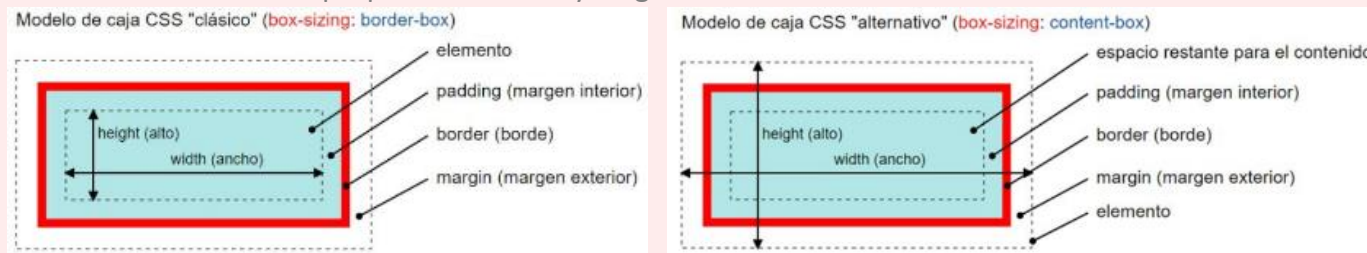
Box-sizing

Indica cómo se debe calcular el ancho y el alto total de un elemento.

Acepta los valores:

- **box-sizing: content-box:** Es el valor que cualquier caja tiene asignada **por defecto**. Las propiedades `width` y `height` no incluyen el borde, padding o margin.
- **box-sizing: border-box:** Las propiedades `width` y `height` incluyen el contenido, padding y borde pero no el margin.

En el modelo de caja CSS "clásico", el borde y los márgenes interior y exterior se añaden al tamaño del elemento definido con las propiedades `width` y `height`.



La propiedad `box-sizing`, permite modificar este comportamiento y hacer que el borde y los márgenes interior y exterior se puedan incluir en el interior del tamaño definido con las propiedades **`width`** y **`height`**. En este caso se reducirá el espacio disponible para el contenido.

POSICIONAMIENTO

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

- **static:** es el valor por defecto, un elemento con este valor no está posicionado con CSS.
- **relative:** mediante las propiedades `top` | `bottom` | `right` y/o `left` el elemento se posiciona en forma relativa a su contenedor.
- **absolute:** hace que un elemento se ubique considerando su contenedor posicionado más cercano. Si no encuentra ningún contenedor cercano, el elemento se colocará respecto al viewport. El resto de elementos de la página ignoran la nueva posición del elemento.
- **fixed:** la caja está posicionada con respecto a la ventana del navegador. Se mantiene en el mismo lugar incluso al hacer scroll en la página. La referencia es el *viewport*, la parte visible del navegador.
- **sticky:** al realizar un scroll y después de alcanzar una posición de desplazamiento determinada, se "pega" al borde elegido.

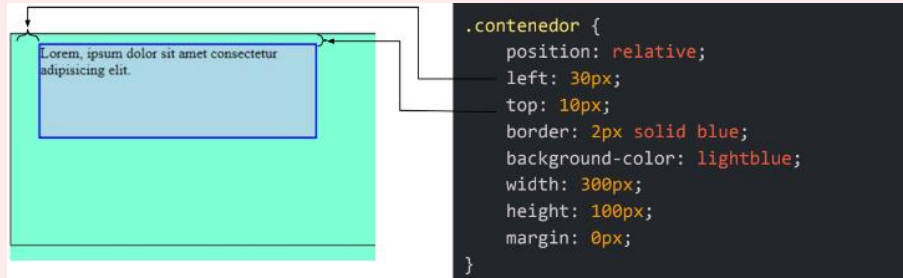
Si utilizamos un modo de posicionamiento diferente al estático (*absolute*, *fixed*, *sticky* o *relative*), podemos emplear una serie de propiedades para modificar la posición de un elemento.

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

Las propiedades **`top`**, **`bottom`**, **`left`** y **`right`** sirven para mover un elemento desde la orientación que su propio nombre indica hasta su extremo contrario. Esto es, si utilizamos **`left`** e indicamos **`20px`**, estaremos indicando mover **desde la izquierda** 20 píxeles hacia la **derecha**.

Posicionamiento relativo (relative)

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



Posicionamiento absoluto (absolute)

Utilizando la palabra clave **absolute** indicamos que el elemento utiliza **posicionamiento absoluto**. Este tipo de posicionamiento coloca los elementos utilizando como punto de origen el primer contenedor con posicionamiento diferente a estático. Si no existe, emplea el documento completo como referencia. Si el contenedor padre tiene posicionamiento estático, se analiza el posicionamiento del padre del contenedor padre, y así sucesivamente hasta encontrar un contenedor con posicionamiento no estático o llegar a la etiqueta `<body>`, situación en la que se comportaría como en el ejemplo.

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

Posicionamiento fijo (fixed) y “pegajoso” (sticky)

El posicionamiento **fijo (fixed)** hace que el elemento se muestre en una posición fija **dependiendo de la región visual del navegador**. Es decir, aunque el usuario haga scroll y se desplace hacia abajo en la página web, el elemento seguirá en el mismo sitio posicionado.

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

El **posicionamiento sticky** se utiliza cuando queremos que un elemento se posicione en un lugar específico de forma fija («sticky», pegajoso).

Ejemplo: Al hacer scroll y llegar a un elemento este podrá quedarse fijo en la parte superior mientras continuamos haciendo scroll. No debemos confundir con el fijo ya que no queda en una posición fija, sino que flota respecto del fondo y se queda adherido a la parte superior.

Profundidad (z-index)

Establece el nivel de profundidad de un elemento sobre los demás. De esta forma, podremos superponer los elementos, quedando “apilados”. Los elementos se posicionan de acuerdo al nivel de profundidad, quedando “encima” los que poseen un valor de index mayor.



SELECTORES AVANZADOS

Utilizan “combinadores”, signos gráficos que establecen la relación entre los elementos y permiten hacer una selección **específica**:

Selector	Carácter	Descripción	Ejemplo
Agrupado	, (coma)	Se utiliza cuando varios elementos comparten una serie de declaraciones iguales, se aplican las reglas CSS a los selectores involucrados.	<pre>p, a, div { /*Reglas CSS*/ }</pre>
Descendiente	(espacio)	Apunta a elementos contenidos dentro de otro en la estructura del documento, sin importar la profundidad o los descendientes interpuestos entre ellos	<pre>div p { /*Reglas CSS*/ }</pre>
Hijos directos	> (mayor)	Selecciona los elementos que sean hijos directos del contenedor padre, descartando nietos y sucesivos.	<pre>span > a { /*Reglas CSS*/ }</pre>
Hermano adyacente	+ (más)	Aplica estilos a elementos que siguen inmediatamente a otros. Deben tener el mismo elemento padre ser inmediatamente siguiente a él.	<pre>div + p { /*Reglas CSS*/ }</pre>
General de hermanos	~ (tilde de la ñ)	Selecciona todos los elementos que son hermanos del especificado, sin necesidad de que sean adyacentes.	<pre>div ~ p { /*Reglas CSS*/ }</pre>

PSEUDOCCLASES

Una pseudoclase es un selector que marca los elementos que están en un estado específico o tienen un comportamiento determinado. Todas las pseudoclases se denominan mediante una palabra precedida por dos puntos y se comportan del mismo modo. Afectan a un fragmento del documento que está en un estado determinado y se comportan como si se hubiera añadido una clase a su HTML.

:first-child

Esta pseudoclase modifica el estilo del primer elemento de un grupo de elementos hermanos dentro de un contenedor, es decir “el primer hijo de su padre”.

```
<div>  
  <p> Párrafo 1 </p>  
  <p> Párrafo 2 </p>  
  <p> Párrafo 3 </p>  
</div>
```

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

Párrafo 1

Párrafo 2

Párrafo 3

:last-child

Se utiliza para representar al último elemento entre un grupo de elementos hermanos dentro de un contenedor, es decir “el último hijo de su padre”.

```
<div>  
  <p> Párrafo 1 </p>  
  <p> Párrafo 2 </p>  
  <p> Párrafo 3 </p>  
</div>
```

```
p:last-child {  
  color: blue;  
}
```

Párrafo 1

Párrafo 2

Párrafo 3

:nth-child (n)

El selector coincide con cada elemento que es el **n-ésimo** hijo, independientemente del tipo, de su padre. **n** puede ser un número, una palabra clave o una fórmula.

```
<div>
  <p> Párrafo 1 </p>
  <p> Párrafo 2 </p>
  <p> Párrafo 3 </p>
</div>
```

```
p:nth-child(2) {
  background: cyan;
}
```

Párrafo 1

Párrafo 2

Párrafo 3

Pseudoclases para hipervínculos

Se aplican a las etiquetas <a>, que pueden tener cuatro estados:

- **:link** se refiere a un enlace que todavía no ha sido visitado.
- **:hover** se refiere a un elemento sobre el que se coloca el puntero del mouse.
- **:visited** se refiere a un enlace que ya ha sido visitado.
- **:active** se refiere a cualquier elemento que ha sido activado por el usuario.

```
<a href="https://google.com" target="_blank">Ir a Google</a>
```

```
a:link {color: red;}
a:hover {background-color: yellow;}
a:visited {color: blue;}
a:active {background-color: green; color: white;}
```

[Contacto](#)

[Contacto](#)

[Contacto](#)

[Contacto](#)

PSEUDOELEMENTOS

Los **pseudoelementos** se añaden a los selectores, pero no describen un estado especial, sino que permiten añadir estilos a una parte concreta del documento.

Se utilizan para darle estilos a partes específicas de un elemento. Están precedida por cuatro puntos (:):

::first-letter se utiliza para darle estilo a la primera letra de un texto:

```
<p>Párrafo con la primera letra de otro color</p>
```

```
p::first-letter{color:blue;}
```

Párrafo con la pri

::first-line se utiliza para darle estilo a la primera línea de un párrafo:

```
p::first-line{background-color: lightgreen;}
```

Lorem ipsum dolor sit amet consectetur adipiscing elit.
Deleniti quaerat asperiores vitae aspernatur ut incidunt
dolores tempora saepe harum at, ullam laudantium

::selection agrega estilos a una parte del documento que ha sido resaltada por el usuario:

```
p::selection{
  background-color: lightsalmon;
}
```

Lorem ipsum dolor sit amet consectetur
harum at, ullam laudantium consectetur
molestias eius, magnam explicabo hic a

::before y **::after** agregan contenido antes y después, respectivamente, del contenido:

```
p::before{ content:"✨";}
p::after{ content:"🙈";}
```

✨ Texto de ejemplo 🙈

TRANSFORMACIONES, ANIMACIONES Y TRANSICIONES

TRANSFORMACIONES

Permiten mover, rotar, escalar y sesgar elementos, es decir, efectos visuales en 2D y 3D. Las propiedades principales para realizar transformaciones son las siguientes:

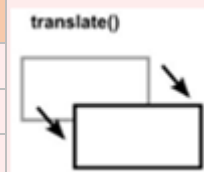
Propiedades	Significado
transform	Aplica una o varias funciones de transformación sobre un elemento.
transform-origin	Cambia el punto de origen del elemento en una transformación.
transform-style	Modifica el tratamiento de los elementos hijos.

Con la propiedad transform podemos indicar una o varias transformaciones para realizar sobre un elemento, ya sean 2D (sobre dos ejes) o 3D (sobre tres ejes).

2D - TRASLACIONES (translate)

Las funciones de traslación son aquellas que realizan una transformación en la que mueven un elemento de un lugar a otro. Si especificamos un valor positivo en el eje X (horizontal), lo moveremos hacia la derecha, y si especificamos un valor negativo, lo moveremos hacia la izquierda. Lo mismo con el eje Y (vertical).

Funciones	Significado
translate(x)	Traslada el elemento una distancia horizontalmente
translate(y)	Traslada el elemento una distancia verticalmente
translate(x, y)	Propiedad de <i>atajo</i> de las dos anteriores.



Por ejemplo, transform: translate(20px, -30px) traslada el elemento 20 píxeles a la derecha y 30 píxeles hacia arriba, que es equivalente a utilizar transform: translateX(20px) translateY(-30px).

2D - ESCALADO (scale)

Las **funciones de escalado** realizan una transformación en la que aumentan o reducen el tamaño de un elemento, basándose en el parámetro indicado, que no es más que un factor de escala:

Funciones	Significado
scale X (fx)	Reescala el elemento a un nuevo tamaño horizontal
scale Y (fy)	Reescala el elemento a un nuevo tamaño vertical
scale (fx, fy)	Propiedad de <i>atajo</i> de las dos anteriores

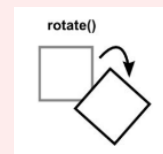


Por ejemplo, transform: scale(2, 2) realiza una transformación de escalado del elemento, ampliándolo al doble de su tamaño original. Si utilizamos scale() con dos parámetros iguales, estamos manteniendo la proporción del elemento, pero si utilizamos diferentes valores, acabaría deformándose.

2D - ROTACIONES (rotate)

Las funciones de rotación simplemente giran el elemento el número de grados indicado:

Funciones	Significado
rotate X (xdeg)	Establece una rotación 2D en grados solo para el eje horizontal
rotate Y (ydeg)	Establece una rotación 2D en grados solo para el eje vertical
rotate (deg)	Establece una rotación 2D en grados sobre si mismo

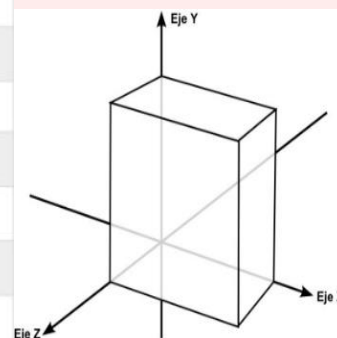


Con transform: rotate(5deg) realizamos una rotación de 5 grados del elemento sobre sí mismo. Utilizando rotateX() y rotateY() podemos hacer lo mismo respecto al eje X o el eje Y respectivamente.

FUNCIONES 3D

A las funciones anteriores, también podemos añadir las funciones equivalentes de CSS para hacer uso del eje Z y con esto acceder a las dimensiones espaciales o “3D”. Las propiedades de transformación 3D son las siguientes:

Funciones	Significado
translateZ(z)	Traslada el elemento una distancia de SIZE z en el eje de profundidad.
translate3d(x, y, z)	Establece una translación 3D, donde aplica los parámetros de SIZE a cada eje.
scaleZ(fz)	Reescala el elemento a un nuevo tamaño con factor NUMBER $\frac{fz}{z}$ de profundidad.
scale3d(fx, fy, fz)	Establece un escalado 3D, donde aplica los factores a cada eje.
rotateZ(zdeg)	Establece una rotación 2D en DIRECTION zdeg grados sólo para el eje de profundidad Z.
rotate3d(x, y, z, deg)	Establece una rotación 3D, aplicando un vector [x_y_z] y el ángulo en DIRECTION deg .
perspective(n)	Establece una perspectiva 3D de SIZE n
matrix3d(n, n, ...)	Establece una matriz de transformación 3D (16 valores)



TRANSFORMACIONES MÚLTIPLES

Si indicamos dos propiedades **transform** en el mismo elemento, con diferentes funciones, la segunda propiedad sobrescribirá a la anterior, como ocurre con cualquier propiedad de CSS:

```
div {  
  transform: rotate(5deg);  
  transform: scale(2,2); /* Esta línea sobrescribe a la anterior */  
}
```

Para evitar esto se suelen emplear múltiples transformaciones, separándolas mediante un espacio. En el siguiente ejemplo, aplicamos una función de rotación, una función de escalado y una función de translación de forma simultánea:

```
div { transform: rotate(5deg) scale(2,2) translate(20px, 40px); }
```

TRANSICIONES

Las **transiciones** CSS le permiten cambiar los valores de una propiedad, durante un período determinado. Se basan en un principio muy básico: conseguir un **cambio de estilo** con un efecto suavizado entre un estado inicial y un estado final.

Para crear un efecto de transición, debemos especificar dos cosas:

- La propiedad CSS a la que desea agregar un efecto (¿qué propiedad modifico?)
- La duración del efecto (¿durante cuánto tiempo?)

Las propiedades relacionadas que existen son las siguientes:

Propiedades	Valor
<code>transition-property</code>	<code>all</code> <code>none</code> <code>propiedad_css</code>
<code>transition-duration</code>	<code>0</code> <code>TIME</code>
<code>transition-timing-function</code>	<code>ease</code> <code>linear</code> <code>ease-in</code> <code>ease-out</code> <code>ease-in-out</code> <code>cubic-bezier(A, B, C, D)</code>
<code>transition-delay</code>	<code>0</code> <code>TIME</code>

- **transition-property:** Indica *la propiedad que será afectada por la transición*. Puede ser una propiedad concreta (width o color, por ejemplo) o simplemente all para que se aplique a todos los elementos. Por otro lado, none hace que no se aplique ninguna transición.
- **transition-duration:** Establece la *duración de la transición*. Se recomienda comenzar con valores cortos, para que las transiciones sean rápidas y elegantes. Una duración demasiado grande producirá una transición con detenciones intermitentes, y pueden resultar molestas a muchos usuarios.
- **transition-timing-function:** indica el *ritmo de la transición* que queremos conseguir. Se recomienda comenzar con linear (ritmo constante) y luego utilizar otros valores para variar el ritmo sea al inicio y/o al final de la transición.

Los valores que puede tomar la propiedad son los siguientes:

Valor	Início	Transcurso	Final	Equivalente en cubic-bezier
<code>ease</code>	Lento	Rápido	Lento	(0.25, 0.1, 0.25, 1)
<code>linear</code>	Normal	Normal	Normal	(0, 0, 1, 1)
<code>ease-in</code>	Lento	Normal	Normal	(0.42, 0, 1, 1)
<code>ease-out</code>	Normal	Normal	Lento	(0, 0, 0.58, 1)
<code>ease-in-out</code>	Lento	Normal	Lento	(0.42, 0, 0.58, 1)
<code>cubic-bezier(A, B, C, D)</code>	-	-	-	Transición personalizada

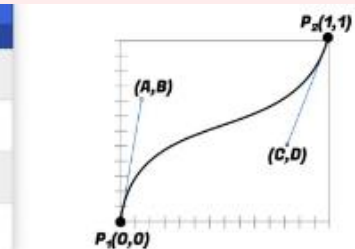
Un valor linear siempre es constante, mientras que ease comienza suavemente, aumenta la velocidad y finaliza suavemente. Ease-in y ease-out son variaciones que van más lento al principio o al final, y ease-in-out una mezcla de ambas.

La función Cubic-Bezier() nos permite configurar con más detalle la transición.

La función de tiempo Cubic-Bezier()

Es una función personalizada. Podemos asignar valores que definen la velocidad que queramos que tenga la transición. En la última columna de la tabla anterior podemos ver los valores equivalentes a cada una de las palabras clave mencionadas. En principio, el formato de la función es cubic-bezier(A, B, C, D), donde:

Parámetro	Valor	Descripción	Pertenece a
A	X_1	Eje X del primer punto que orienta la curva bezier.	P_1
B	Y_1	Eje Y del primer punto que orienta la curva bezier.	P_1
C	X_2	Eje X del segundo punto que orienta la curva bezier.	P_2
D	Y_2	Eje Y del segundo punto que orienta la curva bezier.	P_2



transition-delay nos ofrece la posibilidad de retrasar el inicio de la transición los segundos especificados.

Atajo: Transiciones

Como siempre, podemos resumir todas estas operaciones en una propiedad de atajo denominada transition. Los valores del ejemplo superior, se podrían escribir como se puede ver a continuación (si no necesitas algún valor, se puede omitir):

```
div {  
  /* transition: <property> <duration> <timing-function> <delay> */  
  transition: all 0.2s ease-in;  
}
```

ANIMACIONES

Una animación permite que un elemento cambie gradualmente de un estilo a otro.

Podemos cambiar tantas propiedades CSS como deseemos, tantas veces como sea necesario.

Las animaciones amplían el concepto de transiciones convirtiéndolo en algo mucho más flexible y potente, partiendo del mismo concepto de realizar cambios en ciertos estados inicial y final, pero incorporando más estados, pudiendo realizar cambios desde un estado inicial, a un estado posterior, a otro estado posterior, y así sucesivamente. Además, esto será posible de forma automática, **sin que el usuario tenga que realizar una acción concreta**.

Para utilizar la animación CSS, primero debemos especificar algunos fotogramas clave (**@keyframes**) para la animación, que contendrán los estilos que tendrá el elemento en determinados momentos. Además, tendremos que utilizar las propiedades de las animaciones, que definen el comportamiento de la misma.

Propiedades de animación CSS

Para definir este comportamiento necesitamos conocer las siguientes propiedades CSS:

Propiedades	Valor
<code>animation-name</code>	<code>none</code> <i>nombre</i>
<code>animation-duration</code>	<code>0</code> <code>TIME</code>
<code>animation-timing-function</code>	<code>ease</code> <code>linear</code> <code>ease-in</code> <code>ease-out</code> <code>ease-in-out</code> <code>cubic-bezier(A, B, C, D)</code>
<code>animation-delay</code>	<code>0</code> <code>TIME</code>
<code>animation-iteration-count</code>	<code>1</code> <code>infinite</code> <code>NUMBER</code>
<code>animation-direction</code>	<code>normal</code> <code>reverse</code> <code>alternate</code> <code>alternate-reverse</code>
<code>animation-fill-mode</code>	<code>none</code> <code>forwards</code> <code>backwards</code> <code>both</code>
<code>animation-play-state</code>	<code>running</code> <code>paused</code>

animation-name permite especificar el nombre del fotograma a utilizar. **animation-duration**, **animation-timing-function** y **animation-delay** funcionan exactamente igual que en transiciones.

La propiedad **animation-iteration-count** permite indicar el número de veces que se repite la animación, pudiendo establecer un número concreto de repeticiones o indicando **infinite** para que se repita continuamente. Por otra parte, especificando un valor en **animation-direction** conseguiremos indicar el orden en el que se reproducen los fotogramas, pudiendo escoger un valor de los siguientes:

Valor	Significado
<code>normal</code>	Los fotogramas se reproducen desde el principio al final.
<code>reverse</code>	Los fotogramas se reproducen desde el final al principio.
<code>alternate</code>	En iteraciones par, de forma normal. Impares, a la inversa.
<code>alternate-reverse</code>	En iteraciones impares, de forma normal. Pares, normal.

Por defecto, cuando se termina una animación que se ha indicado que se reproduzca sólo una vez, la animación vuelve a su estado inicial (primer fotograma).

Mediante la propiedad **animation-fill-mode** podemos indicar que debe mostrar la animación cuando ha finalizado y ya no se está reproduciendo; si mostrar el estado inicial (**backwards**), el estado final (**forwards**) o una combinación de ambas (**both**).

La propiedad **animation-play-state** nos permite establecer la animación a estado de reproducción (**running**) o pausarla (**paused**).

Atajo: Animaciones

Nuevamente, CSS ofrece la posibilidad de resumir todas estas propiedades en una sola, para hacer nuestras hojas de estilos más específicas. El orden de la propiedad de atajo sería el siguiente:

```
div {  
  /* animation: <name> <duration> <timing-function> <delay>  
    <iteration-count> <direction> <fill-mode> <play-state> */  
  animation: changeColor 5s linear 0.5s 4 normal forwards running;  
}
```

Debemos ser cuidadosos al indicar el tiempo en las propiedades de duración. Al ser una unidad diferente a las que solemos manejar (px, em, etc..) hay que especificar **siempre** la **s**, por segundos, **aunque sea un valor igual a 0**.

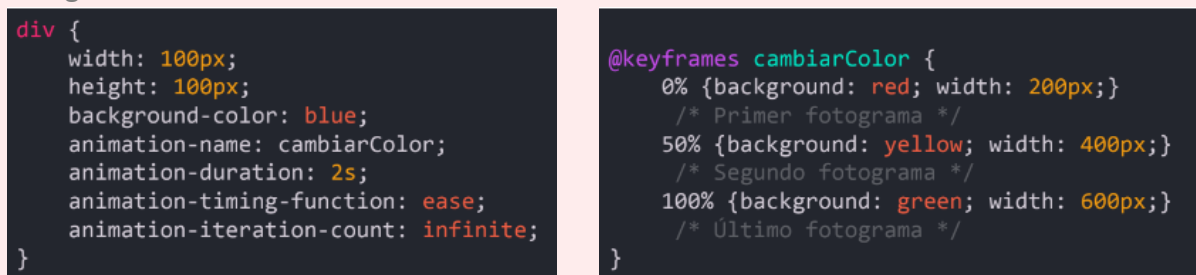
Fotogramas (keyframes)

Para definir los fotogramas de una animación utilizaremos la regla **@keyframes**. Primero elegimos un nombre para la animación (ya que podemos tener varias en una misma página), mientras que podremos utilizar varios selectores para definir el transcurso de los fotogramas en la animación.



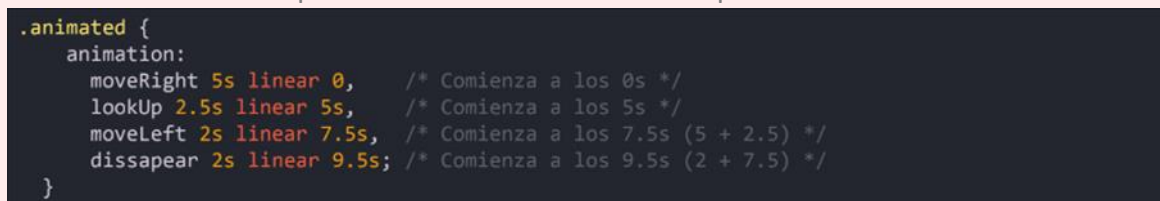
En este ejemplo partimos de un primer fotograma con un elemento con color de fondo rojo. Si observamos el último fotograma, indicamos finalice con color de fondo verde. La regla **@keyframes** creará la animación intermedia para conseguir que el elemento cambie de color.

Los selectores **from** y **to** son realmente sinónimos de 0% y 100%. Al modificarlos podremos ir añadiendo nuevos fotogramas intermedios:



Encadenar animaciones

Es posible encadenar múltiples animaciones, separando con comas las animaciones individuales y estableciendo un tiempo de retardo a cada animación posterior:



Hemos aplicado varias animaciones a la vez, estableciendo un retardo equivalente a la suma de la duración de las animaciones anteriores, encadenando una animación con otra.

Librería de animaciones Animate.css

Podemos utilizar *Animate.css* para dar dinamismo a nuestro contenido.

- En pocos pasos se pueden agregar animaciones CSS a cualquier elemento con esta sencilla librería.
- En la creación de cualquier contenido web resulta interesante incorporar animaciones que nos ayuden a mejorar la experiencia del usuario durante la interacción con el contenido.
- Permite disponer de una gran variedad de animaciones CSS sin necesidad de crearlas nosotros mismos.
- Esta librería permite conseguir que el contenido sea más atractivo y dinámico.

DISEÑO WEB RESPONSIVO

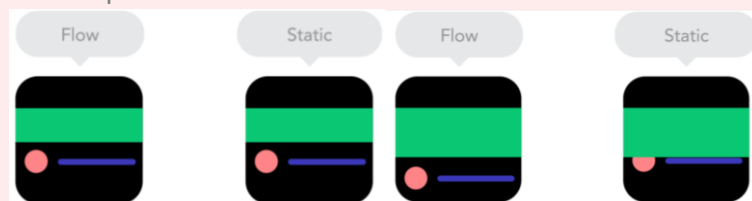
Diseño responsivo vs Diseño adaptativo

Un **diseño responsivo** responde a las dimensiones del dispositivo, mientras que un **diseño adaptativo** se adapta, pero no necesariamente responde en todo momento, tiene cierto delay, estamos hablando casi de lo mismo.

El diseño web responsivo adapta la estructura y los diferentes elementos de cada página web a las dimensiones y características de cada pantalla. Por otro lado, el diseño web adaptativo es menos flexible, y se basa en el uso de tamaños y características preestablecidas. Las diferencias entre ambos métodos se encuentran en el proceso creativo y de diseño, en el resultado final y la experiencia del usuario.

Flujo (The Flow) vs Estático (Static)

Cuando una pantalla se vuelve más pequeña, el contenido comienza a crecer verticalmente ocupando más espacio, y el contenido que se encuentra debajo va a ser desplazado hacia abajo, eso se llama **flujo**. Si es estático ese flujo de elementos no se desplaza, no se adapta al ancho del viewport y se pierde contenido o cierto contenido tapa a otro.



Unidades Relativas vs Unidades Absolutas

La densidad de píxeles de cada dispositivo puede variar, por eso necesitamos unidades que sean flexibles y funcionen sin importar el dispositivo. Ahí es donde las unidades relativas como los porcentajes son útiles. Entonces, hacer algo con un 50% de ancho significa que siempre ocupará la mitad de la pantalla (viewport, el tamaño de la ventana del navegador abierta), independientemente del dispositivo.



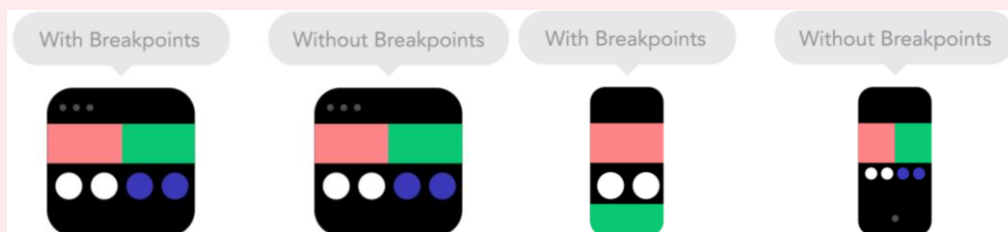
Valores Mínimos y Máximos

En un celular nos puede interesar que determinado contenido ocupe todo el ancho de la pantalla, pero al pasar a un TV 4K podríamos cambiar de idea. Por ejemplo, podríamos tener un `width:100%`, pero con un `max width: 1000px`. En un celular, las imágenes pueden tener un ancho diferente a las que vemos en otras pantallas. El alto no importa tanto en mobile, porque podemos scrollar.



PUNTOS DE CONTROL (Breakpoints)

Estos puntos de control permiten al diseño cambiar en determinados puntos, por ej, en un monitor podemos tener 3 columnas, pero sólo 1 en un celular (que es más angosto). Estos puntos de control o de quiebre se crean con los *media queries*, que nos permiten determinar que, si el mínimo del ancho de la pantalla tiene un valor en lugar de otro, en vez de distribuir el contenido en tres columnas colocarlo en una sola, con varias filas.

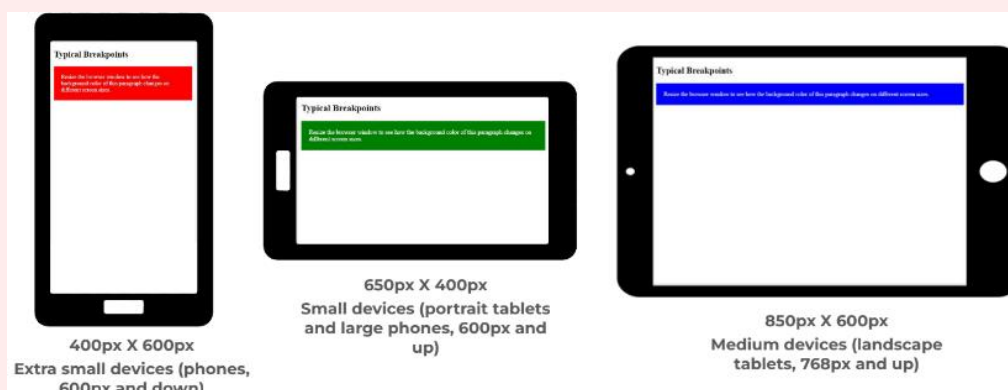


```
.example {  
  padding: 20px;  
  color: white;  
}  
/* Extra small devices (600px and down) */  
@media only screen and (max-width: 600px) {  
  .example {background: red;}  
}  
/* Small devices (600px and up) */  
@media only screen and (min-width: 600px) {  
  .example {background: green;}  
}  
/* Medium devices (768px and up) */  
@media only screen and (min-width: 768px) {  
  .example {background: blue;}  
}
```

```
/* Large devices (992px and up) */  
@media only screen and (min-width: 992px) {  
  .example {background: orange;}  
}  
/* Extra large devices (1200px and up) */  
@media only screen and (min-width: 1200px) {  
  .example {background: pink;}  
}
```

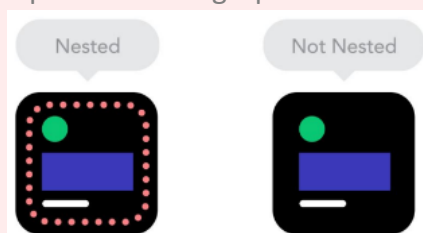
Puntos de corte (según ancho):

- Hasta 600 px: Fondo rojo
- Desde 600 px: Fondo verde
- Desde 768 px: Fondo azul
- Desde 992 px: Fondo naranja
- Desde 1200 px: Fondo rosa



Objetos anidados (Nested Objects)

Tener muchos objetos que dependan de otros puede ser difícil de controlar, sin embargo, agruparlos en contenedores nos puede simplificar las cosas. ¿Por qué usamos contenedores? Porque a la hora de pensar contenido responsive nos va a facilitar posicionar un grupo de elementos en otro lugar.



Mobile first vs Desktop first

- **Mobile first:** Primero nos enfocamos en dispositivos móviles y luego pensamos en otros.
- **Desktop first:** Primero nos enfocamos en dispositivos de escritorio, y luego pensamos en otros.

Estadísticamente, los dispositivos móviles son mayoritariamente los que acceden a los sitios Web. Los dispositivos de escritorio tienden a utilizarse cada vez menos.

System Font vs WebFonts

- **Fuentes de la Web:** son descargadas por lo que, cuantas más haya, más lento cargará el sitio.
- **Fuentes del Sistema:** más rápidas, pero si NO están en el cliente navegador del usuario se usa una por defecto.

Cuando estamos trabajando con dispositivos móviles tenemos que tener en cuenta que todo se carga.

Bitmaps vs Vectors

- **Bitmaps:** JPG, PNG, GIF. Recomendadas para muchos detalles y efectos.
- **Vectors:** SVG (gráficos basados en vectores escalables), si voy a mostrar un ícono uso Icon Fonts, que son más livianos, pero algunos exploradores viejos no los soportan.

TEXTO RESPONSIVO

Recordemos que el tamaño del texto se puede configurar con una unidad "vw", que es el "ancho de la ventana gráfica". De esa forma, el tamaño del texto seguirá el tamaño de la ventana del navegador.

```
<h1 style="font-size:10vw">Hello World</h1>
```

Viewport es el tamaño de la ventana del navegador.

1vw = 1% del ancho de la ventana gráfica.

IMÁGENES RESPONSIVAS

Las imágenes responsivas son imágenes que se escalan bien para adaptarse a cualquier tamaño de navegador.

Si la propiedad CSS **width** se establece en 100%, la imagen responderá y se ampliará y reducirá.

Una imagen grande puede ser perfecta en una pantalla de computadora grande, pero inútil en un dispositivo pequeño. ¿Por qué cargar una imagen grande cuando tiene que reducirla de todos modos? Para reducir la carga, o por cualquier otro motivo, puede utilizar **media queries** para mostrar diferentes imágenes en diferentes dispositivos.

DISPLAY

Cada elemento tiene un valor de display por defecto. Recordemos que los navegadores le dan a los elementos valores por defecto block e inline:

- **block:** el elemento empieza en una nueva línea (div, h1-h6, header, etc)
- **inline:** puede contener algo de texto dentro de un párrafo sin interrumpir el flujo del párrafo.
- **none:** utilizado para ocultar elementos sin eliminarlos, no deja espacio donde el elemento se encontraba.
- **inline-block:** Los elementos inline-block fluyen con el texto y demás elementos como si fueran elementos en línea y además respetan el ancho, el alto y los márgenes verticales.

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

Tipos de display

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

OCULTAR ELEMENTOS

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

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

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

Dicha propiedad **visibility** puede tomar los siguientes valores:

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

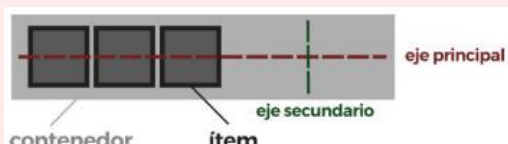
Utilizar **visibility: hidden** es muy interesante si queremos que un elemento y su contenido se vuelva invisible, pero siga ocupando su espacio y así evitar que los elementos adyacentes se desplacen, lo que suele ser un comportamiento no deseado en algunas ocasiones cuando se aplica **display: none**.

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

FLEXBOX

Flexbox es un sistema de **elementos flexibles** en la que los elementos HTML se adaptan y colocan automáticamente y es más fácil personalizar los diseños. Está especialmente diseñado para crear, mediante CSS, **estructuras de una sola dimensión**.

Elementos básicos



- **Contenedor:** Es el elemento padre que tendrá en su interior cada uno de los ítems flexibles.
 - **Eje principal:** Los contenedores flexibles tienen una orientación principal específica. Por defecto es el eje horizontal.
 - **Eje secundario:** La orientación secundaria es perpendicular a la principal.
- **Ítem:** Son los elementos hijos flexibles del contenedor.

Ejemplo:

```
<div class="container"> <!-- Flex container -->
  <div class="item item-1">1</div> <!-- Flex items -->
  <div class="item item-2">2</div>
  <div class="item item-3">3</div>
</div>
```

Para activar el modo **flexbox**, utilizamos sobre el elemento contenedor la propiedad **display** y especificamos el valor **flex** o **inline-flex** (dependiendo de cómo queramos que se comporte el contenedor)

Propiedad display

Tipo de elemento	Descripción
inline-flex	Establece un contenedor en línea, similar a inline-block (ocupa solo el contenido).
flex	Establece un contenedor en bloque, similar a block (ocupa todo el ancho del padre).

display: flex

1 2 3

display: inline-flex

1 2 3

Dirección de los ejes

Existen dos propiedades principales para manipular la dirección y comportamiento de los ítems a lo largo del eje principal del contenedor. Son las siguientes:

Propiedad	Valor	Significado
flex-direction	row row-reverse column column-reverse	Cambia la orientación del eje principal.
flex-wrap	nowrap wrap wrap-reverse	Evita o permite el desbordamiento (multilínea).

Mediante la propiedad **flex-direction** podemos modificar la dirección del **eje principal** del contenedor para que se oriente en horizontal (por defecto) o en vertical. Además, también podemos incluir el sufijo **-reverse** para indicar que coloque los ítems en orden inverso.

row y row-reverse

Determinan el orden de los elementos. Aplicando estas propiedades modificamos el flujo del eje principal

Valor	Descripción
row	Establece la dirección del eje principal en horizontal.
row-reverse	Establece la dirección del eje principal en horizontal (invertido).
column	Establece la dirección del eje principal en vertical.
column-reverse	Establece la dirección del eje principal en vertical (invertido).


```
.container {
  background: steelblue;
  display: flex;
  flex-direction: column;
}

.item {
  background: grey;
}
```

flex-wrap

Existe otra propiedad llamada **flex-wrap** con la que podemos especificar el comportamiento del contenedor respecto a evitar que se desborde (*nowrap*, *valor por defecto*) o permitir que lo haga, en cuyo caso, estaríamos hablando de un **contenedor flexbox multilínea**.

Valor	Descripción
nowrap	Establece los ítems en una sola línea (no permite que se desborde el contenedor).
wrap	Establece los ítems en modo multilínea (permite que se desborde el contenedor).
wrap-reverse	Establece los ítems en modo multilínea, pero en dirección inversa.

Teniendo en cuenta estos valores de la propiedad **flex-wrap**, podemos conseguir cosas como la siguiente:

```
.container {
  background: steelblue;
  display: flex;
  width: 200px;
  flex-wrap: wrap; /* Comportamiento por defecto: nowrap */
}

.item {
  background: grey;
  width: 50%;
}
```

Con **nowrap** los 3 ítems se muestran en una misma línea. El tamaño de los ítems se ajusta al contenedor, manteniendo sus proporciones.

Si especificamos **wrap** el contenedor se puede desbordar, pasando a ser un contenedor multilínea que muestra uno o más elementos en la línea siguiente.

Atajo: dirección de los ejes

Existe una propiedad de atajo (short-hand) llamada **flex-flow**, con la que podemos resumir los valores de las propiedades **flex-direction** y **flex-wrap**, especificándose en una sola propiedad y ahorrándonos utilizar las propiedades concretas:

```
.container {  
  /* flex-flow: <flex-direction> <flex-wrap>; */  
  flex-flow: row wrap;  
}
```

Propiedades de alineación

Disponemos de 4 propiedades relativas a la alineación, la primera relativa al eje principal y las restantes al secundario:

Propiedad	Valor	Eje
justify-content	flex-start flex-end center space-between space-around space-evenly	1
align-content	flex-start flex-end center space-between space-around space-evenly stretch	2
align-items	flex-start flex-end center stretch baseline	2
align-self	auto flex-start flex-end center stretch baseline	2







- **justify-content**: Alinea los ítems del eje principal.
- **align-items**: Alinea los ítems del eje secundario.

Eje principal

La propiedad **justify-content** sirve para colocar los ítems de un contenedor mediante una disposición concreta a lo largo del eje principal:

Valor	Descripción
flex-start	Agrupar los ítems al principio del eje principal.
flex-end	Agrupar los ítems al final del eje principal.
center	Agrupar los ítems al centro del eje principal.
space-between	Distribuye los ítems dejando el máximo espacio para separarlos.
space-around	Distribuye los ítems dejando el mismo espacio alrededor de ellos (izq/dcha).
space-evenly	Distribuye los ítems dejando el mismo espacio (solapado) a izquierda y derecha.

Con estos valores de la propiedad **justify-content** modificamos la disposición de los ítems del contenedor, distribuyéndose como se ve en el siguiente ejemplo (nótese los números para observar el orden de cada ítem):

 flex-start	 space-between
 flex-end	 space-around
 center	 space-evenly

align-content permite manejar contenedores **flex multiline**. Estos contenedores dividen el eje principal en múltiples líneas, dado que los ítems no caben en el ancho disponible. Sus valores son los siguientes:

Valor	Descripción
flex-start	Agrupar los ítems al principio del eje principal.
flex-end	Agrupar los ítems al final del eje principal.
center	Agrupar los ítems al centro del eje principal.
space-between	Distribuye los ítems desde el inicio hasta el final.
space-around	Distribuye los ítems dejando el mismo espacio a los lados de cada uno.
stretch	Estira los ítems para ocupar de forma equitativa todo el espacio.

En un contenedor multilinea de 200 píxeles de alto con ítems de 50px de alto, podemos utilizar la propiedad **align-content** para alinear los ítems de forma vertical de modo que se queden en la zona inferior del contenedor:

```
.container {
  background: #CCC;
  display: flex;
  width: 200px;
  height: 200px;
  flex-wrap: wrap;
  align-content: flex-end;
}
.item {
  background: #777;
  width: 50%;
  height: 50px;
}
```

Eje secundario

align-items alinea los ítems en el eje secundario del contenedor. A diferencia de **align-content**, **align-items** opera sobre el eje secundario. Los valores que puede tomar son los siguientes:

Valor	Descripción
flex-start	Alinea los ítems al principio del eje secundario.
flex-end	Alinea los ítems al final del eje secundario.
center	Alinea los ítems al centro del eje secundario.
stretch	Alinea los ítems estirándolos de modo que cubran desde el inicio hasta el final del contenedor.
baseline	Alinea los ítems en el contenedor según la base del contenido de los ítems del contenedor.

align-self actúa como **align-items**, pero sobre un ítem hijo específico, sobrescribiendo su comportamiento. La propiedad puede tomar los siguientes valores:

Valor	Descripción
flex-start	Alinea los ítems al principio del contenedor.
flex-end	Alinea los ítems al final del contenedor.
center	Alinea los ítems al centro del contenedor.
stretch	Alinea los ítems estirándolos al tamaño del contenedor.
baseline	Alinea los ítems en el contenedor según la base de los ítems.
auto	Hereda el valor de align-items del padre (si no se ha definido, es stretch).

Si se especifica el valor **auto** a la propiedad **align-self**, el navegador le asigna el valor de la propiedad **align-items** del contenedor padre, y en caso de no existir, el valor por defecto **stretch**. Ver segundo ejemplo interactivo aquí.

Existe un atajo para establecer valores de **align-content** y de **justify-content** a la vez, denominada **place-content**. Las dos clases siguientes proporcionan las mismas características:

```
.container {
  display: flex;
  place-content: flex-start flex-end;
}

.container {
  align-content: flex-start;
  justify-content: flex-end;
}
```

Propiedades de hijos

Las siguientes propiedades se aplican sobre los ítems hijos:

Propiedad	Valor	Descripción
flex-grow	0 <small>NUMBER</small>	Número que indica el factor de crecimiento del ítem respecto al resto.
flex-shrink	1 <small>NUMBER</small>	Número que indica el factor de decrecimiento del ítem respecto al resto.
flex-basis	<small>SIZE</small> content	Tamaño base de los ítems antes de aplicar variación.
order	0 <small>NUMBER</small>	Número (peso) que indica el orden de aparición de los ítems.

Si con **flex-grow** indicamos un valor de 1 a todos los ítems, todos serán del mismo tamaño. Si colocamos un valor de 1 a todos, pero a uno le indicamos 2, ese ítem será más grande que los anteriores. Por defecto tienen un valor de 0.

flex-shrink es opuesta a **flex-grow**, aplica un factor de decrecimiento. De esta forma, los ítems que tengan un valor numérico más grande, serán más pequeños, mientras que los que tengan un valor numérico más pequeño serán más grandes, justo al contrario de cómo funciona la propiedad **flex-grow**.

Por último, tenemos la propiedad **flex-basis**, que define el tamaño por defecto (de base) que tendrán los ítems antes de aplicarle la distribución de espacio. Generalmente, se aplica un tamaño (unidades, porcentajes, etc...), pero también se puede aplicar la palabra clave **content** que ajusta automáticamente el tamaño al contenido del ítem, que es su valor por defecto.

Atajo: Propiedades de hijos

Existe una propiedad llamada **flex** que sirve de atajo para estas tres propiedades de los ítems hijos. Funciona de la siguiente forma:

```
.item {
  /* flex: <flex-grow> <flex-shrink> <flex-basis> */
  flex: 1 3 35%;
}
```

Huecos (gaps)

Las propiedades **row-gap** y **column-gap** establecen el tamaño del «hueco» entre ítems desde el elemento contenedor, sin necesidad de utilizar **padding** o **margin** en los elementos hijos.

Propiedad	Valor	Descripción
row-gap	normal size	Espacio entre filas (sólo si flex-direction: column)
column-gap	normal size	Espacio entre columnas (sólo si flex-direction: row)

Sólo una de las dos propiedades tendrá efecto, dependiendo de si **flex-direction** está establecida en **column** o en **row**. Es posible usar ambas cuando **flex-wrap: wrap**, disponiendo de multicolumnas flexbox.

Orden de los ítems

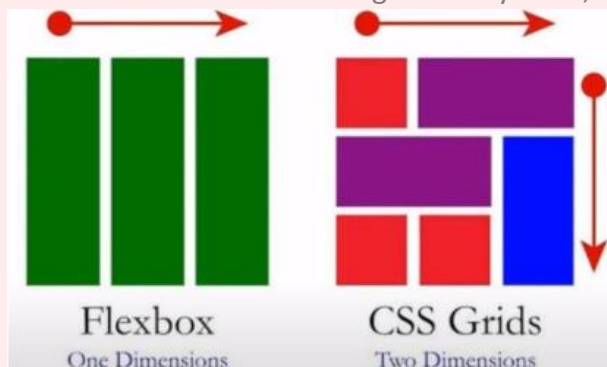
Por último, y quizás una de las propiedades más interesantes, es **order**, que modifica y establece el orden de los ítems según una secuencia numérica.

Por defecto, todos los ítems flex tienen un **order: 0** implícito, aunque no se especifique. Si indicamos un **order** con un valor numérico, se reubican los ítems según su número, colocando primero los ítems con número más pequeño (incluso valores negativos) y después los ítems con números más altos. De esta forma reordenamos fácilmente los ítems, incluso utilizando media queries o responsive design.

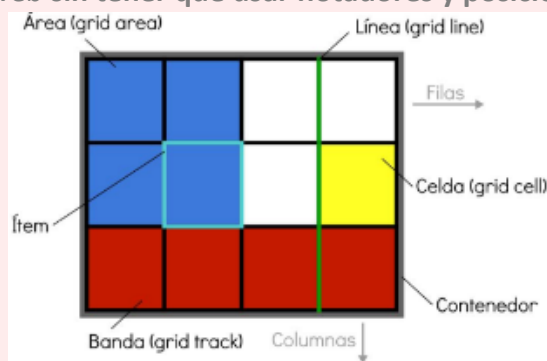
CSS GRID

¿Flexbox vs CSS Grid?

CSS Grid puede convivir con Flexbox. No se trata de tecnologías excluyentes, sino complementarias.



El módulo de diseño de CSS Grid ofrece un sistema de diseño basado en cuadrículas, con filas y columnas, lo que facilita el diseño de páginas web **sin tener que usar flotadores y posicionamiento**.



Conceptos básicos

- **Grid Container:** es nuestro elemento “padre”, donde se asigna un `{display:grid;}` y nos permitirá colocar otras propiedades para manipular nuestro diseño.
- **Grid Item:** son los hijos directos de nuestro container. Estos los manejaremos a nuestra voluntad, nuestras filas y columnas que moveremos a nuestro gusto.
- **Grid Line:** son las líneas divisorias horizontales y verticales.
- **Grid Track:** es el espacio entre dos líneas adyacentes. Filas y columnas.
- **Grid Cell:** nuestras celdas serán el espacio entre dos filas adyacentes y 2 columnas adyacentes.
- **Grid Area:** espacio rodeado por 4 *grid lines*.

Contenedor

```
<div> <!-- Elemento padre -->
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  <div class="grid-item">8</div>
  <div class="grid-item">9</div>
</div>
```

1	2	3
4	5	6
7	8	9

Display

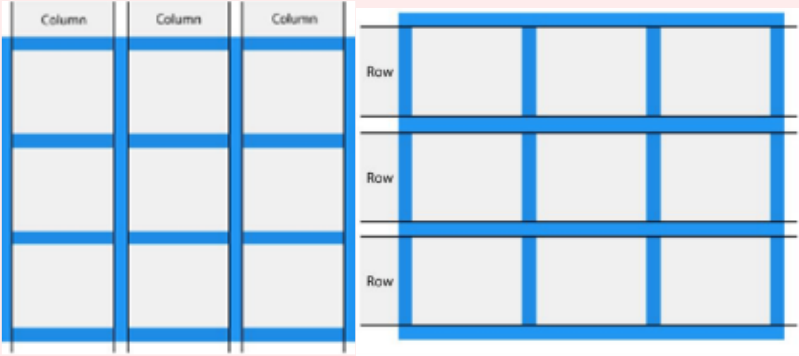
Un elemento HTML se transforma en un contenedor de grilla cuando tiene su propiedad **display** seteada en **grid** o **inline-grid**:

```
.grid-container{
  display: grid;
}
```

```
.grid-container{
  display: inline-grid;
}
```

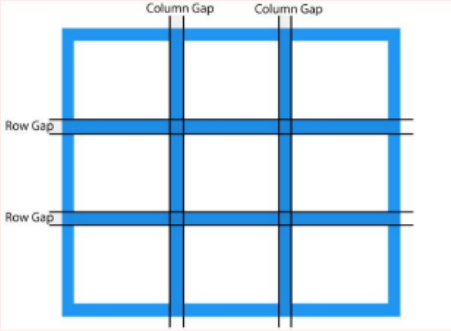
Grid Items

Se pueden referenciar por fila o por columna, aunque no es la única forma.



Grid Gaps

Es el espacio entre los ítems. Se pueden ajustar los tamaños de gap con las siguientes propiedades: **grid-column-gap**; **grid-row-gap** y **grid-gap**.



Grid Lines

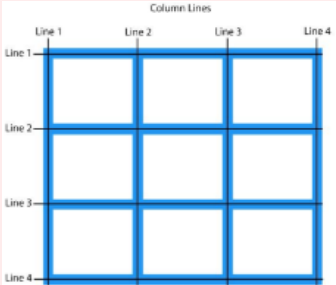
Para colocar un grid-ítem en un contenedor se referencian los números de línea:

Pone un grid item en la línea de la columna 1, que finaliza en la línea de la columna 3.

```
.item{
  grid-column-start: 1;
  grid-column-end: 3;
}
```

Pone un grid item en la línea de la fila 1, que finaliza en la línea de la fila 3.

```
.item{
  grid-row-start: 1;
  grid-row-end: 3;
}
```



Propiedades

Propiedad	Descripción
column-gap	Especifica el espacio entre las columnas.
gap	Propiedad abreviada. Espacio entre filas y entre columnas.
grid	Propiedad abreviada. Filas y columnas, de la cuadrícula, áreas de la cuadrícula, filas y columnas automáticas y propiedades de flujo automático de cuadrícula.
grid-area	Especifica un nombre para el elemento de la cuadrícula. Es una propiedad abreviada para grid-row-start, grid-column-start, grid-row-end y grid-column-end.
grid-auto-columns	Especifica un tamaño de columna predeterminado.
grid-auto-flow	Determina cómo se insertan los elementos en la cuadrícula.
grid-auto-rows	Especifica un tamaño de fila predeterminado.
grid-column	Propiedad abreviada para las propiedades grid-column-start y grid-column-end.
grid-column-end	Especifica dónde termina el elemento de la cuadrícula.
grid-column-gap	Especifica el tamaño del espacio entre columnas.
grid-column-start	Especifica dónde comienza el elemento de la cuadrícula.

Propiedad	Descripción
grid-gap	Una propiedad abreviada para las propiedades grid-row-gap y grid-column-gap.
grid-row	Una propiedad abreviada para las propiedades grid-row-start y grid-row-end.
grid-row-end	Especifica dónde termina el elemento de la cuadrícula.
grid-row-gap	Especifica el tamaño del espacio entre filas.
grid-row-start	Especifica dónde comienza el elemento de la cuadrícula.
grid-template	Una propiedad abreviada para las propiedades de las filas de plantilla de cuadrícula, columnas de plantilla de cuadrícula y áreas de cuadrícula.
grid-template-areas	Especifica cómo mostrar columnas y filas, utilizando elementos de cuadrícula con nombre.
grid-template-columns	Especifica el tamaño de las columnas y cuántas columnas en un diseño de cuadrícula.
grid-template-rows	Especifica el tamaño de las filas en un diseño de cuadrícula.
row-gap	Especifica el espacio entre las filas de la cuadrícula.

Grid Container

Para que un elemento HTML se comporte como un contenedor de cuadrícula, debemos establecer la propiedad **display** en **grid** (cuadrícula) o **inline-grid** (cuadrícula en línea). Los contenedores de cuadrícula consisten en elementos de cuadrícula, colocados dentro de columnas y filas.

```
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
</div>
```

```
.grid-container {
  width: 800px;
  display: grid;
  grid-template-columns: repeat(4, auto);
  grid-gap: 10px;
  background-color: blue;
  padding: 10px;
}
```

```
.grid-container > div {
  background-color: lightblue;
  text-align: center;
  font-size: 20px;
  height: 40px;
}
```

1	2	3	4
5	6	7	8

grid-template-columns

La propiedad **grid-template-columns** define el número de columnas (y el ancho) de la cuadrícula. Se colocan los valores separados por espacios, y cada uno define el ancho de la columna respectiva. Se pueden establecer anchos en **px**, unidades relativas o **%**, aunque es recomendable utilizar la medida **fr**.

```
/*Medidas en px:*/
```

```
grid-template-columns: 300px 200px 400px;
```

1	2	3	
4	5	6	

```
/*Medidas en %:*/
```

```
grid-template-columns: 50% 35% 15%;
```

1	2	3
4	5	6

```
/*Medida automática:*/
```

```
grid-template-columns: auto auto auto;
```

1	2	3
4	5	6

Si una cuadrícula de 4 columnas tiene más de 4 elementos, se agrega automáticamente una nueva fila para colocar los elementos extra.

Unidad fracción restante (fr):

La unidad especial de Grid **fr** (fraction) representa una fracción de espacio restante en el grid.

```
grid-template-columns: 0.5fr 2fr 1fr;
```

1	2	3
4	5	6

La expresión repeat():

Se puede utilizar la expresión **repeat()** para indicar repetición de valores, indicando el número de veces que se repiten y el tamaño en cuestión.

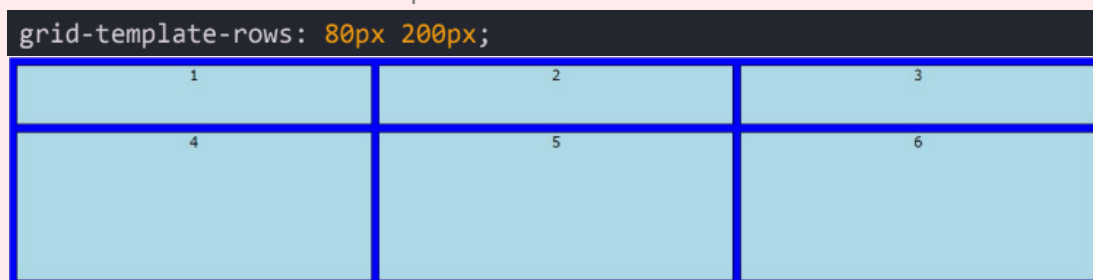
La expresión a utilizar es la siguiente: **repeat([número de veces], [valor o valores])**:

```
grid-template-columns: repeat(3, 1fr);
```

1	2	3
4	5	6

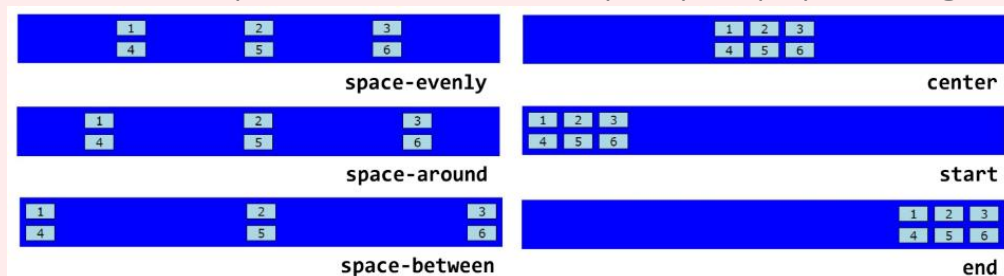
grid-template-rows

La propiedad **grid-template-rows** define la altura de cada fila. El valor es una lista separada por espacios, donde cada valor define el alto de la fila respectiva.



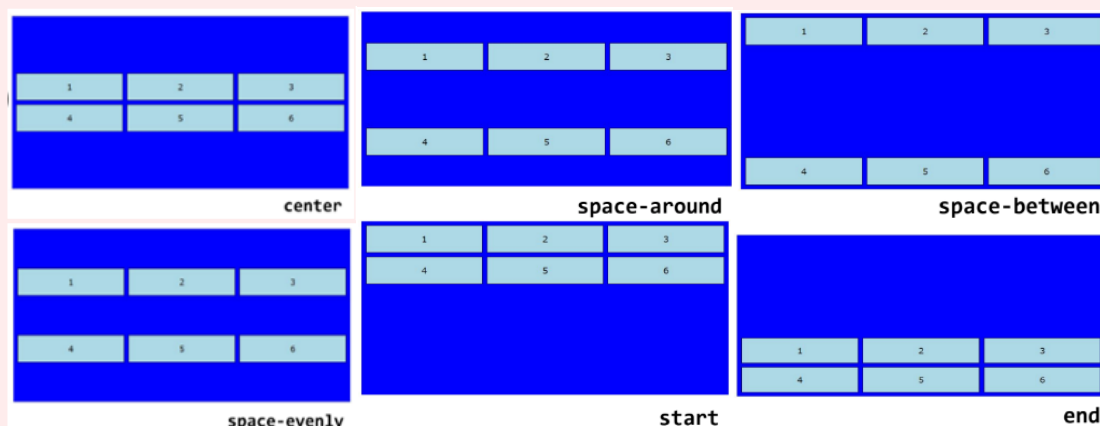
justify-content

La propiedad **justify-content** se utiliza para alinear toda la cuadrícula dentro del contenedor. El ancho total de la cuadrícula debe ser menor que el ancho del contenedor para que la propiedad tenga efecto.



align-content

La propiedad **align-content** se usa para alinear verticalmente toda la cuadrícula dentro del contenedor. La altura total de la cuadrícula debe ser menor que la altura del contenedor para que tenga efecto.



grid-area

La propiedad **grid-area** especifica el tamaño y la ubicación de un elemento de cuadrícula en el diseño, y es una propiedad abreviada para las siguientes propiedades: **grid-row-start**, **grid-column-start**, **grid-row-end** y **grid-column-end**.

La **grid-area** también se puede utilizar para asignar un nombre a un elemento de la cuadrícula. Y se puede hacer referencia a los elementos de cuadrícula con nombre mediante la propiedad **grid-template-areas** del contenedor de cuadrícula.

BOOTSTRAP

es el marco de trabajo HTML, CSS y JavaScript más popular para desarrollar sitios web receptivos y móviles.

Hay dos maneras de usar Bootstrap:

● **Sin conexión:** Descargar desde bootstrap.com. En este caso los archivos deben estar en la misma carpeta del proyecto y ser referenciados en el **<head>** del documento HTML con **<link>**:

```
<link rel="stylesheet" href="css/bootstrap.min.css">
<script src="js/bootstrap.min.js"></script>
```

● **Incluir BootstrapCDN** (*Content Delivery Network*) en el **<head>**. Este método tiene la ventaja de no necesitar instalación alguna, pero nuestro sitio va a estar conectado permanentemente con el sitio Web de Bootstrap, proveyendo los estilos.

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-F3w7mX95PdgyTmZZMECANGseQB83DfGTowi0iMjiWaeVhAn4FJkqJByhZMI3AhiU" crossorigin="anonymous">
```

Un CDN es un grupo de servidores distribuidos por todo el mundo para permitir la entrega rápida del contenido de un sitio web

Class Container

Los contenedores sirven para crear una “caja” o “contenedor” dentro de la que se coloca el contenido de una página web.

Cuando aplicamos a un elemento HTML la clase **container**, a ese elemento se le aplica **un ancho y un padding determinado** y además se centra horizontalmente en la página web.

Bootstrap proporciona 3 tipos de contenedores diferentes, cada uno con sus características distintivas.

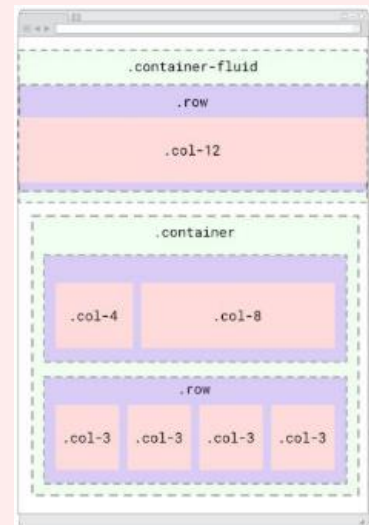
.container: establece un ancho máximo o **max-width** para cualquier tamaño de pantalla o anchos definidos por los breakpoints responsive.

Es **sensible al dispositivo que utilizemos**. Su ancho es determinado por el ancho de viewport. Es el contenedor más usado de Bootstrap.

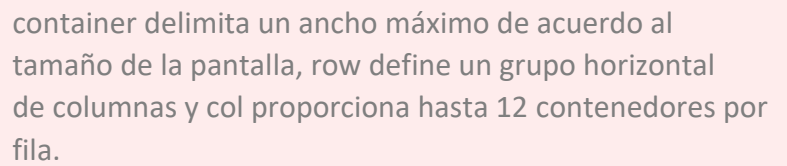
Al modificar el tamaño del viewport, el ancho máximo de este contenedor se corresponde con el “punto de corte” definido.

.container-fluid: establece un width: 100% del viewport en todos los breakpoints. Siempre ocupa el 100% del ancho de la pantalla.

.container- {breakpoint} es similar al **container-fluid**, pero posee un ancho de 100% solamente hasta el breakpoint definido.



		tamaños de pantalla					
		Extra pequeño <576px	Pequeño ≥576px	Medio ≥768px	Grande ≥992px	X-grande ≥1200px	XX-Large ≥1400px
clases	.container	100%	540 px	720px	960 px	1140 px	1320px
	.container-sm	100%	540 px	720px	960 px	1140 px	1320px
	.container-md	100%	100%	720px	960 px	1140 px	1320px
	.container-lg	100%	100%	100%	960 px	1140 px	1320px
	.container-xl	100%	100%	100%	100%	1140 px	1320px
	.container-xxl	100%	100%	100%	100%	100%	1320px
	.container-fluid	100%	100%	100%	100%	100%	100%



Fila 1 - Columna 1 de 2	Fila 1 - Columna 2 de 2	
Fila 2 - Columna 1 de 3	Fila 2 - Columna 2 de 3	Fila 2 - Columna 3 de 3

El sistema de grillas de Bootstrap permite hasta 12 columnas en la página. Es posible agrupar las columnas para crear columnas más amplias. Este sistema es responsivo, por lo tanto, las columnas se reorganizan automáticamente dependiendo del tamaño de la pantalla. Recordemos que siempre deben sumar 12.

Se utiliza una grilla de 12 columnas que se puede dividir en 2, 3, 4... 12 partes.

[illegible]

Este ejemplo crea tres columnas iguales utilizando las clases del sistema grid predefinidas.

Dichas columnas serán centradas en la página con el componente padre `.container`. Las columnas de la cuadrícula que no tengan un `width` específico se distribuirán automáticamente como columnas de igual ancho.

```
<div class="container">
  <div class="row">
    <div class="col-sm">Primer col</div>
    <div class="col-sm">Segunda col</div>
    <div class="col-sm">Tercer col</div>
  </div>
</div>
```

Tres columnas de igual ancho en dispositivos pequeños, medianos, grandes y extra grandes

Primer columna	Segunda columna	Tercer columna
----------------	-----------------	----------------

Clases receptivas

El sistema de cuadrícula Bootstrap tiene seis clases:

- **.col-** (dispositivos extra pequeños: ancho inferior a 576 px)
- **.col-sm-** (dispositivos pequeños: ancho igual o superior a 576 px)
- **.col-md-** (dispositivos medianos: ancho igual o superior a 768 px)
- **.col-lg-** (dispositivos grandes: ancho igual o superior a 992 px)
- **.col-xl-** (dispositivos xlarge: ancho igual o superior a 1200 px)
- **.col-xxl-** (dispositivos xxlarge: ancho igual o superior a 1400 px)

Las clases anteriores se pueden combinar para crear diseños más dinámicos y flexibles.

Cada clase se escala, por lo que si desea establecer los mismos anchos para **sm** y **md**, sólo necesita especificar **sm**.

Componentes

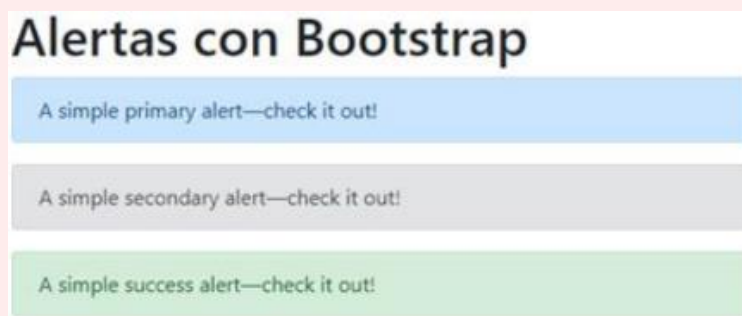
Los componentes de Bootstrap permiten acelerar el proceso de diseño. Son soluciones prediseñadas y personalizables.

Navbar: permite crear una barra de navegación o menú.

Viene preparado con el típico icono de *hamburger* (tres líneas horizontales) que aparece en la versión móvil.



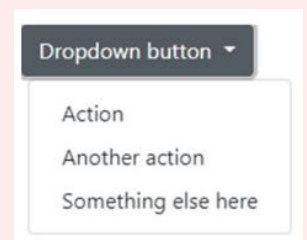
Alerts: son como cajas de texto con cierto tipo de diseño. Se suelen usar para proporcionar información puntual al usuario.



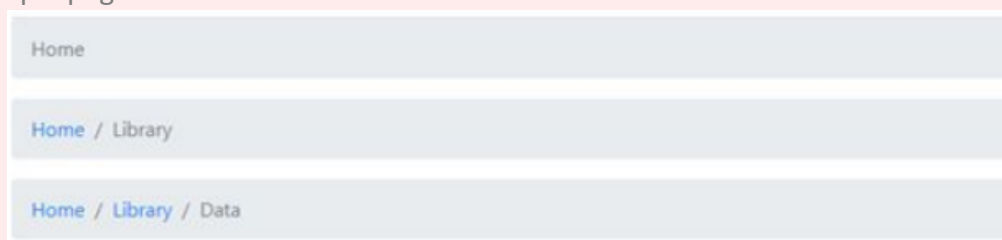
Carousel: Utiliza un sistema de slides para recorrer varios elementos. Permite contener fotografías que van pasando dentro del mismo espacio. Es un componente de presentación de diapositivas.



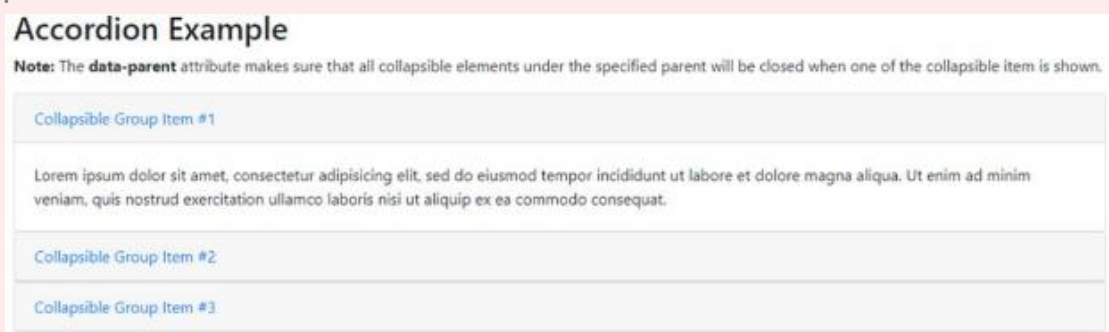
Dropdowns: Sirven para que el usuario pueda escoger una opción en un conjunto de posibilidades. Genera un menú desplegable hacia abajo o hacia a la derecha que permite incluir vínculos. Con el atributo `active` se puede marcar alguna opción del menú. Se pueden alternar para mostrar listas de enlaces y más.



Breadcrumb (o migas de pan): Sirven para mostrar la situación del usuario dentro de una página. Indica al usuario dónde está y de dónde viene. Se agregan dentro de la etiqueta semántica `<nav>`. El atributo **active** es el que indica en qué página estamos ubicados.




Collapse (accordion): Este elemento añade un botón capaz de ocultar o mostrar cierto contenido, es decir, crear elementos colapsables. Son contenidos que se despliegan y su uso es común en la sección “preguntas frecuentes”.



Buttons: Los botones por defecto son elementos **inline**, pero de ser necesario un comportamiento similar a **inline-block** podemos aplicar la clase `btn-block`.



Cards: Las cards o tarjetas, sirven para agrupar contenido. Se suelen utilizar para crear listas de elementos, por ejemplo, artículos de blog, colecciones de elementos, etc.



Título de la tarjeta

Subtítulo

Contenido de la tarjeta

Enlace 1 Enlace 2

Header

Tarjeta con borde

Contenido de la tarjeta

Tarjeta con alineación a la derecha

Contenido alineado a la derecha

Botón dentro de la tarjeta

Forms: Bootstrap aplica estilos a los elementos de tipo formulario para convertirlos en elementos responsive, mejorar su apariencia y permitirnos crear diferentes alineaciones.

Email address

We'll never share your email with anyone else.

Password

☐ Check me out

Submit

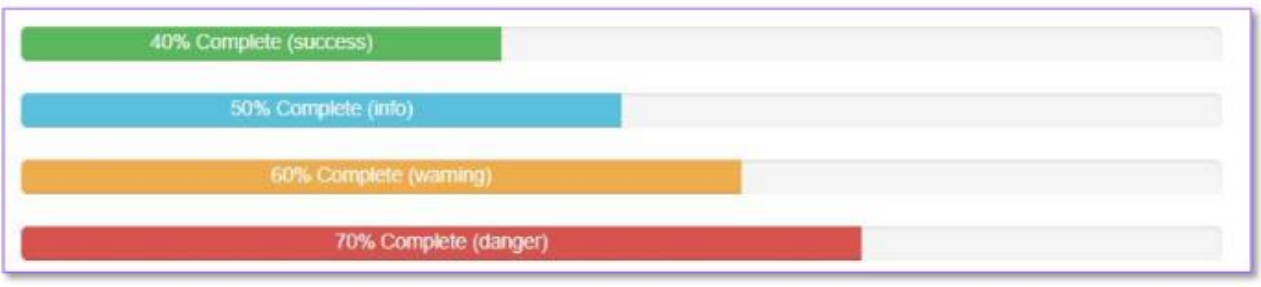
Firstname	Lastname
Default	Defaultson
Success	Doe
Danger	Moe
Info	Dooley
Warning	Refs
Active	Activeson

Firstname	Lastname
John	Doe
Mary	Moe
July	Dooley

Tables: Bootstrap dispone de distintas clases para dar estilo a las tablas, estas son algunas de ellas:

- .table (por defecto)
- .table-hover
- .table-striped

Progress bar (barras de progreso): Otra herramienta que nos presenta la librería Bootstrap son las barras de progreso. Normalmente se las utiliza para indicar cuánto se ha avanzado en una actividad. Para crear una barra de progreso debemos definir un **div** con la clase "**progress**" y un **div** interno al anterior con la clase "**progress-bar**":



Ventanas modales: Son ventanas emergentes que se abren cuando el usuario interactúa con algún elemento. Para funcionar, **modal** usa los atributos **data-toggle** con el valor “**modal**” y **data-target** con el **id** del modal que se crea. Para cerrar el modal se usa la etiqueta html de **data-dismiss=“modal”**. Modal es un contenedor.

```
<button type="button" class="btn btn-primary" data-toggle="modal" data-target="#exampleModal">
  Abrir popup
</button>

<div class="modal fade" id="exampleModal" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel"
  aria-hidden="true">
```

Otros Frameworks CSS



Materialize es un framework CSS que implementa el tema de diseño “*Material Design*”. Ofrece componentes material listos para usar, que se pueden integrar de una manera cómoda en los sitios web, consiguiendo un diseño guiado por las directrices de aplicaciones y sitios de Google.

El framework es sencillo de usar, relativamente ligero, permite optimización y los componentes están altamente personalizados en su diseño.



Tailwind es un framework CSS que ofrece un enfoque diferente a Bootstrap, posee clases y una gran biblioteca que te permitirá acelerar el proceso de diseño de cualquier sitio web. Estos frameworks ofrecen estilos CSS atómicos, aunque también permite crear componentes, lo deja más del lado del desarrollador, que los podrá personalizar a su gusto. Además es muy maleable y se adapta muy bien a las necesidades del desarrollador. Con el framework puedes hacer builds de clases CSS totalmente personalizadas, que se parezcan o no a las que se ofrecen de manera predeterminada.