<codoa codo/>

CSS

Nos vestimos para la ocasión

Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad
Educación

¿Qué es?

CSS es utilizado para diseñar y dar estilo a las páginas web, por ejemplo, alterando la fuente, color, tamaño y espaciado del contenido, dividirlo en múltiples columnas o agregar animaciones y otras características **decorativas**.

Hojas de Estilo en Cascada (del inglés *"Cascading Style Sheets"*), es decir, CSS hace referencia al comportamiento que tiene este lenguaje cuando nuestros estilos entran en conflicto.

Sintaxis de CSS

selector

Elemento de nuestro HTML que deseamos aplicar estilos.

propiedad

Nombre de la propiedad CSS que queremos modificar.

valor

Valor que le asignamos a esa propiedad.

Sintaxis de CSS

```
1  <p>¡CSS es lo máximo!</p>
```

¡CSS es lo máximo!

```
1  p {
2      color: #2b2b2b;
3      font-size: 52px;
4  }
```

Vinculamos CSS a nuestro proyecto.

Interno

Colocamos nuestro código CSS dentro de la etiqueta **<style></style>** en el head de nuestro archivo HTML.

```
<head>
    <style>

    </style>
</head>
```

Externo

Usamos la etiqueta **<link />** donde el **attr *rel*** indica la relación con el recurso y el **attr *href*** la ruta a nuestro archivo css.

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

En línea

Se coloca en el **attr *style*** del elemento/etiqueta HTML a modificar.

```
<p style="color: pink">¡CSS es lo máximo!</p>
```

Selectores CSS

Universal

Selecciona todos los elementos de HTML.

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

Etiqueta

Se utiliza para seleccionar una etiqueta específica.

```
p {
    font-family: Arial, Helvetica, sans-serif;
}
```

clase

Selecciona todos los elementos html que contengan ese atributo ***class***.

```
.cards {
    width: 300px
}
```

id

Selecciona todos los elementos html que contengan ese atributo ***id***.

```
#banner {
    margin-top: 20px;
}
```

Práctica en vivo

Creamos un nuevo archivo llamado ***index.css***, lo vinculamos con nuestro HTML y le aplicamos color y tamaño de fuente a una etiqueta **párrafo** con un **lorem ipsum**.

Herencia

Es un comportamiento de CSS el cual permite que algunas ***propiedades asignadas*** a un elemento ***padre*** sean adquiridas o heredadas por **sus** elementos ***hijos***.

Cabe destacar que esto aplica solo a ***algunas*** propiedades y suelen ser bastante intuitivas, como por ejemplo el color de fuente o la familia tipográfica.

```html
<h1>Nueva sección del sitio</h1>
<article>
    <p>
        Lorem ipsum dolor sit amet consectetur elit.
        Magni laudantium cupiditate odio rem alias
        numquam obcaecati ea, excepturi tempora animi
        provident? Rem eligendi numquam odio velit a,
        consectetur ab blanditiis?
    </p>
</article>
```

```css
body {
    font-family: 'Merriweather Sans', sans-serif;
}

article {
    color: crimson;
}
```

Nueva sección del sitio

Lorem ipsum dolor sit amet consectetur elit. Magni laudantium cupiditate odio rem alias numquam obcaecati ea, excepturi tempora animi provident? Rem eligendi numquam odio velit a, consectetur ab blanditiis?

El elemento ** ** *hereda* el **color** de texto de **<article></article>** y la *tipografía* de **<body></body>** mientras que el **<h1></h1>** sólo hereda la **fuente** y no el **color**.

Es **importante** mencionar que una vez ***asignada la misma propiedad heredada al elemento hijo*** (aunque tenga distinto valor), éste corta la **herencia de su elemento padre**.

Precedencia

Es muy **importante** en CSS ya que de esta depende la jerarquía con la que se aplicarán los estilos a nuestros elementos HTML, por ende, decidirá qué *estilo prevalece* sobre los demás cuando para un elemento apliquen 2 o más estilos.

Junto con el, **también** entra en juego el término de estilos en cascada que como vimos anteriormente, está presente en el nombre de CSS.

Orden de precedencia

etiqueta < clase < id < estilo en línea < !important

¿Cuál de todos estos será el que prevalece? 🤔

```css
p {
    font-size: 16px;
}

.parrafos-main {
    font-size: 18px;
}
```

```css
#parrafo-principal {
    font-size: 20px;
}

.parrafo-diferenciado {
    font-size: 14px!important;
}
```

```html
<p style="font-size: 30px;">
    ¡CSS es lo máximo!
</p>
```

Estilos en Cascada

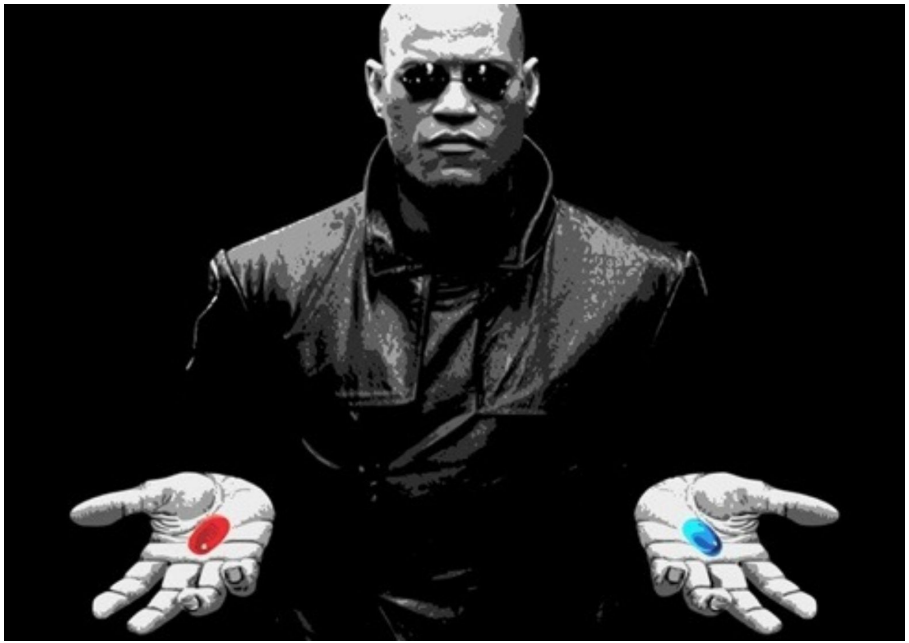
Además de la **precedencia** existe la jerarquía por ***cascada*** que es la responsable de decidir qué estilos **prevalecen** cuando dos o más selectores de un mismo elemento tienen la misma especificidad.

En estos casos, siempre se ***tomará el último*** valor vigente.

Estilos en Cascada

¿Cuál es tu elección?

```
p {
    font-size: 16px;
    color: red;
}

p {
    color: blue;
}
```



Si elegiste la Azul...

¡Ingresaste a la Matrix! 🖐

Veamos algunos ejemplos

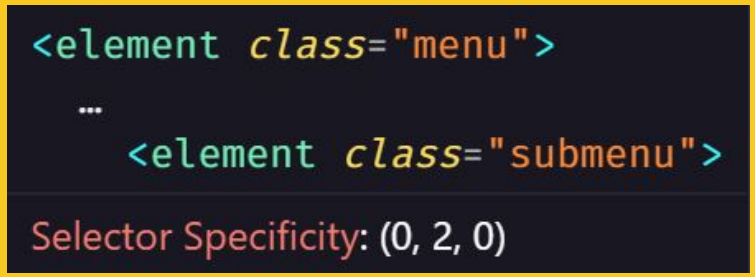
Especificidad

La unión hace la fuerza

Como vimos la clase pasada, dependiendo del tipo de selector que utilicemos ***conseguiremos*** mayor o menor precedencia sobre un elemento, siendo el selector por ***ID*** el más poderoso de todos.

También existe la *posibilidad* de **combinar** distintos *selectores* para poder reforzar la selección o llegar a un elemento mucho más **fácil** teniendo en cuenta el árbol de etiquetas anteriores a él.

Cada **selector** posee una **especificidad** y esta nos va a indicar cuán **directa** es la selección del elemento.



Especificidad según selectores

Si bien es imposible representar todas las combinaciones, cuantos más selectores combinemos más especificidad tendrá nuestra selección.

Etiqueta + Etiqueta:

Seleccionamos todos los ** ** que son hijos de una etiqueta **<article></article>** - **E: (0,0,2)**

```
<article>
  …
    <p>
Selector Specificity: (0, 0, 2)
article p{
    color: crimson;
}
```

Clase + Etiqueta:

Seleccionamos todos los **
** que son hijos de **.menu** - **E: (0,1,1)**

```
<element class="menu">
  …
    <li>
Selector Specificity: (0, 1, 1)
.menu li{
    padding: 10px
}
```

ID + Clase:

Seleccionamos todos los **.shirts** hijos de **#sales**.

```
<element id="sales">
  …
    <element class="shirts">
Selector Specificity: (1, 1, 0)
#sales .shirts {
    color: #2b2b2b;
}
```

Selectores combinados

Esto nos abre una puerta a nuevas combinaciones de selectores.

.menu > li:

Seleccionamos todos los **
** que son hijos directos de un elemento con la clase **.menu**

```
<ul class="menu">
    <li>Este si</li>
    <ul>
        <li>Este No</li>
    </ul>
</ul>
```

h2 + p :

Selecciona la etiqueta ** ** que sea hermana directa y adyacente de una etiqueta **<h2></h2>**

```
<h2>Un título</h2>
<p>Este si</p>
<p>Este no</p>
```

h2 ~ p :

Selecciona todas las etiquetas ** ** que sigan luego de una etiqueta **<h2></h2>**

```
<h2>Un título</h2>
<p>Este si</p>
<p>Este si</p>
<p>Este si</p>
```

Selectores combinados

.link.active:

Seleccionamos la etiqueta con la clase **.link** y la clase .**active**

```
<a href="" class="link active">
    Enlace
</a>
```

#banner .img :

Seleccionamos la etiqueta con la clase **.img** hija del elemento con id **#banner**

```
<section id="banner">
    <picture class="img">

    </picture>
</section>
```

#ofertas.cards :

Seleccionamos la etiqueta con el id **.ofertas** y la clase .**cards**

```
<section id="ofertas" class="cards">

</section>
```

CSS

Nos vamos conociendo

Fuentes

En CSS podemos ***utilizar*** infinidad de fuentes o familias tipográficas.

Por defecto, solo podremos acceder a las que ofrecen los navegadores de forma nativa pero por fortuna siempre **podemos agregar nuevas** opciones.

Profundicemos un poco más.

Tipos de fuente

Serif

Poseen terminaciones con serifa o acabado elegante.

Sans Serif

Tipografías de palo seco con terminaciones rectas o ligeramente redondeadas.

Display

NORMALMENTE UTILIZADAS PARA TÍTULOS POR SU CAPACIDAD DE RESALTAR.

Mono

Cada letra mide lo mismo de ancho, suelen ser las de máquina de escribir o terminal de computadora.

Handwriting

Simulan la escritura hecha a mano.

Propiedad *font-family*

```
body {
    font-family: 'Times New Roman', Times, serif;
}
```

Se divide en varias partes:

- El valor entre comillas ‘ ‘, corresponde al **nombre** de la fuente seleccionada.
- El segundo valor es el **reemplazo** en caso que no se encuentre esa fuente.
- El tercero es la familia tipográfica para que cargue la fuente **por defecto del navegador** para este tipo en caso que las anteriores **no funcionen**.

Importar fuentes externas

URL:

Se importa directamente en nuestra hoja HTML mediante una etiqueta **<link href="" />** a la URL del recurso solicitado (por ejemplo ***Google Fonts***).

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Merriweather+Sans:wght@300;
400;500&display=swap" rel="stylesheet">
```

También se puede importar sobre la hoja de estilos CSS a través de la propiedad ***@import***

```
@import url('https://fonts.googleapis.com/css2?family=Merriweather+Sans:wght@300;400;
500&display=swap');
```

Mas Fuentes en: https://fonts.google.com/

Importar fuentes externas

Archivo:

Se hace mediante la propiedad **@font-face** de CSS.

```
@font-face {
    font-family: <un-nombre-de-fuente-remota>;
    src: <origen> [,<origen>]*;
    [font-weight: <peso>];
    [font-style: <estilo>];
  }
```

```
@font-face {
  font-family: "Bangers", sans-serif;
  src: url("../fonts/bangers.ttf");
}
```

Colores

En el mundo del diseño existen infinidad de sistemas para trabajar los ***colores***, *CMYK, RGB, HSL, Hexadecimal, entre otros* y cada uno tiene una **aplicación distinta**, es decir, podrán ser mejores para imprimir, para sublimar o para ***su proyección en pantallas***.

En este apartado, nos centraremos en este último grupo donde **encontramos** que los códigos más usados son: rgb, rgba, hsl, hexadecimal y el grupo de colores **nativos del navegador**.

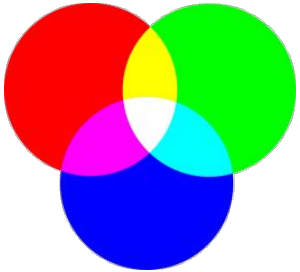
Códigos de Color

Hexadecimal

Se antepone un **#** antes del código que está compuesto por **6** caracteres entre números del **0 al 9** y letras de la **A a la F**.

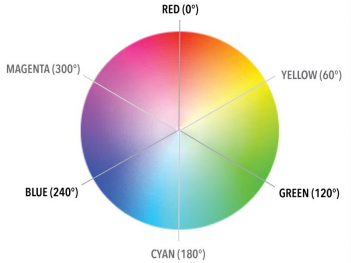
RGB - RGBA

Es la mezcla de los componentes **Red**, **Blue** y **Green** pudiendo además incluir un canal ***"alpha"*** para indicar transparencia.



HSL

Hue, Saturation y Lightness combinación de **matiz** de color (**0 a 360**) y los **porcentajes** de **0 a 100%** de saturación y luminosidad.



Agencia de Aprendizaje
a lo largo de la vida

Códigos de Color

Nativos del navegador

Son los **colores** que por defecto nos ofrecen los **navegadores**. Estos poseen nombres propios como **crimson**, **navy** o **deeppink** y son un total de **150**.

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

Iconos

Son aquellos **recursos gráficos** que nos ayudan a comunicar o explicar **elementos** en diferentes lugares de nuestro sitio.

Normalmente son en formato ***SVG*** aunque en algunos casos, se importan desde librerías que los ponen a disposición a través de etiquetas HTML.

Algunas de las librerías más conocidas son Font Awesome e Iconify.

Font Awesome

La ***importamos*** a nuestro proyecto **descargando** el código fuente o a través de un gestor de paquetes. Luego, la misma librería nos pone a disposición un ***stock de íconos*** que se deben usar mediante una etiqueta **** con una **clase** CSS específica que mostrará el **ícono**.

```
<span style="font-size: 3em; color: Tomato;">
  <i class="fas fa-camera"></i>
</span>

<span style="font-size: 48px; color: Dodgerblue;">
  <i class="fas fa-camera"></i>
</span>

<span style="font-size: 3rem;">
  <span style="color: Mediumslateblue;">
  <i class="fas fa-camera"></i>
  </span>
</span>
```

https://fontawesome.com/

Iconify

Al igual que el anterior se puede instalar mediante un *gestor de paquetes*, a través de una etiqueta **<script></script>**[1] o directamente copiando y pegando el **SVG**[2] en nuestro **HTML**.

2

SVG | SVG with viewBox rectangle | SVG as data: URI

```
<svg xmlns="http://www.w3.org/2000/svg" width="1em" height="1em" preserveAspectRatio="xMidYMid meet" viewBox="0 0 24 24"><path fill="none" stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="m4 15l8-8l8 8"/></svg>
```

Download SVG

1

```
<span class="iconify" data-icon="akar-icons:chevron-up"></span>
```

```
<script src="https://code.iconify.design/2/2.2.1/iconify.min.js"></script>
```

https://iconify.design/

CSS

Estructura en nuestros estilos

Unidades de Medida

Existen **muchas** y cada una tiene una aplicación para cada caso particular.

Lo primero que debemos saber es que al igual que los enlaces en HTML existen de **2 tipos**, unidades **relativas** y **absolutas** cuya diferencia radica en si ese valor siempre va a tomar el mismo tamaño o si va a estar relacionado al tamaño de algo más.

Absolutas

Son medidas **fijas** y no dependen de ningún otro factor.

Ideales en contextos donde las medidas no varían como en los **medios impresos** (documentos, impresiones, etc...), pero **poco adecuadas** para la web, ya que no se adaptan a diferentes resoluciones o pantallas, que es lo que tendemos a hacer hoy en día.

Si bien existen muchas como **cm** (centímetros), **mm** (milímetros), **in** (pulgadas), **pc** (picas), **pt** (puntos), etc...

La más conocida son los ***Pixeles*** por su fácil uso y aplicación práctica en pantallas.

```
img {
    width: 300px;
}
```

Relativas

Mucho más potente y flexible en CSS. Al contrario de las unidades absolutas, dependen de algún otro factor (resolución, tamaño de letra, etc...). Tienen una curva de aprendizaje más compleja, pero son ideales para trabajar en dispositivos con diferentes tamaños, ya que son muy versátiles.

em -> 1em = tamaño de fuente relativo a la **herencia** o al *valor por defecto* del **navegador**.

rem -> 1rem = tamaño de fuente **relativo** al valor por defecto del **navegador**.

vw -> 100vw = total del **ancho** visible del navegador.

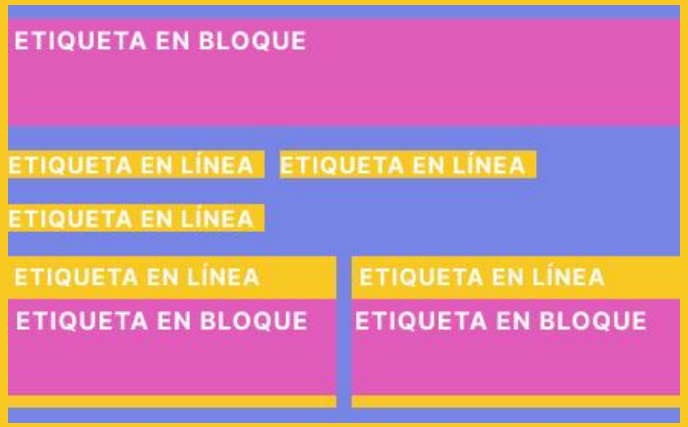
vh -> 100vh = total del **alto** visible del navegador

% Porcentaje Relativo al tamaño del elemento padre.

Displays

Recordemos que, por defecto, cada elemento HTML tiene un tipo de representación concreta. Esos valores eran display **block** o **inline** y estaban relacionados de forma nativa a cada etiqueta.

Sin embargo, estos comportamientos nativos pueden ser *modificados*.



Displays

Esta propiedad *cambia* el tipo de representación del elemento indicado y si bien **puede tomar muchos** valores diferentes, por ahora nos concentraremos en **4** de los cuales ya conocemos algunos.

```
display: block | inline | inline-block | none;
```

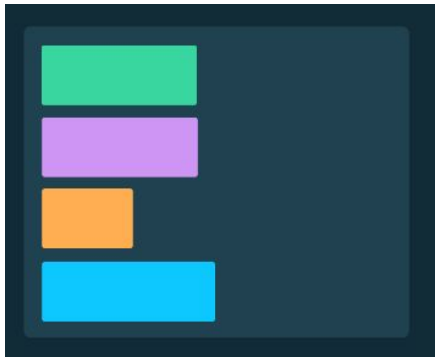
Valores de display

block
Ocupan el **100%** del ancho de su contenedor y comienzan en una nueva línea.

inline
Ocupan el ancho de su contenido y **no aceptan** propiedades de **width**, **height** o **margins** y **paddings** superiores.

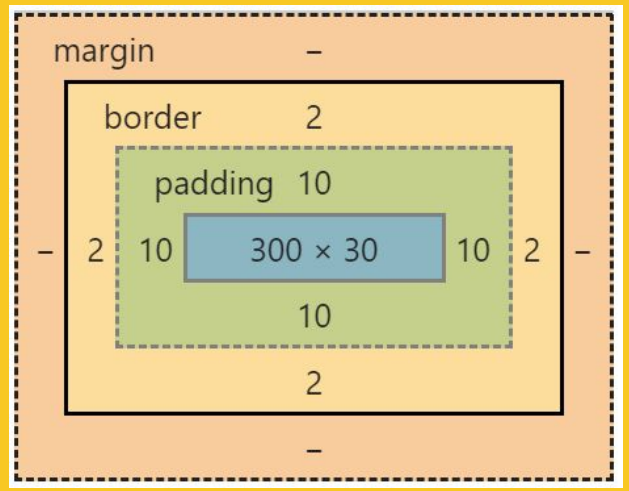
inline-block
La **combinación de los anteriores**, ocupa el ancho de su contenido pero ***sí acepta*** que se modifique su tamaño o sus propiedades de caja.

none
Este valor resulta en que **el elemento seleccionado no sea mostrado** ni ocupe espacio en el lugar donde debería estar.



Modelo de Caja

Es un sistema que tiene el navegador para interpretar las diferentes partes de lo que solemos denominar *cajas*, es decir, un elemento HTML con ciertos límites y dimensiones.

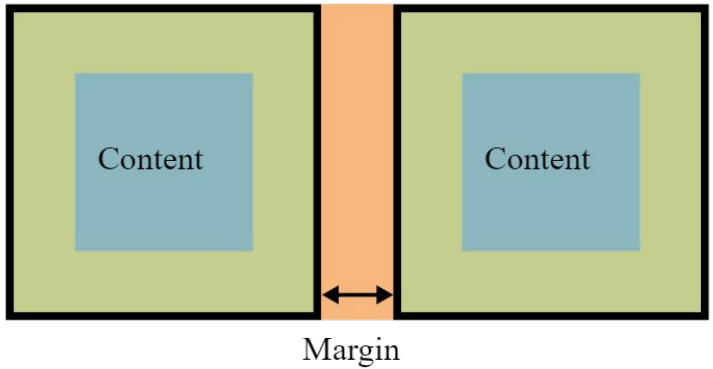


Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad
Educación

Propiedad *margin*

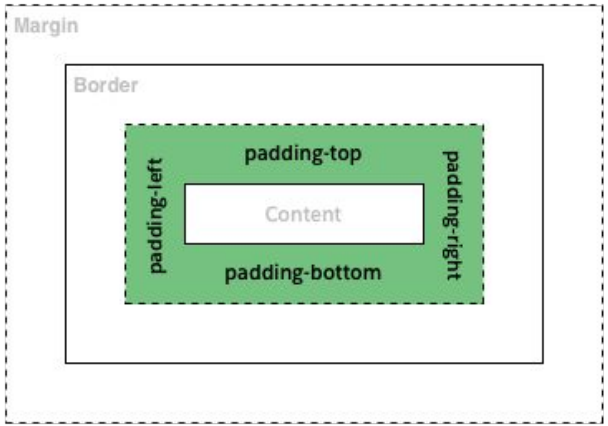
Se utiliza para ***crear espacio*** alrededor de los elementos, **FUERA** de los bordes definidos.




```
/* top right bottom left */
margin: 10px 20px 10px 20px;
/* top right|left bottom */
margin: 10px 20px 10px;
/* top|bottom right|left */
margin: 10px 20px;
/* top|right|bottom|left */
margin: 10px;
```

Propiedad *padding*

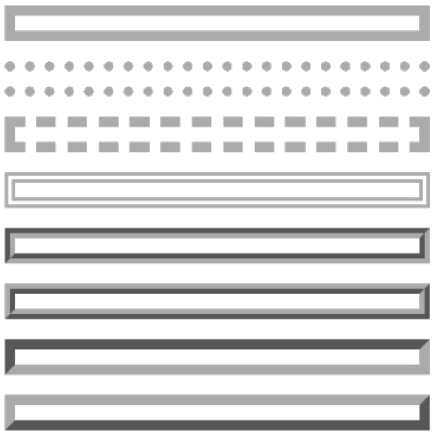
Se utiliza para ***crear espacio*** alrededor de los elementos, **DENTRO** de los bordes definidos.




```
/* top right bottom left */
padding: 10px 20px 10px 20px;
/* top right|left bottom */
padding: 10px 20px 10px;
/* top|bottom right|left */
padding: 10px 20px;
/* top|right|bottom|left */
padding: 10px;
```

Propiedad *border*

Permiten especificar el **estilo**, el **ancho** y el **color** del borde de un elemento.




```
/* ancho estilo color*/
border: 2px solid #f32872;
```

Overflow

Sucede cuando superamos los límites de tamaño de nuestros contenedores.

Dependiendo el caso, puede **generar scroll vertical u horizontal**, ocultar el contenido sobrante o dejarlo simplemente que fluya.

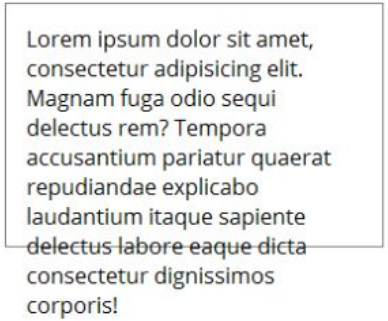
Valores de overflow

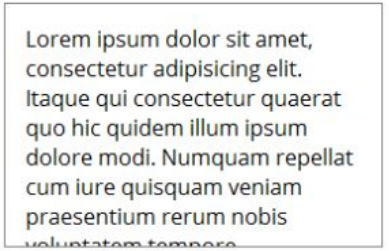
auto
Se colocan **barras de desplazamiento** (sólo las necesarias).

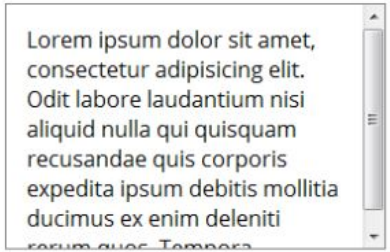
hidden
Se **oculta** el contenido que sobresale.

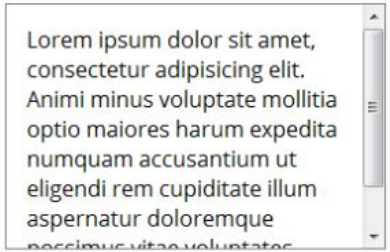
visible
Se **muestra el contenido que sobresale** (comportamiento por defecto)

scroll
Se colocan **barras de desplazamiento** (horizontales y verticales).









POSITIONS

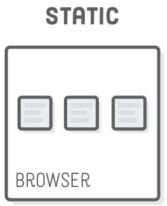
No te dejes vencer por un diseño

Positions

Hasta el momento aprendimos a manejar y posicionar los elementos de una web en base a un flujo estático y contínuo donde las cajas se iban creando en el orden en el cual **fueron escritas** en el HTML.

Gracias a los positions, vamos a poder modificar el flujo estático de nuestros elementos, permitiendo superposiciones o cambios referencia sobre los que las cajas están dispuestas.

Position y sus valores



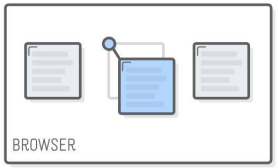
La propiedad `position:` cuenta con los siguientes valores:

```
static | relative | absolute | fixed | sticky
```

`static`

Valor por defecto. Este valor indica que el elemento debe adoptar el flujo natural del sitio.

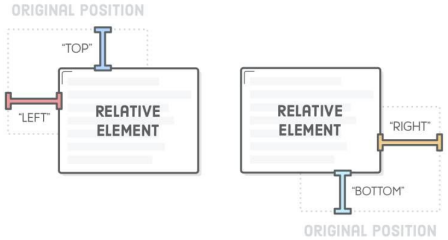
Position y sus valores



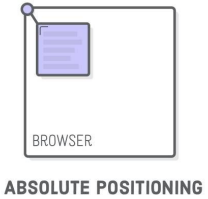
RELATIVE POSITIONING

`relative`

Se comporta igual que static a menos que le agreguemos las propiedades: `top | bottom | right | left` causando un **reajuste en su posición** y *sin modificar el espacio que ocuparía originalmente.*



Position y sus valores



`absolute`

La posición de una caja se establece de forma **absoluta** respecto de su elemento contenedor relative, o el body por defecto.

El resto de elementos de la página ignoran la nueva posición del elemento.

Position y sus valores

`fixed`

Hace que la caja esté posicionada con respecto a la ventana del navegador, lo que significa que **se mantendrá en el mismo lugar** incluso al hacer scroll en la página.

Position y sus valores

`sticky`

La caja **se mantiene static** *hasta que el scroll del navegador llega a ella* y *se comporta como fixed*. Una vez que el tamaño de su contenedor llega a su fin, **vuelve a comportarse como static**.

z-index

En los momentos que nuestras cajas con position se superpongan, podemos utilizar la propiedad **z-index** para manejar el orden de las capas.

CSS

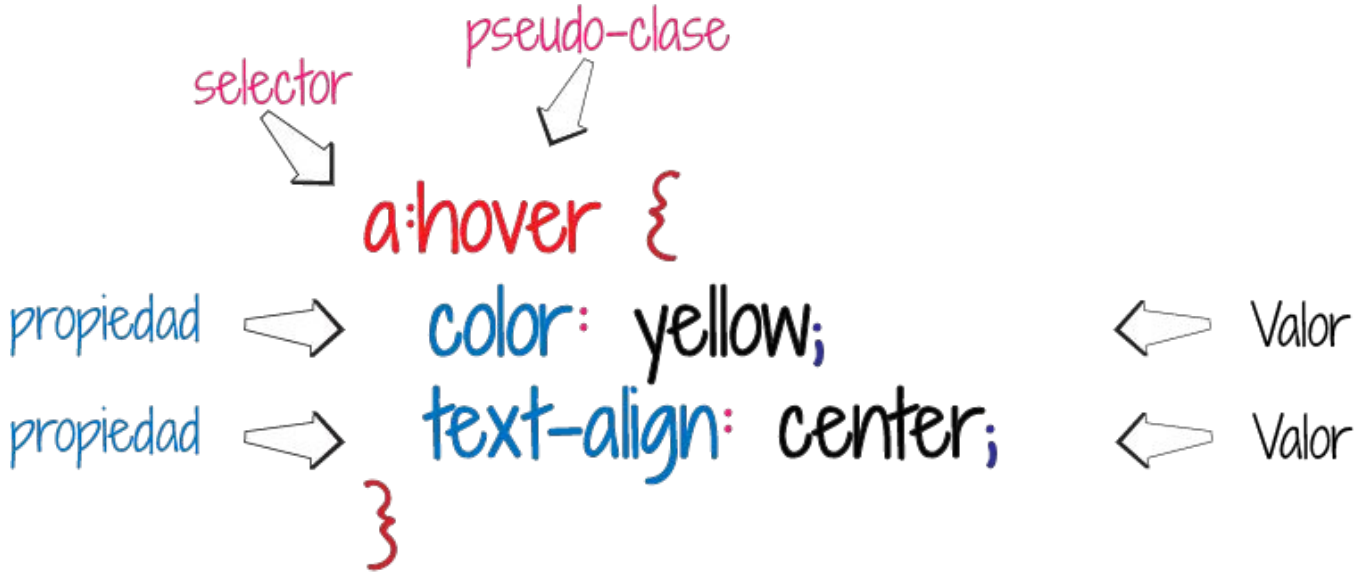
Hasta el infinito y más allá…

Pseudoclases

Una **pseudoclase** en CSS es una palabra clave añadida a un selector que especifica un estado especial del elemento o elementos seleccionados.

Permiten **aplicar un estilo** a un elemento no sólo en relación con el contenido de la estructura del documento, **sino también en relación a los factores externos**.

Definición


selector
pseudo-clase
a:hover {
propiedad
color: yellow;
Valor
propiedad
text-align: center;
Valor
}

pseudoclases de enlaces :link y :visited

:link	Aplica estilos cuando el enlace no ha sido visitado todavía.
:visited	Aplica estilos cuando el enlace ha sido visitado anteriormente.

```
a:link {
  background-color: lightblue;
}
```

Este párrafo contiene un enlace al párrafo siguiente.

Este párrafo es el destino del enlace anterior.

```
a:visited {
  color: red;
}
```

Este párrafo tiene un enlace a sí mismo. Si el navegador reconoce que el enlace se ha visitado, en vez de en morado, se muestra en rojo.

pseudoclases de dinámicas :hover y :active

:hover	Aplica estilos cuando pasamos el ratón sobre un elemento.
:active	Aplica estilos cuando estamos clickeando sobre el elemento.

pseudoclases de formulario :focus y :checked

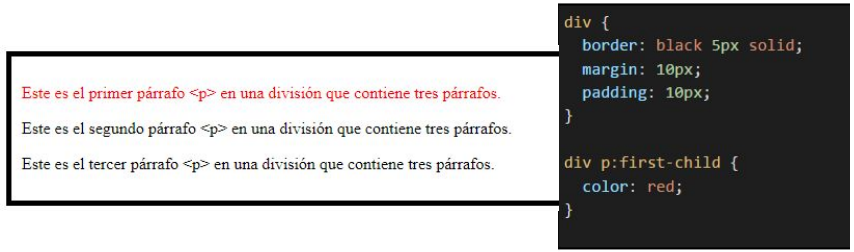
:focus	Aplica estilos cuando el elemento tiene el foco.
:checked	Aplica estilos cuando la casilla está seleccionada.

Gradient border focus fun

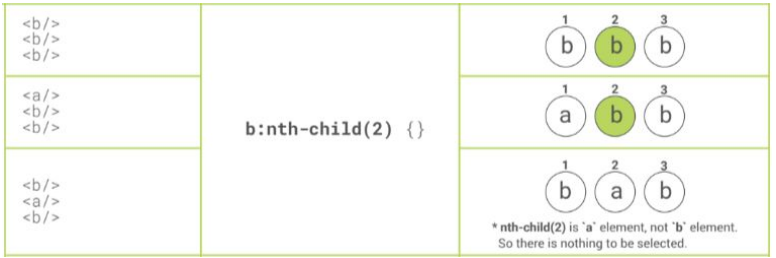
pseudoclases a elementos hijos de distinto tipo

Existen varias pseudoclases que permiten hacer referencias a los elementos del documento HTML según su posición y estructura de los elementos hijos.

`:first-child/:last-child` se utiliza para representar al primer o último elemento entre un grupo de elementos hermanos.



`:nth-child()` coincide con un elemento en función de su posición entre un grupo de hermanos.



pseudoclases a elementos hijos por mismo tipo

`:first-of-type/:last-of-type` se utiliza para representar al primer o último elemento entre un grupo de elementos hermanos de la misma etiqueta.

`:nth-of-type()` coincide con un elemento en función de su posición entre un grupo de hermanos de la misma etiqueta.

First of Type Selector	`<a/>` `<b/>` `<a/>` `<b/>`	`b:first-of-type {}`	a b a b
Nth of Type Selector	`<a/>` `<b/>` `<a/>` `<b/>` `<a/>` `<b/>` `<a/>`	`a:nth-of-type(2) {}`	1 a b 2 a b 3 a b 4 a
		`a:nth-of-type(even) {}`	1 a b 2 a b 3 a b 4 a
		`a:nth-of-type(odd) {}`	1 a b 2 a b 3 a b 4 a
		`a:nth-of-type(2n+1) {}`	1 a b 2 a b 3 a b 4 a

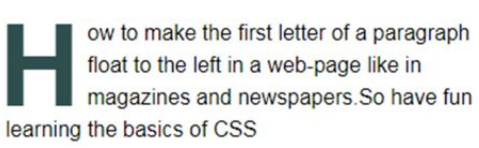
* `n` is an every positive integer or zero value.

Pseudoelementos

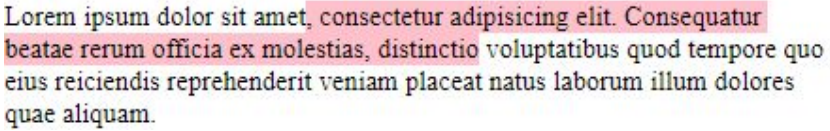
Son otra de las características de CSS que permiten hacer referencias a «comportamientos virtuales no tangibles», o lo que es lo mismo, se le puede dar estilo a elementos que no existen realmente en el HTML, y que se pueden generar desde CSS.

pseudoelementos

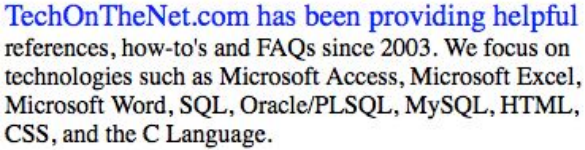
`::first-letter` se utiliza para darle estilo a la primera letra de un texto.



`::selection` se utiliza para darle estilo al texto seleccionado.



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



`::before` puede utilizarse para agregar algo antes del contenido de un elemento.

```
a::before{ content:"✨";}
```

`::after:` coloca contenido después de un elemento

```
a::after{ content:"🙊";}
```

Transiciones

Las transiciones permiten manejar cómo cambia el valor de una o más propiedades en un período de tiempo sobre un evento determinado.

Para crear un efecto de transición, hay que especificar dos cosas:

- La propiedad CSS a la que desea agregar un efecto.
- La duración del efecto.

Propiedad `transition`

`transition-property:` para definir qué propiedad vamos a alterar. `all | none | <prop>`

`transition-duration:` para definir la duración de la transición. `s | ms`

`transition-delay:` definimos el tiempo a esperar antes de que se ejecute la animación. `s | ms`

`transition-timing-function:` curva de velocidad del efecto de la transición. `linear | ease | ease-in | ease-out | ease-in-out | cubic-bezier() | step-end | steps()`

`transition:` shorthand property de las anteriores.

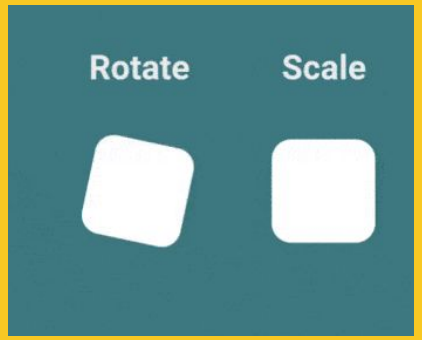
```
transition: width 2s 800ms ease;
```

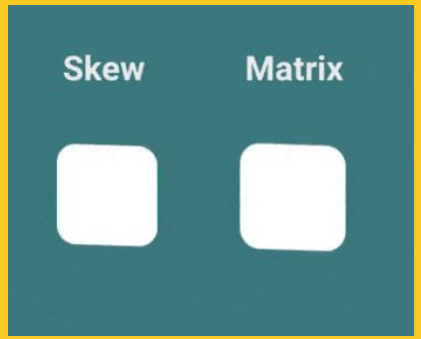
Veamos algunos ejemplos...

Transformaciones

Permiten rotar, torcer, escalar o desplazar los elementos de nuestra página web

Son una de las características de CSS más interesantes y potentes que nos ayudan a convertir las hojas de estilo en un sistema capaz de realizar efectos visuales 2D y 3D.





Propiedad `transform`

Las dos propiedades que nos sirven para **definir las transformaciones** son:

`transform:` propiedad **base** para cualquier transformación. **El punto de origen se ubica en el centro del elemento.**

`transform-origin:` nos permite **cambiar el punto de origen** de nuestra transformación.

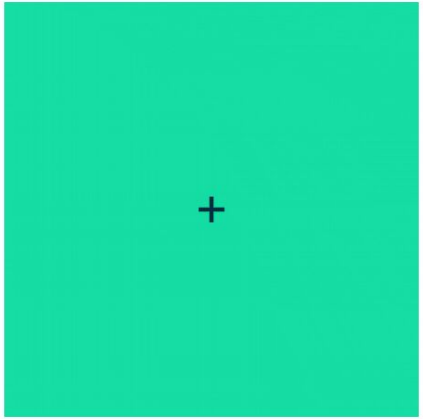
En esta clase nos centraremos en ver las transformaciones 2D más comunes.

scale()

Modifica el tamaño de los elementos. Esta función se establece con uno o dos valores, que representan la cantidad de escala que se aplica **en cada dirección**: scale(x) o scale(x,y).

Valores entre 0 y 1 achican y mayores a 1 agrandan.

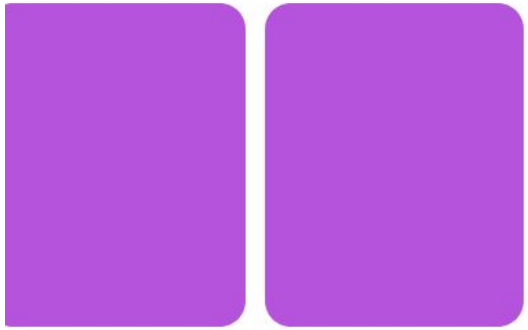
```
div {
    /* 2,5 - X,Y */
    /* 2 - X/Y */
    transform: scale(2,2);
}
```



translate()

Cambia la posición del elemento hacia la izquierda, derecha, arriba o abajo. Esta función se establece con uno o dos valores: translate(x) o translate(x,y).

```
div {
    /*
    Valores positivos mueven a la derecha/abajo.
    Valores negativos mueven a la izquierda/arriba.
    */
    transform: translate(10, 30);
}
```



rotate()

Gira o rota los elementos en grados: rotate(deg).

La unidad de medida es **deg** y puede tomar **valores positivos** (gira hacia la derecha) o **negativos** (gira hacia la izquierda) de 0 a 360°.

```
div {
    transform: rotate(230deg);
}
```

Rotate

skew()

Distorsiona los elementos según el ***ángulo en grados***. Esta función se establece con uno o dos valores: skew(x) o skew(x,y).

La unidad de medida utilizada también es deg al igual que en rotate().

```
div {
    transform: skew(20deg,10deg);
}
```

Animaciones

Las **animaciones** permiten animar la transición entre un estilo CSS y otro. A diferencia de los transitions, estos no se disparan frente a un evento determinado, si no que **comienzan desde que el sitio es cargado**.

Para utilizar animaciones, es necesario crearlas o definirlas previamente mediante la regla css @keyframes.

Propiedades de Animaciones

Constan de dos componentes:

- Propiedades CSS de las animaciones, que definen el comportamiento de la misma.
- Un conjunto de fotogramas que indican su estado inicial y final, así como posibles puntos intermedios en la misma.

Shorthand property

```
animation: mi-animacion 5s linear 0.2s infinite normal;
```

```
animation-name: <nombre de la animación>;
```

```
animation-duration: <duración>s | ms;
```

```
animation-delay: <retardo>s | ms;
```

`animation-timing-function:` efecto de transición. `linear | ease | ease-in | ease-out | ease-in-out | cubic-bezier() | step-end | steps();`

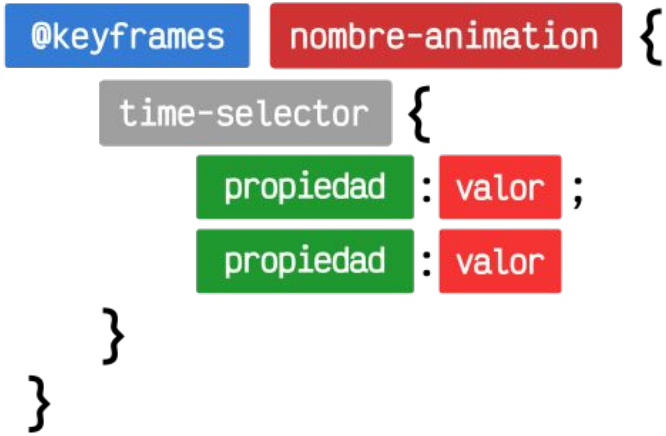
```
animation-direction: normal | reverse | alternate;
```

```
animation-iteration-count: <veces> | infinite;
```

@keyframes

Regla CSS donde vamos a **crear una animación** para **utilizarla después** todas las veces que lo necesitemos.

```css
@keyframes agrandar {
    from {
        width: 100px;
        height: 100px;
        background-color: orange;
    }

    to {
        width: 500px;
        height: 500px;
        background-color: crimson;
    }
}
```



CSS

Estructura en nuestros estilos

Hoy presentamos...

Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad
Educación

FLEXBOX

Nuestro nuevo mejor amigo

¿Qué es Flexbox?

Como mencionamos anteriormente, en el **pasado** para maquetar la estructura de *un sitio web*, era necesario **recurrir** a las tablas o a la propiedad **float** y sus raros comportamientos.

Flexbox es un **sistema** de elementos que nos invita a olvidar estos mecanismos y utilizar una mecánica más potente, limpia y flexible, en la que los elementos HTML *se adaptan y fluyen automáticamente*.

*De este modo **lograremos** que nuestros diseños sean adaptables a diferentes tamaños de pantallas con menos código.*

Flex Container - Flex Item

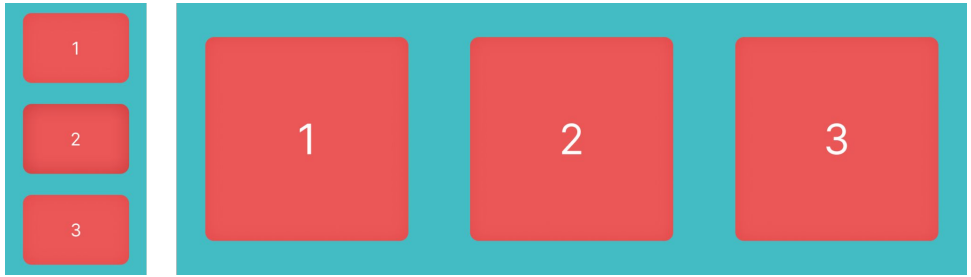
En Flexbox para **posicionar** nuestros elementos debemos asignarle el comportamiento que buscamos al contenedor padre de estos, a diferencia de otras alternativas como ***display inline, inline-block*** o con la propiedad **float** donde la propiedad **se aplicaba** a cada elemento de forma individual.

Esto nos trae un nuevo enfoque de ***"Flex Container"*** que es el elemento padre que tendrá en su interior cada uno de los ítems flexibles y ***"Flex Item"*** que son cada uno de los hijos que tendrá el contenedor en su interior.

Ejes

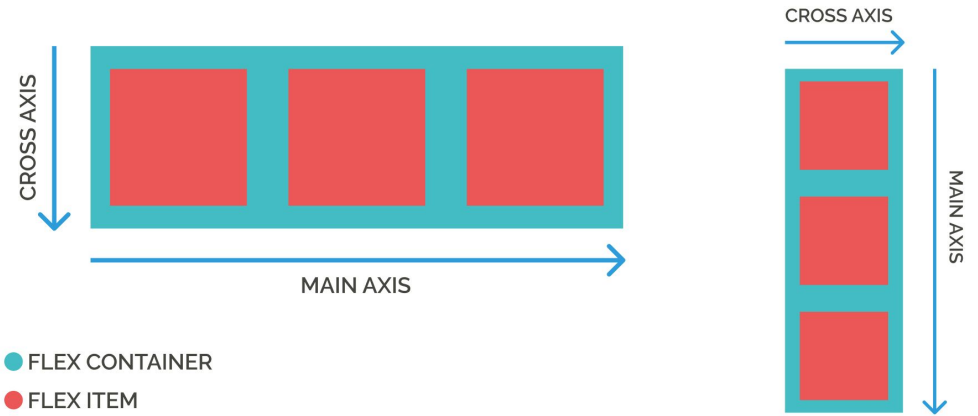
Flexbox es un **conjunto de propiedades CSS** que nos permiten posicionar nuestros elementos HTML con mayor facilidad bajo la premisa de *establecer una dirección vertical u horizontal* para nuestros elementos hijos desde un elemento padre.

Esto nos obliga a **trabajar nuestros contenedores** sobre **un** solo eje al mismo tiempo, ya sea en eje **Y** (vertical) o el eje **X** (horizontal) o como comúnmente se los conoce, **columna** o **fila**.



Main Axis - Cross Axis

Al trabajar de esta manera, nos encontramos con que tenemos un ***Eje Principal*** (main axis) y un ***Eje secundario*** (cross axis).



Propiedades

Flexbox no depende de una única propiedad si no que se encuentra compuesto por un **conjunto** de ellas que nos permiten configurar la disposición de **nuestros elementos** en contenedores flexibles.

En su mayoría, estas propiedades son aplicadas a los elementos padres (flex container) aunque **algunas** se usan en los ***elementos hijos*** (flex item).

Declaración de Contenedor Flexible

Lo primero que debemos hacer **para comenzar a usar flex** es declarar a nuestro *contenedor padre*, que **sus hijos** serán posicionados de forma **flexible**.

Eso lo hacemos con la propiedad `display` que ya conocemos, pero ahora con el valor: `flex;`

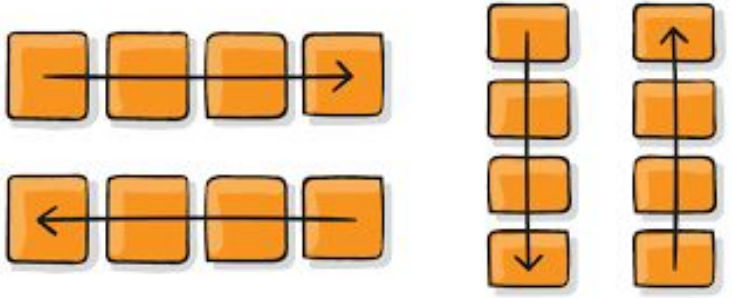
```html
<section class="flex-container">
    <article class="flex-item">
        <h2>Cetaceos</h2>
        <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.
        Quam iure error libero.</p>
    </article>
    <article class="flex-item">
        <h2>Felinos</h2>
        <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.
        Quam iure error libero.</p>
    </article>
</section>
```

```css
.flex-container {
    display: flex;
}
```

Dirección - Multilínea

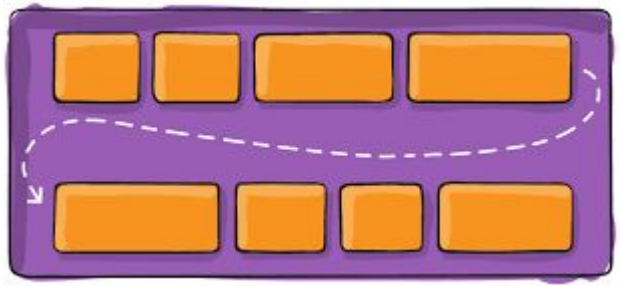
Para manejar la **dirección** de nuestros items vamos a utilizar la propiedad `flex-direction` que puede tomar los valores:

`row* | row-reverse | column |column-reverse`



Si queremos un comportamiento **multilínea** entonces nuestra propiedad es `flex-wrap` que puede tomar los valores:

`wrap | wrap-reverse | nowrap*`



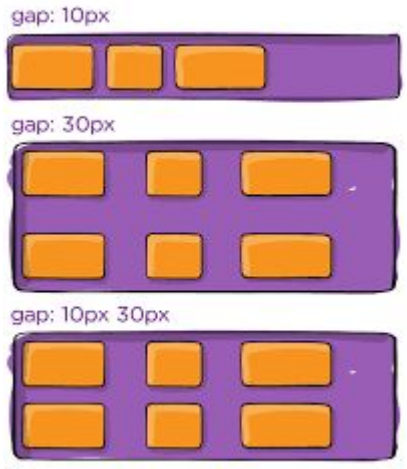
`*` valores por defecto

`flex-direction` y `flex-wrap` se pueden declarar juntos mediante el shorthand `flex-flow`.

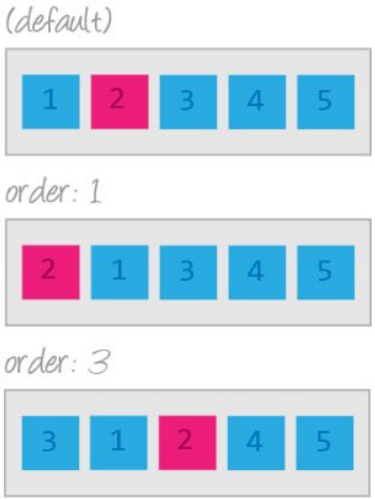
```
.flex-container {
    display: flex;
    flex-flow: row nowrap;
}
```

Espaciado - Orden

Para manejar el espaciado entre nuestros items vamos a utilizar la propiedad `gap` que toma cualquier unidad de medida conocida.



Podemos cambiar el orden de nuestros hijos con `order`, esta propiedad se aplica sobre el item hijo con el valor de posición que debe tomar.




```
.flex-item {
    order: 3;
}
```

Flex Grow

Nos ayuda a **distribuir** el espacio sobrante del contenedor padre entre los items hijos. Se declara en estos últimos a través de la propiedad `flex-grow` y su valor es un **número entero** que indica cuánto le corresponde de ese sobrante.

```
.flex-item-1 {
    flex-grow: 2;
}

.flex-item-1 {
    flex-grow: 4;
}

.flex-item-3 {
    flex-grow: 2;
}
```

`200px*3 = 600px | 400px/8 = 50xp`

Flex Shrink

`flex-shrink` funciona igual que `flex-grow` solo que en este caso va a **determinar** qué sucede cuando *al elemento padre* **le falta** espacio para ubicar a sus hijos, es decir, de quien toma el espacio faltante.

El valor y como se calcula es igual que el anterior.

```
.flex-item-1 {
    flex-shrink: 1;
}

.flex-item-1 {
    flex-shrink: 3;
}

.flex-item-3 {
    flex-shrink: 1;
}
```

Flex Basis

`flex-basis` nos permite determinar el **ancho por defecto** de un item flexible como si de la propiedad **width** se tratara.

```
.flex-item {
    flex-basis: 250px;
}
```

Flex

La propiedad `flex` es el shorthand property de las 3 anteriores.

```
.flex-item {
    /* grow | shrink | basis */
    flex:1 1 250px;
}
```

Alineaciones

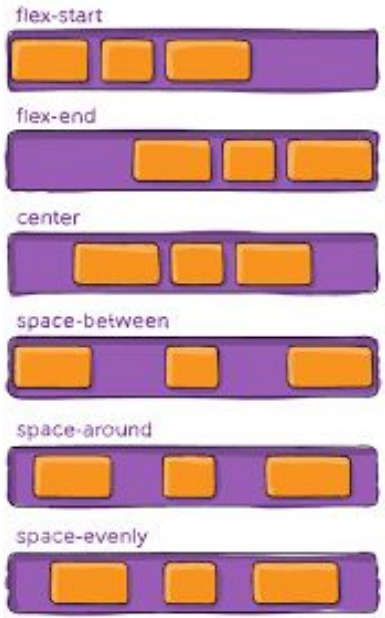
Son la magia de flexbox, con ellos podemos **distribuir** *uniforme y automáticamente* nuestros **elementos hijos** dentro de un contenedor flexible.

Existen múltiples opciones y según cual usemos **actúan** **sobre el main axis** o el **cross axis** dependiendo el flex direction que hayamos aplicado.

justify-content

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

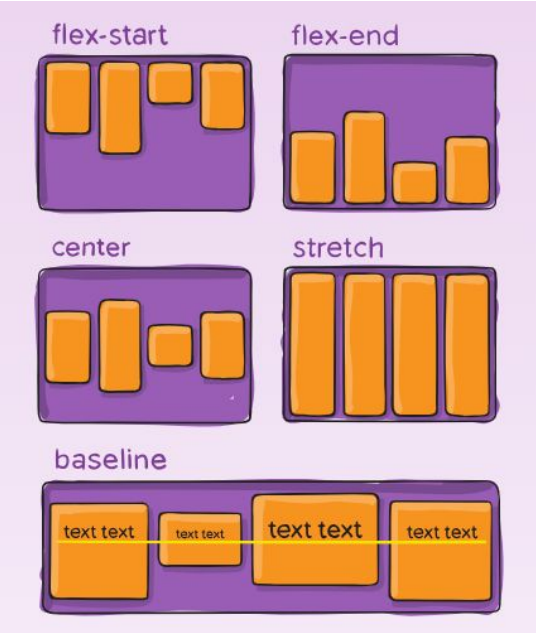
Se utiliza para alinear los ítems del **eje principal** o **main axis**.



align-items

`flex-start | flex-end | center | stretch | baseline`

Se utiliza para alinear los ítems del **eje secundario** o **cross axis**.

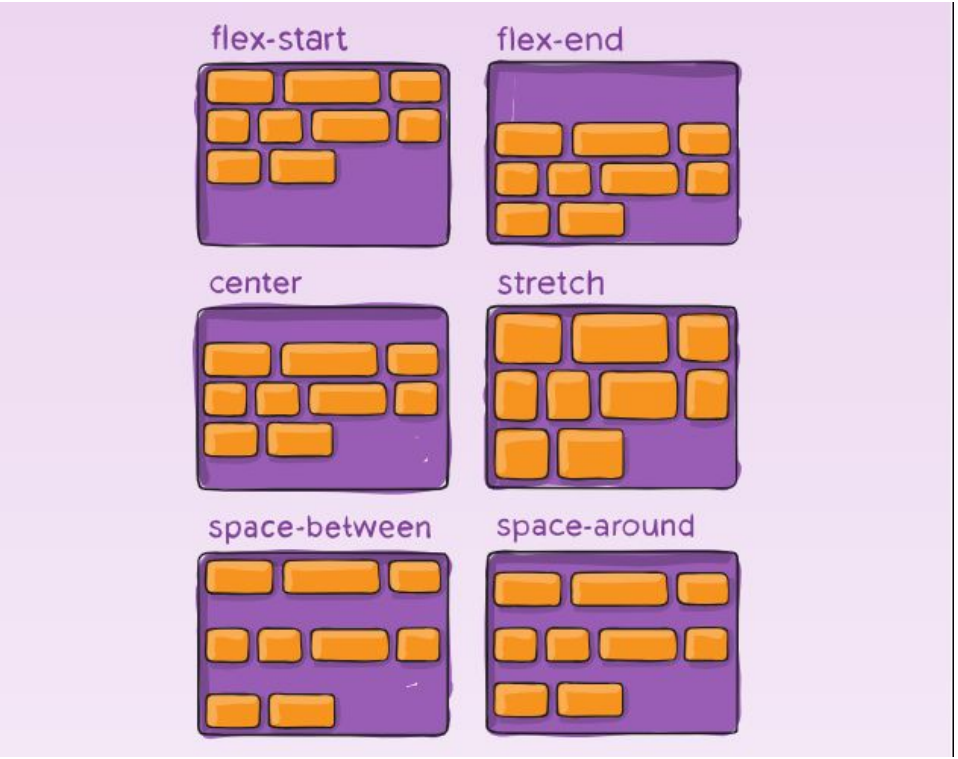


align-items: flex-start;

align-content

`flex-start | flex-end | center | space-between | space-around | space-evenly | stretch`

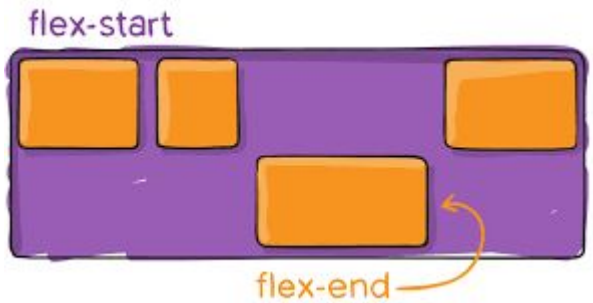
Servirá para **alinear** cada una de las **líneas** del contenedor multilínea.



align-self

`auto | flex-start | flex-end | center | stretch | baseline`

Nos permite asignar un **align-item** diferente a un solo elemento hijo.

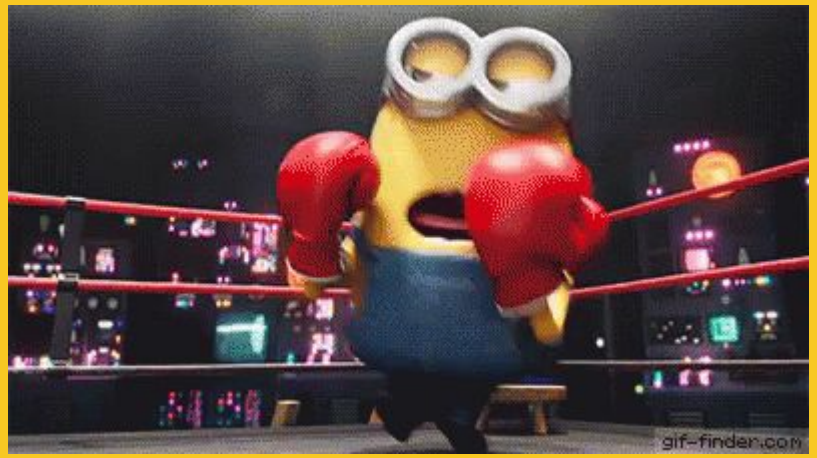



```
.flex-container {
    align-items: flex-start;
}

.flex-item {
    align-self: flex-end;
}
```

CSS

Estructura en nuestros estilos

Segundo round de posicionamiento con...



GRID

Sistema de grillas, para más placer

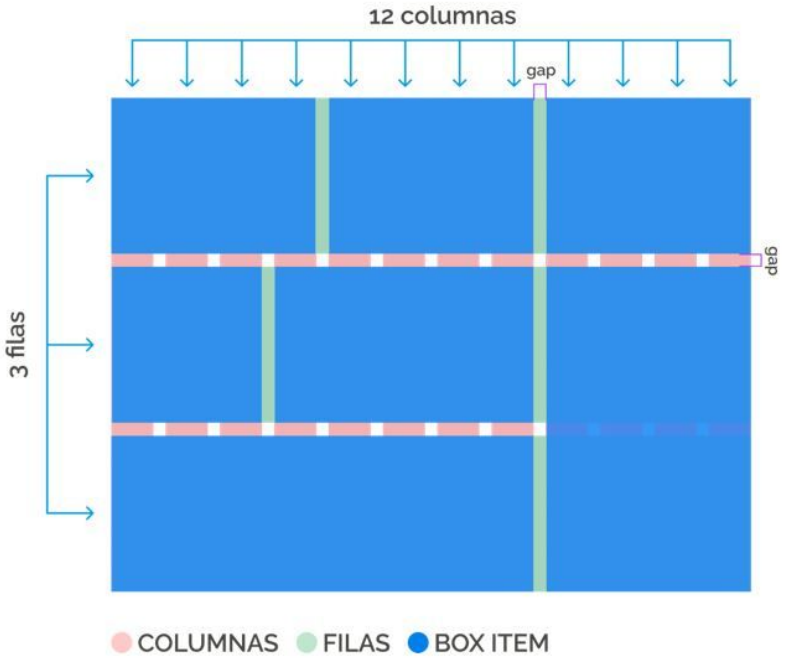
¿Qué es GRID?

A diferencia de **Flexbox**, donde podemos posicionar nuestros elementos en una sola dimensión, con **GRID** tenemos la posibilidad de hacerlo de forma horizontal y vertical al mismo tiempo, es decir, en 2 dimensiones.

GRID toma todas las ventajas de Flexbox para volcarlo en un ***sistema más potente*** que nos permite crear grillas o cuadrículas de una manera muy sencilla.

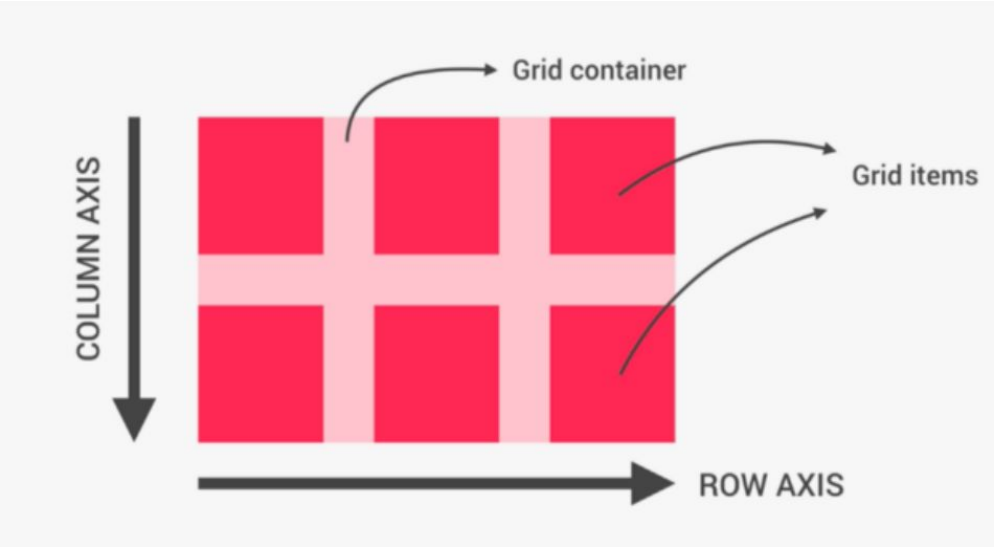
Poder en 2 dimensiones

Como mencionamos anteriormente **GRID** nos permite trabajar en filas y columnas simultáneamente, solo es cuestión de crear una grilla o cuadrícula que se **ajuste a nuestras necesidades** y **posicionar** los elementos ***hijos*** dentro de ella.



Ejes

En el caso de **GRID** se toma como **main axis** al eje **X** (row axis) y como **cross axis** al eje **Y** (column axis), lo que tendrá vital importancia al momento de utilizar las propiedades de **alineación** que veremos en un momento.



Propiedades

Al igual que **Flexbox**, **Grid Layout** está compuesto por un conjunto de propiedades aplicadas a un **elemento padre** que *definirá* una grilla modelo o template con la forma buscada. Sobre este template es que luego posicionaremos los elementos hijos.

Cabe destacar que al ser una **plantilla** esta **no forma parte de la estructura**, sino que genera un lienzo cuadriculado sobre el cual **distribuiremos** nuestros elementos.

Declaración de una Grilla con GRID

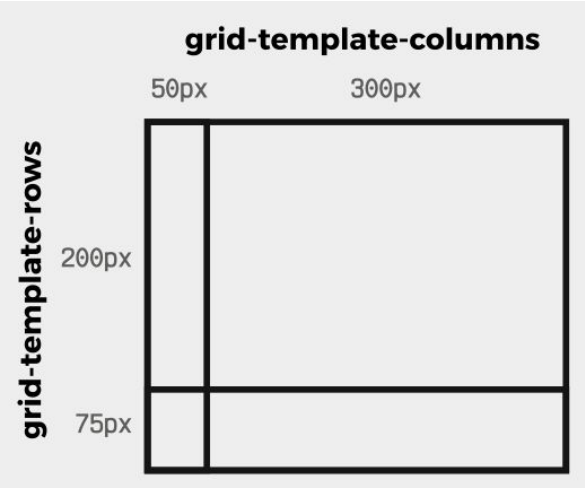
Para comenzar a usar **GRID** es necesario declarar a nuestro *contenedor padre* con un display en forma de grilla. Nuevamente usamos la propiedad `display`, pero ahora con el valor: `grid;`

```html
<section id="gallery">
    <picture class="gallery__img-1">
        <img src="./img/img-0.jpg" alt="">
    </picture>
    <picture class="gallery__img-2">
        <img src="./img/img-1.jpg" alt="">
    </picture>
    <picture class="gallery__img-3">
        <img src="./img/img-2.jpg" alt="">
    </picture>
    <picture class="gallery__img-4">
        <img src="./img/img-3.jpg" alt="">
    </picture>
    <picture class="gallery__im-5">
        <img src="./img/img-4.jpg" alt="">
    </picture>
    <picture class="gallery__img-6">
        <img src="./img/img-5.jpg" alt="">
    </picture>
</section>
```

```css
#gallery {
  display: grid;
}
```

Filas y Columnas

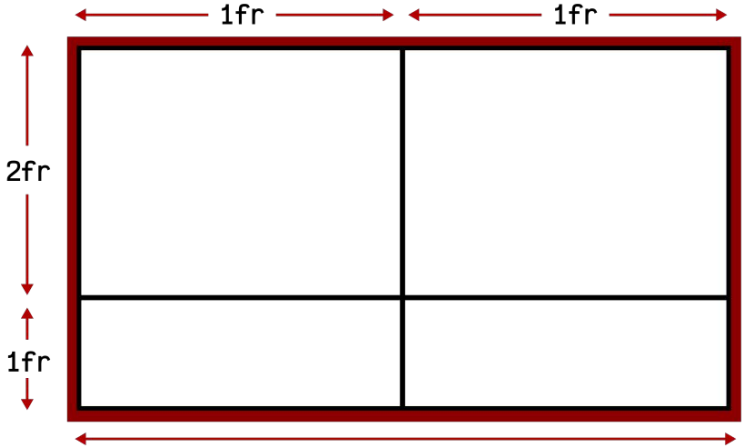
Una vez que **declaramos** nuestro contenedor como una **grilla**, debemos elegir la cantidad de ***filas y columnas*** que vamos a necesitar en nuestro template. Para eso vamos a utilizar las propiedades `grid-template-columns` y `grid-template-rows` de la siguiente manera:




```css
.grid-container {
  display: grid;
  grid-template-columns: 50px 300px;
  grid-template-rows: 200px 75px;
}
```

Fractions

Si bien `grid-template-columns` y `grid-template-rows` aceptan las unidades de medida tradicionales, en GRID existe la unidad fr (fraction) que divide el espacio disponible entre la cantidad de fr declarados y los reparte proporcionalmente.




```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 2fr 1fr;
}
```

En este ejemplo tanto el width cómo el height se dividen por la cantidad de fracciones declaradas y lo distribuye en consecuencia.

Cabe destacar que fr se puede combinar con cualquier otra unidad de medida en la misma declaración.

Repeat

En los casos donde necesitamos crear una **plantilla** con muchas filas o columnas del **mismo tamaño**, podemos utilizar el valor `repeat(cantidad, tamaño);`.

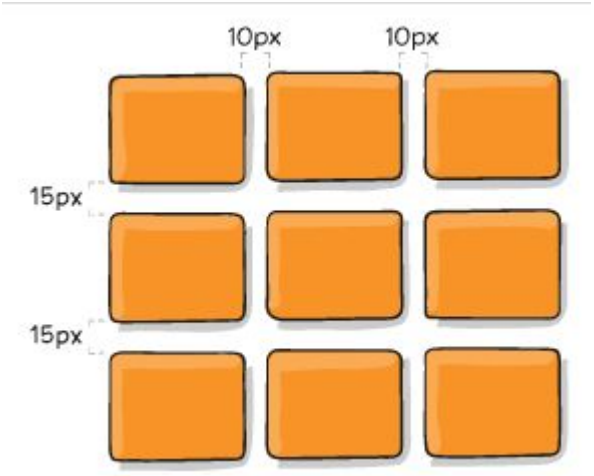
```css
.grid-container {
  display: grid;
  grid-template-columns: repeat(12, 1fr);
  grid-template-rows: repeat(3, 250px);
}
```

cantidad: es el **número** de columnas o filas que necesitamos.

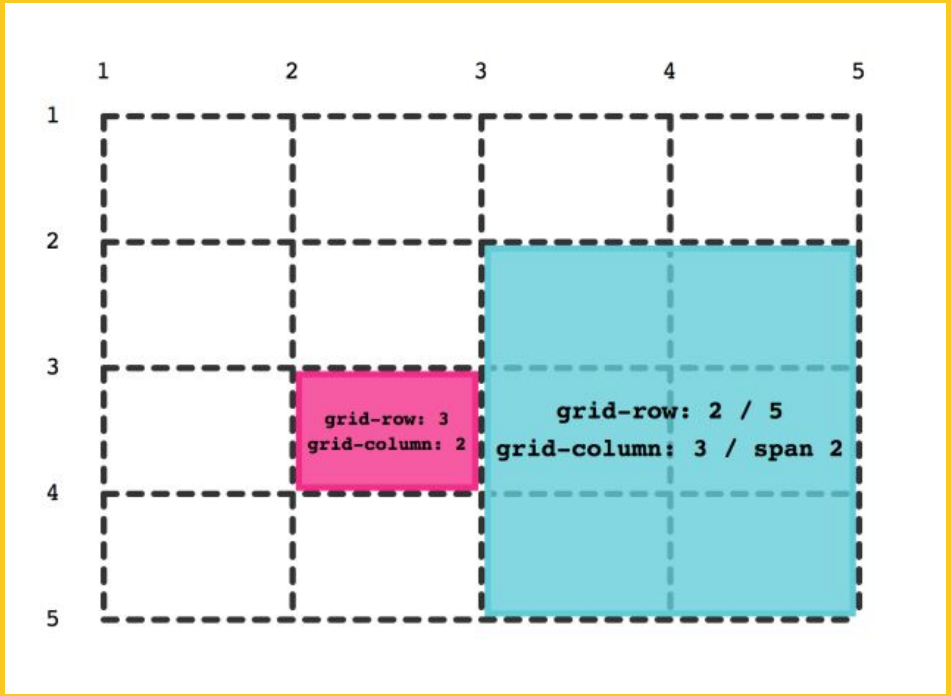
tamaño: el **ancho** o **alto** que ***deberán tomar***.

GAP

Esta propiedad **mágica**, funciona igual que en flexbox y se utiliza para **definir** el **espaciado entre los elementos de una grilla**. En este caso **podemos** hacerlo de forma separada mediante las propiedades `column-gap` y `row-gap` o en conjunto a través de la propiedad `gap`.



GRID POR VALORES



grid-column

Shorthand de `grid-column-start` y `grid-column-end` nos va a permitir indicarle a un grid-item en qué columna debe empezar y en cuál finalizar `start / end;`.

```css
.grid-container {
    display: grid;
    width: 800px;
    grid-template-columns:repeat(12, 1fr);
    grid-template-rows: 60px 200px 60px;
}

.grid-container h1,
.grid-container a {
    grid-column: 1 / 13;
}

.grid-item-1 {
    grid-column: 1 / 5;
}

.grid-item-2 {
    grid-column: 5 / 9;
}

.grid-item-3 {
    grid-column: 9 / 13;
}
```

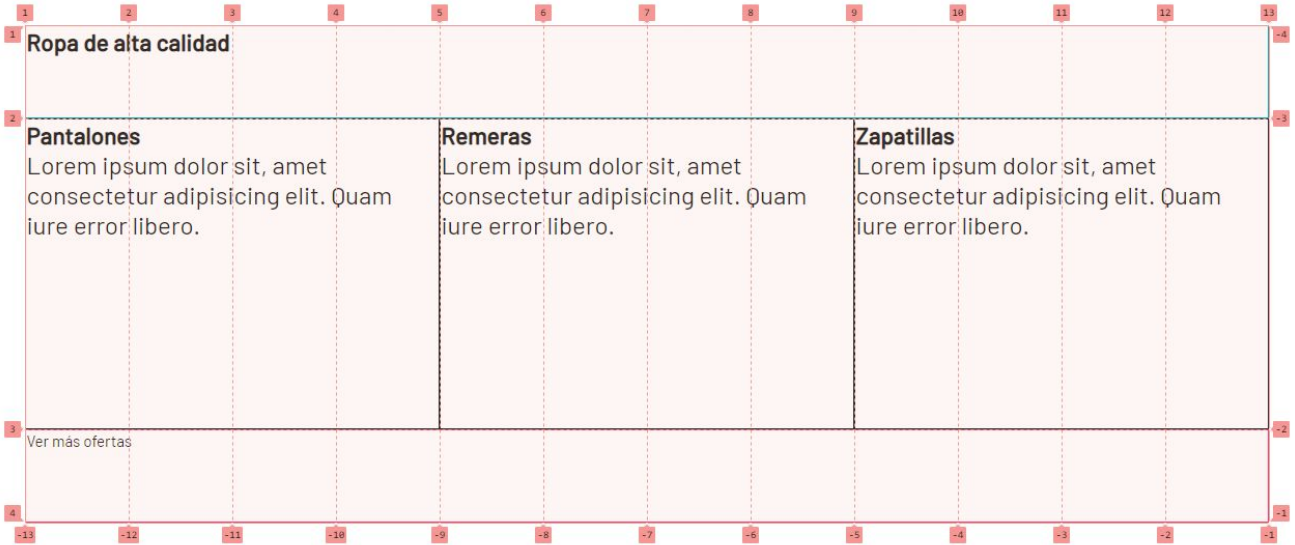
```html
<section class="grid-container">
    <h1>Ropa de alta calidad</h1>
    <article class="grid-item-1">
        <h2>Pantalones</h2>
        <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.
        Quam iure error libero.</p>
    </article>
    <article class="grid-item-2">
        <h2>Remeras</h2>
        <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.
        Quam iure error libero.</p>
    </article>
    <article class="grid-item-3">
        <h2>Zapatillas</h2>
        <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.
        Quam iure error libero.</p>
    </article>
    <a href="./ofertas.html">Ver más ofertas</a>
</section>
```

grid-row

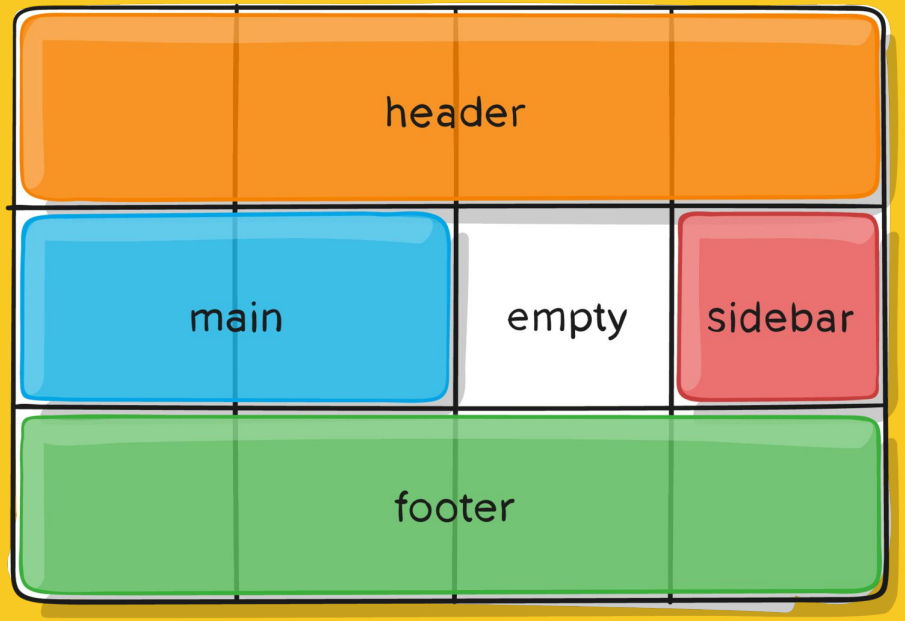
Igual que el anterior, solo que en este caso nos va a permitir indicarle a un grid-item en qué **fila** debe empezar y en cuál finalizar.

```css
.grid-container h1 {
    grid-row: 1;
}
.grid-container a {
    grid-row: 3;
}

.grid-item-1 {
    grid-column: 1 / 5;
    grid-row: 2;
}

.grid-item-2 {
    grid-column: 5 / 9;
    grid-row: 2;
}

.grid-item-3 {
    grid-column: 9 / 13;
    grid-row: 2;
}
```

Así nos queda nuestros elementos distribuidos en la grilla creada:

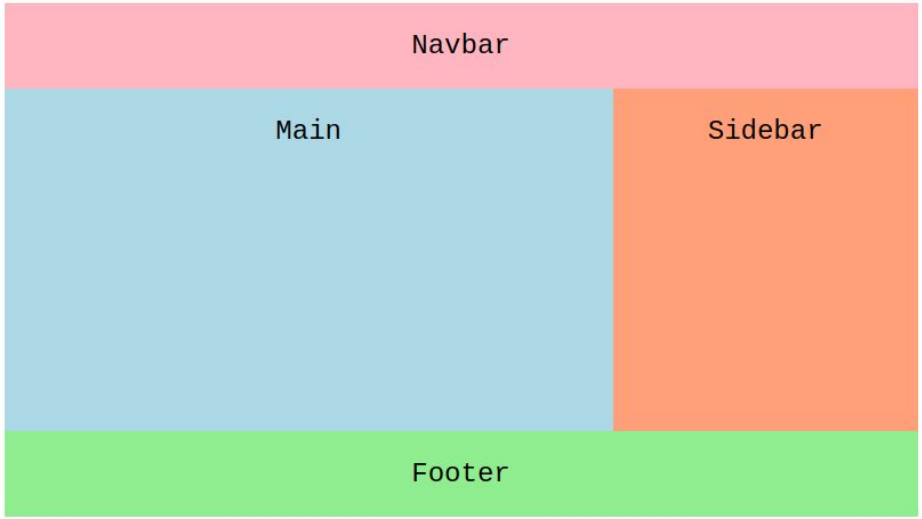


GRID AREAS



Areas

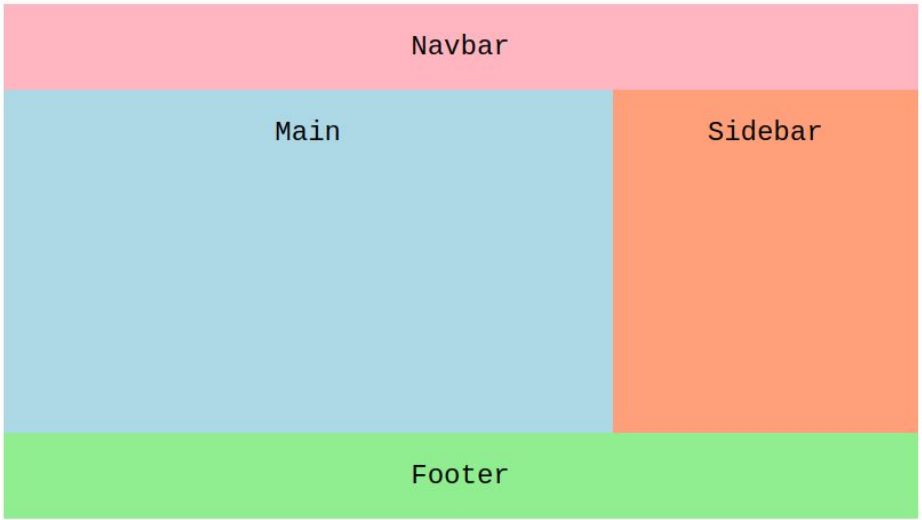
`grid-template-areas:` es una propiedad de GRID que nos permite “dibujar” nuestra plantilla CSS para luego posicionar a los elementos hijos en esa representación.




```css
.grid-container {
  display: grid;
  grid-template-columns: 2fr 1fr;
  grid-template-rows: 120px 1fr 120px;
  grid-template-areas:
  "navbar navbar"
  "main sidebar"
  "footer footer";
}
```

Areas

Estas areas establecidas desde el .grid-container son las que aplicaremos luego a nuestros .grid-items para especificar qué espacio ocupará cada uno.




```css
header {
  grid-area: navbar;
}

main {
  grid-area: main;
}

aside {
  grid-area: sidebar;
}

footer {
  grid-area: footer;
}
```

Alineaciones

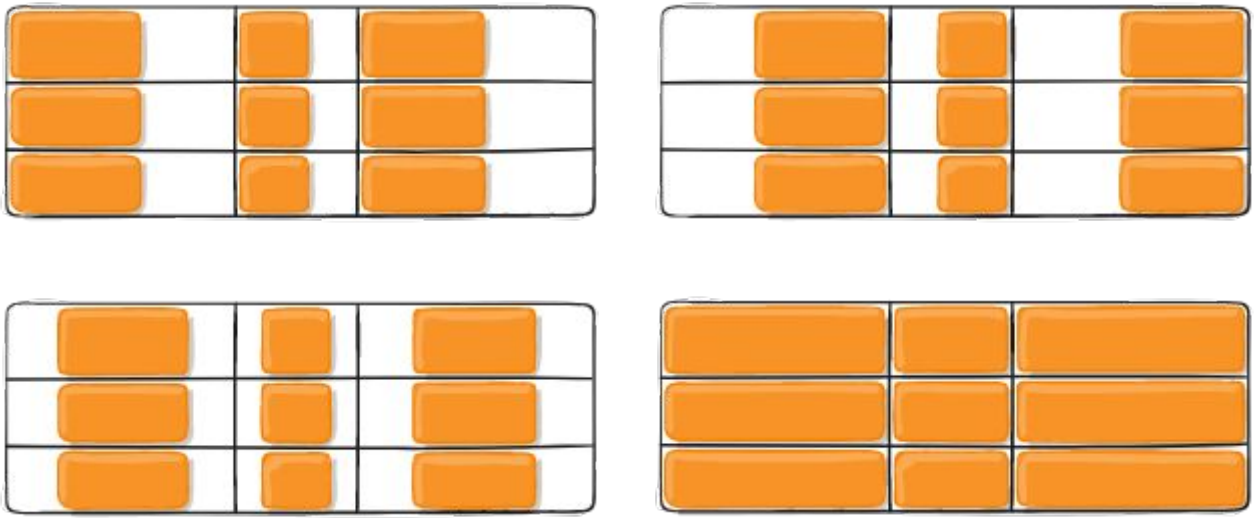
Al tomar las *mismas ventajas de Flexbox* para crear un **sistema** más potente, veremos que en GRID las variantes para alinear nuestros elementos hijos son muy **similares** a las que ya conocemos.

ITEMS

justify-items

```
start | end | center | stretch;
```

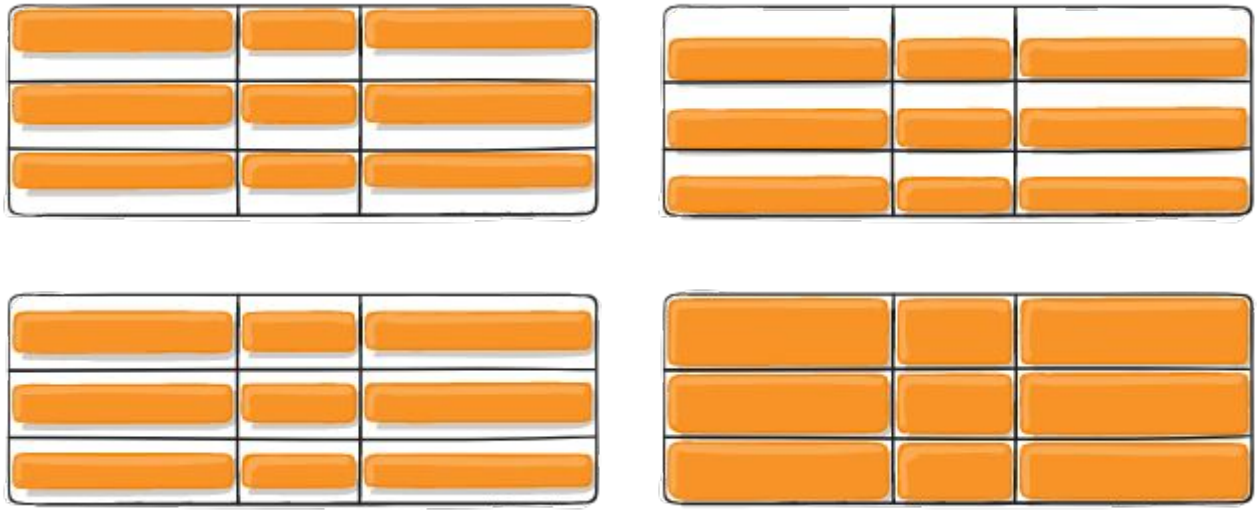
Se utiliza para alinear los ítems en el **eje horizontal** o **row axis**.



align-items

```
start | end | center | stretch | baseline;
```

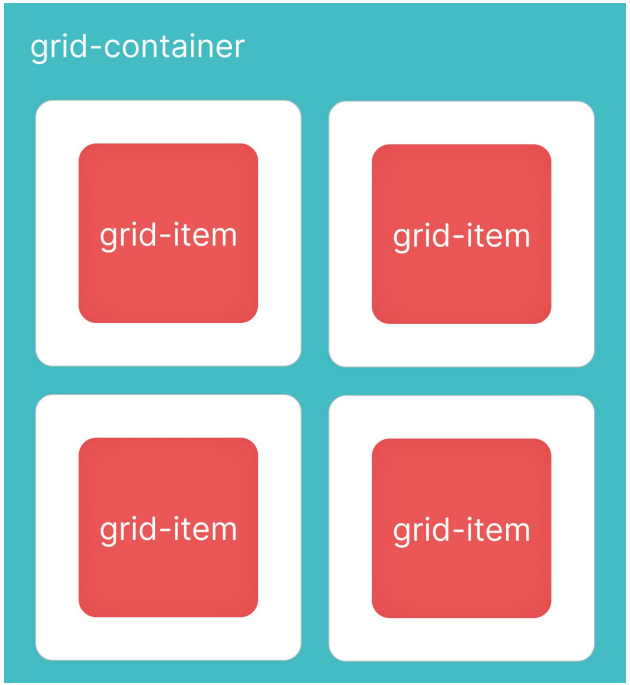
Se utiliza para alinear los ítems en el **eje vertical** o **column axis**.



place-items

`start | end | center | stretch | baseline;`

Nos permite definir `justify-items` y `align-items` con la misma propiedad cuando poseen el mismo valor.



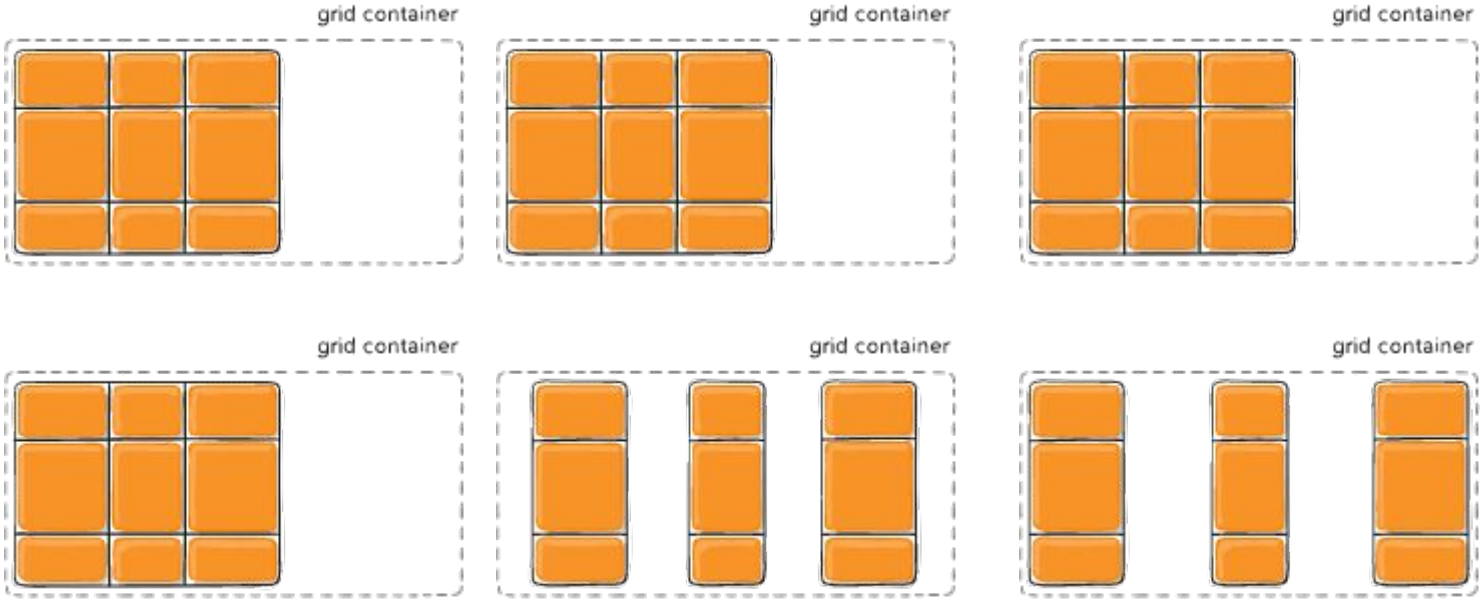

```css
.grid-container {
    display: grid;
    place-items: center;
}
```

CONTENT

justify-content

```
start | end | center | space-between | space-around | space-evenly | stretch;
```

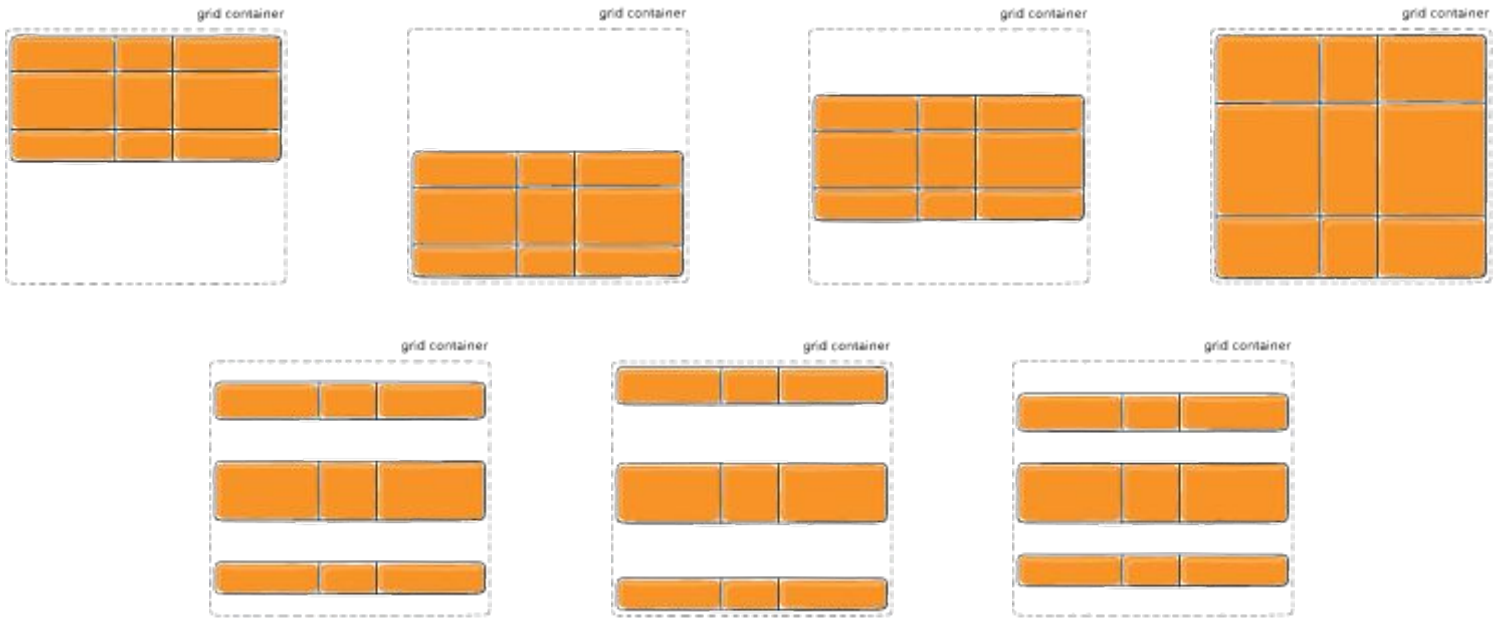
Servirá para **alinear** cada columna (contenido) dentro del contenedor.



align-content

```
start | end | center | space-between | space-around | space-evenly | stretch;
```

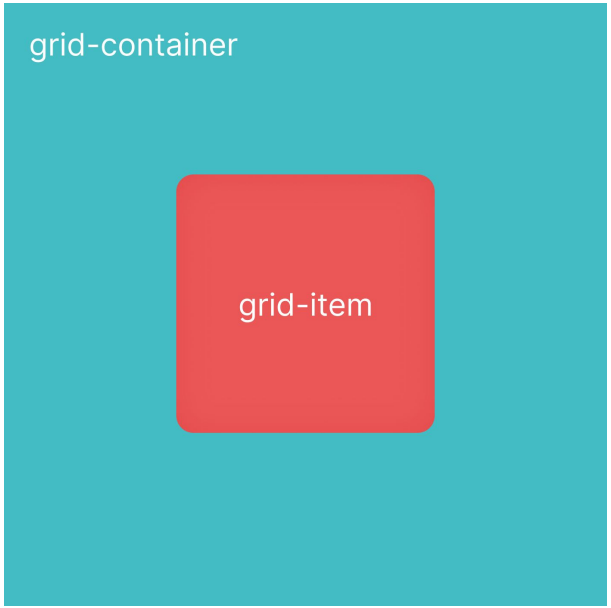
Servirá para **alinear** cada fila (contenido) dentro del contenedor.



place-content

`start | end | center | stretch | baseline;`

Nos permite definir `justify-content` y `align-content` con la misma propiedad cuando poseen el mismo valor.



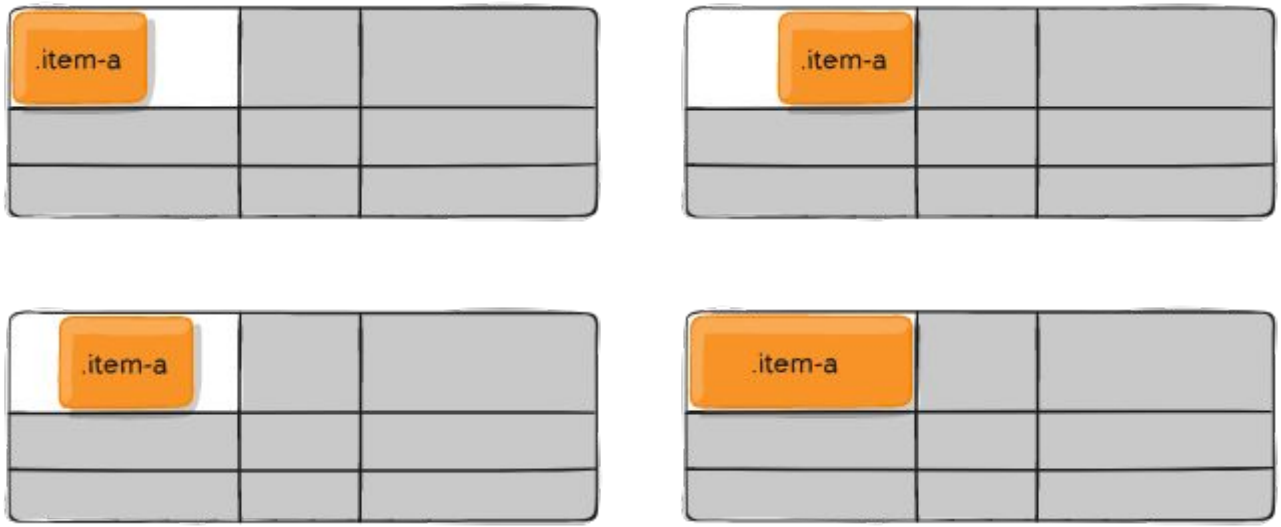

```css
.grid-container {
    display: grid;
    place-content: center;
}
```

SELF

justify-self

```
start | end | center | stretch;
```

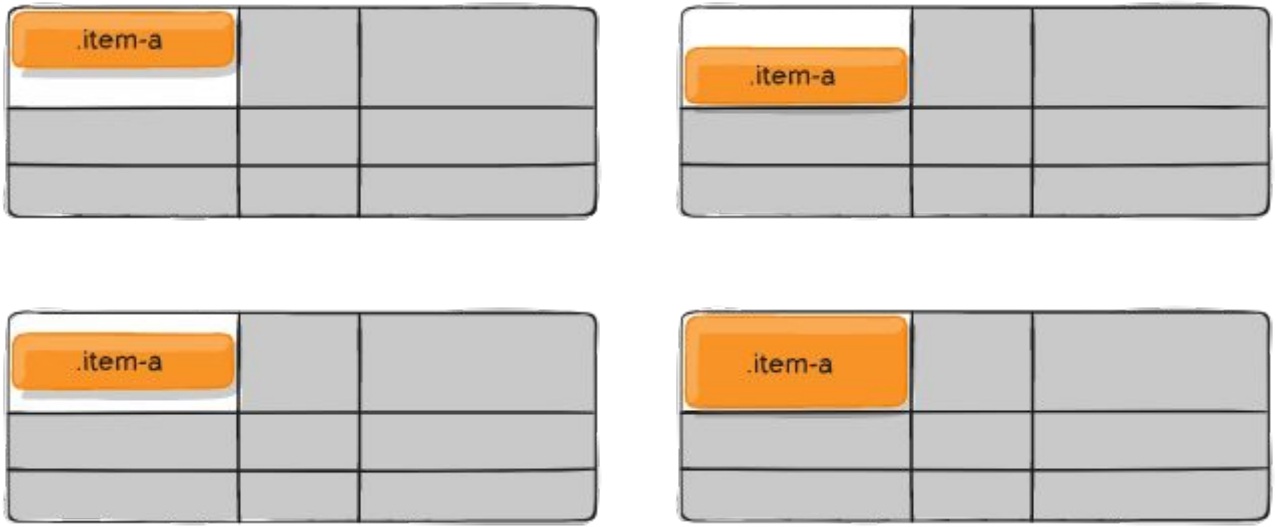
Servirá para **alinear horizontalmente** cada item hijo de forma individual.



align-self

```
start | end | center | stretch;
```

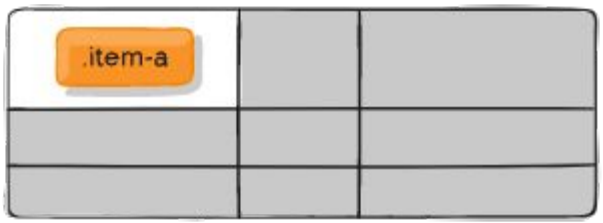
Servirá para **alinear verticalmente** cada item hijo de forma individual.



place-self

`start | end | center | stretch;`

Servirá para **alinear** ambos ejes con la misma propiedad.




```
.grid-container {
    display: grid;
    place-self: center;
}
```

Fuentes

Las imágenes de esta presentación fueron tomadas de:
https://css-tricks.com/snippets/css/complete-guide-grid/

BOOTSTRAP

Framework al rescate

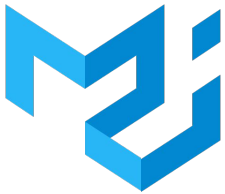
¿Qué es un Framework?

Es un conjunto de *herramientas, librerías, convenciones y buenas prácticas* que pretenden **encapsular** las tareas repetitivas en módulos genéricos fácilmente reutilizables.

Frameworks CSS

Un **framework CSS** es un conjunto de estilos que permiten olvidarse del código repetitivo para centrarse en los *elementos únicos de cada diseño* en los que puede aportar valor.

Si bien existen muchos, los más conocidos son:



¿Qué es Bootstrap?

Es un **Framework CSS** que se añade en los proyectos para tener una serie de estilos ya preparados para utilizar. Incluye estilos para los elementos más comunes de una página web, como por ejemplo, **botones**, tarjetas, navbars, etc.

Además posee clases para crear columnas fácilmente, cuyo principal objetivo es permitir la ***construcción de sitios web responsive*** para dispositivos móviles.

Instalación

Instalando Bootstrap

Originalmente existen *más de una forma* de instalar bootstrap, en este caso conoceremos **dos** de ellas:

- Instalación mediante CDN (content delivery network).
- Descarga de archivos compilados.

Instalación mediante CDN

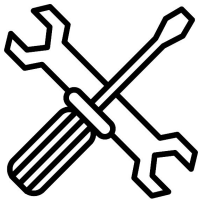
Es la más sencilla de todas, consta de agregar una etiqueta **<link />** para los **estilos** y una etiqueta **<script></script>** para las **funcionalidades**.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-iYQeCzEYFbKjA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT" crossorigin="anonymous">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/js/bootstrap.bundle.min.js"
integrity="sha384-u1OknCvxWvY5kfmNBILK2hRnQC3Pr17a+RTT6rIHI7NnikvbZlHgTPOOmMi466C8"
crossorigin="anonymous"></script>
  </body>
</html>
```

https://getbootstrap.com/docs/5.2/getting-started/introduction/

No todo lo que brilla es oro

Si bien el uso de CDN's es bastante práctico, puede que no sea lo más recomendable para proyectos grandes o muy importantes, ya que no podemos saber si esos enlaces estarán siempre activos y funcionando correctamente.

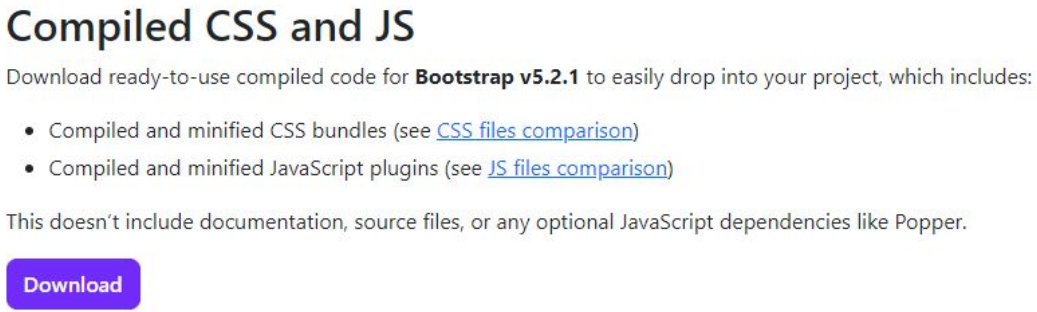


HTTP Error 503
The service is unavailable

Descarga de archivos compilados.

Consiste en **descargar** el código fuente de bootstrap desde su sitio oficial y linkearlo a **nuestro sitio** como una hoja de estilos CSS y scripts Javascript más de nuestro proyecto.

Nos dirigimos a: https://getbootstrap.com/docs/5.2/getting-started/download/ y bajamos el comprimido.



Descarga de archivos compilados.

El paquete de descarga trae múltiples archivos según lo que necesitemos o deseemos usar, sin embargo en este caso ***usaremos*** el aquellos con todo el código de bootstrap minificado para que pese menos.

Ahora que tenemos Bootstrap instalado, empecemos...

Clase Container

Es una *clase predefinida* por bootstrap que nos ayudará a manejar el ancho de nuestras cajas.

Clase .container

Se le aplica principalmente a elementos contenedores y según la clase **.container** que seleccionemos vamos a obtener distintos resultados.

Lo que ocurre es que a ese elemento **se** le aplica un ancho y un padding determinado y además se coloca en el centro de la página web.

Bootstrap viene con tres contenedores diferentes:

.container: que establece un max-width en todos los breakpoints responsive.

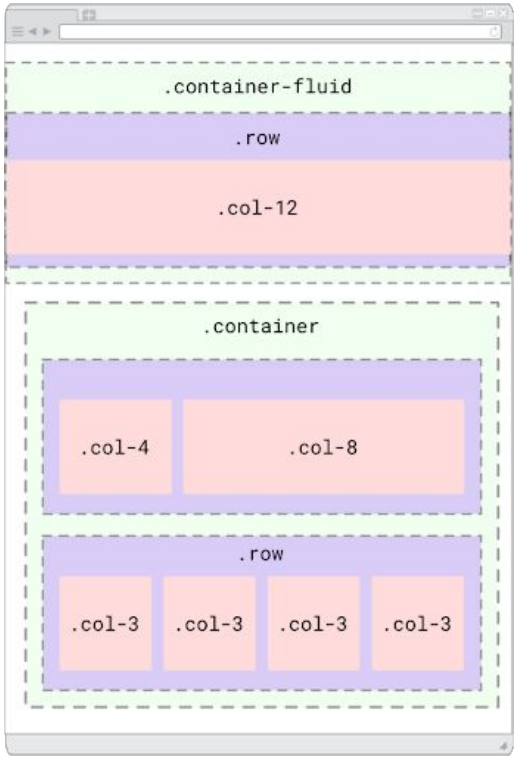
.container-fluid: que establece un width: 100% en todos los breakpoints.

.container-{breakpoint}: que tiene un width: 100% hasta el breakpoint especificado.

Clase .container

“container-fluid” ocupa el 100% del tamaño disponible de la pantalla.

“container” establece un max-width, genera un margen tanto a la izquierda como a la derecha y también se centra.



.container-{breakpoint}

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	X-Large ≥1200px	XX-Large ≥1400px
.container	100%	540px	720px	960px	1140px	1320px
.container-sm	100%	540px	720px	960px	1140px	1320px
.container-md	100%	100%	720px	960px	1140px	1320px
.container-lg	100%	100%	100%	960px	1140px	1320px
.container-xl	100%	100%	100%	100%	1140px	1320px
.container-xxl	100%	100%	100%	100%	100%	1320px
.container-fluid	100%	100%	100%	100%	100%	100%

Grid System

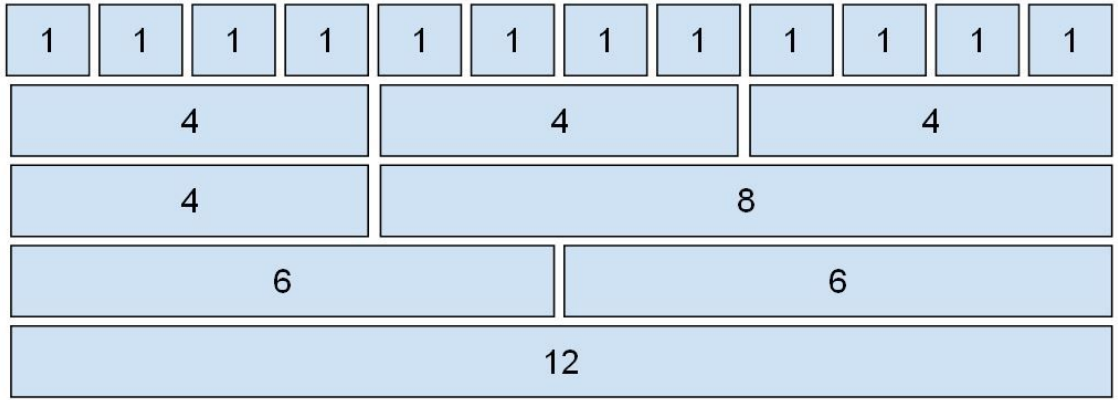
Bootstrap incluye su propia solución a los diseños responsive. Esta **característica** es una de las que hizo que **sea** uno de los Frameworks más famosos y utilizados de CSS.

Se basa en un **sistema de columnas** que se ajustan dinámicamente frente a los distintos tamaños de pantalla.

Grid System

El sistema de grillas de Bootstrap **divide** nuestras cajas en 12 columnas que se utilizan **para repartir** los elementos hijos de cada contenedor.

Para eso se utilizan un conjunto de clases que se **aplican** tanto a padres como a hijos.

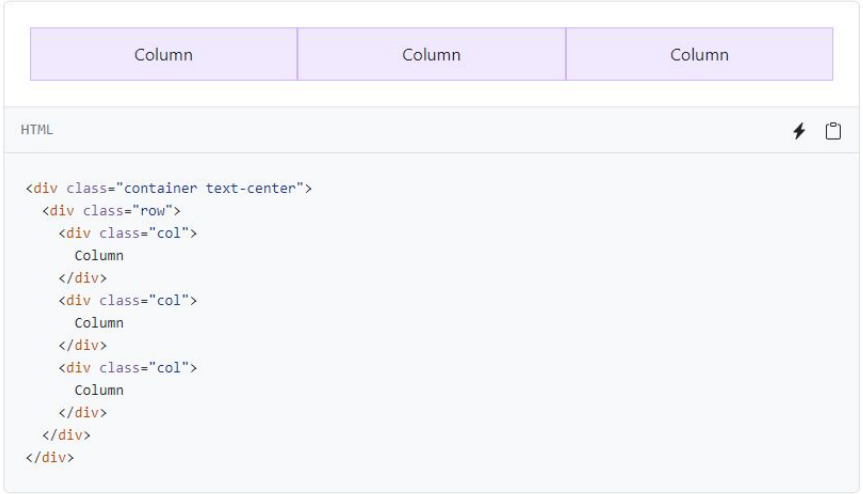


Grid System

Las clases a utilizar son **.row** y **.col** solo que esta última se puede utilizar de diversas maneras.

.row

Define un contenedor bajo el esquema **Grid System**, es la clase que habilita las 12 columnas para distribuir sus elementos hijos.

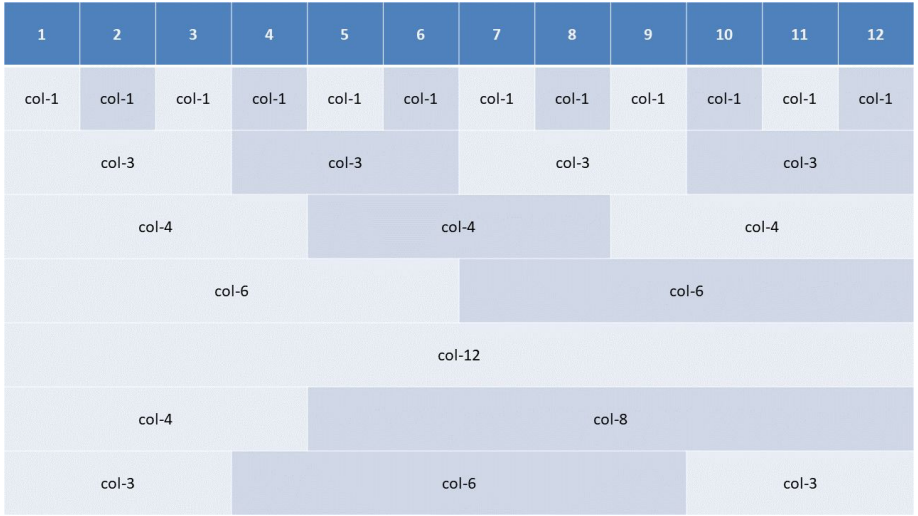


Grid System

.col

Esta clase tiene diversas variantes y puede utilizarse más de una vez en un mismo elemento.

Si la usamos **sola**, entonces le indica al elemento que tiene que ubicarse en fila y **repartir las columnas uniformemente** junto con los demás elementos de clase **.col**



Por otra parte, podemos agregarle un **breakpoint** de pantalla y la cantidad de columnas. Esto nos permite decidir cuántas columnas queremos que ocupen nuestras cajas dependiendo el tamaño de la pantalla, generando *diseños adaptativos y muy dinámicos*.

Breakpoints

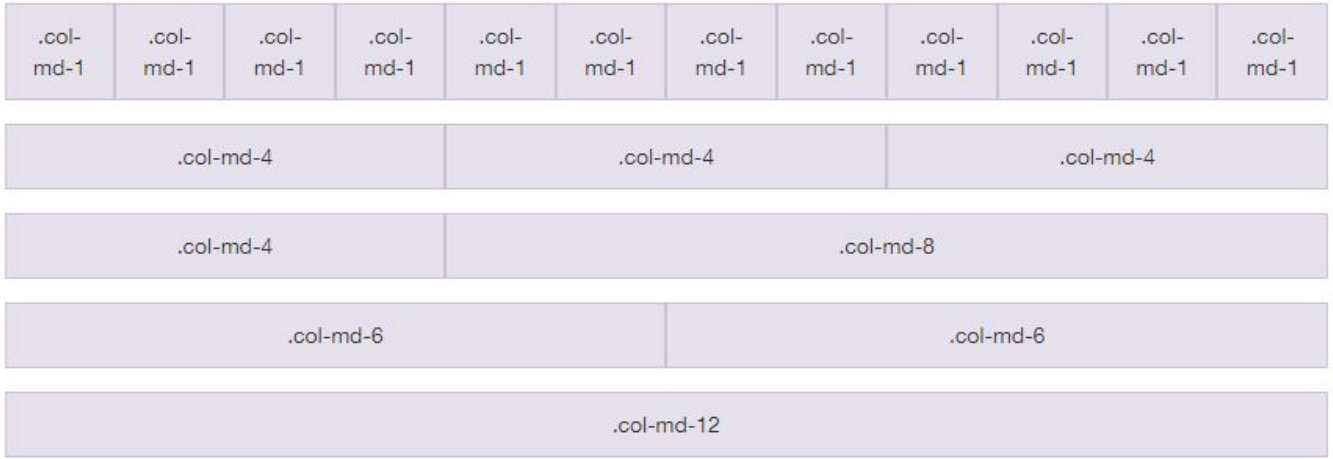
Son los puntos de quiebre (media queries) usadas por bootstrap para distribuir sus diseños. Estos **se aplican en diversas** clases pero como mencionamos recién, el caso más conocido es para el **Grid System**.

Breakpoint	Class infix	Dimensions
Extra small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Veamos cómo aprovechamos los breakpoints para crear distintos layouts con las mismas cajas.

Grid System

Primero elegimos el breakpoint y la cantidad de columnas que deseamos que ocupe nuestro componente. Luego le **aplicamos** la clase **.col-{bkp}-{cols}**.



Así se obtiene un diseño responsive con tan solo un par de clases

Components

Una de las **grandes ventajas** de este framework es que pone a disposición una lista de más de 20 componentes prediseñados con comportamientos propios como **modales interactivos**, tooltips, **menús de navegación**, botones desplegables, entre otros.

Vamos a conocerlos...

Buttons

Con una simple línea HTML obtenemos **botones** de formulario muy estéticos y en **distintos colores** para cada ocasión:

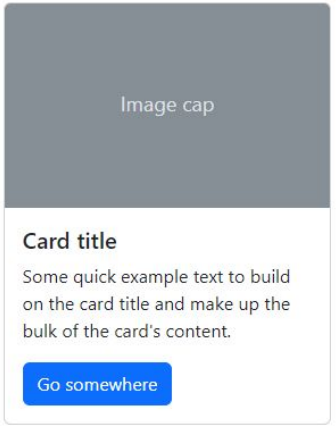
```html
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>

<button type="button" class="btn btn-link">Link</button>
```

Cards

Un **set de tarjetas** prediseñadas multiuso. Son útiles para productos, entradas de blog, mensajes y mucho más.

Para usarlas, simplemente buscamos la que se ajuste a nuestra necesidad, ***copiamos el código HTML*** suministrado por Bootstrap y **lo pegamos en nuestro proyecto**.

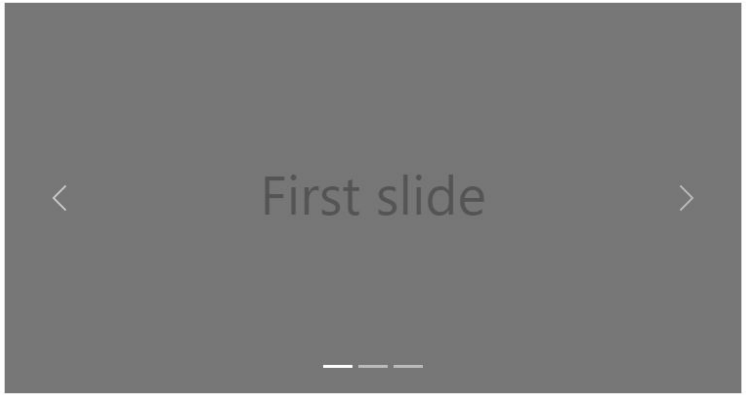
```
<div class="card" style="width: 18rem;">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build on the card title and make up the bul
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```



Carousel

Uno de los componentes más utilizados de Bootstrap.

Simplemente **copiamos el código** y tenemos a disposición un carousel de imágenes para nuestro proyecto, además cuenta con diferentes variantes para cada caso.

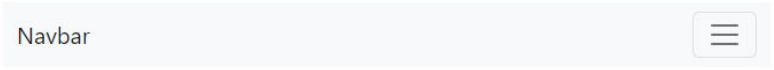



```
<div id="carouselExampleIndicators" class="carousel slide" data-bs-ride="true">
  <div class="carousel-indicators">
    <button type="button" data-bs-target="#carouselExampleIndicators" data-bs-slide-to="0" clas
    <button type="button" data-bs-target="#carouselExampleIndicators" data-bs-slide-to="1" aria
    <button type="button" data-bs-target="#carouselExampleIndicators" data-bs-slide-to="2" aria
  </div>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <button class="carousel-control-prev" type="button" data-bs-target="#carouselExampleIndicator
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-target="#carouselExampleIndicator
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
```

Navbar

Este componente es ideal para tener una barra de navegación responsive.

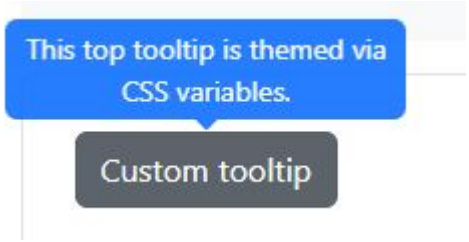
La ventaja es que para tamaños de pantalla pequeños trae integrado un **menú hamburguesa** que esconde nuestros enlaces y los **despliega al hacer click** en el ícono.




```
<nav class="navbar navbar-expand-lg bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#nav
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
      <div class="navbar-nav">
        <a class="nav-link active" aria-current="page" href="#">Home</a>
        <a class="nav-link" href="#">Features</a>
        <a class="nav-link" href="#">Pricing</a>
        <a class="nav-link disabled">Disabled</a>
      </div>
    </div>
  </div>
</nav>
```

Tooltips

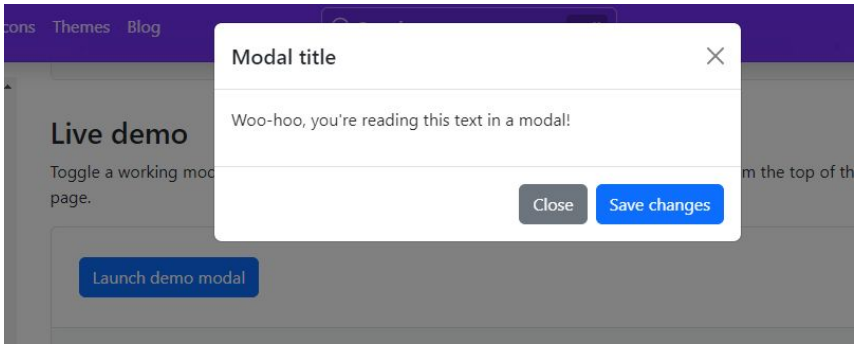
Nos permite tener mensajes que se despliegan al **pasar por encima de un elemento** en particular para dar más información.




```
<button type="button" class="btn btn-secondary"
        data-bs-toggle="tooltip" data-bs-placement="top"
        data-bs-custom-class="custom-tooltip"
        data-bs-title="This top tooltip is themed via CSS variables.">
  Custom tooltip
</button>
```

Modals

Este componente **consiste** en un cuadro de diálogo o mensaje, personalizado que se despliega mediante una acción concreta y **opaca el fondo de la página**.




```
<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel" ari
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Modal title</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></bu
      </div>
      <div class="modal-body">
        ...
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```

<codoa codo/>

CSS

De todos los tamaños

Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad
Educación

RESPONSIVE

Un mismo diseño, muchos dispositivos.

Diseño Responsive

El diseño web responsive tiene como objetivo **establecer** metodologías y estándares que nos permitan adaptar nuestros diseños a distintos tamaños de pantalla.

El fin es **mantener la experiencia** de navegación intacta sin importar el dispositivo donde se esté reproduciendo el sitio web.

Viewport

Cuando hablamos de **viewport** nos referimos a la **ventana visible de nuestro sitio** en el navegador, es decir, el ancho y alto que componen **solamente el sitio que se está proyectando**.

Este concepto **es el mismo en cualquier entorno**, ya que siempre hace referencia espectro visible, así un sitio se proyecte en un **celular**, una **tableta** o un **monitor de escritorio.**

Para **asegurarnos** de mantener siempre la misma escala de pixeles frente a los cambios de resolución de los diferentes dispositivos, debemos utilizar la etiqueta **<meta>** con el name viewport para la escala 1:1.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

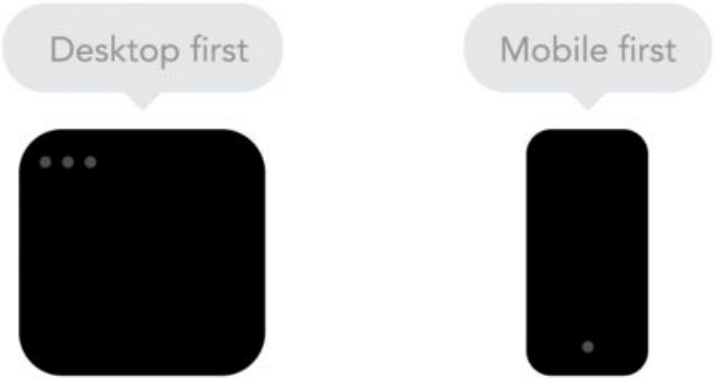
Metodologías Responsive

Responsive Web Design (desktop first)

Propone que **nuestros sitios** sean pensados para verse en una pantalla de escritorio y **adaptar** los estilos para que se ajusten hasta el tamaño de un celular.

Mobile First

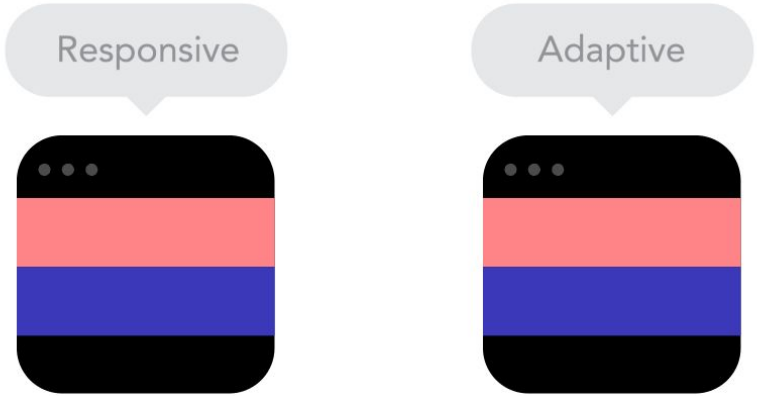
Se piensan los **diseños** teniendo en cuenta los tamaños de celulares para luego ir adaptando los estilos para que el sitio se ajuste a **tamaños de pantalla más grandes**. Pensado para proyectos donde los usuarios acceden en mayor medida desde este tipo de dispositivos.



Principios del diseño responsivo

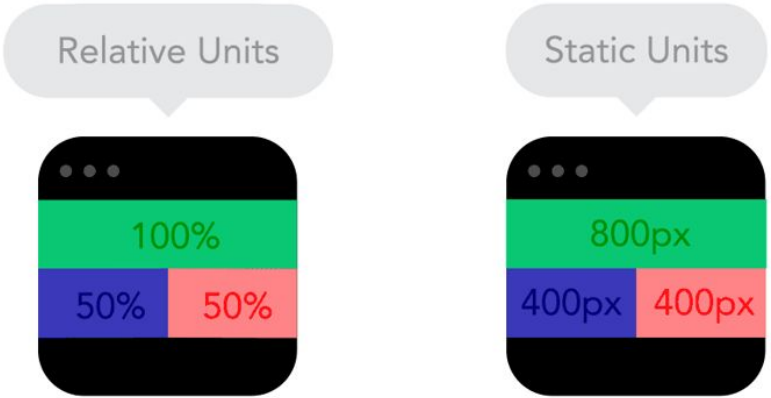
Diseño Responsivo vs Adaptativo

El **Responsivo** es aquel donde nuestro diseño corre de forma fluída ante los cambios de tamaño de las distintas pantallas, mientras que el **Adaptativo** obedece a *cambios más marcados*, donde **el diseño se va adaptando** en los momentos que se **producen quiebres o rupturas** de los elementos del sitio. Si bien ambos son diferentes, en muchos casos se complementan.



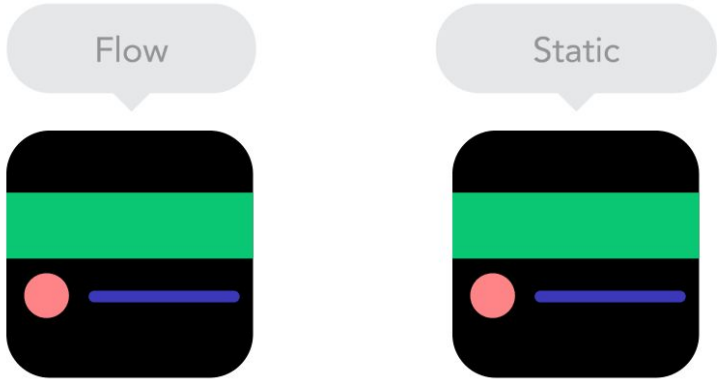
Unidades Relativas

El uso de unidades relativas como **porcentajes**, **rem**, **em**, **vw** y **vh** nos dan la posibilidad de generar proporciones dinámicas que se definan en función de factores inherentes al tamaño de nuestras pantallas, de este modo **evitaremos estructuras rígidas** que favorecen los *overflows* y no logran adaptarse a los diversos cambios de resolución.



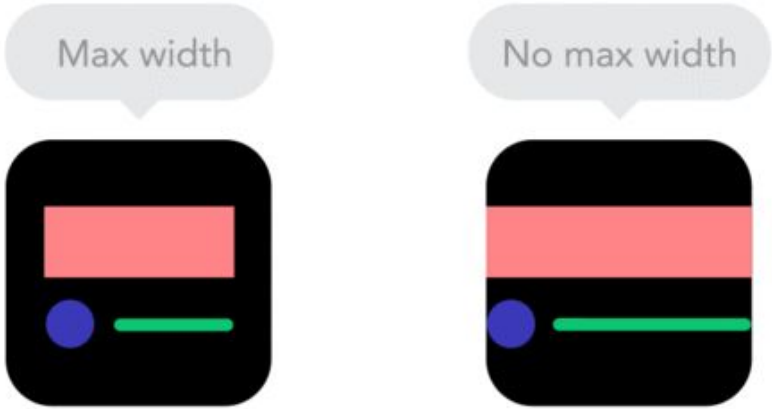
Flujo de los elementos

Gracias al uso de unidades de medida relativas, podemos lograr que nuestros sitios se adapten uniformemente, no solo horizontalmente, sino **también verticalmente** haciendo que nuestras cajas fluyan de forma natural y mantengan siempre sus proporciones.



Máximos y Mínimos

Cuando hablamos de **responsive** solemos concentrarnos en todo lo que sucede cuando nuestra pantalla disminuye de tamaño, pero rara vez pensamos en **qué pasaría** si alguien accede desde **pantallas grandes** como monitores ultra wide o televisores. Para manejar estos extremos siempre es recomendable aprovechar las propiedades `max-width` y `min-width`.



¿Cómo aplicamos responsive?

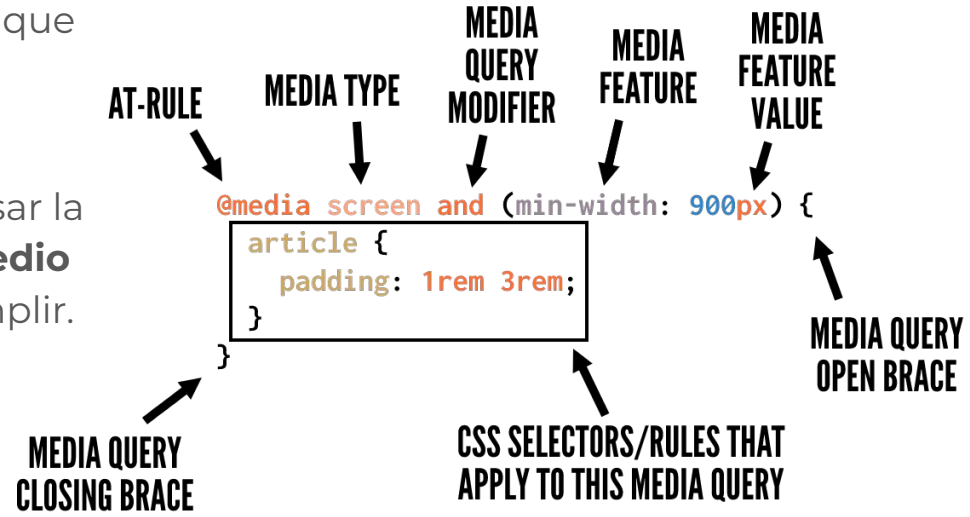
Existen múltiples maneras de generar diseños que se adapten a distintos tamaños de pantallas. Como **hemos visto anteriormente** en el curso podemos valernos de herramientas como **flex** o **grid** y sus propiedades que **nos permiten** manejar la forma en que nuestras cajas fluyen frente ante estos cambios.

Otra forma es el uso de media queries y **breakpoints** predefinidos para establecer qué debe hacer nuestro diseño en los momentos que empieza a deformarse.

Media queries

Las **media queries** son un *bloque de código condicional* donde podremos escribir CSS que sólo se aplicará en caso de **cumplirse** la condición definida en **dicha regla**.

Para definir una media querie debemos usar la palabra reservada @media seguida del **medio** donde debe aplicarse y la condición a cumplir.



Tipos de Media

Los tipos de media query nos ayudan a **establecer** en qué casos debe aplicarse teniendo en cuenta el **medio** donde se proyectará nuestro sitio.

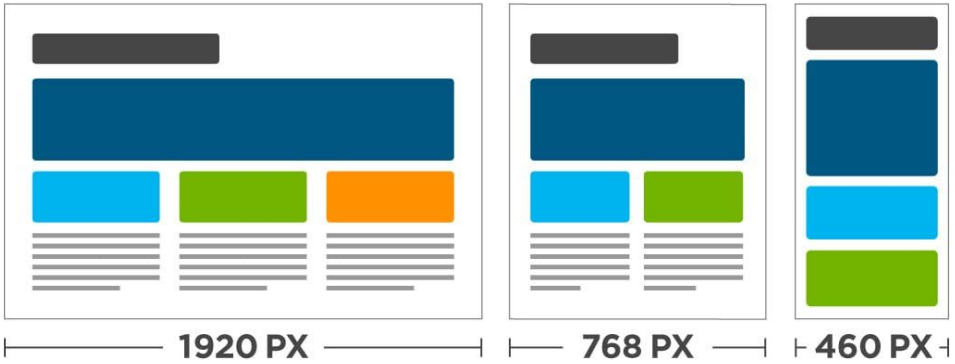
Según el medio elegido, estos **estilos** serán válidos por ejemplo **solo** en *pantallas* o cuando el sitio sea impreso en papel.

screen	Monitores o pantallas de ordenador. Es el más común.
print	Documentos de medios impresos o pantallas de previsualización de impresión.
speech	Lectores de texto para invidentes.
all	Todos los dispositivos o medios. El que se utiliza **por defecto**.

Breakpoints

Son las **reglas condicionales** que le colocamos *a nuestros media queries* para que apliquen nuestros **estilos** según los **distintos escenarios**.

Estos **pueden adoptar diferentes valores** como max-width y **min-width** o **height**, *landscape*, portrait, **device-width**, entre otras.



Breakpoints

Estas reglas condicionales se las conoce como **breakpoints** o *puntos de quiebre* y **dependiendo** el caso puede ser medidas estándar o estar basadas en necesidades puntuales de un sitio en particular.

En este *ejemplo*, definimos **3** comportamientos distintos **para un mismo elemento**, **dependiendo** el tamaño actual que tome la pantalla.

```
@media screen and (max-width: 480px) {
    .element {
        width: 100%;
    }
}

@media screen and (max-width: 768px) {
    .element {
        width: 50%;
    }
}

@media screen and (max-width: 1024px) {
    .element {
        width: 25%;
    }
}
```