Css

Cascading Style Sheets.

@majoledes

losapuntesdemajo

INDICE

- Conectando HTML y CSS ____ 3
- Comentarios en CSS ____ 5
- Selectores ____ 6
- Pseudo-classes ____ 11
- Pseudo-elementos ____ 18
- Especificidad ____ 20
- Valores y unidades ____ 22
- Propiedades de texto ____ 24
- Box Model ____ 26
  - Margin ____ 27
  - Padding ____ 28
  - Border ____ 28
  - Box-sizing ____ 29
  - Border-radius ____ 30
- Outline ____ 32
- Background ____ 33
- Gradientes ____ 37
- Box shadow ____ 39
- Variables ____ 40
- Posicionamiento ____ 41
- Overflow ____ 43
- Display ____ 45
- Flexbox ____ 46
- Grid ____ 53
- Animaciones ____ 60

Conectando HTML-CSS

Hay tres formas de insertar una hoja de estilo:

1. CSS externo
2. CSS interno
3. CSS en línea

1. CSS externo

En el HTML debemos incluir una referencia al archivo de hoja de estilos (.css) dentro del elemento <link>:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
```

extensión: .css

2. CSS interno

Se define dentro de la etiqueta <style> de una página HTML:

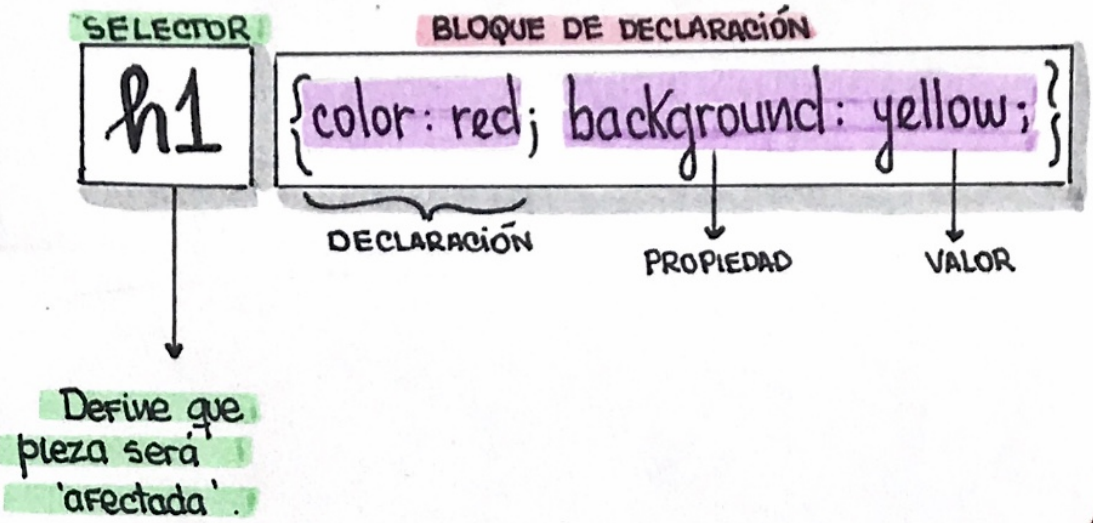
```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: pink;
}

h1 {
    color: blue;
}

</style>
</head>
```

CSS en línea

Se puede usar un estilo en línea para aplicar un estilo único para un solo elemento.

Los estilos en línea se definen dentro del atributo estilo del elemento relevante:

```
<h1 style = "color:blue; font-size: 20px;">¿Qué onda?</h1>
```



comentarios en CSS

```
h1 {color: gray} /* Comentario */
p {color: white} /* Otro comentario */
h2 {color: gray} /* Este es otro pero
                    más largo */
```

selectores

Se utilizan para 'seleccionar' los elementos HTML que queremos diseñar.

• selector de elementos

selecciona elementos HTML en función del nombre del elemento.

```
h2 {color: gray;}
```

• agrupación de selectores

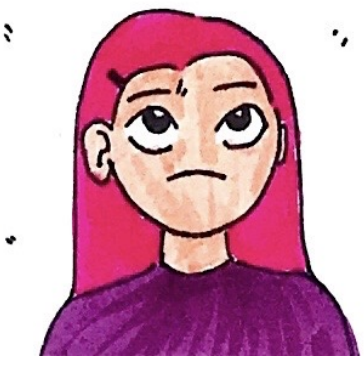
Supongamos que queremos que distintos elementos tengan un mismo estilo.

```
h2 {color: gray}
p {color: gray}
```

```
h2, p {color: gray}
```

↓

de esta manera es más fácil



· selector universal

El selector universal (*) selecciona todos los elementos del HTML.

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

actúa como un 'comodín' que coincide con cualquier elemento.

· selector de clase

Es la forma más común de aplicar estilos.

Para que los selectores de clase funcionen y asociar los estilos de esa clase a algún elemento, simplemente debemos agregar el atributo 'class' a ese elemento:

```
<p class = 'cosito'> Estamos aprendiendo CSS </p>
```

Ahora, agregamos los estilos de la clase:

```
.cosito {
  color: pink;
  font-size: 20px;
}
```

En el HTML es posible agregar más de una clase a un elemento.

```
<p class = "cosito  contenedor"> Blablabla </p>

<h2 class= "cosito contenedor"> Blebleble </h2>
```

```
.cosito { color: pink;
          font-size: 20px;
        }
```

```
h2.cosito { font-weight: bold; }
```

♥ Aquí se agregará la fuente en bold únicamente al elemento h2 con clase 'cosito', no así al p.

selector ID

De cierta forma el selector ID es similar al selector de clase, excepto por :

- ♥ al selector ID lo precede un hash (#)
- ♥ el id puede utilizarse en un único elemento del HTML. (las clases pueden ser asignadas a varios elementos).

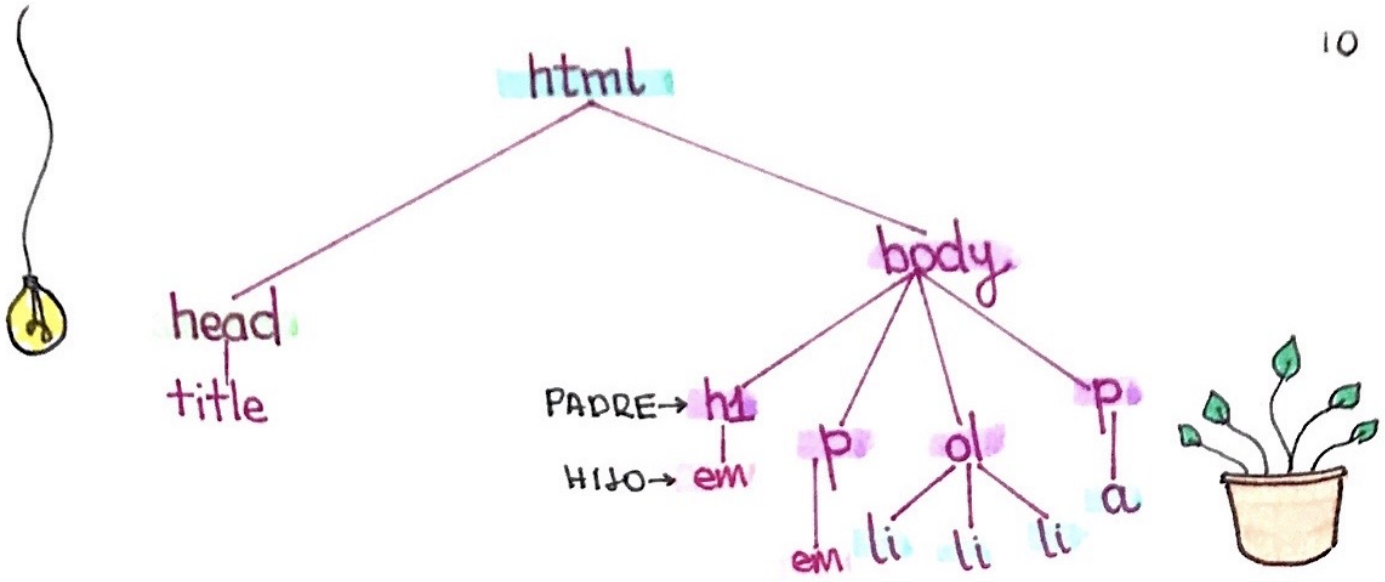
```
<p id="cosito">A la grande le puse Cuca</p>
```

hash ..

```
#cosito {
        background: yellow;
}
```

Ahora vamos a entender como funciona la estructura de un documento HTML, para así comprender la relación de elementos padre-hijo:

```
<html>
<head>
  <title> Apuntes a mano
</head>
<body>
  <h1> Apuntes de <em> Mayo </em> </h1>
  <p> Apuntes <em> disponibles </em> </p>
  <ol>
    <li> JavaScript </li>
    <li> React </li>
    <li> Python </li>
  </ol>
  <p> ¡ Recomendame temas en: <a href = "https://..."> Twitter </a> </p>
</body>
</html>
```



selectores descendientes

El primer beneficio de comprender los selectores descendientes es poder definirlos. Por ejemplo, hagamos de cuenta que tenemos muchos <h1> con como hijos y quisiéramos estilar esos . En vez de colocar una clase a cada , la forma más sencilla sería:

```
h1 em { color: gray; }
```

Esto quiere decir que todos los definidos dentro de un <h1> serán de color gris.

El límite no son dos selectores. Por ejemplo:

```
ul ol ul em { color: gray; }
```

Esto es: cualquier que sea parte de una lista desordenada que sea parte de una lista ordenada que sea parte de una lista desordenada será gris.

En algunos casos, no deseamos seleccionar 'cualquier' elemento descendiente, sino un elemento hijo de otro elemento. Veamos:

Queremos seleccionar el elemento sólo si es 'hijo' de un h1. Para hacer esto, utilizamos el símbolo >.

```
h1 > strong { color: gray; }
```

Esta regla se aplicará al strong del primer h1 pero no del segundo:

1. `<h1> Esto es <strong> muy </strong> importante. </h1>`
2. `<h1> Esto es <em> realmente </strong> muy </strong></em> importante </h1>`

Pseudo-clases

¡Las cosas se están poniendo interesantes!. Veamos:

Es posible combinar pseudo-clases juntas. Por ejemplo:

```
a:link:hover {color: red;}
a:visited:hover {color: pink;}
```

Todas las pseudo-clases, sin excepción, son una palabra precedida por dos puntos (:).

:empty

Con la pseudo-clase :empty podemos seleccionar cualquier elemento que no tenga hijos de ningún tipo, incluyendo textos o espacios en blanco. Debe estar verdaderamente vacío.

Ejemplo: `p:empty {display: none}`

`<p></p>` → será seleccionado

`<p> </p>` } no serán seleccionados

`<p><!--comentario--></p>` } no serán seleccionados

:first-child

Se usa para seleccionar elementos que son los primeros hijos de otros elementos.

```
<div>
  <p> Sigue estos pasos: </p>
  <ul>
    <li> Despierta </li>
    <li> Vive </li>
    <li> Duerme </li>
  </ul>

  <p>Simplemente eso </p>
</div>
```

First-child del div ←

First-child del ul ←

1. p: First-child { Font-weight: bold;}
2. li: First-child {text-transform: uppercase;}

1. **Sigue estos pasos:**
2. DESPIERTA

Del mismo modo, si en lugar de First-child seleccionamos el **:last-child**, se vería así:

p: last-child {Font-weight: bold;} → **Simplemente eso**

li: last-child {text-transform: uppercase;} → DUERME

:first-of-type

Representa el primer elemento de su tipo entre un grupo de elementos hermanos.

```
<h1>Encabezado</h1>
<p> Párrafo 1 </p>  ——→ first-of-type
<p> Párrafo 2 </p>
```

Seleccionará el primer elemento dentro de cada elemento que lo contenga descartando a los elementos hermanos. De manera similar funciona

:last-of-type

pero seleccionando el último.

:nth-child

Coincide con un elemento en función de su posición entre un grupo de hermanos. Por ejemplo, el first-child sería → nth-child(1)

Podemos colocar cualquier número entero que queramos, siempre que tenga un caso de uso.

De una manera más poderosa, podemos usar fórmulas.

Supongamos que queremos seleccionar cada tercer elemento de

una lista en una lista desordenada, comenzando por el primero.

ul > li :nth-child (3n+1) { CSS declaration }

↓

an+b

n: representa la serie 0, 1, 2, 3, 4 ... hasta el infinito.

La fórmula resuelve: los elementos seleccionados serán 1, 4, 7, 10 ...

Si eliminamos el +1, dejando 3n, el resultado sería 0, 3, 6, 9, 12 ...

:nth-child (7) → Representa el séptimo elemento

:nth-child (5n) → Representa los elementos 5, 10, 15...

:nth-child (3n+4) → Representa los elementos 4, 7, 10, 13 ...

- **VALORES:**

odd: Representa elementos cuya posición numérica en una serie de hermanos es impar: 1, 3, 5, etc.
:nth-child(odd) ó :nth-child(2n+1)

even: Representa elementos cuya posición numérica en una serie de hermanos es par: 2, 4, 6, etc.
:nth-child(even) ó :nth-child(2n)

hyperlink pseudo-classes

:link

Se refiere a un enlace. (con atributo href) y apunta a un link que no ha sido visitado.

:visited

Se refiere a un enlace que ya ha sido visitado.

```
a:link {color: blue}    → los links no visitados son azules.
a:visited {color: red}  → los links visitados son rojos.
```

user-action pseudo-classes

Cambian la apariencia en función de acciones tomadas por el usuario.

:focus

Representa un elemento (como una entrada de formulario) que ha recibido el 'foco'. Generalmente se activa cuando

el usuario hace clic, toca un elemento o lo selecciona con la tecla 'Tab'.

:hover

Se refiere a cualquier elemento sobre el que se coloca el puntero del mouse. Por ejemplo, un link.

:active

Se refiere a cualquier elemento que ha sido activado por el usuario. Por ejemplo, un link o un botón. (Cuando se está presionando con el botón del mouse.)

¡Hay muchas más pseudo-clases para explorar!

pseudo-elementos

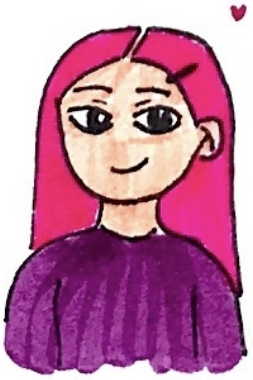
Se utilizan para diseñar partes específicas de un elemento.

Para diferenciarlos de las pseudo-clases, los pseudo-elementos son precedidos por cuatro puntos (::)

::first-letter

Se utiliza para agregar un estilo especial a la primera letra de un texto.

```
p::first-letter {color: red;}
// Esto es un párrafo
```



::first-line

Se utiliza para agregar un estilo especial a la primera línea de un texto.

```
<p>Tu trabajo va a llenar gran parte de tu vida.
la única manera de estar realmente satisfecho
es hacer lo que creas que es un gran trabajo
y la unica manera de hacerlo es amar lo que
hacus </p>

p::first-line{color: pink;}
```

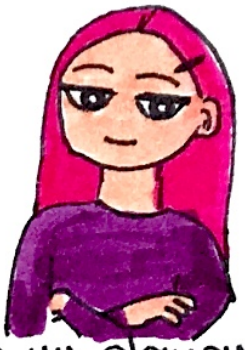
Se aplicará a la primera línea

::before

Puede utilizarse para colocar algo de contenido antes del contenido de un elemento.


```
a::before {
    content: "♥";
}
```

añadirá un corazón antes de los enlaces



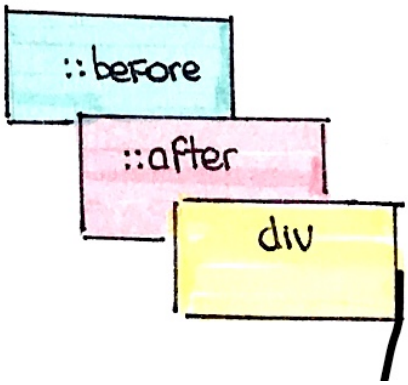
::after

Coloca algo de contenido después del contenido de un elemento

Tip: Cuando 'dibujo' con CSS suelo colocar pseudo-elementos detrás del elemento padre. Para poder hacerlo:

```
div::after {
    position: absolute;
    z-index: -10;
}
div::before {
    position: absolute;
    z-index: -20;
}
```

z-index negativo



::selection

Aplica reglas/estilos a una porción del documento que ha sido resaltado por el usuario (cuando hace clic y arrastra el mouse por el texto 😂)

sólo un pequeño grupo de propiedades pueden ser utilizadas con este pseudo-elemento: color, background, background-color, text-shadow.

```
p::selection {color: grey;
              background: orange;}
```

dibuja ~~en~~ cualquier texto seleccionado en color gris sobre un fondo naranja.

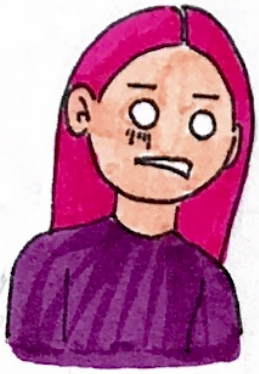
ESPECIFICIDAD

Es la manera mediante la cual los navegadores deciden qué valores de una propiedad CSS son más relevantes para un elemento y, por lo tanto, serán aplicados.

Jerarquía de especificidad

Cada selector tiene su lugar en la jerarquía de especificidad. Hay cinco que definen el nivel de especificidad de un selector:

!important	1,0,0,0,0
inline styles	0,1,0,0,0
#id	0,0,1,0,0
.class (pseudo-class también)	0,0,0,1,0
etiquetas (elementos, pseudoelementos)	0,0,0,0,1



♥ Ejemplo:

A: h1

B: #content h1

C: <h1 style="color: #FFFFFF">Heading</h1>

- La especificidad de A es 1 (es una etiqueta)
- La especificidad de B es 101 (un #(id) y una etiqueta)
- La especificidad de C es 1000 (es un estilo en línea)

Mayor nivel de especificidad por lo tanto se aplicará.

!important ⚠

Cuando se emplea !important en una declaración de estilo, esta declaración sobreescribe a cualquier otra. Sin embargo, el uso de !important es una mala práctica y debería evitarse. Buscá siempre la manera de emplear la especificidad antes de considerar su uso.

Valores y unidades

• KEYWORDS:

- none: "ninguno"

```
a:link, a:visited {text-decoration:none;}
```

- inherit: el valor de una propiedad en un elemento será heredada del elemento padre.

- initial: establece el valor de una propiedad en el valor inicial. Es decir, 'restablece' el valor.

- unset: restablece el valor de un elemento al valor heredado, si es que heredó uno y a su valor inicial si no es así.

medidas

ABSOLUTAS:

Pixels (px): Al usar esta medida, esta no cambiará, sin importar el tamaño del proyecto.

in, cm, pc, mm, Q.

RELATIVAS:

em: se utiliza para hacer referencia al tamaño actual de la fuente que ha sido establecido en el navegador.

rem: representa el tamaño (font-size) del elemento raíz (por ejemplo, el tamaño de fuente del elemento <html>) Cuando se aplica a font-size del elemento raíz, representa su valor inicial.

"FLEXIBLES": Relativas al tamaño del viewport.

vw: se calcula con respecto al ancho del viewport.

El 100vw corresponde al 100% de la ventana.

Al arrastrar la ventana a más ancho o más estrecho, el valor se mantendrá

- **vh**: [viewport height] Se calcula con respecto a la altura del viewport.
- **vmin/vmax**: se usan para utilizar el porcentaje de ancho o alto del viewport, dependiendo cuál sea más pequeño o más grande de los dos.

Font

- ♡ Font-size: tamaño de la fuente.
- ♡ Font-style: normal, italic, oblique
- ♡ Font-weight: normal, bold, 100-900 { normal: 400, bold: 700
- ♡ Font-Family: especifica el tipo de fuente.

Propiedades de texto

ALINEACIÓN DE TEXTO:

text-align: Valores: start, end, left, right, center, justify.

ALTURA DE LÍNEA:

line-height: Valores: <number>, <length>, <porcentaje>, normal

ESPACIADO ENTRE PALABRAS :

word-spacing valores : <length>, normal

ESPACIADO ENTRE LETRAS :

letter-spacing valores : <length>, normal

TRANSFORMACIÓN DE TEXTO:

text-transform valores :

- uppercase : texto en mayúsculas
- lowercase : texto en minúsculas
- capitalize: Primera letra en mayúsculas
- none

DECORACIÓN DE TEXTO :

text-decoration valores :

- none
- underline : subrayado
- overline : línea arriba del texto
- line-through : línea sobre el texto

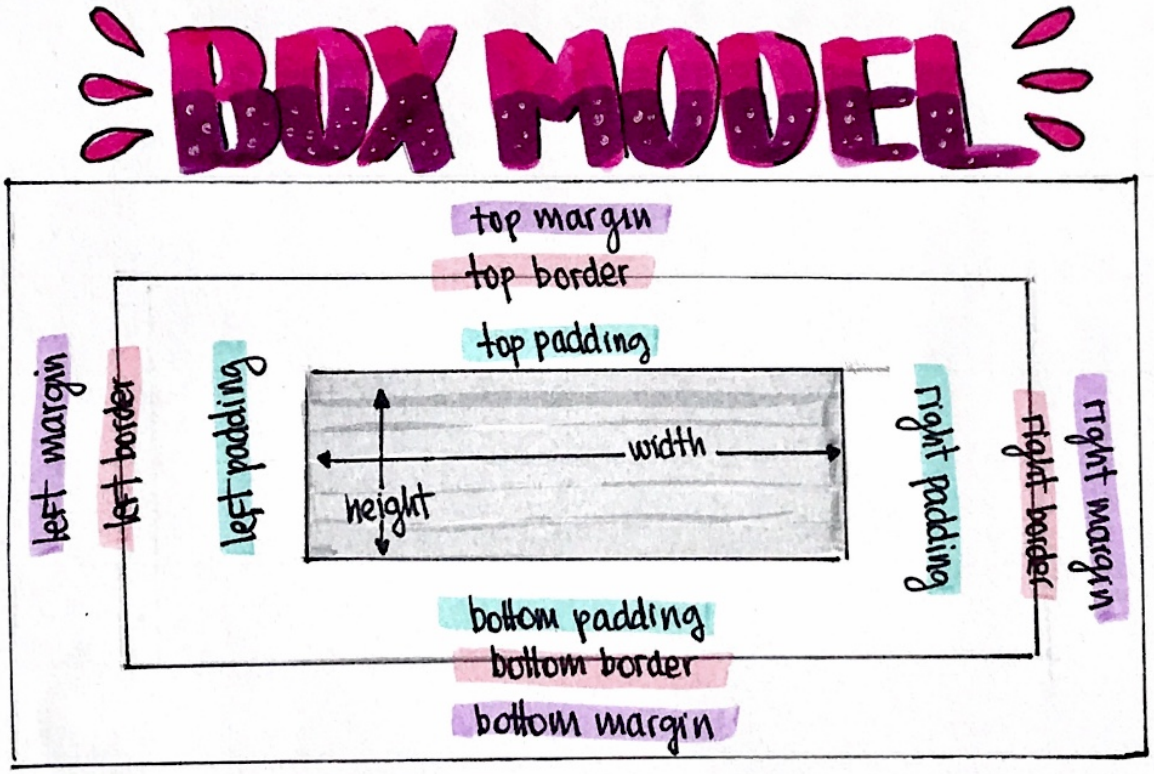
SOMBRA DE TEXTO :

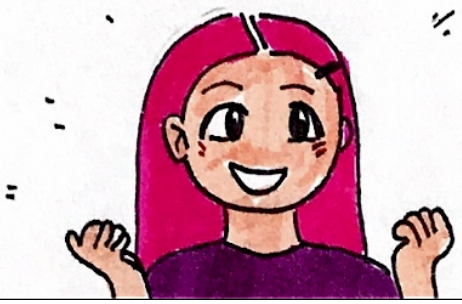
text-shadow valores: none, <length> <length> <length>

Ejemplo: text-shadow: green 5px 1px 4px → blur

- color
- desplazamiento horizontal de la sombra
- desplazamiento vertical de la sombra

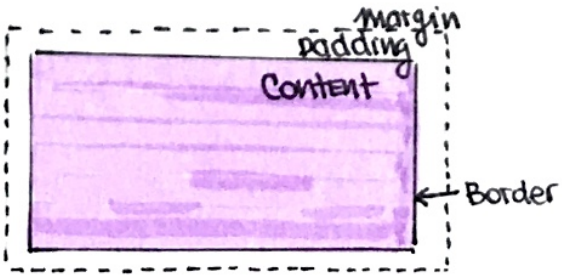
♥ ¡Hay muchísimas más propiedades para un texto! ♥





margin

Se utilizan para crear espacio alrededor de los elementos, fuera de los bordes definidos.



CSS tiene propiedades para especificar el margen para cada lado del elemento:

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

VALORES:
- **auto**: El navegador calcula el margin.
- **longitud**: en px, cm, etc.
- **porcentaje**: especifica un margen en % del ancho del elemento contenedor.

TOP, BOTTOM

`margin: 25px 50px 75px 100px;`

RIGHT, LEFT

`margin: 25px 75px 100px;`

TOP, RIGHT y LEFT, BOTTOM

`margin: 25px 75 px`

TOP y BOTTOM, RIGHT y LEFT

`margin: 25px`

TOP, BOTTOM, RIGHT, LEFT

padding

Se utiliza para generar espacio alrededor del contenido de un elemento dentro de los bordes definidos.

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

} valores → longitud: px, cm, etc.
→ porcentaje: %. En relación al ancho (width) del contenedor.

padding: 25px 5px 10px 10px;
↓ top ↓ right ↓ left ↓ bottom

padding: 25 px 5px 10px;
↓ top ↓ right y left ↓ bottom

padding: 5px 10px;
↓ top y bottom ↓ right y left

padding: 10px;
↓ top bottom right left

border

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

border, border-top, border-right, border-bottom, border-left.

- border-color: color de borde.
- border-style: estilo de borde [dashed, solid, dotted, double, groove, inset, outset, none, hidden]
- border-width: ancho de borde. Puede tener cuatro valores diferentes (Ej: 0 4px 8px 12px) ↓ top ↓ right ↓ bottom ↳ left

Se puede establecer:

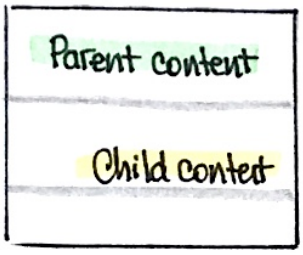
```
border: 3px solid red;
```

box-sizing

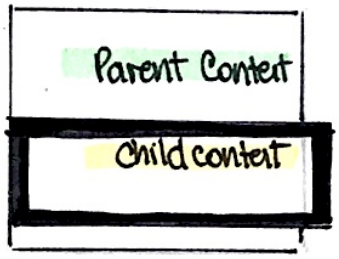
Esta propiedad indica como se debe calcular el ancho y el alto total de un elemento.

Valores: content-box | border-box.

- content-box: Es el valor inicial y por defecto. Las propiedades width y height no incluyen el borde, padding o margin.



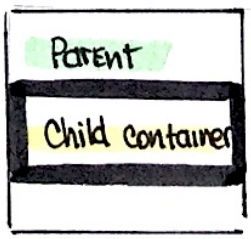

```
box-sizing: content-box;
width: 100%;
```




```
box-sizing: content-box;
width: 100%;
border: solid black 10px;
padding: 2px;
```

- border-box: Las propiedades width y height incluyen el contenido, padding y borde, pero no incluyen margin.

ancho (width) = border + padding + ancho
alto (height) = border + padding + alto




```
box-sizing: border-box;
width: 100%;
border: solid black 10px;
padding: 2px;
```

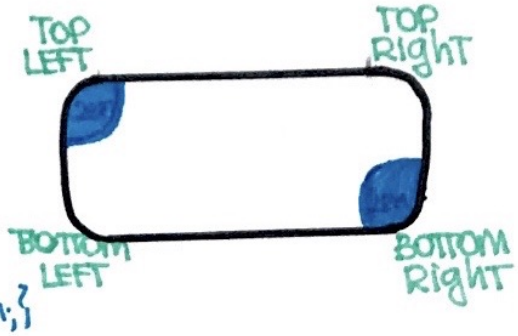
border-radius ♥

Esta propiedad se utiliza para generar bordes redondeados en un elemento.

#ejemplo: {border-radius: 2em;}

↓ Equivale a los cuatro bordes

{border-radius: 2em 2em 2em 2em;}



Funciona como el margin o padding, es decir,

#ejemplo {border-radius:

4 valores {
- 1em → TOP LEFT
- 2em → TOP RIGHT
- 2em → BOTTOM RIGHT
- 3em → BOTTOM LEFT

}

Si hay 3 valores:

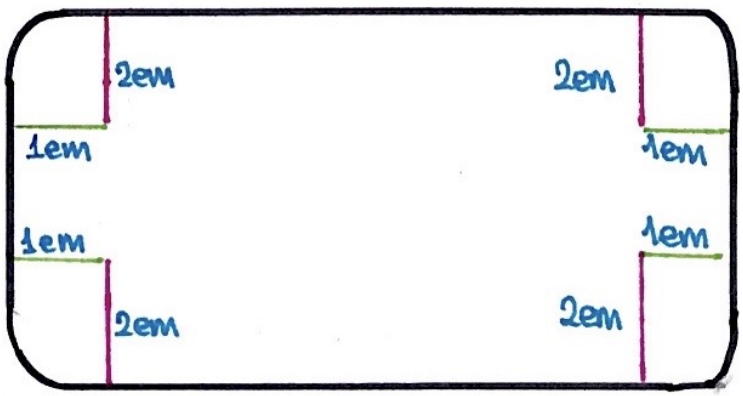
{border-radius:
- 1em → TOP LEFT
- 2em → TOP RIGHT | BOTTOM LEFT
- 3em → BOTTOM RIGHT

}

¡Un poco más complicado!

#ejemplo {border-radius: 1em / 2em;} donde 1em es horizontal y 2em es vertical.

↓ Equivale a: 1em 1em 1em 1em / 2em 2em 2em 2em;
(1em 1em 1em 1em: HORIZONTAL; 2em 2em 2em 2em: VERTICAL)



```
#ejemplo { border-radius: 1em 2em / 2em 3em; }
```

horizontal vertical

1em 2em : TOP LEFT | BOTTOM RIGHT
2em 3em : TOP RIGHT | BOTTOM LEFT

¿POR QUÉ?

Acordate de la regla cuando tenemos sólo 2 valores:

4 valores
1em TOP LEFT ①
2em TOP RIGHT ②
3em BOTTOM RIGHT ③
4em BOTTOM LEFT ④
→ 2 valores
1em TOP LEFT BOTTOM RIGHT ① y ③
2em TOP RIGHT BOTTOM LEFT ② y ④

Además, no te confundas! No creas que el 1em 2em a la izquierda del slash (/) define el valor de una esquina y el 2em 3em de otra. Recordá que los valores horizontales se colocan ANTES del slash y los verticales, después.

Entonces:

1em 2em / 2em 3em;

1em 2em 2em 3em.

Si quisieras, también podes modificar un sólo borde con las siguientes propiedades:

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

>outline<

Los bordes y los contornos son muy similares. Sin embargo, difieren en lo siguiente:

- Los contornos no ocupan espacio, son dibujados por encima del elemento. No suma en el tamaño de la caja.
- Los contornos pueden no ser rectangulares. Si el elemento se distribuye en varias líneas, el contorno no forma un rectángulo para encerrar a todo el elemento.

Ejemplo: contorno aplicado a varias líneas.

El contorno no está abierto en algunos lados.

Ejemplo: borde aplicado a varias líneas.

La propiedad outline es una manera reducida para establecer una o más de las propiedades individuales de outline (contorno):

outline-style, outline-width, outline-color.

- **outline-style:** establece el estilo del contorno de un elemento.

  Valores: none, dotted, dashed, solid, double, inset, outset...

- **outline-width:** establece el grosor del contorno de un elemento.

  Valores: thin (generalmente 1px), medium (generalmente 3px), thick (generalmente 5px), medidas (px, em...)

- **outline-color:** establece el color del contorno de un elemento.

BACKGROUND

Para declarar un color de fondo de un elemento, utilizamos la propiedad background-color, el cual acepta cualquier color como valor.

```
<p> Los apuntes de Mayo </p>

p { background-color: pink; } // Los apuntes de Mayo.
```

Podemos extender el background un poco, del elemento usando padding:

```
p{background-color: pink;
    padding: 10px;
}
```

→ Los apuntes de Mayo

También, podemos colocar una imagen de fondo utilizando la propiedad background-image.

Ejemplo: `body{background-image: url(...);}`

Además, existe la propiedad background-position para posicionar el fondo.

Valores: left, center, right, top, bottom, porcentaje, medidas. (px, em...)

Ejemplo:

```
body {background-position: top right;}
```

top right: arriba y a la derecha. → esquina superior derecha

Los valores pueden colocarse en pares (como el ejemplo anterior) o no. Luego, también, están los valores en longitud (50px) y finalmente, los valores porcentuales (43%).

Los 'res que se colocan en pares pueden aparecer en cualquier orden, siempre que no haya más de dos, uno para horizontal y uno para vertical. Si usamos dos horizontales (right right;) o dos

verticales (top top;) se ignora el valor completo.

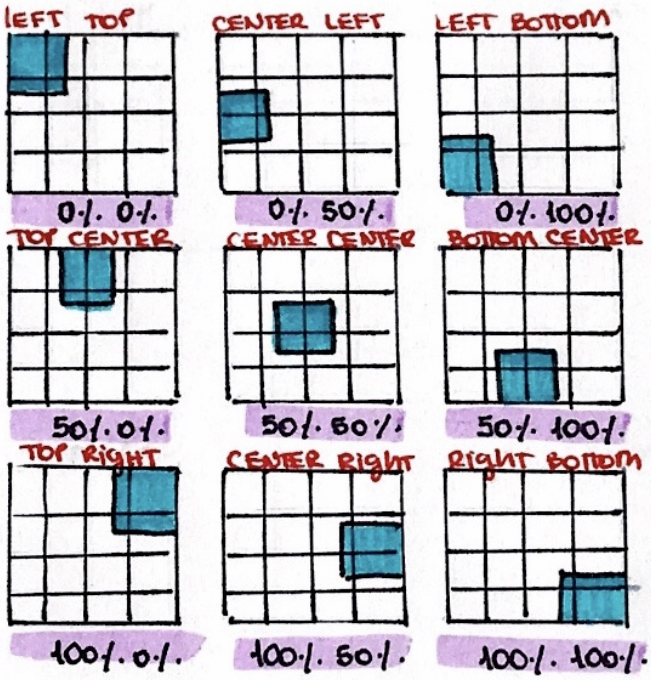
Si solo aparece una palabra, se asume que la otra es el centro.

Valores porcentuales:

Para centrar una imagen dentro de su elemento, es bastante fácil:

```
p {background-image: url(chrome.jpg);
   background-position: 50% 50%;
  }
```

Veamos algunos ejemplos para una mejor comprensión:



valores de longitud:

Estos se interpretan como desplazamientos de la esquina superior izquierda. Si establecemos los valores 20px 30px;, la esquina superior izquierda de la imagen será 20px a la derecha y 30px hacia abajo.

También existe la regla background-repeat, que como su nombre lo indica, repite una imagen tanto horizontal como verticalmente.

Mostrar la imagen sólo una vez también lo especifica esta propiedad ☺

Valores: repeat, no-repeat, space, round.

Si proporcionamos dos valores, el primero se aplica en la dirección horizontal y el segundo en la vertical. Si hay un sólo valor se aplica tanto en horizontal como en vertical.

Equivale a:

repeat-x → repeat no-repeat
repeat-y → no-repeat repeat
repeat → repeat repeat
no-repeat → no-repeat no-repeat
space → space space
round → round round

Todas las propiedades de background pueden juntarse en una sola propiedad: background.

```
body {background-color: black;
      background-image: url(hola.png);
      background-position: top left;
      background-repeat: no repeat;
      background-size: 50%. 50%.;}
```

tamaño del background.

```
body {background: black url(hola.png) no repeat top left / 50%. 50%.;}
```

⚠ El background-size debe aparecer después del background-position.

gradients

CSS tiene dos tipos de gradientes: linear-gradients y radial-gradients.

linear-gradients:

Los degradados lineales son 'degradados' que avanzan a lo largo de un vector lineal.

El degradado va de arriba (top) a abajo (bottom) porque la dirección es así de forma predeterminada, que es lo mismo que 180 grados.

La sintáxis básica de un gradiente lineal es:

```
linear-gradient (ángulo-dirección, color-stop1,
                 color-stop2 ....
                )
```

Ejemplo:

```
#ejemplo {background: linear-gradient (90deg,
          red, orange, yellow, blue, red); }
#ejemplo2 {background: linear-gradient (to left,
          purple, gold); }
```

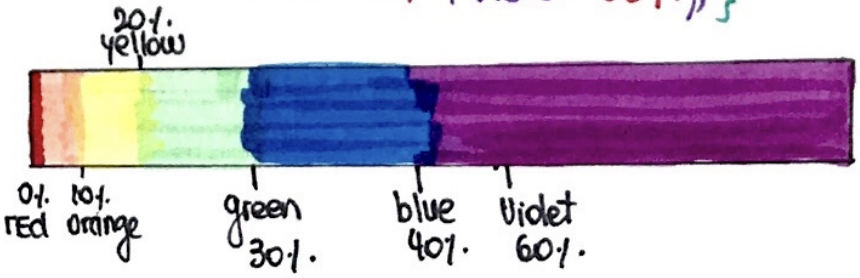
Además, los colores pueden tener una posición:

```
#espectro {background-image: linear-gradient (90deg,
          red, orange 25px, yellow 50px, green 75px,
          blue 100px, violet 150px); }
```

→ ¡imaginen un gradiente!

En cuanto a los porcentajes, se calculan con respecto a la longitud total del gradiente.

```
#espectro {background-image: linear-gradient(90deg,
    red, orange 10%, yellow 20%, green 30%,
    blue 40%, violet 60%);}
```



radial-gradients:

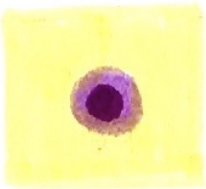
Sintaxis:

radial-gradient ([Forma-tamaño] [posición] [color1] [color2] ...)

Para [forma-tamaño] hay 2 posibles formas: círculo y elipse.

Ejemplos:

radial-gradient(50px, purple, yellow) → significa: "un radial-gradient circular con un centro púrpura a amarillo, con una distancia de 50px desde el centro"



ELIPSE

radial-gradient(50px 100px, purple, yellow)

¡Para las elipses podemos usar porcentajes!

Herramienta: cssgradient.io

BOX-SHADOW

Esta propiedad agrega sombra a los elementos (al igual que text-shadow agrega sombra a los textos)

Su valor por defecto es none.

Sintaxis:

```
box-shadow: sombra horizontal  sombra vertical  blur  spread  color ;  inset | initial | inherit
```

- sombra horizontal: Un valor positivo coloca la sombra hacia el lado derecho del elemento y un valor negativo hacia el lado izquierdo.
- sombra vertical: Un valor positivo coloca la sombra debajo del elemento y un valor negativo, por arriba del elemento.
- blur: Es el grado de desenfoque. A mayor valor, la sombra será más borrosa.
- spread: Valor de propagación. Un valor positivo aumenta el tamaño de la sombra y uno negativo, la disminuye.
- color: color de la sombra.

Si especificamos la sombra como inset, ésta sera interna.

initial: Establece la propiedad en su valor predeterminado.

inherit: Hereda esta propiedad de su elemento padre.

Variables

Las variables en CSS deben declararse dentro de un selector.

Generalmente se utiliza :root para que la variable sea global.

El nombre de una variable debe comenzar con dos guiones(--) y distingue entre mayúsculas y minúsculas.


```
:root {
    --my-color: yellow;
}
body {
  color: var(--my-color)
}
```

→ todos los textos, párrafos, títulos, encabezados, etc serán de color amarillo por HERENCIA.

```
h1 {
--my-color: red;
}
```

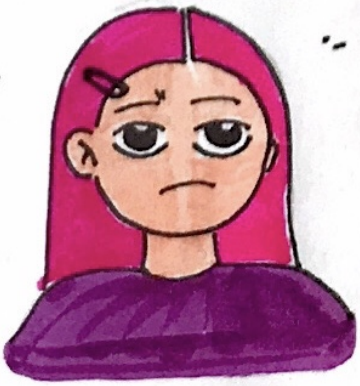
→ Pero todos los h1 serán rojos porque utilizamos la CASCADA para redefinir el valor de la propiedad --color

posicionamiento

position

- valores -

static | relative | sticky | absolute | fixed



- **static**: valor por defecto. un elemento con posición: static no está posicionado. este elemento no se verá afecrado por las propiedades top, bottom, right, left.

Las posiciones relative, sticky, absolute y fixed desbloquean las propiedades top, bottom, right y left.

- **relative**: se comporta de la misma forma que static a menos que agreguemos las offset properties (top, bottom, left y right) causando un reajuste en su posición. otro elemento no podrá ajustarse para adaptarse a cualquier hueco dejado por el elemento.

- **absolute**: un elemento con position: absolute; está posicionado en relación a su ancestro posicionado (≠ static) más cercano, si no tiene ancestro posicionado usará el elemento body del documento, y se seguirá moviendo al hacer scroll en la página.

- **fixed**: está posicionado con respecto a la ventana del navegador, lo que significa que se mantendrá en el mismo lugar incluso al hacer scroll en la página.
  No dejará espacio en el lugar de la página donde estaba ubicado normalmente.

- **sticky**: se posiciona según la posición de desplazamiento del usuario.
  se "pega" en su lugar, luego de alcanzar una posición de desplazamiento determinada.

z-index

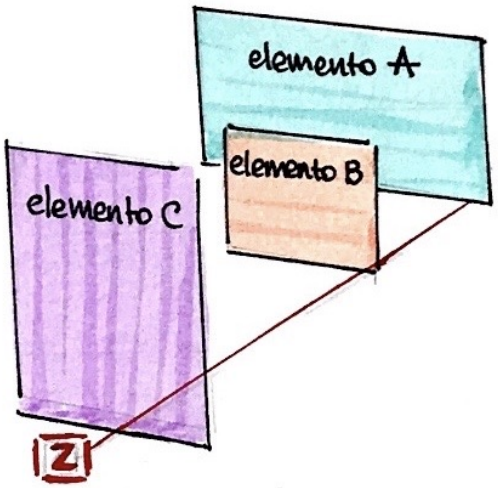
Inevitablemente habrá una situación en la que dos elementos intentarán existir en el mismo lugar, visualmente hablando. Uno de ellos tendrá que superponerse a otro.

Pero ¿cómo controlamos qué elemento estará por encima?

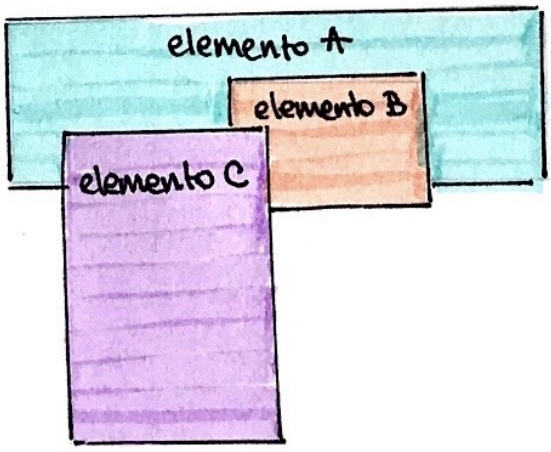
Para eso existe la propiedad `z-index`.

Esta propiedad nos permite modificar la forma en que los elementos se superponen entre sí.

Un elemento con un valor de z-index más alto se superpondrá a los que tengan un menor valor.




```
elementoC {
    z-index: 8;
}

elementoB {
    z-index: 4;
}

elementoA {
    z-index: 1;
}
```



Overflow

Esta propiedad controla lo que sucede con el contenido que es demasiado grande para caber en un área.
Es decir, especifica si se recorta el contenido o se agregan barras de desplazamiento cuando el contenido de un elemento es demasiado grande para caber en el área especificada.

valores:

- **visible**: valor por defecto. El desbordamiento es `visible`, lo que significa que no está recortado, y se renderiza fuera de la caja del elemento.
- **hidden**: el desbordamiento se recorta y el resto del contenido se oculta.
- **scroll**: el desbordamiento se recorta y se agrega una barra de desplazamiento para 'desplazarse' dentro del cuadro..
  ⚠ (agregará una barra de desplazamiento tanto horizontal como vertical)
- **auto**: El valor `auto` es similar a `scroll`, pero agrega barras de desplazamiento sólo cuando es necesario.

overflow-x y overflow-y

Especifican si se debe cambiar el desbordamiento de contenido solo horizontal o verticalmente (o ambos):

overflow-x: especifica qué hacer con los bordes izquierdo / derecho del contenido.

overflow-y: especifica qué hacer con los bordes superior / inferior del contenido.

```
div {
  overflow-x: hidden;
  overflow-y: scroll;
}
```

:opacity:

opacity: 1.0;

opacity: 0.6;

opacity: 0.3;

Esta propiedad especifica la ~~propiedad~~ opacidad - transparencia de un elemento.

Puede tomar un valor de 0,0 a 1,0. Cuanto menor sea el valor, más transparente se verá el elemento.

Cuando se utiliza la propiedad opacity para agregar transparencia al fondo de un elemento, todos sus elementos secundarios heredan la misma transparencia.

Se puede aplicar transparencia utilizando RGBA.

rgba → (rojo, verde, azul, alfa)

Parámetro alfa: número entre 0.0 (completamente transparente) y 1.0 (completamente opaco)

```
div { background: rgba(76, 175, 80, 0.3)
    }
```

Display

Propiedad muy (o la más) importante para controlar estructuras.

Cada elemento tiene un valor de display por defecto. Usualmente es `block` (de bloque) o `inline` (en línea).

BLOCK : un elemento block comienza en una nueva línea y se estira hasta la derecha e izquierda tan lejos como pueda. Ejemplo: `<div>` `<p>`, `<header>`, `<footer>` ...

INLINE : un elemento inline puede contener algo de texto dentro de un párrafo sin interrumpir el flujo del párrafo. Ejemplo: `<span>` ó `<a>` ya que se usa para links.

NONE : comúnmente utilizados para ocultar elementos sin eliminarlos. usar `display: none` no dejará espacio donde el elemento se encontraba, como `visibility: hidden` que deja un espacio vacío.

otros valores de display: list-item, table.

INLINE-BLOCK: Permite establecer un ancho (width) y un alto (height) de un elemento (esto no se puede hacer con `display: inline`). Además se respetan los márgenes / paddings superior e inferior.
`display: inline-block` no agrega un salto de línea después del elemento (a diferencia de `display: block`)

FLEXBOX

Para comenzar a usar flexbox, primero debemos definir un contenedor flexible:

```
<div class="container">
    <div> 1 </div>
    <div> 2 </div>
    <div> 3 </div>
</div>
```

→

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

• PROPIEDADES DE FLEXBOX:

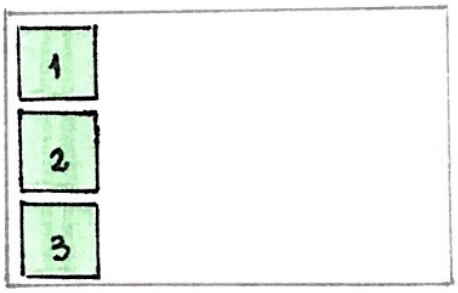
- **flex-direction:** define en qué dirección el contenedor apilará los flex-items (en este caso, los divs 1, 2 y 3)

valores:

- column

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

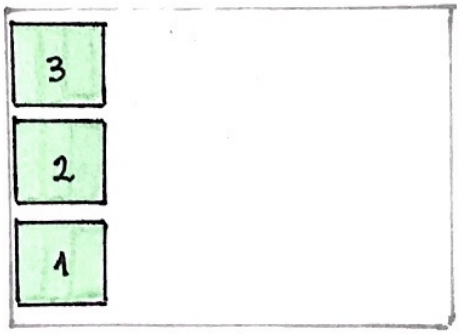
→



APILA VERTICALMENTE DE ARRIBA A ABAJO.

- column-reverse

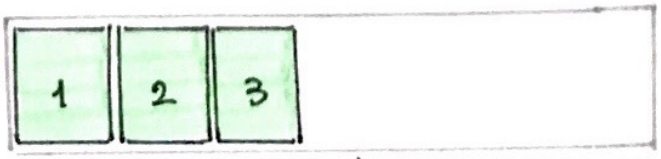
```
.container {
    display: flex;
    flex-direction: column-reverse;
}
```



APILA VERTICALMENTE DE ABAJO HACIA ARRIBA.

- row:

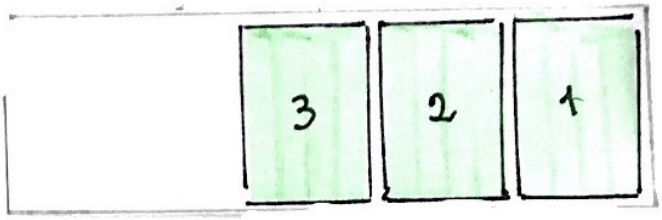
```
.container {
  display: flex;
  flex-direction: row;
}
```



APILA LOS ITEMS HORIZONTALMENTE, DE IZQUIERDA A DERECHA.
ES EL VALOR POR DEFECTO.

- row-reverse:

```
.container {
  display: flex;
  flex-direction: row-reverse;
}
```



APILA LOS ITEMS HORIZONTALMENTE DE DERECHA A IZQUIERDA.

- Propiedad flex-wrap: especifica si los items son obligados a permanecer en una misma línea o pueden fluir en varias líneas.

  valores:

  - nowrap: especifica que los flex item son distribuidos en una sola línea.

  - wrap: especifica que los flex items son colocados en varias líneas. cuando los items sobrepasan el ancho de su contenedor se genera un 'quiebre' para que se continuen apilando por debajo, en varias líneas.

  - wrap-reverse: especifica que los flex items serán colocados en varias líneas pero en orden inverso.

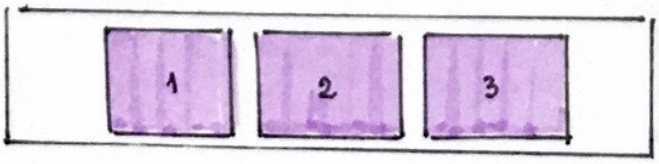
- La propiedad FLEX-FLOW establece las propiedades FLEX-DIRECTION y FLEX-WRAP juntas.

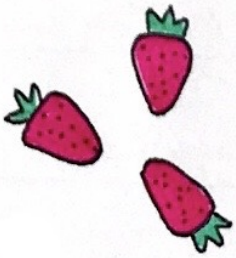
```
Flex-flow: row wrap;
```

- **Justify-content**: se utiliza para alinear flex-items.

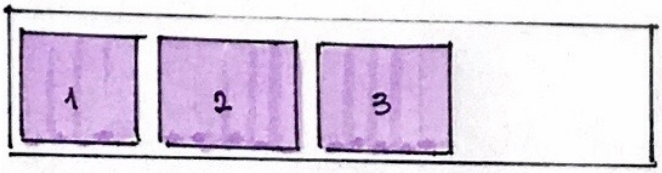
valores: center, flex-start, flex-end, space-around, space-between.

▲ CENTER: alinea los elementos en el centro del contenedor.

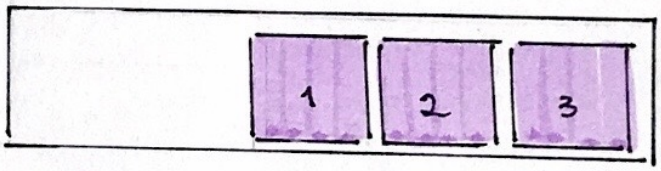




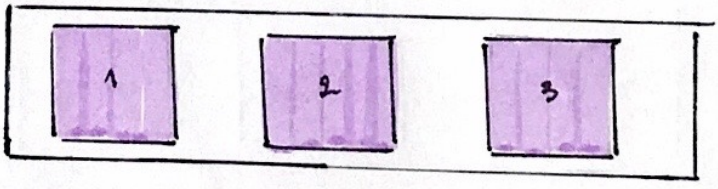
▲ flex-start: alinea los elementos al primeipro del contenedor. Es el valor por defecto.



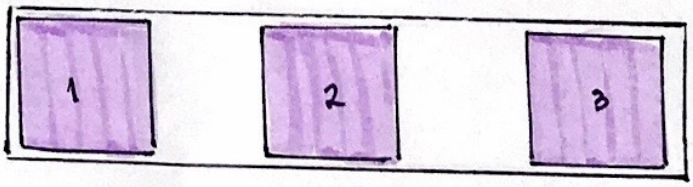
▲ flex-end: alinea los elementos al final del contenedor.



▲ space-around: muestra los elementos con espacio antes, entre y después de las líneas.



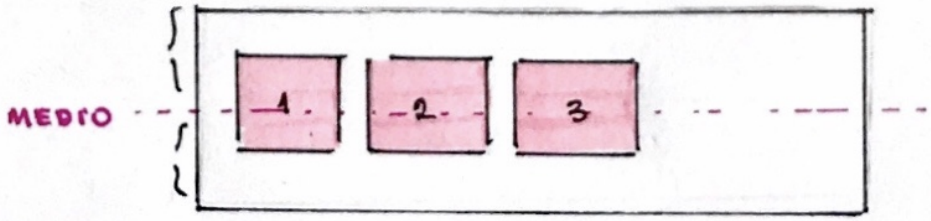
▲ space-between: muestra los elementos con espacio entre las líneas.



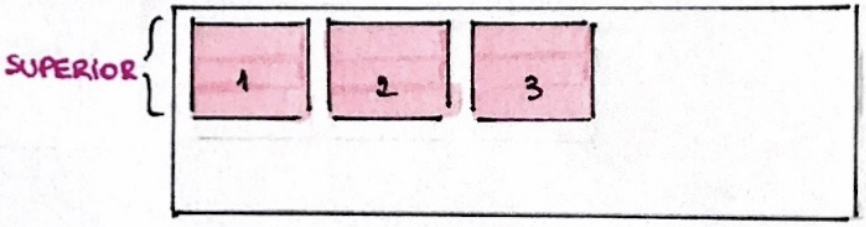
♥ Align-items: se utiliza para alinear flex-items.

valores: center, flex-start, flex-end, stretch, baseline.

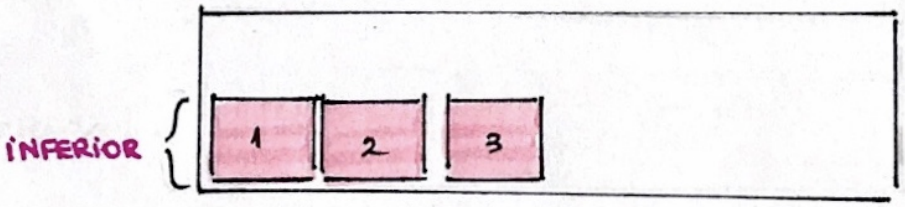
△ center: alinea los elementos en el medio del contenedor. (IDEAL PARA CENTRADO VERTICAL ♥)



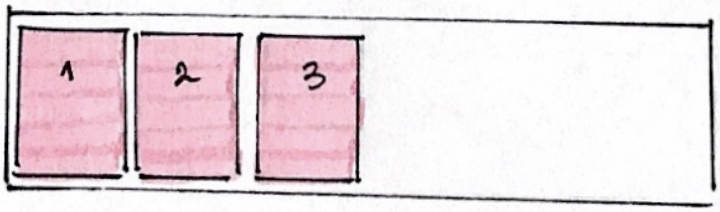
△ Flex-start: alinea los elementos en la parte superior del contenedor.



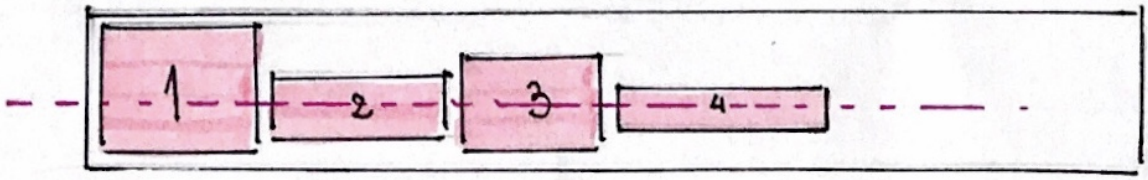
△ Flex-end: alinea los elementos en la parte inferior del contenedor.



△ stretch: estira los elementos para llenar el contenedor.



△ baseline: alinea los elementos, como sus líneas de base se alinean.



- Align-content: ajusta las líneas dentro de un contenedor Flex.

Valores: Flex-start, Flex-end, center, space-between, space-around, stretch.

▲ Flex-start: Las líneas son ajustadas a partir del eje transversal. El borde transversal de inicio de la primera línea y el del contenedor Flex quedan unidos.

▲ Flex-End: Las líneas son ajustadas d partir del final del eje transversal. El borde transversal final de la última línea y el del contenedor Flex quedan unidos.

▲ center: Las líneas son ajustadas hacia el centro del contenedor Flex. Las líneas son unidas entre sí, y centradas dentro del contenedor.

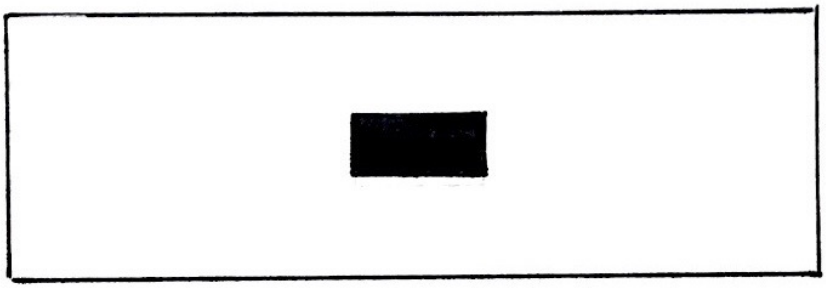
▲ space-between: Las líneas son distribuidas de manera uniforme en el contenedor Flex. El espaciado entre cualquier par de elementos adyacentes es el mismo. Los bordes transversales de inicio y de fin del contenedor son unidos a los bordes de la primera y última línea.

▲ space-around: Las líneas son distribuidas uniformemente de modo que el espacio entre cualquier par de elementos adyacentes sea el mismo. El espacio vacío antes de la primera línea y el espacio después de la última es igual a la mitad del espacio entre los elementos.

▲ stretch: Las líneas son estiradas para usar el espacio sobrante.

centrado perfecto

¿Cómo lo logramos?



El contenedor padre estará definido con la propiedad `display: flex;`. Luego, establecemos las propiedades:

```
justify-content: center;
align-items: center;
```

Y voilá!

¡Es tan fácil centrar verticalmente en CSS utilizando flex!

- **Flex-grow**: ESPECIFICARÁ CUANTO CRECERÁ UN ELEMENTO FLEX EN RELACIÓN CON EL RESTO DE ELEMENTOS FLEXIBLES.
  El valor debe ser un número.
  El valor predeterminado es cero.

```
<div class="container">
  <div style="flex-grow: 1"> 1 </div>
  <div style="flex-grow: 1"> 2 </div>
  <div style="flex-grow: 8"> 3 </div>
</div>
```

¡EL TERCER ELEMENTO FLEXIBLE CRECERÁ 8 VECES MÁS RÁPIDO!

- **Flex-shrink**: ESPECIFICA CUÁNTO SE ENCOGERÁ UN ELEMENTO FLEXIBLE EN RELACIÓN CON EL RESTO DE LOS ELEMENTOS FLEXIBLES.
  El valor debe ser un número.
  El valor por defecto es uno.

```
<div class="container">
  <div> 1 </div>
  <div> 2 </div>
  <div> 3 </div>
  <div style="flex-shrink: 0"> 4 </div>
</div>
```

¡ESTE ELEMENTO NO SE ENCOGERÁ TANTO COMO LOS DEMÁS!

- **Flex-basis**: ESPECIFICA la longitud inicial de un elemento flexible.

```
<div class="container">
  <div> 1 </div>
  <div style="flex-basis: 200px"> 2 </div>
</div>
```

LA LONGITUD INICIAL DEL ELEMENTO 2 SERÁ DE 200px

CSS grid

CSS grid ofrece un sistema de diseño basado en cuadrículas, con filas y columnas, lo cual facilita el diseño sin tener que usar posicionamiento.

Debemos comenzar definiendo el display: grid para convertir el contenedor en una grilla.

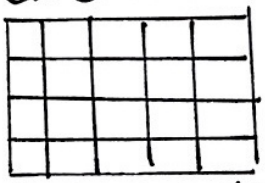
* Todos los hijos directos del contenedor de grilla se convierten automáticamente en elementos de grilla.

Hay dos tipos de 'grids':

- display: grid;
- display: inline-grid;

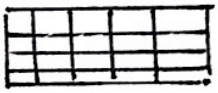
display: grid;

Esta 'grid box' se coloca en el medio

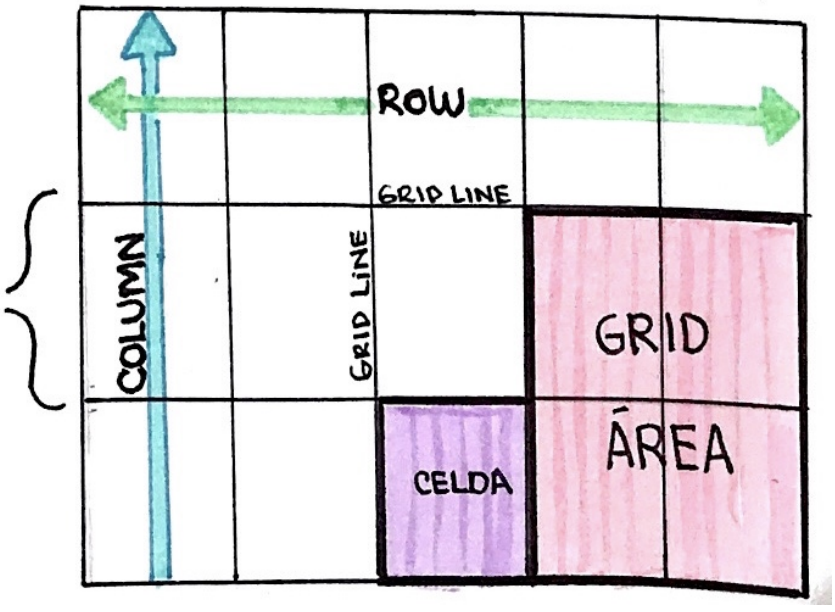


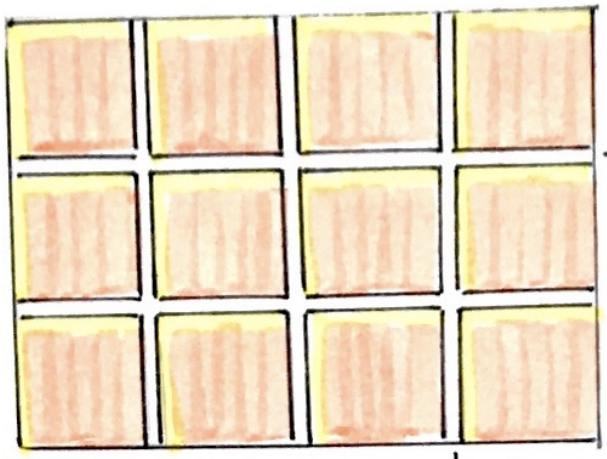
de una sentencia. Genera un 'block box' rompiendo el flujo del contenido.

display: inline-grid;

En cambio, esta 'grid box' está tambien en el medio de  una sentencia, pero genera un 'inline box'.

COMPONENTES DE GRID





→ LOS ESPACIOS ENTRE FILAS SE LLAMAN ROW GAP

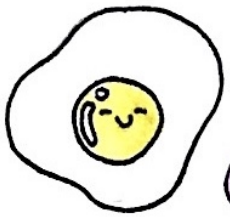
LOS ESPACIOS ENTRE COLUMNAS SE LLAMAN COLUMN GAP

LA PROPIEDAD GRID-GAP ES UNA ABREVIATURA DE AMBAS.

```
grid-gap: 50px 2px;
```

✱ SI EL ESPACIO ENTRE FILAS Y EL ESPACIO ENTRE COLUMNAS SERÁ EL MISMO, TAMBIÉN USAMOS grid-gap:

```
grid-gap: 50px;
```



grid-template-rows
grid-template-columns

♥ **GRID-TEMPLATE-COLUMNS**: ESTA PROPIEDAD DEFINE EL NÚMERO DE COLUMNAS EN LA GRILLA, Y ADEMÁS, PUEDE DEFINIR EL ANCHO DE CADA COLUMNA.

LOS VALORES SE SEPARAN POR ESPACIOS, DONDE CADA VALOR DEFINE EL ANCHO DE LA COLUMNA RESPECTIVA.

Si quisiéramos 4 columnas:

```
grid-template-columns: 20px 10px 10px 20px;
```

Si las 4 columnas deben tener el mismo ancho, podemos especificar:

grid-template-columns: auto auto auto auto;

- **GRID-TEMPLATE-ROWS:** ESTA PROPIEDAD DEFINE LA ALTURA DE CADA FILA. EL VALOR ES UNA LISTA SEPARADA POR ESPACIOS, DONDE CADA VALOR DEFINE LA ALTURA DE LA FILA RESPECTIVA.

grid-template-rows: 80px 200px;
↓ Fila1 ↓ Fila2

- Valores para grid-template-column y grid-column-row:
  - none: no hay cuadrícula explícita.
  - <length>: longitud no negativa (20px)
  - porcentaje: valor no negativo relativo al tamaño del contenedor grid.
  - auto: palabra clave que es idéntica al contenido máximo si es un máximo. como un mínimo representa al mínimo más grande.
  - repeat(): cuando los valores que definimos son repetidos.

"grid-template-columns: repeat(3, 20px);
- 3 → cantidad de columnas
- 20px → cuanto medirá cada columna

grid-template-areas

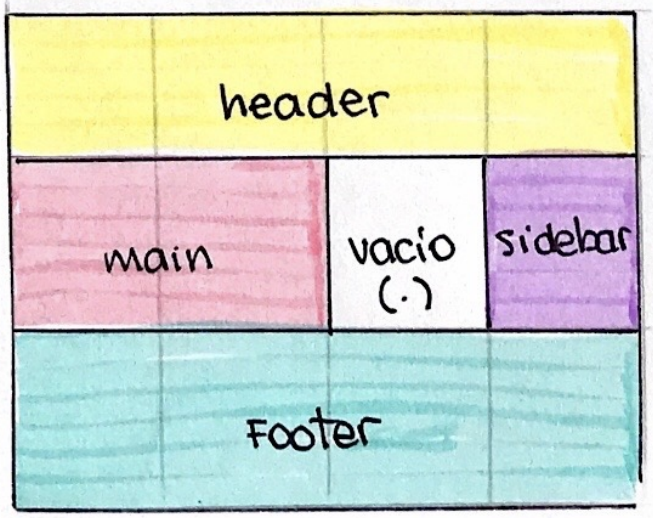
Esta propiedad especifica nombres para cada una de las grid áreas.

valores:

- none : no hay grid áreas definidas.
- nombre de grid área: el nombre especificado con grid-area.
- . (punto): significa una celda vacía.

```
.container {
  grid-template-areas:
    " | . | none | ... "
    "..." ;
}
```

```
.item-a { grid-area: header; }
.item-b { grid-area: main; }
.item-c { grid-area: sidebar; }
.item-d { grid-area: footer; }
```



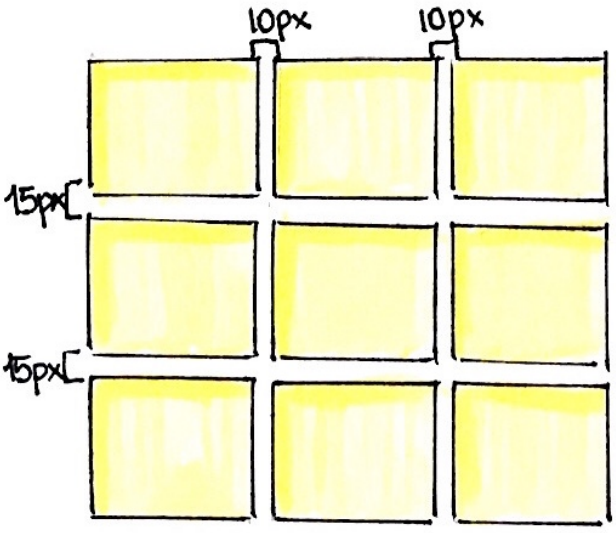

```
.container {
  display: grid;
  grid-template-columns: repeat(4,50px);
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer"
}
```

"grid-template" es una abreviatura de grid-template-rows, grid-template-columns y grid-template-areas.

column-gap
row-gap

} ESPECIFICAN EL TAMAÑO DE LAS LÍNEAS DE LA CUADRÍCULA/GRILLA. ES DECIR, EL ESPACIADO ENTRE COLUMNAS Y ENTRE FILAS.

```
.container {
grid-template-columns: 100px 50px 100px;
grid-template-rows: 80px auto 80px;
column-gap: 10px;
row-gap: 15px;
}
```



grid-gap

Es una abreviatura de column-gap y row-gap en una sola declaración.

sintaxis:

```
grid-gap: "row-gap" "column-gap"
```

propiedades de grid-items

- grid-column-start : línea donde la columna comienza
- grid-column-end : línea donde la columna termina
- grid-row-start : línea donde la fila comienza
- grid-row-end : línea donde la fila termina

#Ejemplo:

```
.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

grid-column

ES UNA PROPIEDAD ABREVIADA DE grid-column-start y grid-column-end.

```
grid-column: 1 / 3;
```

1 → EMPIEZA EN 1

3 → TERMINA EN 3

#EJEMPLO:

```
.item1 {
  grid-row-start: 1;
  grid-row-end: 4;
}
```

grid-row

ES UNA PROPIEDAD ABREVIADA DE grid-row-start y grid-row-end

```
grid-row: 1 / 4;
```

el valor 'span' define la cantidad de columnas o filas que abarcará:

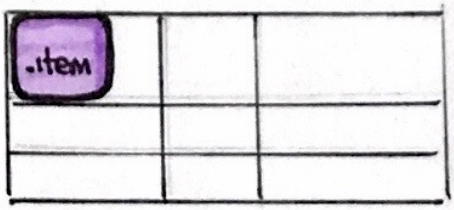
```
grid-column: 1 / span 2;
```

- 1 → comienza en la columna 1.
- span 2 → abarca 2 columnas

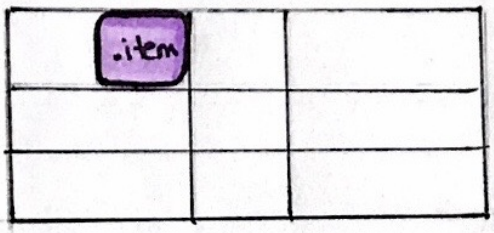
✦ Justify-self

Alinea un grid-item dentro de una celda a lo largo de la Fila.

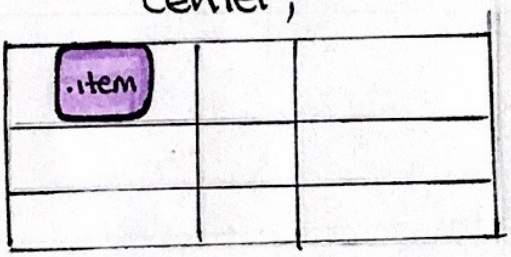
start;



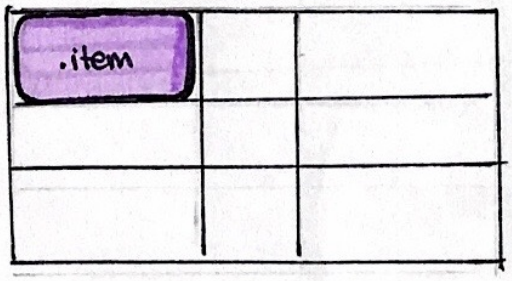
end;



center;



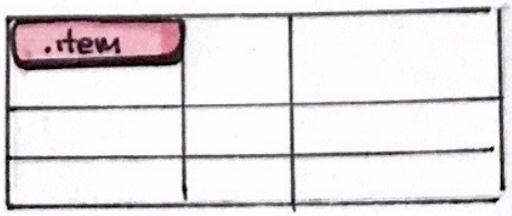
stretch;



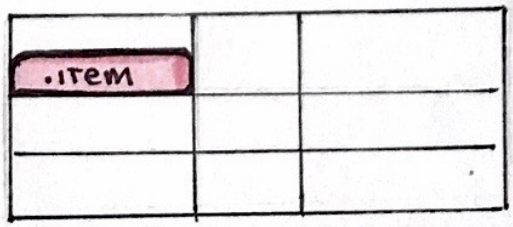
✦ align-self

Alinea un grid-item dentro de una celda a lo largo de la columna.

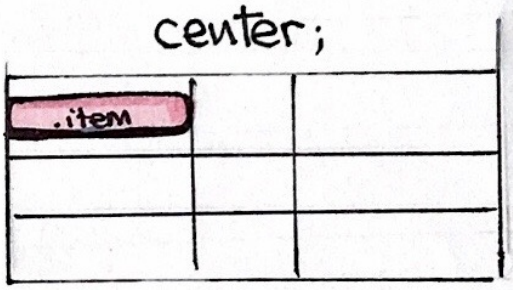
start;



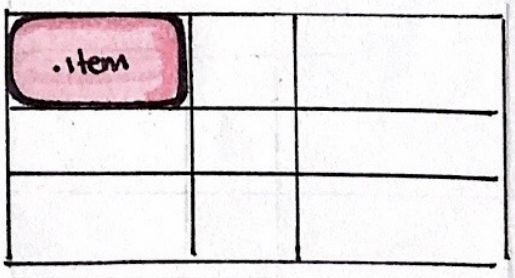
end;



center;



stretch;



ANIMACIONES

CSS permite la animación de elementos HTML.

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

@keyframes

Para animar un elemento, necesitamos establecer el nombre y crear una animación reutilizable mediante la regla @keyframes.

El nombre que definamos se utilizará para adjuntar la animación a un elemento, o pseudo-elemento.

nombre que le colocamos

```
@keyframes ejemplo {
    from {
        opacity: 1;
    }
    to {
        opacity: 0;
    }
}
```

En el ejemplo anterior, cambiamos la opacidad usando las palabras clave from y to (que representa 0% (inicio) y 100% (completo)).

También es posible usar porcentajes. Por ejemplo, cambiamos el color de fondo de un elemento, cuando la animación esté 25% completa, 50% completa y 100% completa:

```
@keyframes ejemplo {
  0% {background-color: black;}
  25% {background-color: grey;}
  50% {background-color: green;}
  100% {background-color: yellow;}
```

Luego, aplicación la animación al elemento:

```
div {
    width: 100px;
    height: 100px;
    background-color: black;
    animation-name: ejemplo;
    animation-duration: 2s;
}
```

→ DEFINE CUANTO TIEMPO TARDARÁ LA ANIMACIÓN EN COMPLETARSE.

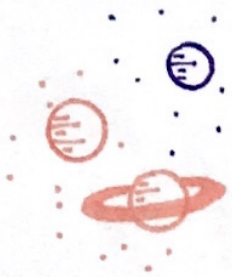
⚠ SI NO SE ESPECIFICA LA PROPIEDAD animation-duration NO SE PRODUCIRÁ NINGUNA ANIMACIÓN PORQUE SU VALOR POR DEFECTO ES CERO.

▵ **animation-delay**: especifica un retraso para el inicio de una animación. (También permite valores negativos)

▵ **animation-iteration-count**: especifica el número de veces que debe ejecutarse una animación. (El valor también puede ser infinito)

▵ **animation-direction**: especifica si una animación debe reproducirse hacia adelante, hacia atrás o en ciclos alternos.

- normal: La animación se reproduce hacia adelante. Es el valor por defecto.
- reverse: La animación se produce en sentido inverso.
- alternate: La animación se reproduce primero hacia adelante y luego hacia atrás.
- alternate-reverse: La animación se reproduce primero hacia atrás y luego hacia adelante.



▵ **animation-timing-function**: especifica la velocidad de la animación.

- ease: inicio lento, luego rápido y termina lento. Es el valor por defecto.
- linear: misma velocidad de principio a fin.
- ease-in: inicio lento.
- ease-out: Final lento.
- ease-in-out: inicio y final lentos.

Utilizando la propiedad abreviada animation podemos usar las propiedades de animación juntas ☺