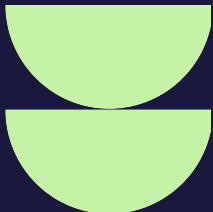




# M5

GRID & FLEX



**SENPAI**  
academy

# *Posicionamiento y visibilidad*



# ***Propiedad display***

# Inline o block

Los elementos HTML se dividen en dos grandes tipos:

→ **Block** que tienden a ocupar el espacio disponible a todo lo ancho y en caso de existir varios se sitúan unos debajo de otros.

→ **Inline** que ocupan el espacio necesario dentro de una línea y en caso de existir varios se sitúan uno junto a otro en la misma línea (siempre que haya espacio).

## Ejemplos de tags HTML block

- `<div>`
- `<h1>` - `<h6>`
- `<p>`
- `<form>`
- `<header>`
- `<footer>`
- `<section>`

## Ejemplos de tags HTML inline

- `<span>`
- `<a>`
- `<img>`

# Valores que puede tomar display

Valores posibles para esta propiedad

inline (el elemento se muestra en una caja inline)

block (el elemento se muestra en una caja block).

none (el elemento no se muestra; el efecto es como si no existiera, por lo que su espacio será ocupado por otros elementos)

list-item (el elemento se comporta como si fuera un elemento li)

inline-block (el elemento genera una caja block pero que se comporta como si fuera inline admitiendo otros elementos en la misma línea; el comportamiento se asemeja al de los elementos img)

Otros que llevan a que el elemento simule el comportamiento de otro (inline-table, table, table-caption, table-cell, table-column, table-column-group, table-footer-group, table-header-group, table-row, table-row-group)

Otros avanzados (flex, inline-flex, grid, inline-grid, run-in)

inherit (se heredan las características del elemento padre).

***Además de las dimensiones, las márgenes y los bordes, los elementos pueden posicionarse de otras maneras.***

# ***Propiedad position***

## Posición *static*

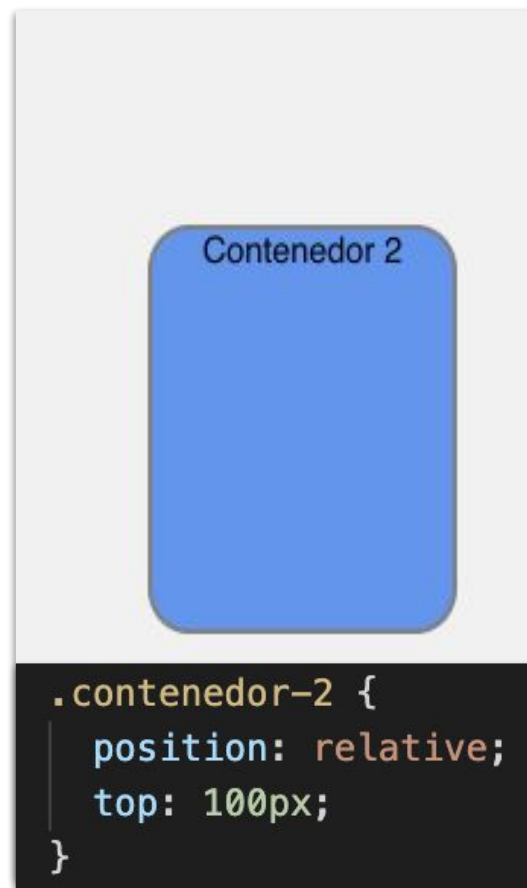
Un elemento con **position: static** se comporta de manera “natural” respecto al resto del contenido de la página. Éste valor es el valor por defecto para la propiedad **position**.





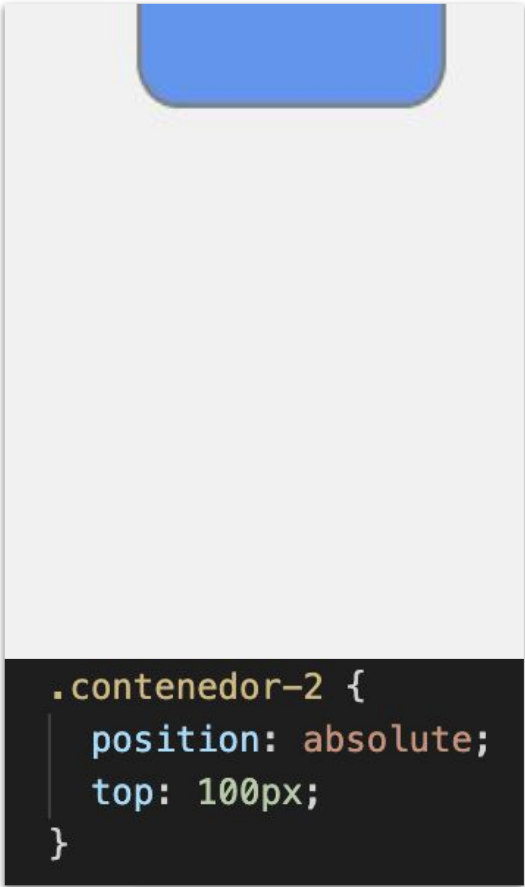
## Posición *relative*

Al igual que con el valor **static**, un elemento con **position: relative** se comporta de manera “natural” respecto al resto del contenido de la página. La principal diferencia, es que un elemento con posición relativa se considera un elemento **posicionado**.



## Posición *absolute*

Un elemento con **position: absolute** se ubica en relación con su contenedor ancestro **posicionado** más cercano sin ocupar espacio real.  
Su posición final está determinada por los valores de **top, right, bottom y left**.

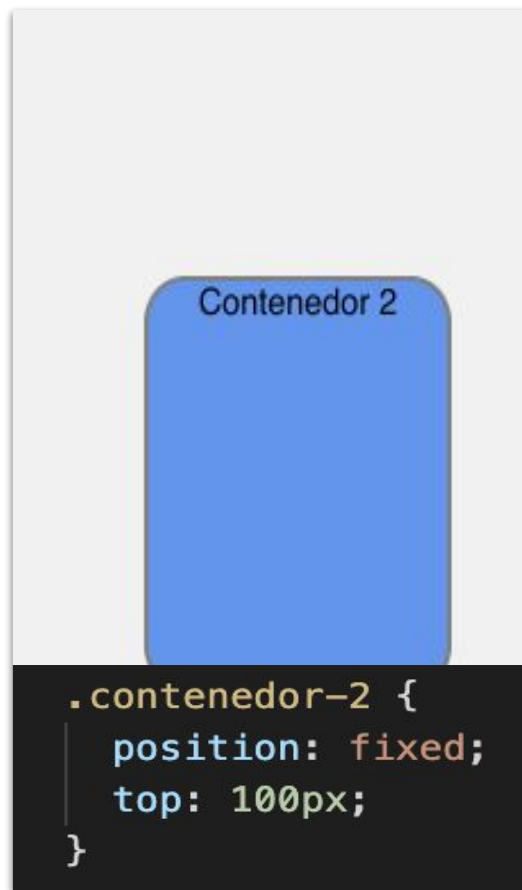
A diagram illustrating absolute positioning. It features a light gray rectangular container with a blue rounded rectangle at the top. Below the container, a dark gray box contains CSS code. The code defines a class '.contenedor-2' with 'position: absolute;' and 'top: 100px;'.

```
.contenedor-2 {  
  position: absolute;  
  top: 100px;  
}
```

## Posición *fixed*

Un elemento con **position: fixed** se mantendrá en el espacio asignado, siempre en una misma posición incluso cuando el usuario se desplace en la pantalla.


Igual que en **absolute**, su posición final está determinada por los valores de **top**, **right**, **bottom** y **left**.



## Posición *sticky*

Un elemento con `position: sticky` es un híbrido de posicionamiento `relative` y `fixed`.

El elemento se trata como una posición relativa hasta que cruza un umbral especificado, en cuyo punto se trata como una posición fija.

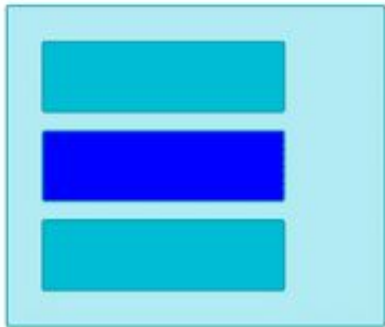


```
.contenedor-2 {  
  position: sticky;  
  top: 0;  
}
```

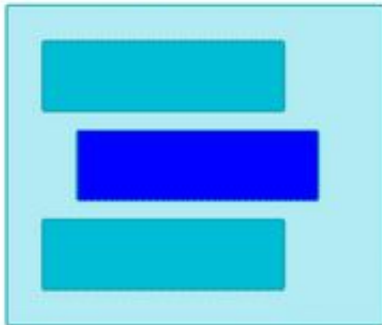
## Comparando las posiciones

En estos tres ejemplos podemos ver visualmente las diferencias entre tres valores que puede tomar la propiedad **position**.

`position: static`



`position: relative`  
`left: 20px`



`position: absolute`  
`bottom: 0; right: 0`



# *CSS - Grid*



# CSS - Grid

nos permite diagramar una página (o partes de ella) como si fuera una grilla.

- No es un reemplazo de Flexbox (Flex), pero puede lograr muchas de las mismas cosas.
- Con Grid podemos definir filas, columnas e incluso áreas.
- Además de esto, podemos nombrar regiones o colocar elementos declarativamente.
- Para empezar a utilizar Grid, sólo tenemos que especificarlo de la siguiente manera:

```
.contenedor {  
    display: grid;  
}
```

Más información sobre Grid: [link 1](#), [link 2](#), [link 3](#)

# ¿Qué hay en un contenedor Grid?



**Grid:** la grilla completa.



**Grid lines:** son las líneas verticales u horizontales que dividen los tracks.



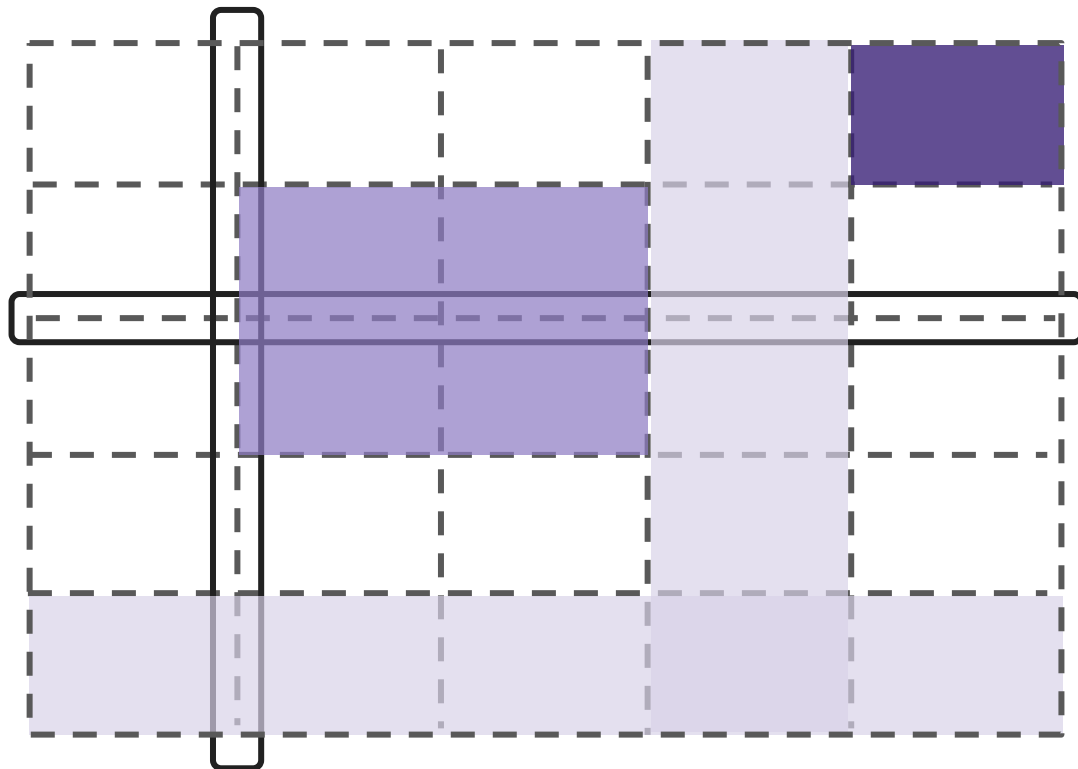
**Grid Tracks:** filas o columnas completas.



**Grid Areas:** conjunto rectangular de celdas



**Grid Cell:** una única celda.





# *Ejemplos*

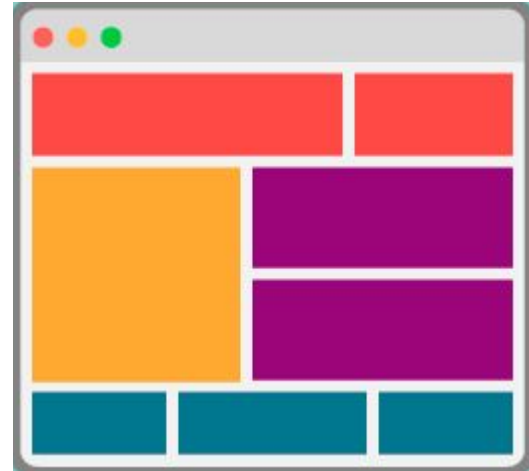
## *CSS - Grid*



## CSS · Ejemplos

Con CSS Grid se pueden hacer muchas cosas, estos son algunos ejemplos:

- <https://codepen.io/oliviale/pen/mgWjpg>
- <https://codepen.io/andybarefoot/pen/PBPrex>
- <https://codepen.io/andybarefoot/pen/MWgvLvq>
- <https://codepen.io/oliviale/details/ZmvPPd>
- <https://codepen.io/oliviale/pen/WqwOzv>
- <https://codepen.io/oliviale/pen/MZZYyO>



## CSS Grid · Filas y columnas

Podemos crear filas y columnas de manera explícita utilizando `grid-template-rows` y `grid-template-columns`.

```
.contenedor {  
  display: grid;  
  /* Valores absolutos */  
  grid-template-rows: 50px 150px; /* Dos filas con tamaños distintos*/  
  grid-template-columns: 100px 100px; /* Dos columnas de 100px */  
}
```

Existen más formas de definir filas y columnas, revisar: [columnas](#), [filas](#)

```
.contenedor {  
  display: grid;  
  /* Fracciones */  
  grid-template-rows: 1fr 1fr; /* Dos filas que ocupan el 50% del espacio disponible */  
  grid-template-columns: 3fr 1fr; /* Dos columnas, la primera mide el triple que la segunda */  
}
```

## CSS Grid · Separar columnas y filas (gap)

Podemos separar celdas utilizando `gap`, `column-gap` y/o `row-gap`. Si utilizamos `gap`, podemos definir la distancia entre filas y columnas en una única declaración.

Algunos ejemplos:

```
.contenedor {  
  display: grid;  
  row-gap: 15px;  
}
```

```
.contenedor {  
  display: grid;  
  /* Primero fila luego columna */  
  gap: 35px 15px;  
}
```

```
.contenedor {  
  display: grid;  
  column-gap: 15px;  
}
```

```
.contenedor {  
  display: grid;  
  /* Ambas distancias iguales */  
  gap: 35px;  
}
```

## CSS Grid · Crear áreas o secciones

Podemos nombrar filas y columnas utilizando **grid-template-areas** y luego asignar elementos a esas "áreas" utilizando **grid-area**.

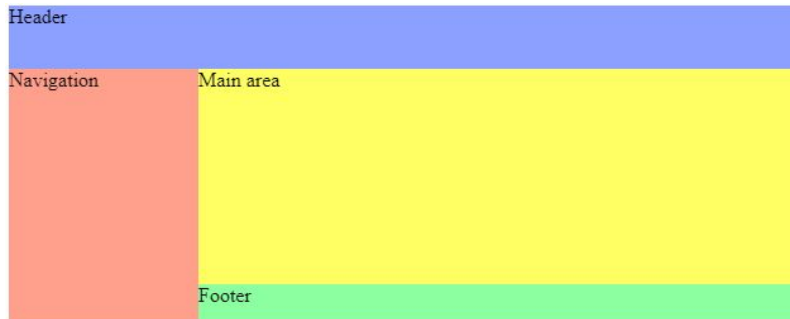
```
#page {
  display: grid;
  width: 100%;
  height: 250px;
  grid-template-areas: "head head"
                      "nav main"
                      "nav footer";
  grid-template-rows: 50px 1fr 30px;
  grid-template-columns: 150px 1fr;
}

header {
  grid-area: head;
  background-color: #8ca0ff;
}

nav {
  grid-area: nav;
  background-color: #ffa08c;
}

main {
  grid-area: main;
  background-color: #ffff64;
}

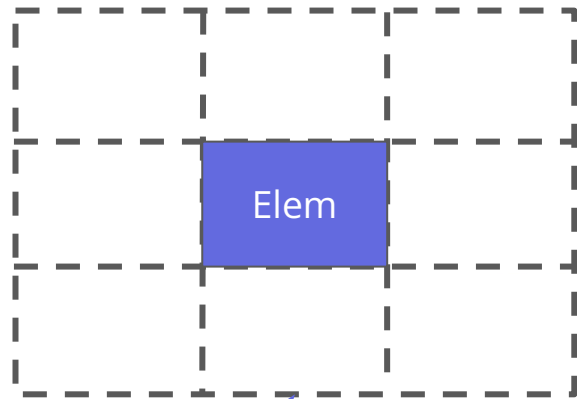
footer {
  grid-area: footer;
  background-color: #8cffa0;
}
```



## CSS Grid · Posicionar elementos por columna y fila

Además de utilizando “áreas”, podemos posicionar un elemento utilizando `grid-column` y `grid-row`.

```
.contenedor {  
  display: grid;  
  grid-template-rows: repeat(3, 1fr);  
  grid-template-columns: repeat(3, 1fr);  
}  
  
.elemento {  
  grid-column: 2;  
  grid-row: 2;  
}
```

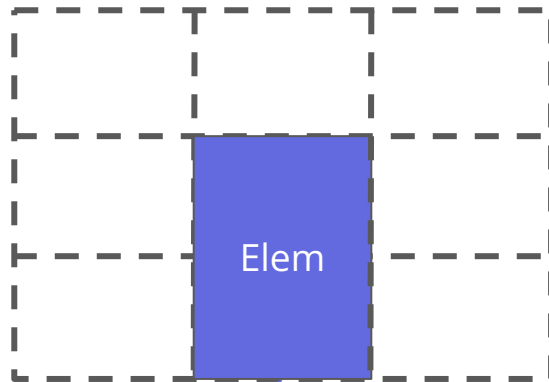


## CSS Grid · Posicionar elementos por columna y fila

Podemos también definir cuántas filas o columnas debe abarcar utilizando la palabra clave **span**.

En el siguiente ejemplo, el elemento está siendo ubicado en la segunda fila y abarcará 2 filas.

```
.contenedor {  
  display: grid;  
  grid-template-rows: repeat(3, 1fr);  
  grid-template-columns: repeat(3, 1fr);  
}  
  
.elemento {  
  grid-column: 2;  
  grid-row: 2 / span 2;  
}
```

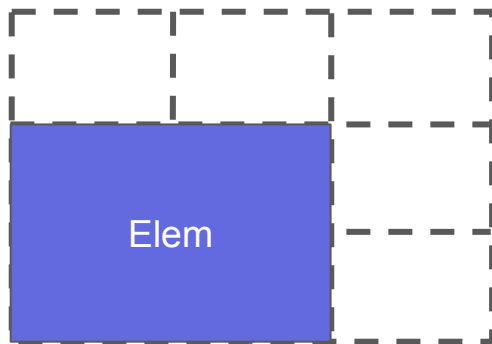


## CSS Grid · Posicionar elementos por columna y fila

Además de utilizando “áreas”, podemos posicionar un elemento utilizando `grid-column` y `grid-row`.

En el siguiente ejemplo sólo especificamos la cantidad de columnas que abarca el elemento, pero no especificamos en cual comienza.

```
.contenedor {  
  display: grid;  
  grid-template-rows: repeat(3, 1fr);  
  grid-template-columns: repeat(3, 1fr);  
}  
  
.elemento {  
  grid-column: span 2;  
  grid-row: 2 / span 2;  
}
```





## CSS Grid · Posicionar elementos por columna y fila

La función `repeat()` nos facilita la creación de varias filas o columnas del mismo tamaño.

```
.contenedor {  
  display: grid;  
  /* Tres columnas iguales: */  
  grid-template-columns: 1fr 1fr 1fr;  
  /* Utilizando repeat(): */  
  grid-template-columns: repeat(3, 1fr);  
}
```

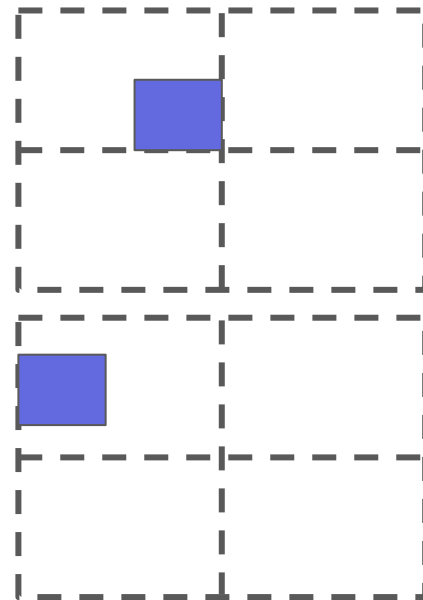
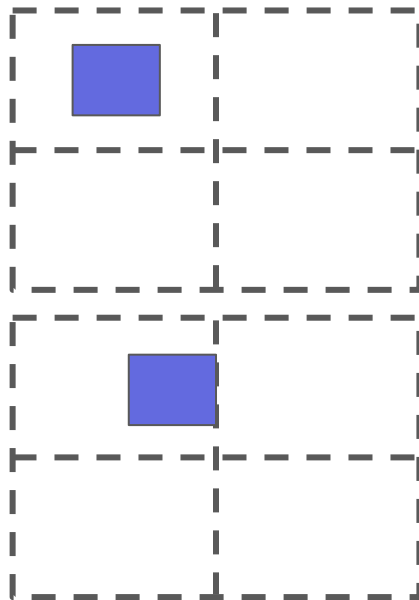
Hay más formas de utilizar la función `repeat`, revisar: [función repeat\(\)](#)



## CSS Grid · Alinear elementos

En Grid podemos utilizar `justify-content`, `justify-items`, `align-content` y `align-items` para cambiar la alineación.

`justify-items` y `align-items` alinean los elementos dentro de sus respectivas celdas.



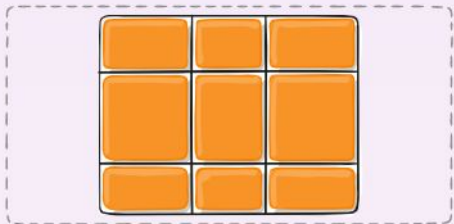
## CSS Grid · Alinear elementos

Al igual que en flex, podemos alinear y justificar elementos. En Grid podemos utilizar `justify-content`, `justify-items`, `align-content` y `align-items`.

`justify-content` y `align-content` alinean la grilla dentro del contenedor (sólo si no ocupa el 100%).

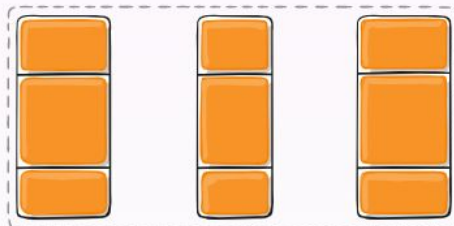
```
.container {  
  justify-content: center;  
}
```

grid container



```
.container {  
  justify-content: space-between;  
}
```

grid container



# *CSS - Flexbox*



**“Flexbox fue diseñado como un modelo unidimensional de layout, y como un método que pueda ayudar a distribuir el espacio entre los ítems de una interfaz y mejorar las capacidades de alineación.”**

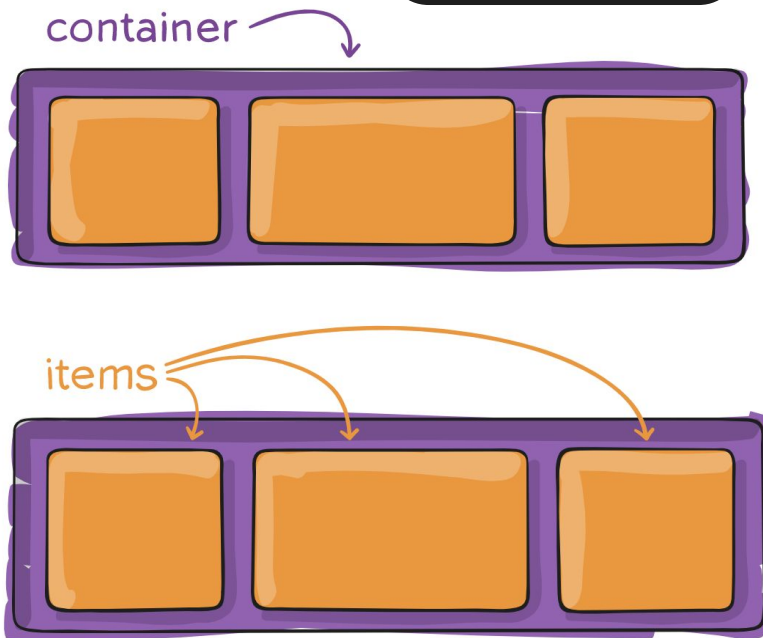


# Primeros Pasos

Aquí tenemos un **contenedor** y varios **ítems**. A ambos le podemos aplicar la propiedad **flex**.

Existe diferentes propiedades para cada uno, tanto para el **ítem flex** como para el **contenedor flex**.

```
.contenedor {  
  display: flex;  
  width: 250px;  
  height: 80;  
}
```



# Container vs Item flex

## Container flex

- `display`
- `flex-direction`
- `flex-wrap`
- `justify-content`
- `align-items`
- `align-content`

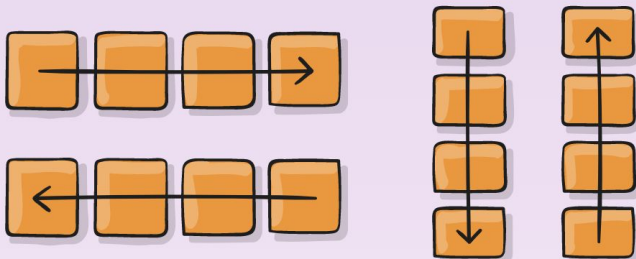
## Item flex

- `order`
- `flex-grow`
- `align-self`
- `flex`

# Direccionando el contenido en un contenedor flex

```
.contenedor {  
  display: flex;  
  flex-direction: column | column-reverse | row | row-reverse;  
}
```

## flex-direction



**Valores posibles:** column, column-reverse, row, row-reverse.

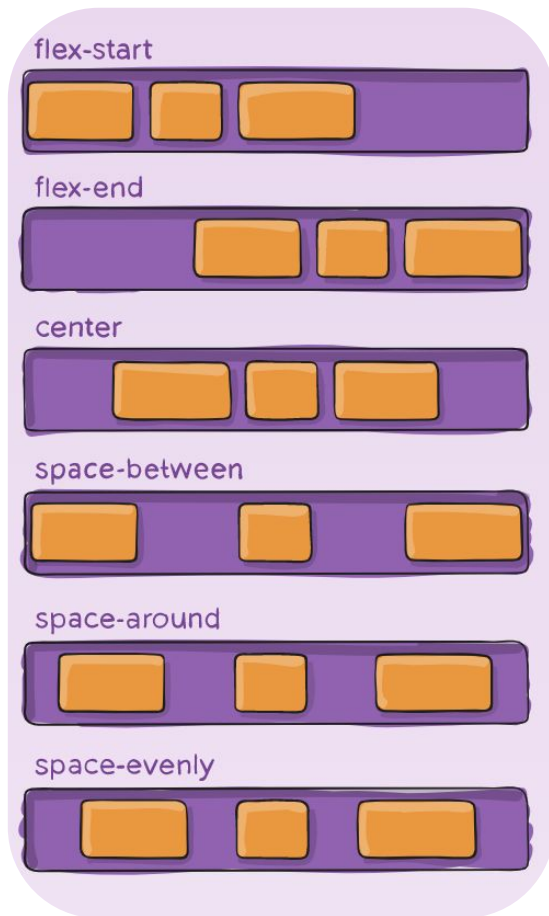


# Distribuyendo elementos en un contenedor flex



```
.container {  
  justify-content: flex-start | flex-end | center | space-between ...;  
}
```

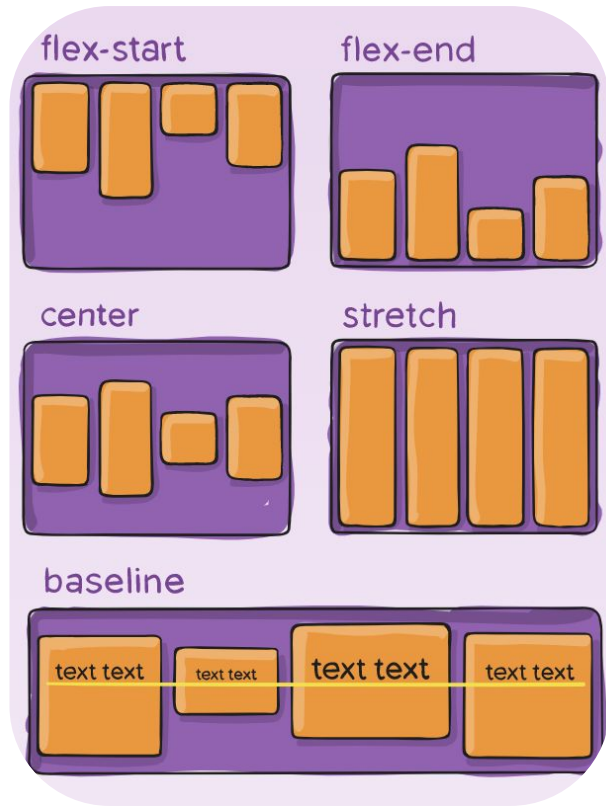
**Valores posibles:** flex-start, flex-end, start, end, left, right, center, space-between, space-around y space-evenly.



# Alineando elementos dentro de un contenedor flex

```
.container {  
  align-items: flex-start | flex-end | center | baseline | start | end ...;  
}
```

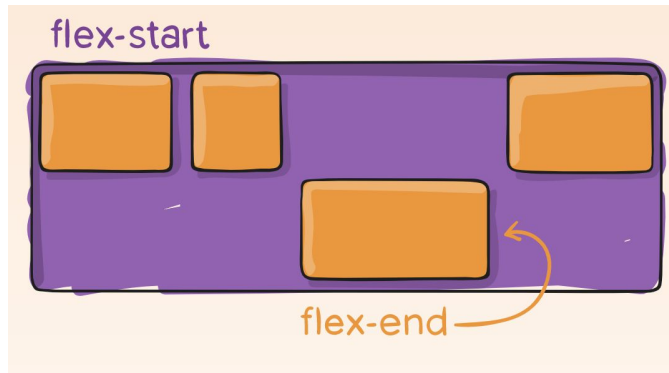
Valores posibles: flex-start, flex-end, center, stretch y baseline.



# Alineando un único elemento dentro de un contenedor flex

```
.item {  
  align-self: auto | flex-start | flex-end | center | baseline |  
}
```

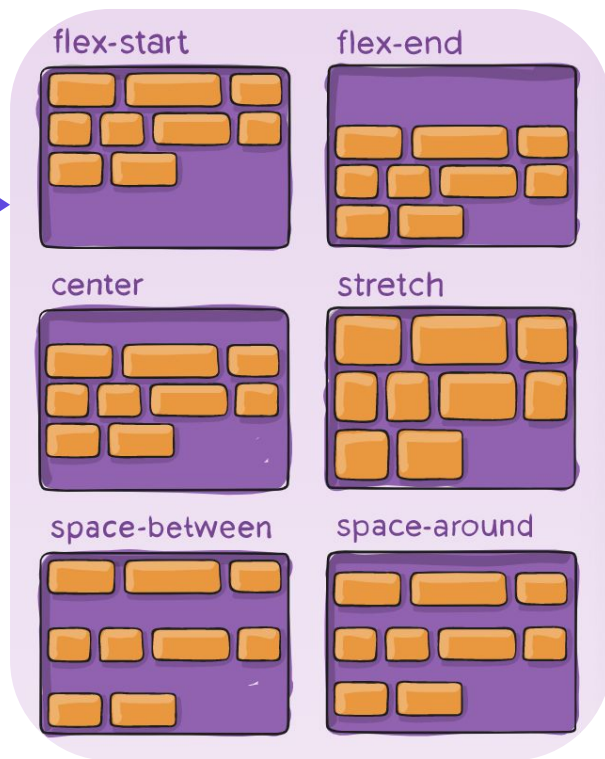
**Valores posibles:** flex-start, flex-end, center, stretch y baseline.



# ¿Cómo alineamos nuestro contenido en un contenedor flex?

```
.container {  
  align-content: flex-start | flex-end | center  
}
```

**Valores posibles:** flex-start, flex-end, center, stretch, space-between, space-around y space-evenly..





**¡GRACIAS!**