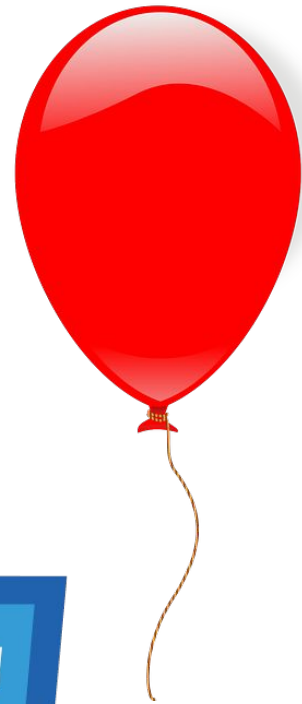




CSS: position, display, float y flexbox



FLEXBOX

FLEXBOX EVERYWHERE

made on imgur



Position... no dónde sino cómo

La propiedad `position` de [CSS](#) especifica cómo un elemento es posicionado en el documento. Las propiedades [top](#), [right](#), [bottom](#), y [left](#) determinan la ubicación final de los elementos posicionados.



Tipos de posicionamiento

- Un **elemento posicionado** es un elemento cuyo valor `computado` de `position` es `relative`, `absolute`, `fixed`, o `sticky`. (En otras palabras, cualquiera excepto `static`).
- Un **elemento posicionado relativamente** es un elemento cuyo valor `computado` de `position` es `relative`. Las propiedades `top` y `bottom` especifican el desplazamiento vertical desde su posición original; las propiedades `left` y `right` especifican su desplazamiento horizontal.
- Un **elemento posicionado absolutamente** es un elemento cuyo valor `computado` de `position` es `absolute` o `fixed`. Las propiedades `top`, `right`, `bottom`, y `left` especifican el desplazamiento desde los bordes del `bloque contenedor` del elemento. (El bloque contenedor es el ancestro relativo al cual el elemento está posicionado). Si el elemento tiene márgenes, se agregarán al desplazamiento. el elemento establece un nuevo contexto de formato de bloque para su contenido
- Un **elemento posicionado fijamente** es un elemento cuyo valor de `position` `computado` es `sticky`. Es tratado como un elemento posicionado relativamente hasta que su `bloque contenedor` cruza un límite establecido (como por ejemplo dando a `top` cualquier valor distinto de `auto`), dentro de su flujo principal (o el contenedor dentro del cual se mueve), desde el cual es tratado como "fijo" hasta que alcance el borde opuesto de su `bloque contenedor`.



When you use



position: absolute;

imgflip.com



Posicionamiento relativo

Elementos posicionados relativamente son desplazados una cantidad dada de su posición normal en el documento, pero sin que su desplazamiento afecte a otros elementos. En el ejemplo siguiente, nótese cómo los demás elementos se ubican como si "Two" estuviera ocupando el lugar de su ubicación normal.



Posicionamiento absoluto

Los elementos posicionados relativamente se mantienen en el flujo normal del documento. Por el contrario, un elemento posicionado absolutamente es removido del flujo de esta manera, los demás elementos se posicionan como si el mismo no existiera. El elemento posicionado absolutamente se posiciona relativamente a su *ancestro posicionado más cercano* (es decir, el ancestro más cercano que no es `static`). Si no hay ningún ancestro posicionado se ubica relativo al bloque contenedor inicial. En el ejemplo siguiente, la caja "Two" no tiene un ancestro posicionado, por lo tanto se posiciona relativo al `<body>` del documento.



HTML

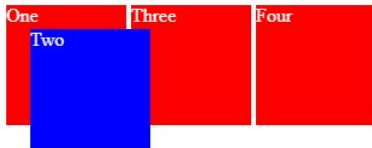
```
<div class="box" id="one">One</div>
<div class="box" id="two">Two</div>
<div class="box" id="three">Three</div>
<div class="box" id="four">Four</div>
```



CSS

```
.box {
  display: inline-block;
  width: 100px;
  height: 100px;
  background: red;
  color: white;
}

#two {
  position: absolute;
  top: 20px;
  left: 20px;
  background: blue;
}
```



Posicionamiento sticky

El posicionamiento sticky puede considerarse un híbrido de los posicionamientos relativo y fijo. Un elemento con posicionamiento sticky es tratado como un elemento posicionado relativamente hasta que cruza un umbral especificado, en cuyo punto se trata como fijo hasta que alcanza el límite de su padre. Por ejemplo...

```
#one { position: sticky; top: 10px; }
```



...posicionaría el elemento con id *uno* relativamente hasta que el viewport sea desplazado de manera tal que el elemento esté a menos de 10 píxeles del límite superior. Más allá de ese umbral, el elemento sería fijado a 10 píxeles del límite superior.

Dato de color sobre sticky: internet explorer no lo soporta, así que es una pesadilla. Se puede usar algún script para simular un sticky pero... siempre se ve horrible.



Bueno, pero antes a entender un poco qué es display

La propiedad CSS **display** especifica si un elemento es tratado como [block or inline element](#) y el diseño usado por sus hijos, como [flow layout](#)(Diseño de Flujo), [grid](#)(Cuadrícula) o [flex](#)(Flexible).

Además de los Diferentes Tipos de caja de Visualización, el valor de **none** permite Desactivar la Visualización DE UN Elemento; cuando se utiliza **none**, todos los elementos descendentes también quedan desactivados.



<display-outside>

Estas palabras clave especifican el tipo de pantalla externa del elemento, que es esencialmente su función en el diseño de flujo: A continuación se definen:

Valor	Descripción
<code>block</code>	El elemento genera un cuadro de elemento de bloque.
<code>inline</code>	El elemento genera uno o más cuadros de elemento en línea.



<display-inside>

Estas palabras clave especifican el tipo de pantalla interna del elemento, que define el tipo de contexto de formato que establece su contenido (suponiendo que es un elemento no reemplazado).


Se definen como sigue:

table	Estos elementos se comportan como elementos HTML <table> . Define un cuadro de nivel de bloque.
flex	El elemento se comporta como un elemento de bloque y establece su contenido de acuerdo con el modelo de flexbox .
grid	El elemento se comporta como un elemento de bloque y establece su contenido de acuerdo con el modelo de cuadrícula .



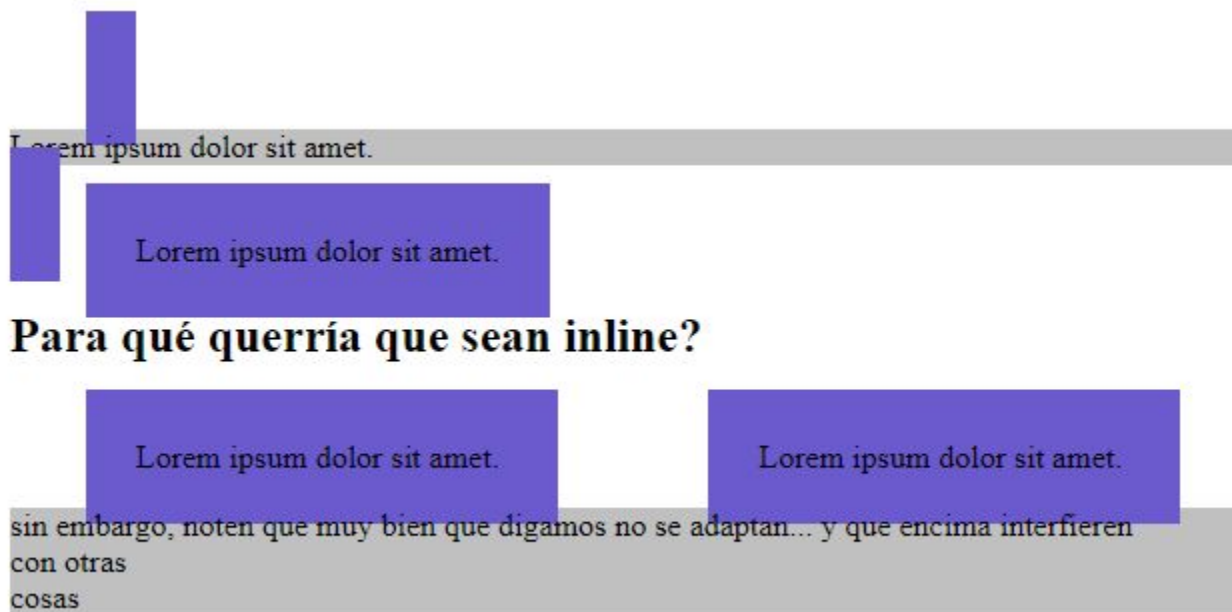
Vayamos por orden. Ya block e inline los vimos.

Sí, pero los vimos de forma nativa! un div es nativamente block, un span es nativamente inline, pero podemos cambiar eso en el css:

```
.soyInline {  
  background-color:  slateblue;  
  display: inline;  
}  
  
span {  
  display: block;  
}
```



Sin embargo, como habrán notado, trabajar con inline y block puede fácilmente convertirse en un infierno. Hay que usarlos con bastante cuidado, o:



EVERYBODY FLOATS

imgflip.com



Float es... viejo. Y un poco problemático también.

La propiedad CSS `float` ubica un elemento al lado izquierdo o derecho de su contenedor, permitiendo a los elementos de texto y en línea aparecer a su costado. El elemento es removido del normal flujo de la página, aunque aún sigue siendo parte del flujo (a diferencia del [posicionamiento absoluto](#)).

Hoy en día se usa poco y tiene mala fama... es muy complicado de manejar bien y no romper todo.



Valores

left

El elemento debe flotar a la izquierda de su bloque contenedor.

right

El elemento debe flotar a la derecha de su bloque contenedor.

none

El elemento no deberá flotar.

inline-start

El elemento debe flotar en el costado de inicio de su bloque contenedor. Esto es el lado izquierdo con scripts `ltr` y el lado derecho con scripts `rtl`.

inline-end

El elemento debe flotar en el costado de término de su bloque contenedor. Esto es el lado derecho con scripts `ltr` y el lado izquierdo con scripts `rtl`.



Todo bien con el float, pero esto se ve re word en los 90 cuando ponías una imagen.

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Laboriosam harum aliquid fuga earum in iste nisi praesentium deserunt eveniet dolore eos dolor facere, ea repudiandae est saepe consectetur quisquam! Ullam?.

Lorem
ipsum
dolor sit
amet

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Exercitationem ratione hic voluptate explicabo delectus quia officiis error ad sint molestias, nesciunt quo quidem rem autem fuga nam quae, voluptatibus accusantium!

consectetur adipisicing elit. Harum libero nobis inventore expedita autem ullam cumque, officia maxime temporibus, at hic quaerat, assumenda provident delectus tempora voluptatibus odio eligendi corporis!



I HAVE A PARTICULAR SET OF SKILLS

I CAN MOVE THINGS AROUND USING
FLEXBOX



Lo que todos esperaban: flexbox!

El Módulo de Caja Flexible, comúnmente llamado 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. Este artículo hace un repaso de las principales características de flexbox, las que exploraremos con mayor detalle en el resto de estas guías.

Cuando describimos a flexbox como unidimensional destacamos el hecho que flexbox maneja el layout en una sola dimensión a la vez — ya sea como fila o como columna. Esto contrasta con el modelo bidimensional del [Grid Layout de CSS](#), el cual controla columnas y filas a la vez.

Flexbox es un poquito más complejo de aprender, pero vale la pena.



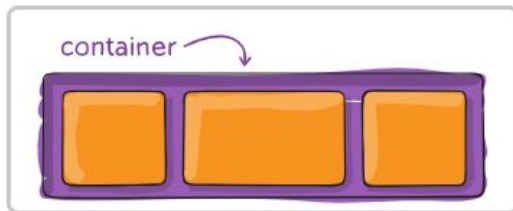
En flex tenemos que distinguir lo que es del padre y de los hijos

Vamos a declarar el display y su dirección, entre otras cosas, en el padre, mientras que vamos a trabajar en el orden y tamaño en los hijos.

Toda esta info está acá, este sitio se lo guardan como inicio de chrome, así de fácil:

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>





🔗 Properties for the Parent (flex container)

🔗 display

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

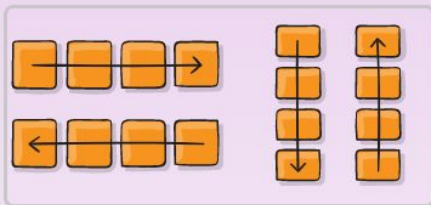
```
.container {  
  display: flex; /* or inline-flex */  
}
```

CSS

Note that CSS columns have no effect on a flex container.



flex-direction



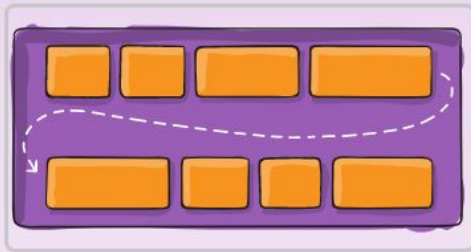
This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

```
.container {  
  flex-direction: row | row-reverse | column | column-reverse  
}
```

- row (default): left to right in ltr ; right to left in rtl
- row-reverse : right to left in ltr ; left to right in rtl
- column : same as row but top to bottom
- column-reverse : same as row-reverse but bottom to top



flex-wrap



By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property.

```
.container {  
  flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

CSS

- nowrap (default): all flex items will be on one line
- wrap : flex items will wrap onto multiple lines, from top to bottom.
- wrap-reverse : flex items will wrap onto multiple lines from bottom to top.



🔗 flex-flow

This is a shorthand for the `flex-direction` and `flex-wrap` properties, which together define the flex container's main and cross axes. The default value is `row nowrap`.

```
.container {  
  flex-flow: column wrap;  
}
```

CSS

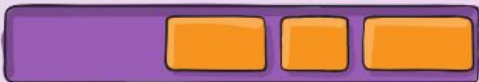


justify-content

flex-start



flex-end



center



space-between



space-around



space-evenly

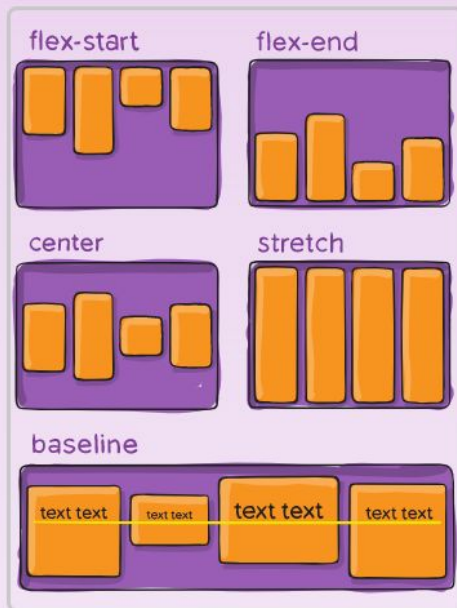


This defines the alignment along the main axis. It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

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



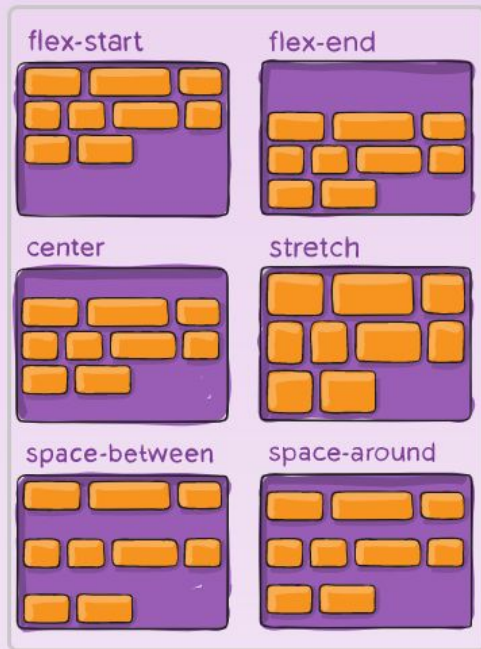
align-items



This defines the default behavior for how flex items are laid out along the **cross axis** on the current line. Think of it as the `justify-content` version for the cross-axis (perpendicular to the main-axis).

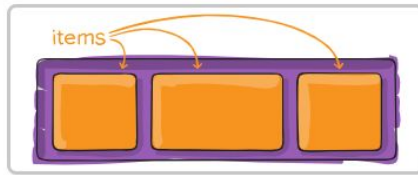


align-content



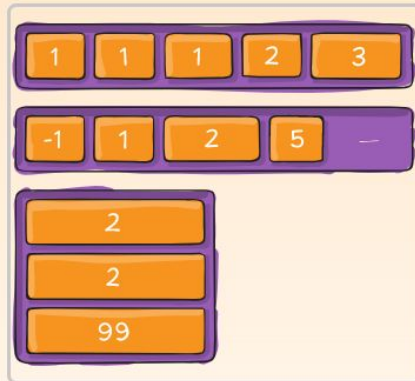
This aligns a flex container's lines within when there is extra space in the cross-axis, similar to how `justify-content` aligns individual items within the main-axis.





Properties for the Children (flex items)

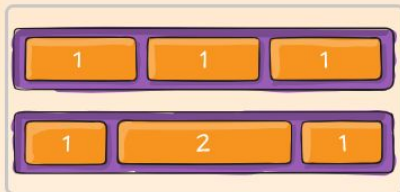
 **order**



By default, flex items are laid out in the source order. However, the `order` property controls the order in which they appear in the flex container.



flex-grow



This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

If all items have `flex-grow` set to 1, the remaining space in the container will be distributed equally to all children. If one of the children has a value of 2, the remaining space would take up twice as much space as the others (or it will try to, at least).

```
.item {  
  flex-grow: 4; /* default 0 */  
}
```

Negative numbers are invalid.

flex-shrink

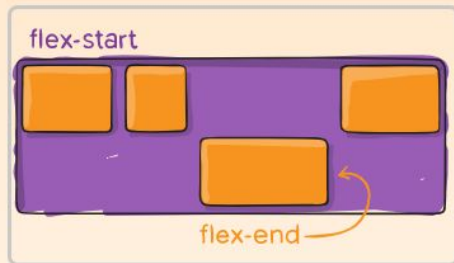
This defines the ability for a flex item to shrink if necessary.

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

Negative numbers are invalid.



align-self



This allows the default alignment (or the one specified by `align-items`) to be overridden for individual flex items.

Please see the `align-items` explanation to understand the available values.

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

Note that `float`, `clear` and `vertical-align` have no effect on a flex item.





NEW THINGS?

I'M TERRIFIED OF THAT.

makeameme.org

