Fuente: lenguajehtml.com y lenguajecss.com

Unidades de Medidas

Unidades absolutas

Se llaman absolutas porque su valor es el declarado. No hay cálculo del mismo. Es el que es y de ahí no se mueve

- ### Pixel

  Px: Los píxeles son unidades de tamaño fijo. Un pixel es igual a un punto en la pantalla del ordenador (la división más pequeña de la resolución de su pantalla) y por lo tanto, es indivisible.

  Su problema es que no es escalable, atentado contra la accesibilidad. Era una cuestión que muchos obviaban porque les permitía un control "al milímetro" de sus realizaciones.

  Pero hoy a esa falla en la accesibilidad por algunos colectivos se suma el comportamiento en dispositivos de pequeñas dimensiones y grandes resoluciones.

  Al usar pixeles para definir las dimensiones de los objetos y los estilos de fuente no solo estamos definiendo tamaños rígidos, sino que también estamos ignorando las configuraciones que cada usuario pueda tener en su navegador.

  Ni permite escalar a más los textos a los usuarios con problemas visuales ni a menos en dispositivos como los smartphones.

- ### Puntos

  Pt: El punto es una herencia de los medios impresos. Un punto es igual a 1/72 de pulgada (según diversas fuentes), aprox. 0´04mm.

  Al ser también una medida absoluta tiene todos los problemas de los px aumentados.

- ### Otras unidades absolutas:

  - mm, cm, in: Milímetros, centímetros, o pulgadas.
  - pc: picas (12 puntos.)

Unidades relativas

El valor final resultante (computado) está en función de un valor previo.

- ## Em

  Es básicamente el tamaño de una letra «M» (bien podría ser cualquier otra letra) del elemento al cual se esté aplicando esta medida. Es decir, si en elemento tiene aplicado un tamaño de fuente de 16 pixeles, entonces 1 em será igual a 16px (los navegadores de manera predeterminada definen un font-size de 16px al elemento HTML, por lo tanto, por defecto 1em es igual a 16px).

  La unidad em es escalable y siempre depende de su elemento padre. Por ejemplo, si el elemento body tiene un tamaño de fuente de 16px y un elemento hijo tiene una fuente con tamaño 1.3em, este texto se mostrará de un tamaño un 30% más grande que el del body (20.8px), mientras que si dentro de ese elemento tenemos otro hijo con un font-size de 1.3 em, el tamaño de fuente de este objeto sería un 30% más grande que el tamaño de su padre (27.04px).

  Body = 1em (16px)
  Hijo = 1.3em (16px x 1.3 = 20.8px)
  Nieto = 1.3em (20.8px x 1.3 = 27.04px)

- ## Porcentaje %

  La unidad de medida porcentual es la que se usa por defecto en los elementos HTML en donde de manera predeterminada cada elemento de bloque usa un ancho del 100%, es por eso que cuando achicamos la ventana del navegador con una página que no tenga estilos, la página se adapta, ya que siempre usará el ancho total visible. Pero nosotros podemos utilizar los porcentajes de una manera más avanzada tratando de generar layouts más complejos. Supongamos, por ejemplo, que tienes un div que contiene todos los elementos de la página y, según el diseño, este elemento debiera medir 1200 pixeles. En lugar de caer en la tentación de simplemente usar esa medida en pixeles, te recomendaría usar una medida en porcentajes, en donde el máximo ancho del elemento sea esos 1200px:

  CSS
  ```
  .container {
     margin: 0 auto;
     width: 90%;
     max-width: 1200px;
  }
  ```

  Con estas 3 propiedades de CSS conseguimos que a) El elemento se centre en la página, b) tenga un ancho del 90% de la ventana y c) su ancho nunca sea superior a 1200 píxeles.

  Es decir, hemos conseguido que el elemento con la clase container sea responsive sin la necesidad siquiera de escribir un media query.

El uso de los porcentajes también lo podemos llevar a elementos interiores del layout, en donde, por ejemplo, podemos asignar a la columna principal de contenido y a la barra lateral unas medidas de ancho del 70% y el 30% respectivamente, haciendo que sean completamente adaptables al tamaño de su elemento contenedor.

- ## Rem:

La unidad de medida rem es muy similar a em, con la única diferencia de que no es escalable, esto quiere decir que no depende del elemento padre, sino del elemento raíz del documento, el elemento HTML. Rem significa «Root Em«, o sea, es un em basado en la raíz.

Esto significa que si el elemento HTML tiene un tamaño de fuente de 16px (como es por defecto), entonces 1rem, sería igual a 16px, y si queremos aplicar un tamaño basado en rem a cualquier elemento de la página, no importará cual sea el tamaño de fuente que tenga asociado ese elemento, ya que 1 rem siempre será igual a 16 pixeles a no ser que se modifique el elemento raíz.

Usar rem nos permite cierta estructura para poder definir ciertas partes del layout, pero al mismo tiempo nos entrega cierta escalabilidad para respetar las configuraciones de cada usuario.

Esta unidad de medida es recomendable para aplicar a elementos del layout que requieran medidas fijas y eventualmente también para textos que deseemos que tengan un tamaño de fuente que no dependa de su elemento padre.

Los rems, son una unidad muy interesante también para definir los media queries de CSS.

Si quisiéramos refinar el ejemplo anterior para no usar pixeles deberíamos usar algo como lo siguiente:
CSS

```
.container {
   margin: 0 auto;
   width: 90%;
   max-width: 75rem;
}
```

Para poder convertir una medida de pixeles a rem solo tienes que multiplicar el tamaño que quieres obtener por el número 0.0625, eso te dará el tamaño que debes usar en rem. Así es como se define que 75rem es igual a 1200px:
75rem = 1200px x 0.0625

Para evitarnos este cálculo se puede definir el font-size del elemento html con un tamaño de 62.5%, de esta forma, conseguimos que 1 rem sea equivalente a 10px, haciendo más fácil el cálculo.

● ex, ch: Son respectivamente la altura de la x minúscula, y el ancho del número 0. Aunque no son tan soportadas por los navegadores como los rems.
● vw, vh: Estas son respectivamente 1/100 del ancho de la ventana, y 1/100 de la altura de la ventana. Tampoco son tan soportadas como los rems.

Posicionamiento

A grandes rasgos, si tenemos varios elementos en línea (uno detrás de otro) aparecerán colocados de izquierda hacia derecha, mientras que si son elementos en bloque se verán colocados desde arriba hacia abajo. Estos elementos se pueden ir combinando y anidando (incluyendo unos dentro de otros), construyendo esquemas más complejos.

Hasta ahora, hemos estado trabajando sin saberlo en lo que se denomina posicionamiento estático (static), donde todos los elementos aparecen con un orden natural según donde estén colocados en el HTML. Este es el modo por defecto en que un navegador renderiza una página.

Sin embargo, existen otros modos alternativos de posicionamiento, que podemos cambiar mediante la propiedad position, que nos pueden interesar para modificar la posición en donde aparecen los diferentes elementos y su contenido.
A la propiedad position se le pueden indicar los siguientes valores:

Valor	Significado
static	Posicionamiento estático. Utiliza el orden natural de los elementos HTML.
relative	Posicionamiento relativo. Los elementos se mueven ligeramente en base a su posición estática.
absolute	Posicionamiento absoluto. Los elementos se colocan en base al contenedor padre.
fixed	Posicionamiento fijo. Idem al absoluto, pero, aunque hagamos scroll no se mueve.

Si utilizamos un modo de posicionamiento diferente al estático (absolute, fixed o relative), podemos utilizar una serie de propiedades para modificar la posición de un elemento. Estas propiedades son las siguientes:

Propiedad	Valor	Significado
top:	auto | TAMAÑO	Empuja el elemento una distancia desde la parte superior hacia el inferior.
bottom:	auto | TAMAÑO	Empuja el elemento una distancia desde la parte inferior hacia la superior.
left:	auto | TAMAÑO	Empuja el elemento una distancia desde la parte izquierda hacia la derecha.
right:	auto | TAMAÑO	Empuja el elemento una distancia desde la parte derecha hacia la izquierda.
z-index:	auto | nivel	Ordena en el eje de profundidad, superponiendo u ocultando.

Antes de pasar a explicar los tipos de posicionamiento, debemos tener claras las propiedades top, bottom, left y right, que sirven para mover un elemento desde la orientación que su propio nombre indica hasta su extremo contrario. Esto es, si utilizamos left e indicamos 20px, estaremos indicando mover desde la izquierda 20 píxeles hacia la derecha.

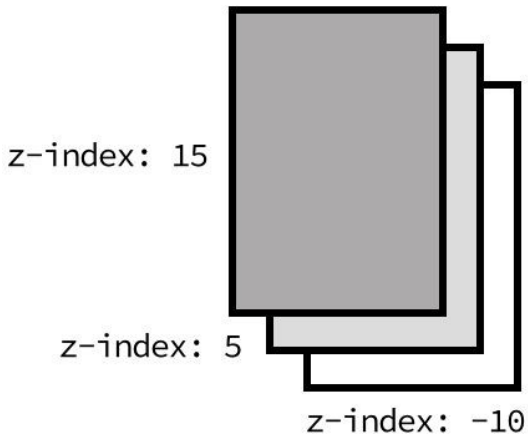
Pero pasemos a ver cada tipo de posicionamiento por separado y su comportamiento:

Posicionamiento relativo
Si utilizamos la palabra clave relative activaremos el modo de posicionamiento relativo, que es el más sencillo de todos. En este modo, los elementos se colocan exactamente igual que en el posicionamiento estático (permanecen en la misma posición), pero dependiendo del valor de las propiedades top, bottom, left o right variaremos ligeramente la posición del elemento.
Ejemplo: Si establecemos left:40px, el elemento se colocará 40 píxeles a la derecha desde la izquierda donde estaba colocado en principio, mientras que si especificamos right:40px, el elemento se colocará 40 píxeles a la izquierda desde la derecha donde estaba colocado en principio.

Posicionamiento absoluto
Si utilizamos la palabra clave absolute estamos indicando que el elemento pasará a utilizar posicionamiento absoluto, que no es más que utilizar el documento completo como referencia. Esto no es exactamente el funcionamiento de este modo de posicionamiento, pero nos servirá como primer punto de partida para entenderlo.
Ejemplo: Si establecemos left:40px, el elemento se colocará 40 píxeles a la derecha del extremo izquierdo de la página. Sin embargo, si indicamos right:40px, el elemento se colocará 40 píxeles a la izquierda del extremo derecho de la página.
Como mencionaba anteriormente, aunque este es el funcionamiento del posicionamiento absoluto, hay algunos detalles más complejos en su modo de trabajar. Realmente, este tipo de posicionamiento coloca los elementos utilizando como punto de origen el primer contenedor con posicionamiento diferente a estático. Por ejemplo, si el contenedor padre tiene posicionamiento estático, pasamos a mirar el posicionamiento del padre del contenedor padre, y así sucesivamente hasta encontrar un contenedor con posicionamiento no estático o llegar a la etiqueta <body>, en el caso que se comportaría como el ejemplo anterior.

Posicionamiento fijo
Por último, el posicionamiento fijo es hermano del posicionamiento absoluto. Funciona exactamente igual, salvo que hace que el elemento se muestre en una posición fija dependiendo de la región visual del navegador. Es decir, aunque el usuario haga scroll y se desplace hacia abajo en la página web, el elemento seguirá en el mismo sitio posicionado.
Ejemplo: Si establecemos top:0 y right:0, el elemento se colocará justo en la esquina superior derecha y se mantendrá ahí aunque hagamos scroll hacia abajo en la página.

Profundidad (niveles)
Es interesante conocer también la existencia de la propiedad z-index, que establece el nivel de profundidad en el que está un elemento sobre los demás. De esta forma, podemos hacer que un elemento se coloque encima o debajo de otro.
Su funcionamiento es muy sencillo, sólo hay que indicar un número que representará el nivel de profundidad del elemento. Los elementos un número más alto estarán por encima de otros con un número más bajo, que permanecerán ocultos detrás de los primeros.

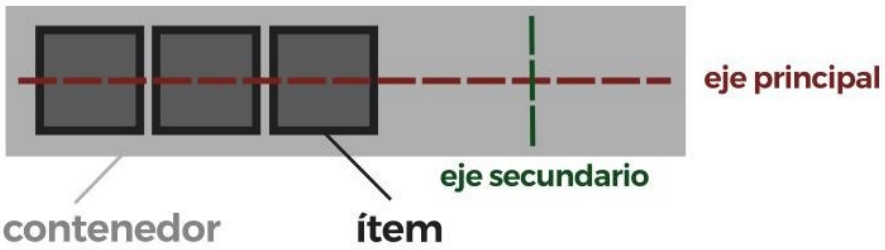


Nota: Los niveles z-index, así como las propiedades top, left, bottom y right no funcionan con elementos que estén utilizando posicionamiento estático. Deben tener un tipo de posicionamiento diferente a estático.

CSS Flexbox

Flexbox es un sistema de **elementos flexibles** que llega con la idea de acostumbrarnos a una mecánica más potente, limpia y personalizable, en la que los elementos HTML se adaptan y colocan automáticamente y es más fácil personalizar

los diseños. Está especialmente diseñado para crear, mediante CSS, estructuras de **una sóla dimensión.**



Conceptos

Para empezar a utilizar **flexbox** lo primero que debemos hacer es conocer algunos de los elementos básicos de este nuevo esquema, que son los siguientes:

■ Contenedor: Existe un elemento padre que es el contenedor que tendrá en su interior cada uno de los ítems flexibles y adaptables.
■ Eje principal: Los contenedores flexibles tendrán una orientación principal específica. Por defecto, es en horizontal (fila).
■ Eje secundario: De la misma forma, los contenedores flexibles tendrán una orientación secundaria, perpendicular a la principal. Si la principal es en horizontal, la secundaria será en vertical, y viceversa.
■ Ítem: Cada uno de los hijos flexibles que tendrá el contenedor en su interior.

Imaginemos el siguiente escenario:

```
<div id="contenedor">                    <!-- contenedor flex -->
   <div class="item item-1">1</div>      <!-- cada uno de los ítems flexibles -->
   <div class="item item-2">2</div>
   <div class="item item-3">3</div>
</div>
```

Para activar el modo **flexbox** hay que utilizar sobre el elemento contenedor la propiedad display, y especificar el valor **flex.**

Por defecto, y sólo con esto, observaremos que los elementos se disponen todos sobre una misma línea. Esto ocurre porque estamos utilizando el modo **flexbox** y estaremos trabajando con ítems flexibles básicos, garantizando que no se desbordaran.

Dirección de los ejes

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

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

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

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

Esto nos permite tener un control muy alto sobre el orden de los elementos en una página. Veamos la aplicación de estas propiedades sobre el ejemplo anterior, para modificar el flujo del eje principal del contenedor:

```
#contenedor {
    background: #CCC;
    display: flex;
    flex-direction: column;
}
.item {
    background: #777;
}
```

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

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

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

```
#contenedor {
    background: #CCC;
    display: flex;
    width: 200px;
    flex-wrap: wrap; /* Comportamiento por defecto: nowrap */
}
.item {
    background: #777;
    width: 50%;
}
```

En el caso de especificar **nowrap** (u omitir la propiedad flex-wrap) en el contenedor, los 3 ítems se mostrarían en una misma línea del contenedor. En ese

caso, cada ítem debería tener un 50% de ancho (o sea, 100px de los 200px del contenedor). Un tamaño de **100px** por ítem, sumaría un total de **300px**, que no cabrían en el contenedor de **200px**, por lo que flexbox reajusta los ítems flexibles para que quepan todos en la misma línea, manteniendo las mismas proporciones.

Sin embargo, si especificamos wrap en la propiedad flex-wrap, lo que permitimos es que el contenedor se pueda desbordar, pasando a ser un contenedor **multilínea,** que mostraría el **ítem 1 y 2** en la primera linea (con un tamaño de 100px cada uno) y el **ítem 3** en la línea siguiente, dejando un espacio libre para un posible **ítem 4.**

Atajo: Dirección de los ejes

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

```
#contenedor {
   /* flex-flow: <flex-direction> <flex-wrap>; */
   flex-flow: row wrap;
}
```

Propiedades de alineación de ítems

Ahora que tenemos un control básico del contenedor de estos ítems flexibles, necesitamos conocer las propiedades existentes dentro de flexbox para disponer los ítems dependiendo de nuestro objetivo. Vamos a echar un vistazo a cuatro propiedades interesantes para ello:

Propiedad	Valor	Actúa sobre
justify-content:	**flex-start** | flex-end | center | space-between | space-around	Eje principal

align-content:	flex-start | flex-end | center | space-between | space-around | **stretch**	Eje principal
align-items:	flex-start | flex-end | center | **stretch** | baseline	Eje secundario
align-self:	**auto** | flex-start | flex-end | center | stretch | baseline	Eje secundario

De esta pequeña lista, nos centraremos en la primera y la tercera propiedad, que son las más importantes (las otras dos son casos particulares que explicaremos más adelante):

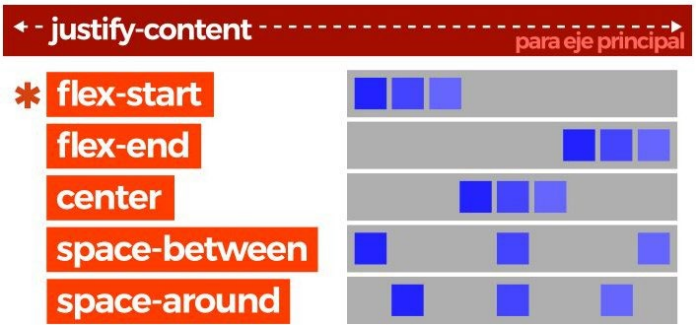
■ justify-content: Se utiliza para alinear los ítems del eje principal (por defecto, el horizontal).
■ align-items: Usada para alinear los ítems del eje secundario(por defecto, el vertical).

Sobre el eje principal

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

Valor	Descripción
flex-start	Agrupa los ítems al principio del eje principal.
flex-end	Agrupa los ítems al final del eje principal.
center	Agrupa los ítems al centro del eje principal.
space-between	Distribuye los ítems dejando (el mismo) espacio entre ellos.
space-around	Distribuye los ítems dejando (el mismo) espacio a ambos lados de cada uno de ellos.

Con cada uno de estos valores, modificaremos la disposición de los ítems del contenedor donde se aplica, pasando a colocarse como se ve en la imagen siguiente (nótense las diferentes tonalidades azules para indicar las posiciones de cada ítem):



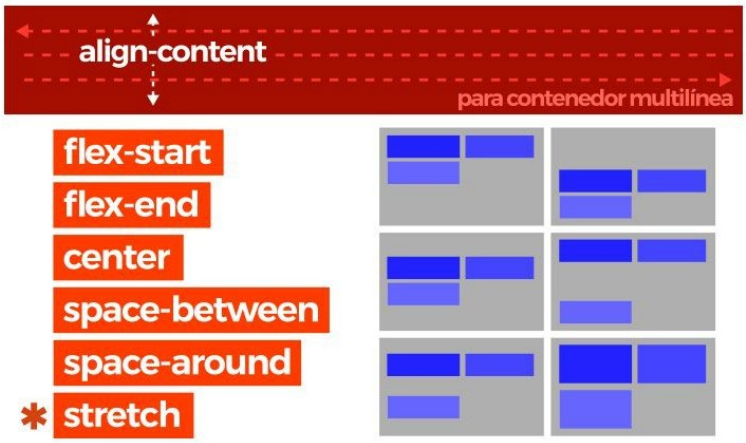
Una vez entendido este caso, debemos atender a la propiedad align-content, que es un caso particular del anterior. Nos servirá cuando estemos tratando con un

contenedor flex multilínea, que es un contenedor en el que los ítems no caben en el ancho disponible, y por lo tanto, el eje principal se divide en múltiples líneas.

De esta forma, align-content servirá para alinear cada una de las líneas del contenedor multilinea. Los valores que puede tomar son los siguientes:

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

Con estos valores, vemos cómo cambiamos la disposición en vertical (porque partimos de un ejemplo en el que estamos utilizando flex-direction: row, y el eje principal es horizontal) de los ítems que están dentro de un contenedor multilínea.



En el ejemplo siguiente, veremos que al indicar un contenedor de 200 píxels de alto con ítems de 50px de alto y un flex-wrapestablecido para tener contenedores multilinea, podemos utilizar la propiedad align-content para alinear los ítems de forma vertical de modo que se queden en la zona inferior del contenedor:

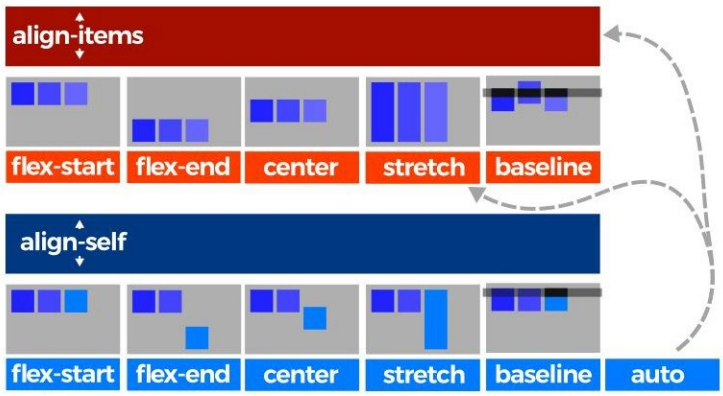
```
#contenedor {
    background: #CCC;
    display: flex;
    width: 200px;
    height: 200px;
    flex-wrap: wrap;
    align-content: flex-end;
}
.item {
    background: #777;
    width: 50%;
    height: 50px;
}
```

Sobre el eje secundario

La otra propiedad importante de este apartado es align-items, que se encarga de alinear los ítems en el eje secundario del contenedor. Hay que tener cuidado de no confundir align-content con align-items, puesto que el primero actúa sobre cada una de las líneas de un contenedor multilinea (no tiene efecto sobre contenedores de una sola línea), mientras que align-items lo hace sobre la línea actual. Los valores que puede tomar son los siguientes:

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

Por otro lado, la propiedad align-self actúa exactamente igual que align-items, sin embargo es la primera propiedad de flexbox que vemos que se utiliza sobre un ítem hijo específico y no sobre el elemento contenedor. Salvo por este detalle, funciona exactamente igual que align-items.



Gracias a ese detalle, align-self nos permite cambiar el comportamiento de align-items y sobreescribirlo con comportamientos específicos para ítems concretos que no queremos que se comporten igual que el resto. La propiedad puede tomar los siguientes valores:

Valor	Descripción
flex-start	Alinea los ítems al principio del contenedor.
flex-end	Alinea los ítems al final del contenedor.
center	Alinea los ítems al centro del contenedor.
stretch	Alinea los ítems estirándolos al tamaño del contenedor.
baseline	Alinea los ítems en el contenedor según la base de los ítems.

auto Hereda el valor de align-items del padre (o si no lo tiene, stretch).

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

Propiedades de ítems hijos

A excepción de la propiedad align-self, todas las propiedades que hemos visto hasta ahora se aplican sobre el elemento contenedor. Las siguientes propiedades, sin embargo, se aplican sobre los ítems hijos. Echemos un vistazo:

Propiedad	Valor	Descripción
flex-grow:	**0** | *[factor de crecimiento]*	Número que indica el crecimiento del ítem respecto al resto.
flex-shrink:	**1** | *[factor de decrecimiento]*	Número que indica el decrecimiento del ítem respecto al resto.
flex-basis:	**content**	Tamaño base de los ítems antes de aplicar variación.
order:	**0** | *[número]*	Número (peso) que indica el orden de aparición de los ítems.

En primer lugar, tenemos la propiedad **flex-grow** para indicar el factor de crecimiento de los ítems en el caso de que no tengan un ancho específico. Por ejemplo, si con flex-grow indicamos un valor de 1 a todos sus ítems, tendrían el mismo tamaño cada uno de ellos. Pero si colocamos un valor de 1 a todos los elementos, salvo a uno de ellos, que le indicamos 2, ese ítem será más grande que los anteriores. Los ítems a los que no se le especifique ningún valor, tendrán por defecto valor de 0.

En segundo lugar, tenemos la propiedad **flex-shrink** que es la opuesta a flex-grow.
Mientras que la anterior indica un factor de crecimiento, flex-shrink hace justo lo contrario, aplica un factor de decrecimiento. De esta forma, los ítems que tengan un valor numérico más grande, serán más pequeños, mientras que los que tengan un valor numérico más pequeño serán más grandes, justo al contrario de como funciona la propiedad flex-grow.

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

Atajo: Propiedades de ítems hijos

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

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

Orden de los ítems

Por último, y quizás una de las propiedades más interesantes, es order, que

modificar y establece el orden de los ítems según una secuencia numérica.

Por defecto, todos los ítems flex tienen un order: 0 implícito, aunque no se especifique. Si indicamos un order con un valor numérico, irá recolocando los ítems según su número, colocando antes los ítems con número más pequeño (incluso valores negativos) y después los ítems con números más altos.