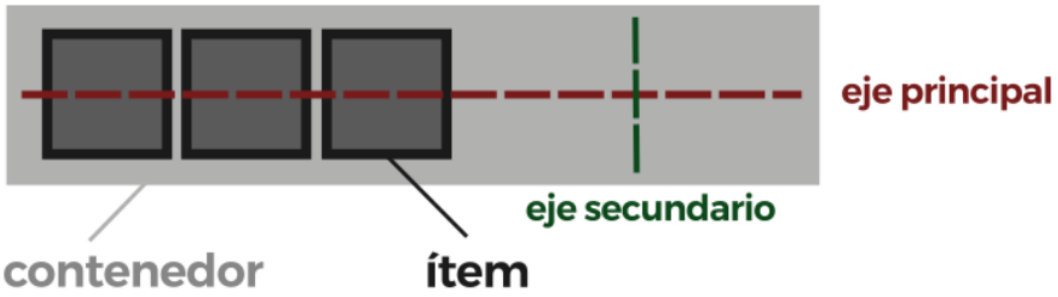
FLEX

1. Introducción

Este nuevo valor para la propiedad `display` ha reemplazado a `float` y `position` como las propiedades clave en la maquetación web. Su utilidad radica en que convierte los elementos HTML en flexibles, pudiendo adaptar su posición, anchura o altura como deseemos mediante CSS.

2. Conceptos previos

Digamos que tenemos un contenedor padre donde almacenaremos nuestros elementos que queremos que sean flexibles.



En la imagen, tenemos:

- **Contenedor:** Elemento padre 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 horizontal.
- **Eje secundario o transversal:** Es el eje perpendicular al principal (*cross-axis*). Si el principal es horizontal, el secundario será vertical y viceversa.
- **Ítem:** Cada uno de los hijos flexibles que tendrá el contenedor en su interior.

Imaginemos el siguiente escenario, donde tenemos un contenedor y 3 ítems en su interior:

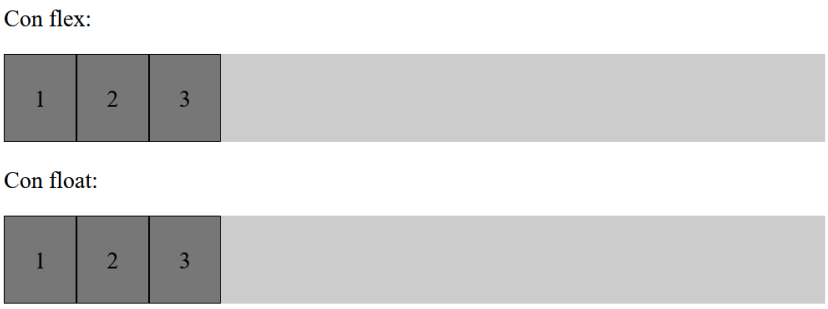
```
<div id="contenedor"> <!-- contenedor flex -->
  <div class="item item">1</div> <!-- ítem flexible -->
  <div class="item item">2</div> <!-- ítem flexible -->
  <div class="item item">3</div> <!-- ítem flexible -->
</div>
```

Sobre el elemento contenedor aplicaremos la propiedad `display` con el valor `flex` o `inline-flex` dependiendo de cómo queramos que se comporte el contenedor, si como un elemento en línea o como un elemento en bloque.

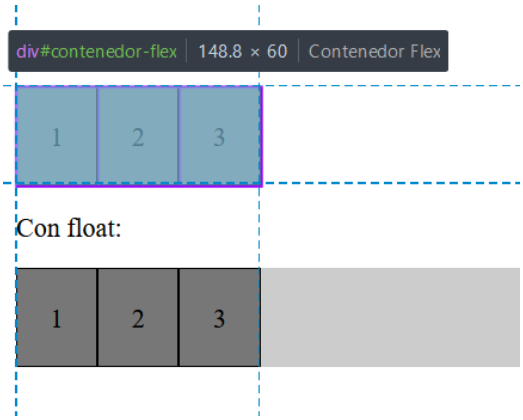
- `flex` establece un contenedor flexible en bloque, de forma equivalente a *block*.
- `inline-flex` establece un contenedor flexible en línea, de forma equivalente a `inline-block`.

De esta forma, los elementos se dispondrán todos sobre una misma línea, con lo que conseguimos el mismo efecto que en maquetación tradicional conseguíamos con `float`. Observa el siguiente ejemplo en codepen:

El resultado es:



El contenedor `flex` ocupa el 100% de la anchura disponible porque es un elemento de bloque (`display: flex`). Si quisiéramos que se comportase como un elemento de línea tendríamos que indicarlo con `display: inline-flex`. En ese caso, el contenedor sólo ocupa el ancho estrictamente necesario para albergar a los hijos:



¡Importante! Fíjate que los ítems del contenedor flex no se estiran en la dirección del eje principal, pero sí en la del eje transversal (si fuera necesario). En la imagen anterior no se aprecia porque el contenedor flex tiene la altura necesaria para albergar a sus hijos. Sin embargo, imagina qué debería ocurrir en estas dos situaciones:

a) ¿Y si el contenedor tiene más altura que cualquiera de sus hijos?

b) ¿Y si uno de los hijos tiene más altura que el resto?

Podemos simular ambas situaciones. Para la primera, basta con añadir al contenedor `height: 100px`.

Vemos que los hijos se estiran en la dirección del eje transversal hasta alcanzar la altura del contenedor.

Para la segunda situación quitamos la altura que hemos dado antes al contenedor y se la ponemos a alguno de los hijos, añadiendo:

```
#contenedor-flex>*:first-child {
  height: 100px;
}
```

El resultado es exactamente el mismo que en la situación anterior.

Otro aspecto **importante** es que flex no se hereda. Es decir, si indicamos `display: flex` a un elemento sólo se aplica a ese elemento, no a sus hijos o descendientes. Puedes observar esto añadiendo un par de `divs` a un ítem en el ejemplo anterior, por ejemplo:

```
<div id="contenedor-flex">
  <div class="item">
    <div>hola</div>
    <div>adios</div>
  </div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>
```

Y darle un pequeño estilo para diferenciarlos:

```
.item > * {
  border: 1px solid red;
}
```

Observarás el comportamiento normal de dos elementos de bloque:

Pero si añadimos `display: flex` a ese ítem si disponen uno al lado del otro:

3. Dirección de los ejes: *flex-direction* y *flex-wrap*

Existen dos propiedades para manipular la dirección de los ítems a lo largo del eje principal del contenedor. Son `flex-direction` y `flex-wrap`.

Propiedad	Valores posibles	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).

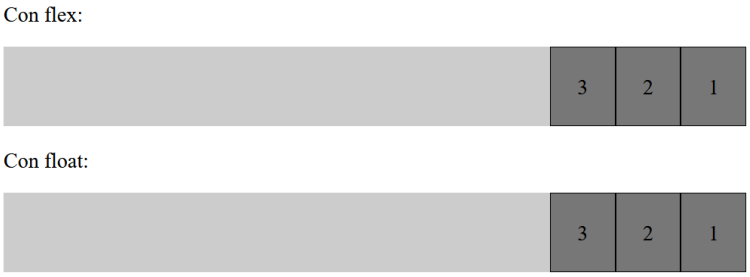
Tanto en esta tabla como en las siguientes, el valor subrayado indica el valor por defecto de la propiedad.

3.1. *flex-direction*

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 que coloque los ítems en orden inverso al que aparecen en el documento HTML.

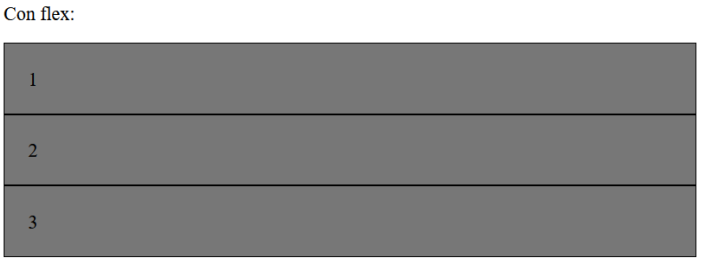
Valor	Descripción
`row`	Establece la dirección del eje principal en horizontal.
`row-reverse`	Establece la dirección del eje principal en horizontal invertido.
`column`	Establece la dirección del eje principal en vertical.
`column-reverse`	Establece la dirección del eje principal en vertical invertido.

Veamos el ejemplo anterior (Flex. Flex vs float), pero flotando a la derecha y con `row-reverse`. Simplemente hay que cambiar `float: left` por `float: right` y añadir al contenedor flex `flex-direction: row-reverse`. El resultado será:



Si añadimos `flex-direction: column` al ejemplo anterior, el resultado será:

```
#contenedor-flex {
  background: #ccc;
  display: flex;
  flex-direction: column;
}
```



Si recuerdas, un poco más arriba comentábamos que los ítems del contenedor flex no se estiran en la dirección del eje principal, pero sí en la del eje transversal. Ahora que hemos intercambiado los ejes principal y transversal, podemos ver el mismo comportamiento dando al contenedor una altura determinada. Por ejemplo, 200px:

Ahora los ítems se están estirando en la dirección del eje transversal (horizontal), pero no en la del eje principal (vertical).

Puedes ver el comportamiento de `flex-direction` en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_flex-direction

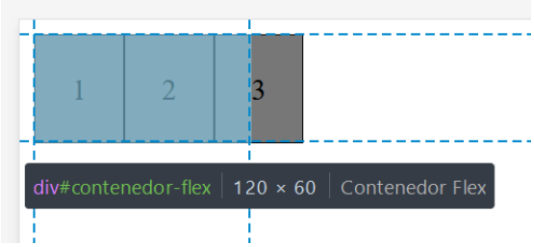
También en el siguiente codepen.

3.2. *flex-wrap*

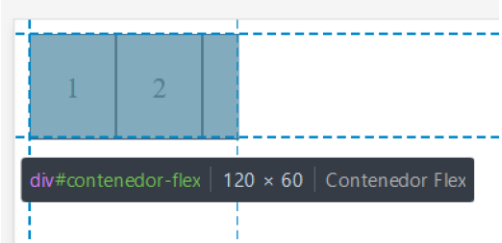
Con la propiedad `flex-wrap` especificamos el comportamiento del contenedor en caso de desbordamiento. Los valores posibles son:

Valor	Descripción
`nowrap`	Establece los ítems en una sola fila o columna, dependiendo del eje principal. 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.

Observa el siguiente codepen. Si reducimos la anchura del contenedor a 120px manteniendo el valor por defecto `flex-wrap: nowrap`, la anchura del contenedor será menor que la suma de la anchura total de los ítems. El resultado es el esperado, aunque el contenedor sea más pequeño los elementos se siguen viendo con normalidad.



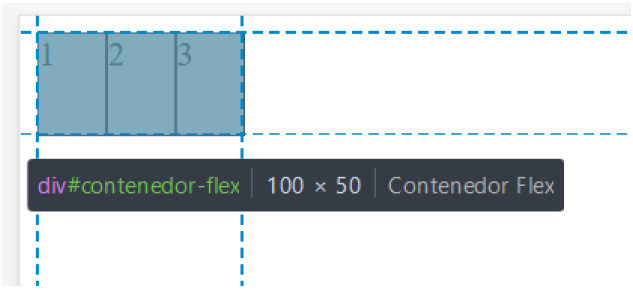
Si no queremos que se vea el contenido desbordado, podemos indicárselo al contenedor con `overflow: hidden`.



Hay que tener **cuidado** con esta propiedad porque si los hijos tienen una anchura establecida por encima de su contenido se encogerán al encogerse el contenedor.

Esto lo podemos observar quitando el `padding` y añadiéndole `width`:

```
.item {
  background-color: #777;
  /* padding: 20px; */
  width: 50px;
  height: 50px;
  border: 1px solid black;
}
```

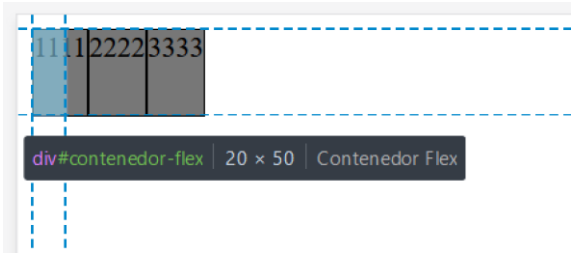


Los hijos se encogerán hasta que su contenido lo permita. Es decir, si hacemos que su contenido sea más ancho (aumentado el `padding` o aumentando el texto), aunque disminuyamos la anchura del contenedor los hijos van a seguir ocupando el espacio que necesitan.

Prueba añadiendo al ejemplo anterior un poco más de texto a uno de los hijos:

```
<div id="contenedor-flex">
  <div class="item">1111</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>
```

A medida que reducimos la anchura del contenedor los ítems hijos se van encogiendo, igual que antes, pero cuando el primer hijo llega a la anchura mínima deja de encogerse mientras que los otros dos siguen encogiéndose:



Si ahora activamos `flex-wrap: wrap` o `wrap-reverse` los ítems que no quepan se irán a la línea siguiente.

También puedes ver el comportamiento de esta propiedad en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_flex-wrap

3.3. *flex-flow* (atajo)

Podemos utilizar la propiedad `flex-flow` para resumir los valores de las propiedades `flex-direction` y `flex-wrap`, especificándolas en una sola propiedad:

```
flex-flow: <flex-direction> <flex-wrap>;
```

4. Espacios (gaps): *row-gap*, *column-gap* y *gap*

Estas propiedades permiten establecer un *gap* (espacio) entre los ítems hijos.

Según la especificación del W3C, estas propiedades se aplican a contenedores flex, contenedores grid (que veremos más adelante) y a contenedores multicolumna, donde sólo se aplica la propiedad `column-gap`.

Recordemos que un contenedor multicolumna es aquel en el que se ha establecido un valor distinto de `auto` a la propiedad `column-width` o `column-count`. Puedes ver un ejemplo sobre esto en codepen.

Volviendo a las propiedades `row-gap` y `column-gap`, puedes ver sus posibles valores y descripción en la siguiente tabla y un ejemplo de su uso en este codepen.

Propiedad	Posibles valores	Descripción
`row-gap`	`normal` longitud-porcentaje [0, ∞]	Espacio entre filas.
`column-gap`		Espacio entre columnas.

Dado que flex es un sistema para diseños de una sola dimensión, sólo una de las dos propiedades tendrá efecto al mismo tiempo. Si `flex-direction` está establecida en `column` utilizaremos `row-gap` y si está establecida en `row` utilizaremos `column-gap`.

Sin embargo, es posible usar ambas si tenemos `flex-wrap` establecida en `wrap`. En este caso el contenedor será multicolumna, ya que en este caso sí podemos separar elementos por filas y por columnas. Veremos un ejemplo un poco más adelante.

Ten en cuenta que **estas propiedades establecen espacios entre elementos, no entre un elemento hijo y su contenedor padre**.

Sobre los **posibles valores** que pueden tomar hay cosas a tener en cuenta:

- El valor `normal` (por defecto) representa `1em` para contenedores multicolumna y `0px` en cualquier otro caso. Esto significa que en un contenedor multicolumna como en el del codepen anterior `column-gap` es `1em` por defecto. Puedes probar esto indicando `column-gap: 1em` y eliminándolo luego, no verás cambios en la separación de las columnas.
- Cuando se indican valores de longitud en unidades absolutas o relativas (`px`, `pt`, `em`, `rem`…), se aplican directamente.
- Cuando se indica un valor en porcentaje éste se resuelve computando el tamaño del *content box* del contenedor.
- Los valores negativos son inválidos según la especificación.

La propiedad `gap` es un atajo para establecer el valor de las dos propiedades:

```
gap: <row-gap> <column-gap>
```

Si solo se indica un valor éste se toma para las dos propiedades.

5. Propiedades de alineación de ítems

En esta sección conoceremos las propiedades que necesitamos para colocar los ítems dependiendo de nuestro objetivo. Vamos a echar un vistazo a 4 propiedades interesantes para ello:

Propiedad	Posibles valores		Descripción

`justify-content`	`flex-start (start)`
	`flex-end (end)`
	`center`
	`left | right`
		`space-between`
		`space-around`
		`space-evenly`
		`strech`
			Alinea los ítems en el eje principal (por defecto, el horizontal).

`align-items`	`flex-start`
	`flex-end`
	`center`
		`stretch`
		`baseline`
			Alinea los ítems en el eje transversal (por defecto, el vertical).

`align-content`	`flex-start`
	`flex-end`
	`center`
		`space-between`
		`space-around`
		`stretch`
			Alinea los ítems en el eje transversal, pero solo tiene efecto en contenedores flex multilínea.

`align-self`	`auto`
	`flex-start`
	`flex-end`
		`center`
		`stretch`
		`baseline`	

5.1. *justify-content*

La propiedad `justify-content` define cómo se distribuye el espacio *entre* y *alrededor de* los ítems a lo largo del eje principal de un contenedor flex.

Los posibles valores lo puedes encontrar en la tabla siguiente, que es un resumen de la sintaxis formal descrita en la especificación o en developer.mozilla.org:

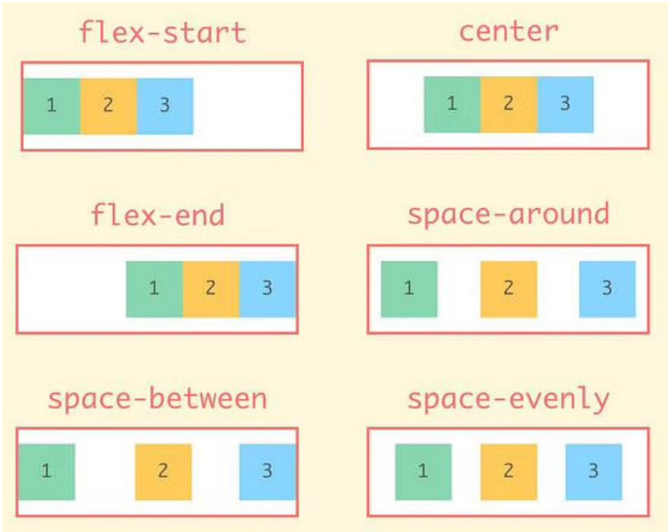
Para resumir y por motivos didácticos, hemos establecido en la tabla que el valor por defecto es `flex-start` o `start`, aunque esto no es realmente así. El valor por defecto (`initial`) es `normal`, que se comporta como `stretch`, y como `strech` se comporta como `start` en contenedores flex, `normal` también se comporta como `start`.

Además, la especificación indica que puede tomar dos valores que no vienen recogidos en la tabla, `safe` y `unsafe`. El soporte de los navegadores para estas propiedades es aún muy bajo, por lo que nos centraremos en el resto, que están ampliamente soportados.

Valor		Descripción
content position	`flex-start (start)`	Agrupa los ítems al comienzo del eje principal.
	`flex-end (end)`	Agrupa los ítems al final del eje principal. De uso exclusivo en layouts flex.
	`center`	Agrupa los ítems en el centro del eje principal.
content distribution	`space-between`	Distribuye el espacio entre los ítems dejando uno al inicio y otro al final.
	`space-around`	Distribuye el espacio entre ítems dejando el mismo espacio entre los ítems (como si fuera un cilindro).
	`space-evenly`	Distribuye el espacio entre ítems dejando el mismo espacio entre los ítems y hasta los bordes.
	`stretch`	Equivalente en esta propiedad a `flex-start` o `start`.
	`left | right`	Posiciona todos los ítems juntos a la derecha o izquierda del contenedor.

Puedes ver esta propiedad en acción en el este codepen.

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:



En este punto el lector podría preguntarse por qué hay dos pares de valores que parecen hacer lo mismo, `flex-strat` y `start` por un lado y `flex-end` y `end` por otro. En realidad, no son exactamente lo mismo. Los valores `flex-start` y `flex-end` son valores de uso exclusivo con layouts flex, mientras que `start` y `end` son valores que se pueden usar en propiedades con otros tipos de layouts, como grid, que veremos más adelante.

La reciente especificación *Box Alignment* añadió una serie de valores, como `start`, `end`, `right` o `left`, que no existían antes. Con esta nueva especificación, W3C pretende establecer un lenguaje universal para alinear elementos en CSS. Es posible que, con el tiempo, los valores de *Box Alignment* acaben reemplazando los valores particulares definidos para flex, pero por el momento **es mejor utilizar** `flex-start` **y** `flex-end`, dado que el soporte de los navegadores es mucho mejor para estos valores. Puedes ver que es soportado por todos los navegadores desde 2014, mientras que `start` y `end` solo desde 2022.

Por otra parte, **los valores** `flex-strat`**,** `start`**,** `flex-end` **y** `end` **son relativos al flujo del texto**. Esto significa que `flex-strat` y `start` siempre se orientan hacia el inicio del texto (arriba a la izquierda para idiomas de izquierda a derecha y de arriba abajo, como el español o el inglés) y `flex-end` y `end` se orientan hacia el final del texto. Puedes probar este comportamiento modificando en el contenedor flex la dirección del flujo de texto de derecha a izquierda con `direction: rtl`.

Puedes ver el comportamiento de `justify-content` en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_justify-content

5.2. align-items

La otra propiedad importante de este apartado es `align-items`, que se encarga de **alinear los ítems en el eje transversal** del contenedor. Hay que tener cuidado de no confundir `align-content` con `align-items`, puesto que la primero actúa sobre cada una de las líneas de un contenedor multilínea (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 transversal.
`flex-end`	Alinea los ítems al final del eje transversal.
`center`	Alinea los ítems al centro del eje transversal.
`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.

Puedes ver el comportamiento de `align-items` en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_align-items

5.3. *align-content*

La propiedad `align-content` es un caso particular de `align-items`.

Puedes ver esta propiedad en acción en este codepen.

Cuando el valor de `flex-wrap` es `nowrap`, esta propiedad no hace nada, no entra en acción. Pero cuando es `wrap` o `wrap-reverse` tenemos un contenedor multilínea y podemos diferenciar dos situaciones:

- Cuando los ítems sí caben en el contenedor: Esta propiedad se comporta exactamente igual que `align-items`.
- Cuando los ítems no caben en el contenedor: Esta propiedad alinea los ítems en el eje transversal, pero dividiendo el espacio disponible entre el número de líneas necesarias para albergar a los ítems.

Los valores que puede tomar son los siguientes:

Valor	Descripción
`flex-start`	Agrupa los ítems al principio del eje transversal.
`flex-end`	Agrupa los ítems al final del eje transversal.
`center`	Agrupa los ítems al centro del eje transversal.
`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.
`space-evenly`	Distribuye el espacio entre ítems dejando el mismo espacio entre los ítems.
`stretch`	Estira los ítems para ocupar de forma equitativa todo el espacio.

Puedes ver el comportamiento de esta propiedad en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_align-content

5.4. *place-content* (atajo)

Existe una propiedad de atajo con la que se pueden establecer los valores de las propiedades `align-content` y `justify-content` de una sola vez. Dicha propiedad es `place-content` y funciona de la siguiente forma:

Con 1 parámetro:

```
place-content: flex-start;
```

Es equivalente a:

```
align-content: flex-start;
justify-content: flex-start;
```

Con 2 parámetros:

```
place-content: flex-start flex-end;
```

Es equivalente a:

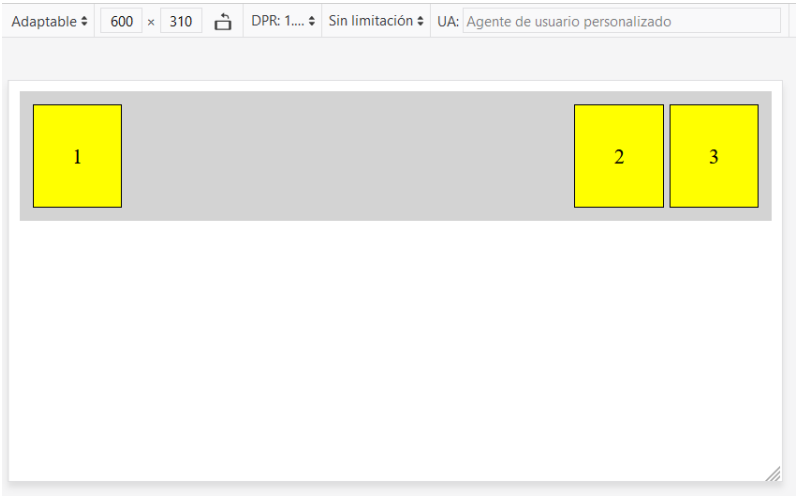
```
align-content: flex-start;
justify-content: flex-end;
```

Con 3 parámetros establece las 3 propiedades en este orden: `flex-grow`, `flex-shrink` y `flex-basis`. Por defecto es `0 1 auto`.

```
place-content: <flex-grow> <flex-shrink> <flex-basis>
```

Ejercicio propuesto 1. (solución en codepen**)**

Utilizando flex, intenta conseguir la siguiente disposición de elementos, donde cada elemento flexible tiene un padding de 20px sin altura ni anchura definida.

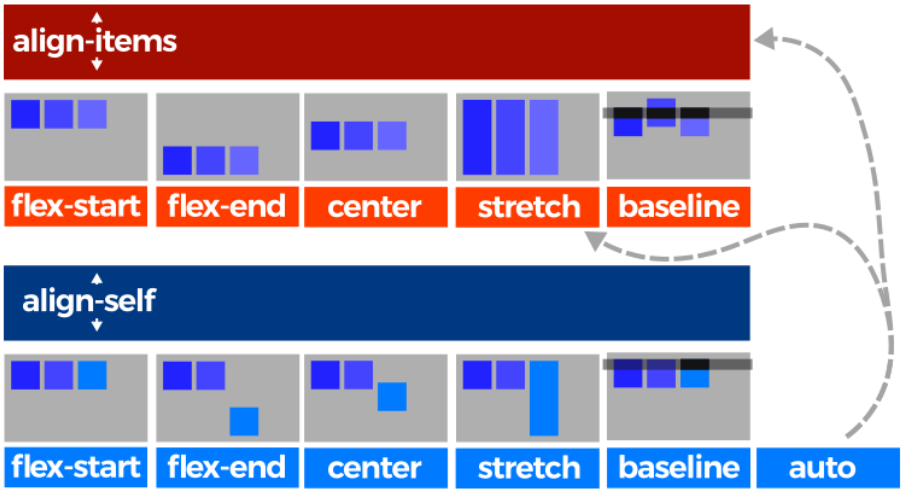


Ejercicio propuesto 2. (solución en codepen**)**

Utilizando flex, intenta conseguir la siguiente disposición de elementos, donde cada elemento flexible tiene un padding de 20px sin altura ni anchura definida.

5.5. align-self

Por otro lado, la propiedad `align-self` actúa exactamente igual que *align-items*, salvo 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 sobrescribirlo 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. En caso de que no se haya establecido, el valor por defecto `stretch`.

Puedes ver el comportamiento de esta propiedad en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_align-self

6. Propiedades de flexibilidad

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	Valores posibles	Descripción
`flex-basis`	`Tamaño | content`	Define el tamaño base de los ítems antes de aplicar la distribución de espacio. El valor *content* ajusta automáticamente el tamaño al contenido del ítem.
`flex-grow`	`0 | [número]`	Número que indica el factor de crecimiento del ítem.
`flex-shrink`	`1 | [número]`	Número que indica el factor de decrecimiento del ítem.

6.1. flex-grow

La propiedad `flex-grow` especifica cómo se reparte el espacio restante entre cada ítem dentro de un contenedor `flex` en la dirección del eje principal. Su valor por defecto es 0 (el ítem no crece para rellenar el espacio libre).

El espacio restante es el tamaño del contenedor menos la suma del tamaño de todos los elementos flexibles juntos. Si todos los ítems dentro del contenedor tienen el mismo factor de crecimiento, todos los elementos reciben la misma cantidad del espacio restante. De lo contrario, el espacio restante se distribuye en función de los diferentes factores de crecimientos de cada ítem.

Puedes ver el comportamiento de esta propiedad en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_flex-grow

6.2. *flex-shrink*

La propiedad `flex-shrink` es la opuesta a `flex-grow`. Mientras que la anterior indica un factor de crecimiento, `flex-shrink` aplica un factor de decrecimiento en el eje principal. Su valor por defecto es 1.

6.3. *flex-basis*

Define el tamaño base por defecto que tendrán los ítems antes de aplicarle una cierta distribución de espacio. Se puede aplicar un tamaño específico (en unidades, píxeles, porcentajes…), pero también se puede aplicar la palabra clave `content`, que ajusta automáticamente el tamaño al contenido del elemento. Este es el valor por defecto. La palabra clave `auto` se comporta del mismo modo.

Si al contenedor aplicamos `flex-direction: column` invertiremos los ejes primario y transversal y `flex-basis` actuará sobre `height` en lugar de sobre `width`.

Si se aplica un valor a `flex-basis` y a `width` al mismo tiempo (o a `height` si `flex-direction: column`), prevalecerá el valor de `flex-basis`.

También puedes probar `flex-basis` en el siguiente enlace:

https://www.w3schools.com/cssref/tryit.asp?filename=trycss3_flex-basis

6.3.1. Diferencias entre *flex-basis* y *width*

Según la especificación de W3C, `flex-basis` se resuelve del mismo modo que `width` para valores distintos de `auto` y `content`. Para esos dos valores, `flex-basis` ajusta la anchura a la anchura de su contenido.

6.3.2. *flex-basis* con *flex-grow*

Observa este ejemplo (ver en codepen):

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>flex-basis + flex-grow</title>
  <style>
    * {box-sizing: border-box;}
    body {padding: 20px 0;}
```

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

    .item {
      flex-basis: 220px;
      height: 100px;
      background-color: aquamarine;
      border-radius: 6px;
    }

    .item:nth-child(1) {
      background-color: #EA3556;
      flex-grow: 1;
    }

    .item:nth-child(2) {background-color: #61D2D6;}
    .item:nth-child(3) {background-color: #EDE5E2;}
    .item:nth-child(4) {background-color: #ED146F;}
    .item:nth-child(5) {background-color: #EDDE45;}
    .item:nth-child(6) {background-color: #9BF0E9;}
  </style>
</head>

<body>
  <div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
  </div>
</body>

</html>
```

Aunque todos los ítems tienen establecido `flex-basis: 200px`, el primero de ellos ocupa más de esos 200px debido a que tiene establecido `flex-grow: 1`.

Si eliminamos ese `flex-grow` del primer ítem:

Lo mismo ocurrirá con flex-basis, como veremos a continuación.

6.3.3. *flex-basis* con *flex-shrink*

Hay que recordar que el valor por defecto de `flex-shrink` es 1, lo cual permite que el ítem flexible se encoja por defecto. Esto es así para evitar que los ítems hijos desborden la anchura del contenedor.

Este valor por defecto tiene una consecuencia importante. Aunque nosotros indiquemos `flex-basis: 100px`, puede ocurrir que el ítem realmente no ocupe 100px porque se esté encogiendo.

Para mantener esos 100px y evitar que el ítem se encoja, tenemos que cambiar el valor de `flex-shrink`.

```
div {
  width: 100px;
  flex-shrink: 0;
}
```

O también:

```
div {
  flex-basis: 100px;
  flex-shrink: 0;
}
```

O, aún mejor como recomienda la especificación, usar la propiedad `flex`:

```
flex: 0 0 100px;  /* no crezcas, no te encojas, quédate en 100px */
```

En el ejemplo anterior en codepen, puedes probar a establecer `flex-basis: 1000px` a cada ítem y eliminar `flex-grow` del primer ítem. Obtendrás:

Es evidente que esos ítems no ocupan 1000px de anchura, se están encogiendo. ¿Por qué? Porque, por defecto, `flex-shrink` es 1 y obliga a los ítems a encogerse para evitar desbordamientos. Pero si ponemos `flex-shrink: 0` a la clase ítem, veremos un desbordamiento masivo y un horrible scroll horizontal.

Por supuesto, podemos evitar ese desbordamiento con `flex-wrap: wrap` en el contenedor. Otra cosa es que sea ese comportamiento el que deseamos.

6.4. Atajo: flex

Existe una propiedad llamada `flex` que sirve de atajo para estas 3 últimas propiedades. Funciona de la siguiente forma:

- Con 1 parámetro establece `flex-basis` o `flex-grow` (dependiendo de si el valor es un número o un tamaño). Establecer `none` equivale a `0 0 auto`.
- Con 2 parámetros establece `flex-grow` y `flex-shrink`.
- Con 3 parámetros establece las 3 propiedades es el orden: `flex-grow`, `flex-shrink` y `flex-basis`. Por defecto es `0 1 auto`.

7. Orden de los ítems: *order*

La propiedad `order` establece el orden de los ítems independientemente de su ubicación en el código.

Por defecto, todos los ítems flex tienen `order: 0`. 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. De esta forma podemos recolocar fácilmente los ítems incluso utilizando *media queries*.

Puedes ver el comportamiento de esta propiedad en:

https://www.w3schools.com/cssref/playdemo.php?filename=playcss_order

8. Webgrafía

- https://lenguajecss.com/css/maquetacion-y-colocacion/flex/