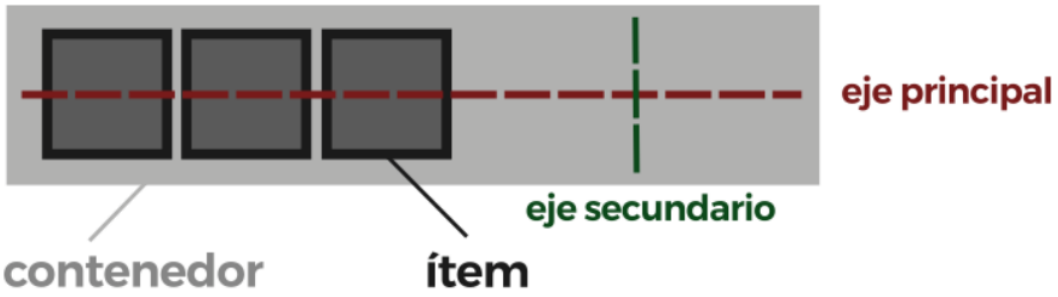
FLEX

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:

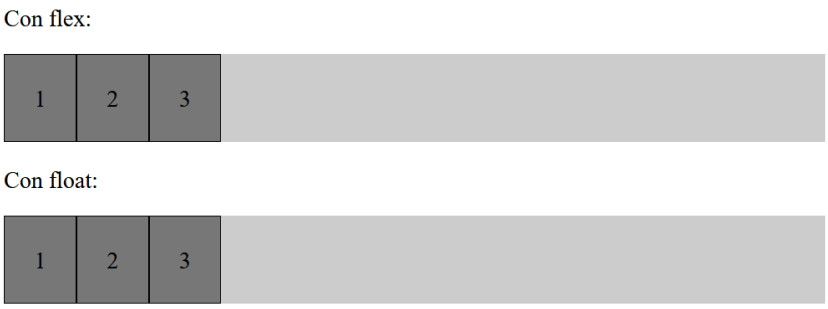
```
<div id="contenedor"> <!-- contenedor flex -->
  <div class="item item">1</div> <!-- ítem flexible -->
  <div class="item item">2</div> <!-- ítem flexible -->
  <div class="item item">3</div> <!-- ítem flexible -->
</div>
```

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.

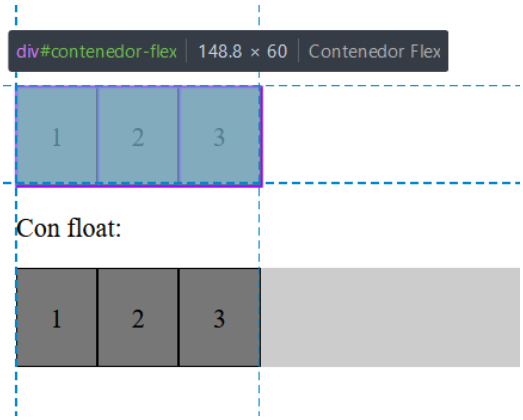
- `flex` establece un contenedor flexible en bloque, de forma equivalente a *block*.
- `inline-flex` establece un contenedor flexible en línea, de forma equivalente a `inline-block`.

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

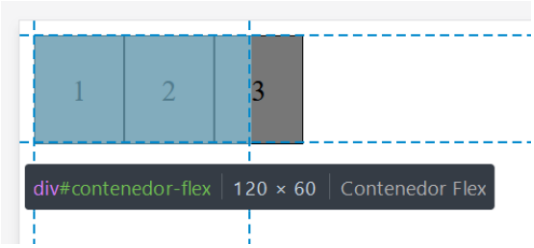
También en el siguiente codepen.

3.2. `flex-wrap`

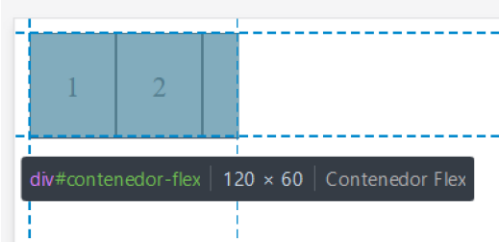
Con la propiedad `flex-wrap` especificamos el comportamiento del contenedor en caso de desbordamiento. Los valores posibles son:

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.



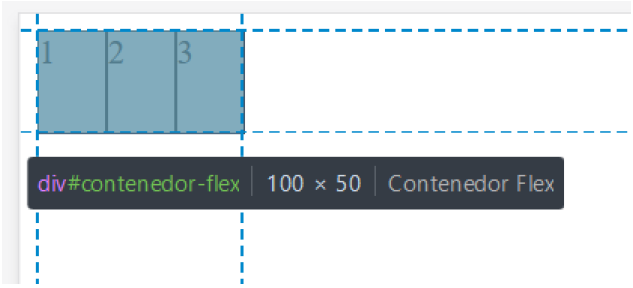
Si no queremos que se vea el contenido desbordado, podemos indicárselo al contenedor con `overflow: hidden`.



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;
}
```

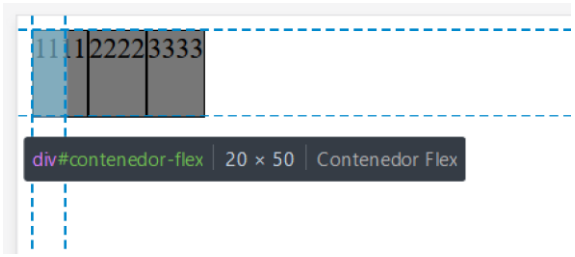


Los hijos se encogerán hasta que su contenido lo permita. Es decir, si hacemos que su contenido sea más ancho (aumentado el `padding` o aumentando el texto), aunque disminuyamos la anchura del contenedor los hijos van a seguir ocupando el espacio que necesitan.

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`

Las siguientes 3 propiedades, conocidas como ***gutters***, permiten establecer un *gap* (separación) entre los ítems hijos de un contenedor flex. Estas propiedades no formaban parte de la especificación inicial de flex, aparecieron con grid más tarde.

Según la especificación del W3C, estas propiedades se aplican a contenedores flex, contenedores grid 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 el siguiente 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.

Hay que resaltar que casi siempre sólo una de las dos propiedades tendrá efecto al mismo tiempo. Si tenemos `flex-direction: column` utilizaremos `row-gap` y si `flex-direction: row` utilizaremos `column-gap`. La excepción está en `flex-wrap: wrap`. En este caso el contenedor será multicolumna y podremos aplicar ambas propiedades. Veremos un ejemplo un poco más adelante.

Ten en cuenta que **estas propiedades establecen espacios entre ítems flexibles, no entre un ítem 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 sólo 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 ver 4 propiedades interesantes:

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 secundario (por defecto, el vertical).

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

`align-self`	`auto`
	`flex-start`
	`flex-end`
		`center`
		`stretch`
		`baseline`
			Actúa igual que `align-items`, pero se utiliza sobre un ítem específico y no sobre el contenedor.

5.1. `justify-content`

La propiedad `justify-content` define cómo **distribuir los ítems en el 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:

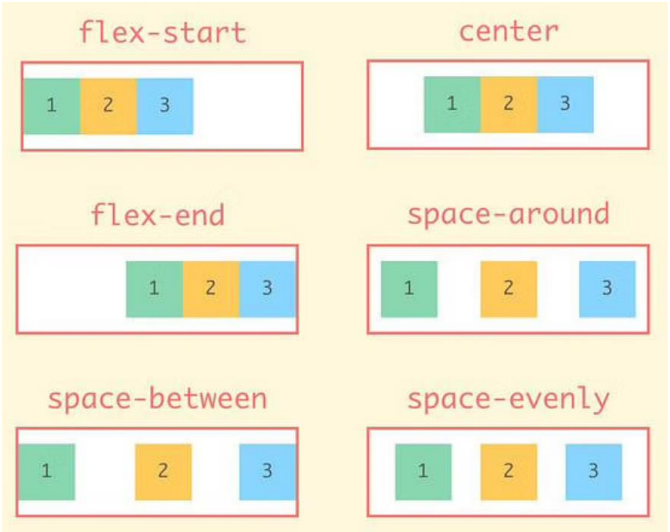
Para resumir y por motivos didácticos, hemos establecido en la tabla que el valor por defecto es `flex-start` o `start`, aunque esto no es realmente así. El valor por defecto (`initial`) es `normal`, que se comporta como `stretch`, y como `strech` se comporta como `start` en contenedores flex, `normal` 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 (desde 2014) que para `start` y `end` (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 **distribuir 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 a lo largo de su línea de base, que es la línea imaginaria en la que se colocan las letras en un texto.

Puedes ver esta propiedad en acción en este codepen y también 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:

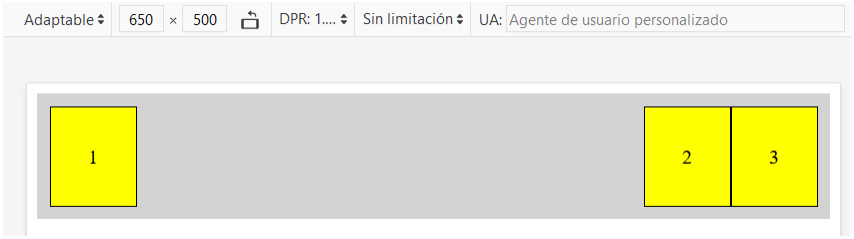
```
place-content: flex-start flex-end;
```

Es equivalente a:

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

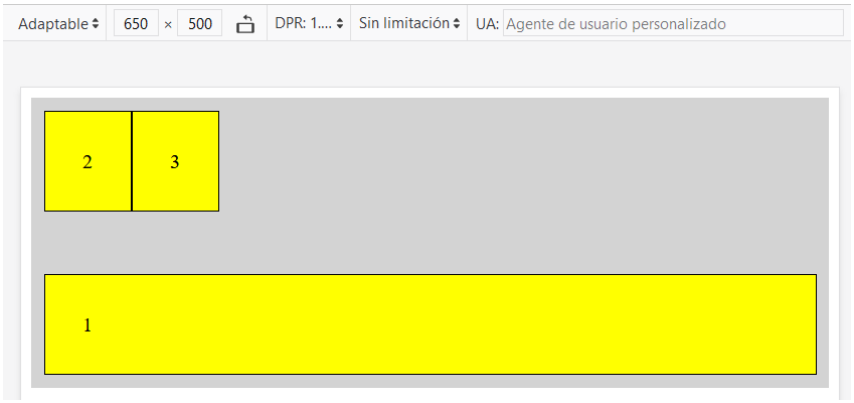
Ejercicio propuesto 1. (solución en codepen)

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



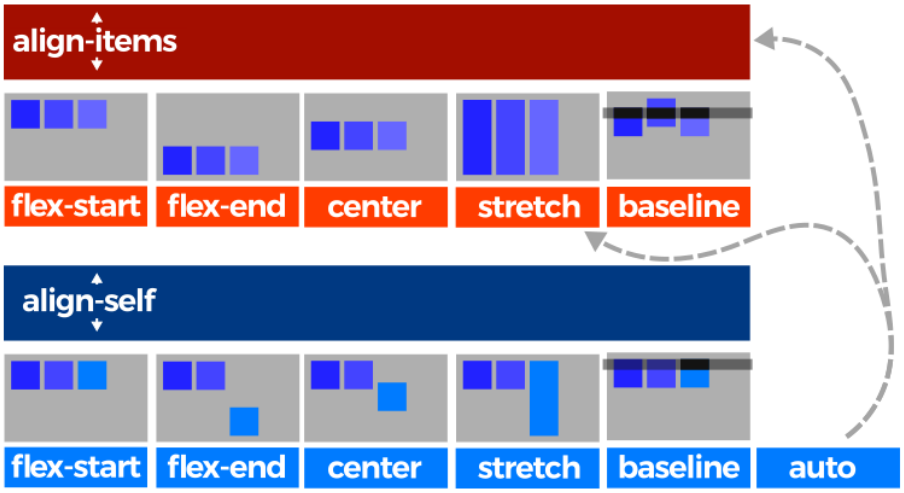
Ejercicio propuesto 2. (solución en codepen)

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`

La propiedad `align-self` **actúa exactamente igual que** `align-items`**, salvo que se utiliza sobre un ítem hijo específico** y no sobre el contenedor.



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.

Esta 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 (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, su valor por defecto, que es `stretch`.

Puedes ver esta propiedad en acción en este codepen y también 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	Posibles valores	Descripción
`flex-grow`	`0 | [número]`	Número que indica el factor de ensanchamiento del ítem en el eje principal.
`flex-shrink`	`1 | [número]`	Número que indica el factor de encogimiento del ítem en el eje principal.
`flex-basis`	`auto | content [tamaño]`	Define el tamaño base de los ítems antes de aplicar cualquier distribución de espacio.

Es importante entender cómo flex calcula los tamaños de los ítems flexibles para entender cómo funcionan estas tres propiedades y cómo interactúan entre ellas.

6.1. `flex-grow`

La propiedad `flex-grow` **especifica cómo se reparte el espacio restante entre cada ítem dentro de un contenedor flex en el eje principal**. Acepta valores numéricos, siendo 0 por defecto, por lo que el ítem no crece para rellenar el espacio libre.

El espacio restante es el tamaño del contenedor menos la suma de todos los tamaños de los elementos flexibles juntos.

Si todos los ítems dentro del contenedor flex tienen el mismo valor de `flex-grow`, entonces 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 este codepen o en w3schools.

Un aspecto **importante** que debemos conocer sobre `flex-grow` es que el tamaño inicial de los ítems importa. El navegador primero calcula cuánto miden los ítems flexibles, suma esas cantidades y, si en el contenedor aún queda espacio, reparte ese espacio entre todos los ítems proporcionalmente a su valor de `flex-grow`. Puedes ver un ejemplo de esto en codepen.

6.2. `flex-shrink`

La propiedad `flex-shrink` es la opuesta a `flex-grow`. Mientras que la anterior aplica un factor de crecimiento, `flex-shrink` aplica un **factor de contracción** en el eje principal. Como en el caso de `flex-grow`, acepta valores numéricos, siendo 1 por defecto.

Este número especifica el factor de contracción, que determina cuánto se contraerá el elemento flexible en relación con el resto de los elementos flexibles en el contenedor cuando se distribuya el espacio libre negativo.

> El factor de contracción flexible se multiplica por el tamaño base antes de distribuir el espacio negativo. Esto distribuye el espacio negativo en proporción a cuánto puede encogerse el objeto. De este modo, un elemento pequeño no se reducirá a cero antes de que un elemento más grande se haya reducido notablemente.

- Si `flex-shrink: 1` (por defecto) entonces si no hay suficiente espacio disponible en el eje principal del contenedor, el elemento se encogerá en un factor de 1.
- Si `flex-shrink: 0` el elemento no se encogerá. Conservará el ancho que necesita, por lo que existe la posibilidad de que el contenido se desborde del contenedor.
- Si `flex-shrink: 2`. Debido a que el valor de contracción flexible es relativo, su comportamiento depende del valor de sus elementos hermanos. Un mayor valor con respecto al de sus hermanos implica una contracción más rápida a medida que el contenedor se hace más pequeño.
- Si asignamos el mismo valor distinto de 0 a todos, todos se encogerán en la misma proporción.

Cuando `flex-shrink` entra en acción, hay que tener en cuenta que los ítems flexibles se encogerán hasta el valor que indique su `min-content`.

Puedes ver el comportamiento de esta propiedad en este codepen.

6.3. `flex-basis`

Esta propiedad establece el tamaño inicial de un elemento flexible en el eje principal antes de que el espacio adicional se distribuya entre los elementos restantes.

Las referencias que haremos a `width` para esta propiedad son para el caso de que el eje principal sea el eje horizontal. Si se ha cambiado a vertical con `flex-direction`, entonces habría que hablar de `height` y no de `width`.

Valor	Descripción
`auto`	El tamaño se establecerá según la propiedad `width`. Si `width` es también `auto`, entonces se aplicará `content`.
`content`	Ajusta el tamaño al contenido. Cualquier valor asignado a `width` es ignorado.
Valores de `width`	Se aplica el tamaño de la misma forma que se haría con `width`.

Según la especificación, `flex-basis` se resuelve de la misma manera que `width` salvo para los valores `auto` y `content`[1], aunque en la mayoría de los casos estos se comportan de la misma manera:

- Cuando el valor es `auto`, el tamaño se establecerá según el valor de `width`. Si `width` es también `auto`, entonces se aplicará `content`.
- Cuando el valor es `content`, el tamaño se establecerá según el tamaño del ítem flexible. Suele ser equivalente a `max-content`.

Cuando trabajamos con elementos flexibles, `flex-basis` tiene prioridad sobre `width`, salvo que se especifique el valor `auto`.

Puedes ver el comportamiento de esta propiedad en este codepen o en:

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

6.3.1. `flex-basis` con `flex-grow`

Observa el siguiente ejemplo en codepen.

Inicialmente todos los ítems tienen `flex-basis: 100px`, lo que establece 100px de anchura a cada uno de ellos. Cuando establecemos `flex-grow: 1` al primer ítem lo que estamos haciendo es decirle que absorba todo el espacio disponible en el contenedor. Lo mismo ocurrirá con `flex-shrink`, como veremos a continuación.

6.3.2. `flex-basis` con `flex-shrink`

Observa el siguiente codepen.

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.

[1] Esto es así para escritura tipos de escritura horizontal, pero como no vamos a diseñar páginas en chino o japonés, esta premisa nos vale.

6.4. `flex` (atajo)

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 en 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. *Responsive design* con `flex`

Vamos a ver cómo amoldar el tradicional sistema de 12 columnas flotantes a las características de flex.

Recordemos, en maquetación tradicional teníamos un sistema de 12 columnas flotantes. Cada columna flotaba a la izquierda, tenía un cierto `padding` e indicábamos que ocupase el 100% de la anchura del contenedor para móviles:

```
[class*="col-"] {
  float: left;
  width: 100%;
  padding: 15px;
}
```

Cada grupo de columnas era envuelto por una fila, la clase `row`, que, en esencia, era lo mismo que el *hack* `clearfix` pero con otro nombre.

```
.row::after {
  content: "";
  clear: both;
  display: table;
}
```

Finalmente, insertábamos las *media queries* que describían zonas delimitadas por breakpoints. Por ejemplo, para tablets:

```
@media only screen and (min-width: 600px) {
  .col-t-1  { width: 8.33%;  }
  .col-t-2  { width: 16.66%; }
  .col-t-3  { width: 25%;    }
  .col-t-4  { width: 33.33%; }
  .col-t-5  { width: 41.66%; }
  .col-t-6  { width: 50%;    }
  .col-t-7  { width: 58.33%; }
  .col-t-8  { width: 66.66%; }
  .col-t-9  { width: 75%;    }
  .col-t-10 { width: 83.33%; }
  .col-t-11 { width: 91.66%; }
  .col-t-12 { width: 100%;   }
}
```

Para escritorio solo necesitábamos repetir la *media query* para escritorio, cambiando el breakpoint de 600px y los nombres de las clases, de `col-t-X` a `col-e-X`. Con esto ya teníamos listo nuestro CSS para comenzar a implementar layouts adaptativos. Podíamos hacer algo así:

```
<div class="container">
  <div class="row">
    <div class="col-e-4 col-t-6">Columna 1</div>
    <div class="col-e-4 col-t-6">Columna 2</div>
    <div class="col-e-4 col-t-12">Columna 3</div>
  </div>
</div>
```

Puedes ver este ejemplo en funcionamiento en el este codepen.

Veamos ahora qué debemos modificar para adaptar este código a las características y propiedades de flex.

Comencemos por la clase `row`. Como las columnas ya no van a usar `float`, nos sobra el hack *clearfix*. Tenemos que añadir `display: flex`, por supuesto, pero también `flex-wrap: wrap` para que el navegador apile las columnas horizontalmente en dispositivos pequeños.

```
.row::after {
  content: "";
  clear: both;
  display: table;
}

.row {
  display: flex;
  flex-wrap: wrap;
}
```

Las columnas ya no flotan, por lo que nos sobra `float`. Sí vamos a mantener `width: 100%` porque nos sigue haciendo falta especificar que las columnas ocupen el 100% en dispositivos pequeños. El `padding` lo mantendremos también.

Con estos cambios ya sería suficiente para que todo funcionara perfectamente. Sin embargo, vamos a añadir `flex: 0 0 auto` a las columnas:

```
[class*="col-"] {
  float: left;
  width: 100%;
  padding: 15px;
  flex: 0 0 auto;
  /* Atajo para:
       flex-grow: 0;
       flex-shrink: 0;
       flex-basis: auto;
  */
}
```

Usamos el atajo `flex: 0 0 auto` para establecer 3 propiedades importantes:

- `flex-grow: 0`.
- `flex-shrink: 0`.
- `flex-basis: auto`.

Los valores de `flex-grow` y `flex-shrink` son 0 porque no queremos que los ítems se estiren o se encojan libremente, ya que vamos a ser nosotros los que indiquemos explícitamente cuánto ocupará cada uno. El valor de `flex-basis` es `auto` por defecto, por lo que es redundante, pero incluso la especificación anima a los desarrolladores a utilizar esta fórmula para evitar sobrescribir los valores y obtener comportamientos indeseados:

> *Authors are encouraged to control flexibility using the flex shorthand rather than with its longhand properties directly, as the shorthand correctly resets any unspecified components to accommodate common uses.*

Puedes ver el ejemplo completo en codepen.

Con esto ya hemos igualado las opciones que teníamos en maquetación tradicional. Sin embargo, no nos vamos a parar aquí. **Vamos a mejorar el comportamiento y las posibilidades de diseño** explotando las propiedades de flex.

Cuando diseñamos un layout con maquetación tradicional tenemos que especificar para cada columna cuánto ocupará en cada zona delimitada por los breakpoints. Por tanto, si queremos añadir una nueva columna tenemos que darle espacio en la fila quitándoselo a alguna o algunas de sus hermanas modificando sus clases. Con flex, podemos añadir o quitar columnas tranquilamente y que el espacio se distribuya automáticamente.

Imagina el siguiente escenario, donde **queremos que nuestras columnas ocupen lo mismo**. El ejemplo completo está disponible en codepen.

```
<div class="row">
  <div class="col">Lorem ipsum dolor sit amet consectetur adipisicing elit.
Aperiam nihil, quibusdam magnam possimus minus cumque maiores qui, architecto
sed quam recusandae impedit eaque soluta vel tenetur illo, laborum accusamus
quis</div>
  <div class="col">Lorem</div>
  <div class="col">Lorem ipsum dolor sit amet</div>
</div>
```

Queremos que el sistema redistribuya el espacio disponible para repartirlo equitativamente al añadir o quitar columnas. Lo podemos hacer fácilmente con flex.

```
.col {
  background-color: #eae0fe;
  border: 1px solid #c5a9fc;
  padding: 15px;
  flex: 1 1 0%;
  /* Atajo para:
    flex-grow: 1;
    flex-shrink: 1;
    flex-basis: 0%;
  */
}
```

La clave aquí es que actúan `flex-grow: 1` y `flex-basis: 0%` conjuntamente.

- Con `flex-basis` indicamos que queremos que, de inicio, cada columna ocupe exactamente el 0%, es decir, 0px.
- Con `flex-grow: 1` indicamos que todas las columnas se repartan el espacio disponible equitativamente.

Incluso podemos ir más allá y combinar columnas en las que indicamos su anchura por zonas y columnas sin anchura delimitada. Por ejemplo:

```
<div class="row">
  <div class="col"></div>
  <div class="col-e-6 col-t-3"></div>
  <div class="col"></div>
</div>
```

Fíjate en la segunda columna. Estamos indicando que queremos un 50% para esta columna en la versión de escritorio y un 33.3% en la de tablets. Y las otras dos columnas se deben repartir el espacio restante en todas las situaciones.

Puedes ver este ejemplo completo en codepen.

Se puede incluso mejorar y añadir más opciones, pero lo que hemos visto hasta ahora es suficiente para hacerse una idea de las tremendas posibilidades de diseño que ofrece flex.

9. Webgrafía

- https://www.w3.org/TR/css-flexbox-1/
- https://www.w3schools.com/css/css3_flexbox.asp
- https://developer.mozilla.org/es/
- https://lenguajecss.com/css/maquetacion-y-colocacion/flex/