ÍNDICE

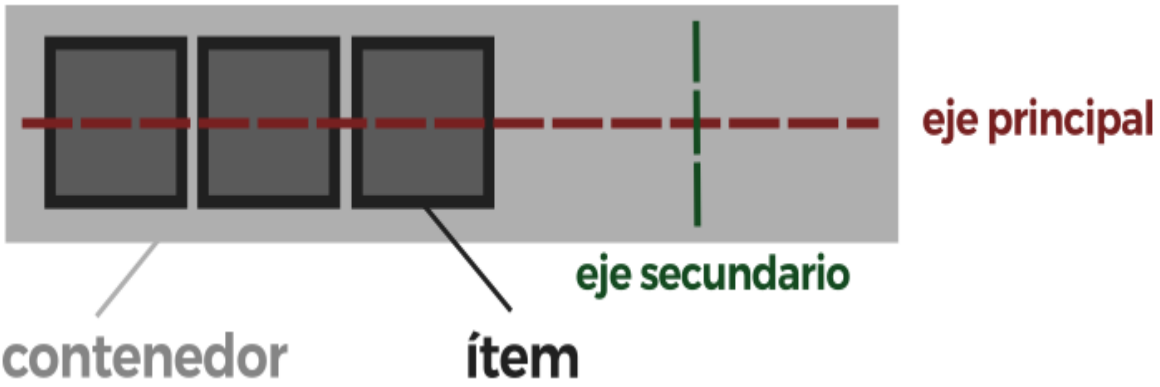
Flexbox CSS

Tradicionalmente, en CSS se ha utilizado el posicionamiento (static, relative, absolute...), los elementos en línea o en bloque (y derivados) o los **float**, lo que a grandes rasgos no dejaba de ser un sistema de creación de diseños bastante tradicional que no encaja con los retos que tenemos hoy en día (sistemas de escritorio, dispositivos móviles, múltiples resoluciones, etc.).

Flexbox es un sistema de **elementos flexibles** que llega con la idea de olvidar estos mecanismos y 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.

1. 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.
- **Ítem**: Cada uno de los hijos flexibles que tendrá el contenedor en su interior.
- **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.

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` que vimos en un capítulo previo, y especificar 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.

inline-flex	Establece un contenedor de ítems flexible en línea, de forma equivalente a inline-block.
flex	Establece un contenedor de ítems flexible en bloque, de forma equivalente a block.

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 desbordarán ni mostrarán los problemas que tienen los porcentajes sobre elementos que no utilizan flexbox.

2. 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:

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 (multilinea).

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.

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**.

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 diseños como el 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 línea (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

Recuerda que 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;
}
```

3. 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:

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):

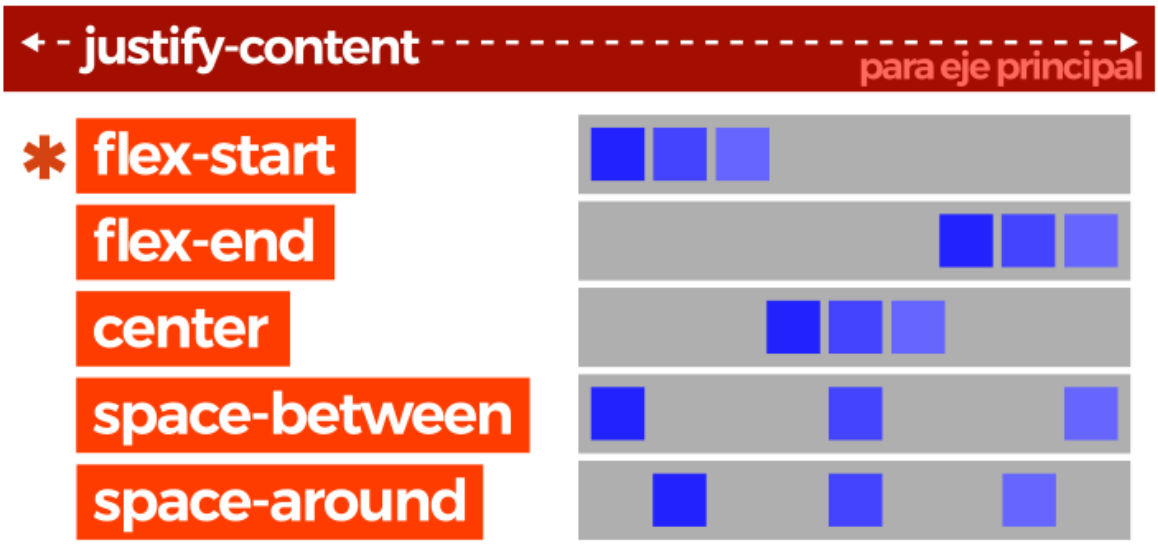
- `justify-content`: Se utiliza para alinear los ítems del **eje principal** (según flex-direction, por defecto, el horizontal).
- `align-items`: Usada para alinear los ítems del **eje secundario** (según flex-direction, por defecto, el vertical).

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** (según flex-direction, por defecto, el horizontal):

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):



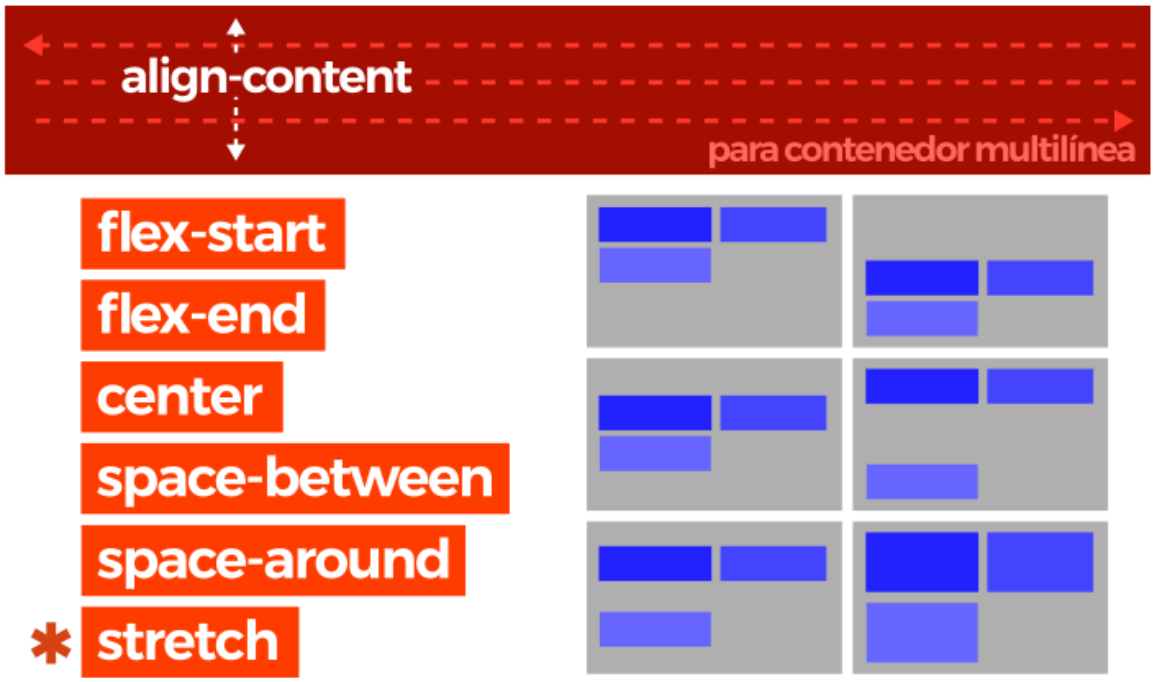
Aunque no aparece en la especificación, se ha añadido posteriormente el valor **space-evenly** similar a **space-around** pero permitiendo compensar el doble espaciado entre los elementos internos y así lograr un diseño más uniforme.

Una vez entendido este caso, debemos atender a la propiedad `align-content`, que es un caso particular del anterior. Nos servirá especialmente 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 hecho, `align-content` solo funciona con `flex-wrap: wrap`

De esta forma, `align-content` servirá para agrupar o distribuir cada una de las líneas del contenedor multilínea. Los valores que puede tomar son los siguientes:

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 como 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-wrap** establecido para tener contenedores multilínea, 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;
}
```

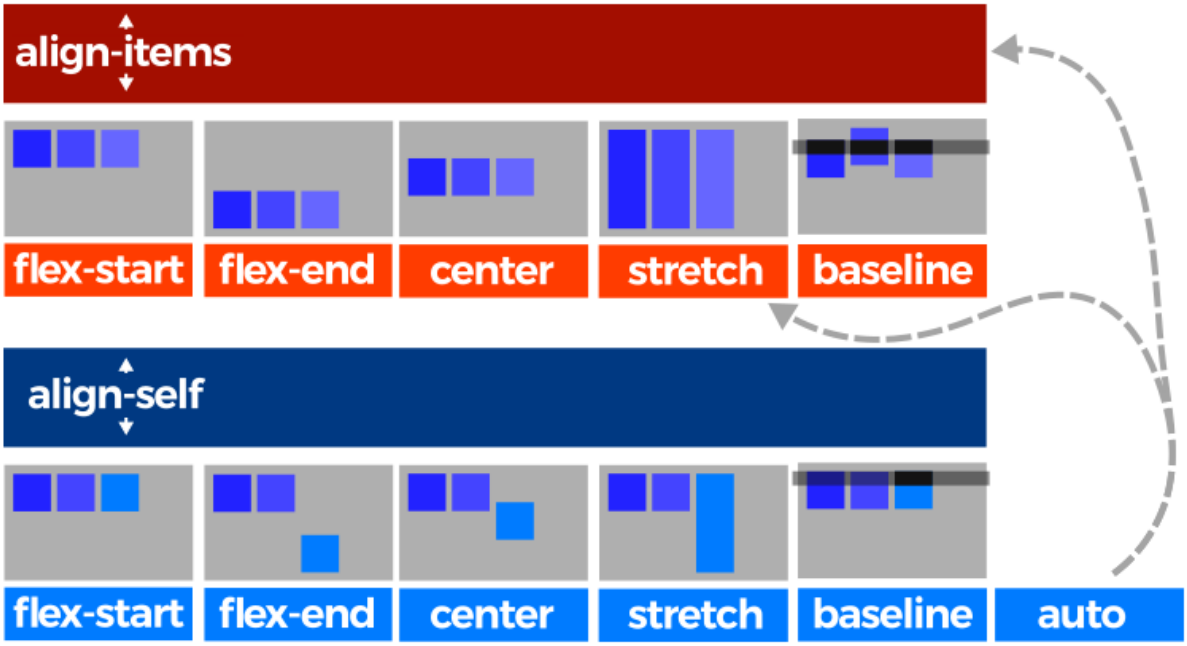
Al igual que con `justiy-content`, se ha añadido posteriormente el valor **space-evenly** con objetivo similar para el estilo `align-content`.

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** (según flex-direction, por defecto, el vertical). Hay que tener cuidado de no confundir `align-content` con `align-items`. El primero puede agrupar las líneas de un contenedor multilínea, mientras que `align-items` distribuye las diferentes líneas por igual en el espacio disponible en el eje secundario y se aplica la alineación por cada una de ellas. Los valores que puede tomar son los siguientes:

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 sobrescribirlo con comportamientos específicos para ítems concretos que no queremos que se comporten igual que el resto. La propiedad puede tomar los siguientes valores:

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**.

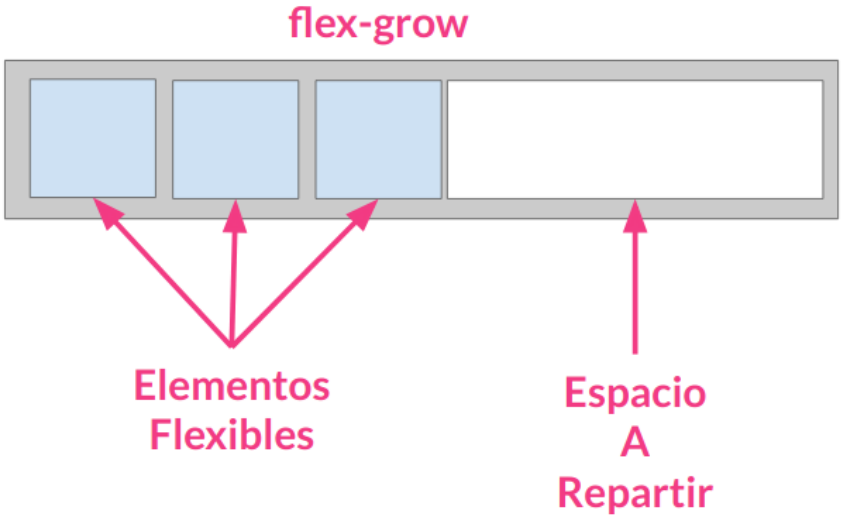
4. Propiedades de ítems hijos

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:

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:	*[tamaño base]* | **content**	Tamaño base de los ítems antes de aplicar variación.
order:	*[número]*	Número 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**.

La propiedad `flex-grow` en definitiva especifica cuánto debe crecer nuestros contenedores flex al rellenar el espacio sobrante en el **eje principal**. Si el valor es **0** como se indica por defecto, entonces no se rellenará este espacio. Si es mayor, se rellenará proporcionalmente.



Dado el siguiente CSS:

```
.contenedor { width: 500px; }
.item-1 { flex-basis: 150px ; flex-grow: 1 }
.item-2 { flex-basis: 150px ; flex-grow: 2 }
.item-3 { flex-basis: 150px ; flex-grow: 1 }
```

Como vemos en el ejemplo anterior, la suma de los flex es menor al tamaño del padre (sobran 50px), en ese escenario el flex de mayor tamaño sería el item-2 siguiendo la fórmula presentada anteriormente quedaría:

```
unit grow = 50/4 = 12.5
```

Sería de 12.5px por unidad:

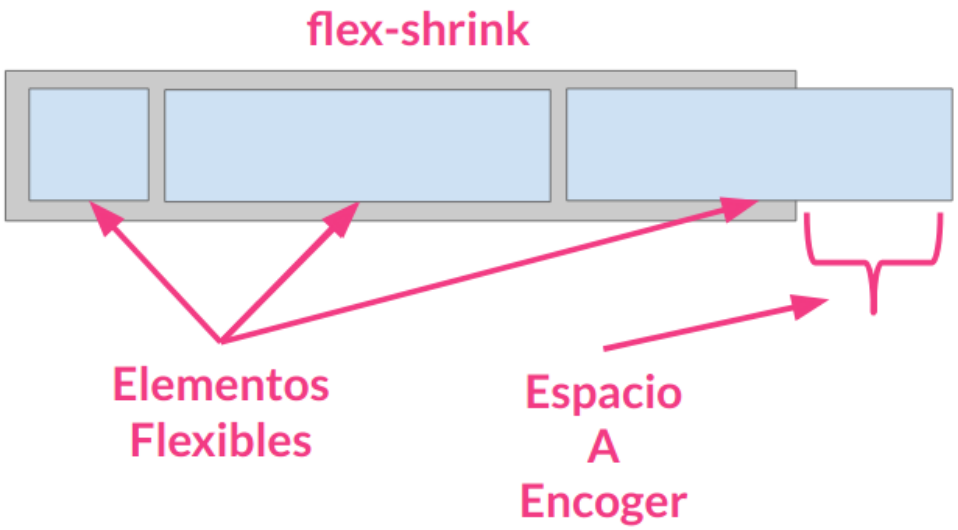
```
.item-1: 12.5px
.item-2: 25px
.item-3: 12.5px
```

Dando el tamaño total:

```
.item-1: 150px + 12.5px = 170.5px
.item-2: 150px + 25px = = 185px
.item-3: 150px + 12.5px = 170.5px
```

En segundo lugar, tenemos la propiedad `flex-shrink` que es la complementaria 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 cómo funciona la propiedad `flex-grow`. Existe un valor especial que es **0**. En realidad, el valor de flex-shrink a partir de **1** permite que aunque se desborde el contenedor, los ítems se adapten aplicando el factor de decrecimiento. Sin embargo, si el valor es **0** los ítems desbordarán el contenedor.

En definitiva, `flex-shrink` especifica el comportamiento de los ítems flexibles cuando el tamaño del contenedor es menor que los propios ítems que lo contienen.



Dado el siguiente CSS:

```
.contenedor { width: 400px; }
.item-1 { flex-basis: 150px ; flex-shrink: 1 }
.item-2 { flex-basis: 150px ; flex-shrink: 2 }
.item-3 { flex-basis: 150px ; flex-shrink: 1 }
```

Siguiendo el ejemplo anterior, el flex con la propiedad doble se encogería el doble de espacio faltante del padre que el flex con la clase simple; si quisiéramos que tuviera se encogiera el triple del espacio faltante del padre se establecería el número a establecer sería 3; en resumen:

```
unit shrink = 50/4 = 12.5
```

Sería de 12.5px por unidad:

```
.item-1: 12.5px
.item-2: 25px
.item-3: 12.5px
```

Dando el tamaño total:

```
.item-1: 150px - 12.5px = 137,5px
.item-2: 150px - 25px = = 124px
.item-3: 150px - 12.5px = 137,5px
```

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 **auto** que ajusta automáticamente el tamaño al contenido del ítem, que es su valor por defecto.

La propiedad `flex-basis` se aplica al eje principal. Si flex-direction es row, tenemos lo siguiente:

```
flex-direction: row;
height: 250px;
flex-basis: 400px;
```

Y si el valor de flex-direction es column:

```
flex-direction: column;
width: 250px;
flex-basis: 400px;
```

Básicamente la propiedad `flex-basis` permite sobrescribir width o height, en función del tipo de contenedor, para posteriormente adaptar estas dimensiones según las propiedades `flex-grow` y `flex-shrink`. Se podría resumir el comportamiento en lo siguiente:

- `flex-basis` controla width o height según la dirección de los items.
- `flex-basis` sobrescribe width o height, a no ser que su valor sea auto.

- Si la dirección de los ítems es row, habrá que especificar el alto con height. Al contrario, si es column, que habría que especificar el ancho con width.
- Las propiedades max-width, max-height, min-width y min-height siguen funcionando y los ítems deben respetar los límites establecidos.

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 manera:

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

Incluso hay otras opciones para abreviar aún más como se indica en el siguiente enlace: https://developer.mozilla.org/es/docs/Web/CSS/flex

Una de las más empleadas es flex: ancho, por ejemplo `flex: 25%`. Aunque el valor de `flex-grow` por defecto es 0, al asignar directamente flex con unas dimensiones (flex-basis), en realidad se aplica el equivalente a `flex: 1 1 25%` (se sobrescribe automáticamente el valor de flex-grow a 1).

5. 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.

6. Espaciado entre ítems

Hasta ahora hemos empleado las propiedades clásicas de **`padding`** y **`margin`** para establecer el espaciado entre los elementos de un contenedor flexible.

Existe la propiedad **`gap`** que facilita la separación interna entre ítems. En realidad, **`gap`** es la abreviatura de **`row-gap`** y **`column-gap`** que asignan de manera individual el espaciado entre filas y columnas respectivamente. Se define como en el código de debajo:

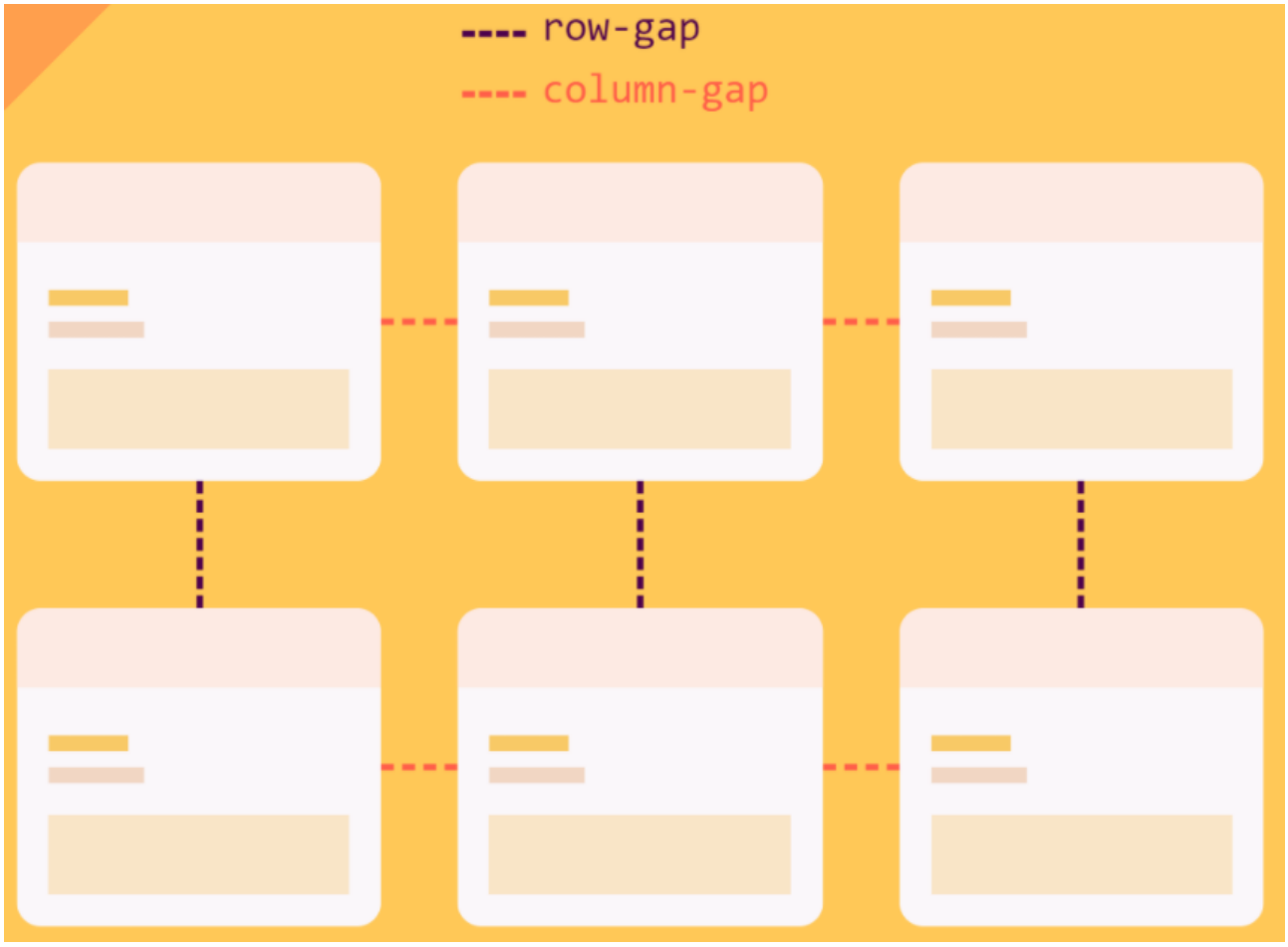
```
#contenedor {
  display: flex;
  row-gap: 20px;
  column-gap: 10px;
}
```

Y la forma abreviada sería tal y como se indica a continuación:

```
#contenedor {
  display: flex;
  gap: 20px 10px;
}
```

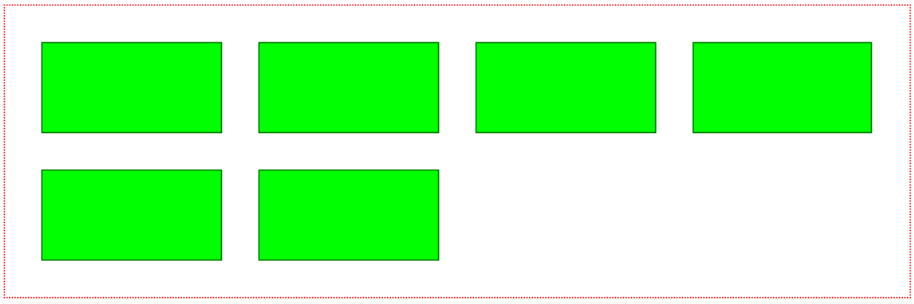
Si el espaciado a nivel de fila y columna es el mismo, entonces bastaría con añadir `gap: 20px`

En definitiva, el diseño sería similar a la siguiente imagen:

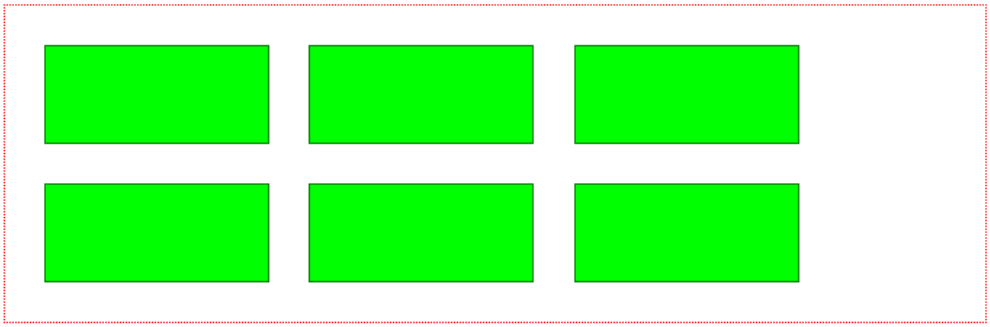


Cabe destacar que `gap` solo funciona en contenedores flexibles, grid o multi-column. No es posible aplicarlo a layouts más clásicos con `float` o `inline-block`.

Por último, hay que tener cuidado, ya que las dimensiones de gap se suman aparte del resto de elementos. Por ejemplo, supongamos el siguiente diseño:



Si el espaciado interno entre las cajas verdes se asigna con `gap: 20px` (el espaciado externo entre el borde rojo y las cajas es un `padding`) y el ancho de cada caja es 25% (cuatro cajas por línea con flex-wrap), entonces hay que compensar el tamaño con `calc` o de alguna otra manera para evitar que haya un salto de línea en una de las cajas como ocurre en la imagen de debajo.



Ahora tenemos en total por cada línea 25% (ancho de cada caja) * 4 = 100%. Y a esto le añadimos los tres espacios internos que se han creado con gap, con un resultado de 20px * 3 = 60px. Por tanto, 100% + 60px nos dejaría sin espacio suficiente para añadir cuatro cajas en la misma línea.