

# Guía FlexBox

---



# FlexBox

---

## Pre-flexbox Float dolor de cabeza

---

Vamos a ver como trabajar con elementos flotantes, que limitaciones tiene y porque hacía falta un nuevo estándar como *Flexbox*.

Vamos a ver esto a base de ejemplos.

Partimos de un *codesandbox* vanilla *javascript*.

Borramos el contenido del *body* de *index.html*

*./index.html*

```
<body>
-   <div id="app"></div>
-
-   <script src="src/index.js">
-   </script>
</body>
```

Y añadimos una imagen así como un texto genérico (*lorem ipsum* lo podéis obtener usando este [generador de texto](#)):

```
<body>
+ <div class="container">
+   
+     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus.
+ </div>
</body>
```

Podemos ver el aspecto que tiene:



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus. Nullam eget gravida elit. Mauris aliquam elementum lacus. Cras ac efficitur nulla. Sed fringilla rhoncus neque et faucibus.

Vamos a darle algo de estilo: crearmos un contenedor y vamos a añadirle un borde para poder verlo claro como queda.

./src/miestilo.css

```
.container {  
    border: 1px solid black;  
    padding: 15px;  
}
```

Ahora a las imágenes que estén bajo ese container vamos a asignarle un ancho de 245px y un margen de 5px

```
.container {  
    border 1px solid black;  
    padding: 15px;  
}  
  
+ .container img {  
+   width: 245px;  
+   margin: 5px;  
+ }
```

Importamos esa hoja de estilo en nuestro html:

*./index.html*

```
<head>  
    <title>Parcel Sandbox</title>  
    <meta charset="UTF-8" />  
+    <link rel="stylesheet" type="text/css" href=".src/miestilo.css" />  
</head>
```

El resultado que tenemos ahora es:



*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus. Nullam eget gravida elit. Mauris aliquam elementum lacus. Cras ac efficitur nulla. Sed fringilla rhoncus neque et faucibus. Nulla finibus orci vel auctor pellentesque. Duis maximus arcu non sem suscipit, sed volutpat arcu vehicula. Morbi hendrerit fringilla velit, sed consequat diam pretium quis. Donec euismod consectetur nisl. Pellentesque varius vestibulum pretium. Mauris mauris sapien, scelerisque at diam a, gravida ornare nibh. Etiam fringilla lacus a massa malesuada venenatis.*

¿Qué pasa si queremos poner el texto a la derecha de la imagen? Podemos plantearnos utilizar la propiedad de css *float:left* y aplicarla a la imagen que tenemos:

```
.container img {  
    width: 245px;  
    margin: 5px;
```

```
+ float:left;
}
```

De esta manera la imagen se queda flotando a la izquierda y a la derecha tenemos el texto, veamos el resultado:



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus.

¡¡ Uy !! Lo tenemos casi, si te fijas sale el texto a la derecha pero se descuadra la imagen, el contenido del elemento flotante sale fuera del contenedor, ya que no sabe cuánto ocupa el elemento flotante, y tendríamos que hacer algo para controlar que ese contenido no sobresalga, podríamos añadir otra propiedad que se llama *overflow* en el que le digamos que hacer cuando se salga todo el contenido del padre.

Antes de continuar vamos a hacer un pequeño ejemplo aparte para explicar esto.

Prueba a quitar la imagen, y vamos a añadir varias líneas de *lorem ipsum*

```
<div class="container">
-   
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc
facilisis,
    quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc
id mi.
    Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla
pellentesque, elementum dignissim libero. Nunc placerat nisl non
blandit
    tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis
quis.
    Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque
at mi
    non nisi tempor accumsan eleifend vel purus.

+   Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc
facilisis,
+   quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc
id mi.
+   Phasellus eu ultricies neque. Maecenas ex elit, consectetur id
```

```

nulla
+     pellentesque, elementum dignissim libero. Nunc placerat nisl non
blandit
+     tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis
mollis quis.
+     Sed placerat nunc est, et molestie turpis efficitur rhoncus.
Quisque at mi
+     non nisi tempor accumsan eleifend vel purus.

+     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc
facilisis,
+     quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc
id mi.
+     Phasellus eu ultricies neque. Maecenas ex elit, consectetur id
nulla
+     pellentesque, elementum dignissim libero. Nunc placerat nisl non
blandit
+     tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis
mollis quis.
+     Sed placerat nunc est, et molestie turpis efficitur rhoncus.
Quisque at mi
+     non nisi tempor accumsan eleifend vel purus.

</div>
</body>

```

Y en los estilos vamos aadir un height en pixeles al contenedor:

`./src/miestilo.css`

```

.container {
    border: 1px solid black;
    padding: 15px;
+ height: 200px;
}

```

Si te fijas ahora el contenido se sale del espacio que hay disponible:

*“Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus.*

¿Qué podemos hacer? indicarle con CSS que si se sale del espacio oculte el contenido, o añade un scroll  
¿Cómo se hace esto? utilizando la propiedad *overflow*, si queremos que no salga el contenido:

```
.container {  
    border: 1px solid black;  
    padding: 15px;  
    height: 200px;  
    + overflow: hidden;  
}
```

Tenemos el siguiente resultado:

*“Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit,*

Si queremos que aparezca un scroll:

```
.container {  
    border: 1px solid black;  
    padding: 15px;  
    height: 200px;
```

```
- overflow: hidden;  
+ overflow: scroll;  
}
```

También podemos usar *overflow: auto*

Ahora podemos ver si aplicamos la misma aproximación a la imagen, restauramos el contenido que teníamos antes en el body y eliminamos el de prueba que hemos añadido:

*./index.html*

```
<body>  
  <div class="container">  
      
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc  
facilisis,  
    quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id  
mi.  
    Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla  
pellentesque, elementum dignissim libero. Nunc placerat nisl non  
blandit  
    tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis  
quis.  
    Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque  
at mi  
    non nisi tempor accumsan eleifend vel purus.  
  </div>  
</body>
```

Y dejamos el CSS como estaba antes:

*./src/miestilo.css*

```
body {  
  font-family: sans-serif;  
}  
  
.container {  
  border: 1px solid black;  
  padding: 15px;  
}  
  
.container img {  
  width: 245px;  
  margin: 5px;  
  float: left;  
}
```

Bueno pues en principio nos las vemos muy felices, sería algo tan fácil como meterle un *overflow hidden* a nuestro contenedor y ¿todos tan contentos no? El trozo de imagen que se sale ya no se vería..

Vamos a meterle a nuestro contenedor el *overflow*

`./src/miestilo.css`

```
body {  
    font-family: sans-serif;  
}  
  
.container {  
    border: 1px solid black;  
    padding: 15px;  
+ overflow: hidden;  
}
```

Si lo probamos veremos que... la foto se ve completa y el contenedor se adapta para tomar todo el espacio de la foto... es decir no se corta, se redimensiona ¿Pero qué clase de brujería es esta?



`Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc facilisis, quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc id mi. Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla pellentesque, elementum dignissim libero. Nunc placerat nisl non blandit tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis quis. Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque at mi non nisi tempor accumsan eleifend vel purus.`

Estos son los trucos "raros" de css que hacen que sea complicado maquetar de forma antigua:

Existen más workaround para arreglar esto, pero no dejan de ser *ñapas* por eso es necesario tirar de un estándar más moderno como es *flexbox*

Vamos a ver otro ejemplo, y vamos a montar un layout, con dos columnas, una con un menú y otra con un contenido.

`./src/index.html`

```
<body>  
- <div class="container">  
-   
```

```

-   Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc
facilisis,
-   quam ac dictum dictum, orci est cursus dui, non accumsan nulla nunc
id mi.
-   Phasellus eu ultricies neque. Maecenas ex elit, consectetur id nulla
-   pellentesque, elementum dignissim libero. Nunc placerat nisl non
blandit
-   tincidunt. Cras vestibulum ultrices quam, quis feugiat turpis mollis
quis.
-   Sed placerat nunc est, et molestie turpis efficitur rhoncus. Quisque
at mi
-   non nisi tempor accumsan eleifend vel purus.
- </div>
+   <div class="menu">
+     <ul>
+       <li>
+         <a href="https://www.google.com">Google</a>
+       </li>
+       <li>
+         <a href="https://www.apple.es">Apple</a>
+       </li>
+     </ul>
+   </div>
+   <div class="content"></div>
</body>

```

Esta vez el css de *miestilo* lo borramos entero y añadimos lo siguiente:

- Estilos para la clase body, queremos que ocupe toda la pantalla, eliminamos márgenes.
- Creamos una clase para el menú:
  - Le decimos que flote a la izquierda.
  - Le damos un ancho en pixeles.
  - El alto le decimos que ocupe todo el *body* (100 viewport height).
  - Le ponemos un color de fondo para diferenciarlo (*cadetblue*).
- Y para el contenido creamos una clase content:
  - Le decimos que flote a la izquierda del que ya estaba (uno al lado del otro)
  - El height el 100% del viewport.
  - El color del fondo *bisque*
  - Y le damos un ancho para que se vea, queremos todo el disponible, eso es: el 100% del view port menos los 150 pixeles que ocupa nuestro menú, para calcular eso usamos la función *calc* de css.

*./src/miestilo.css*

```

body {
  margin: 0px;
  padding: 0px;
}

.menu {

```

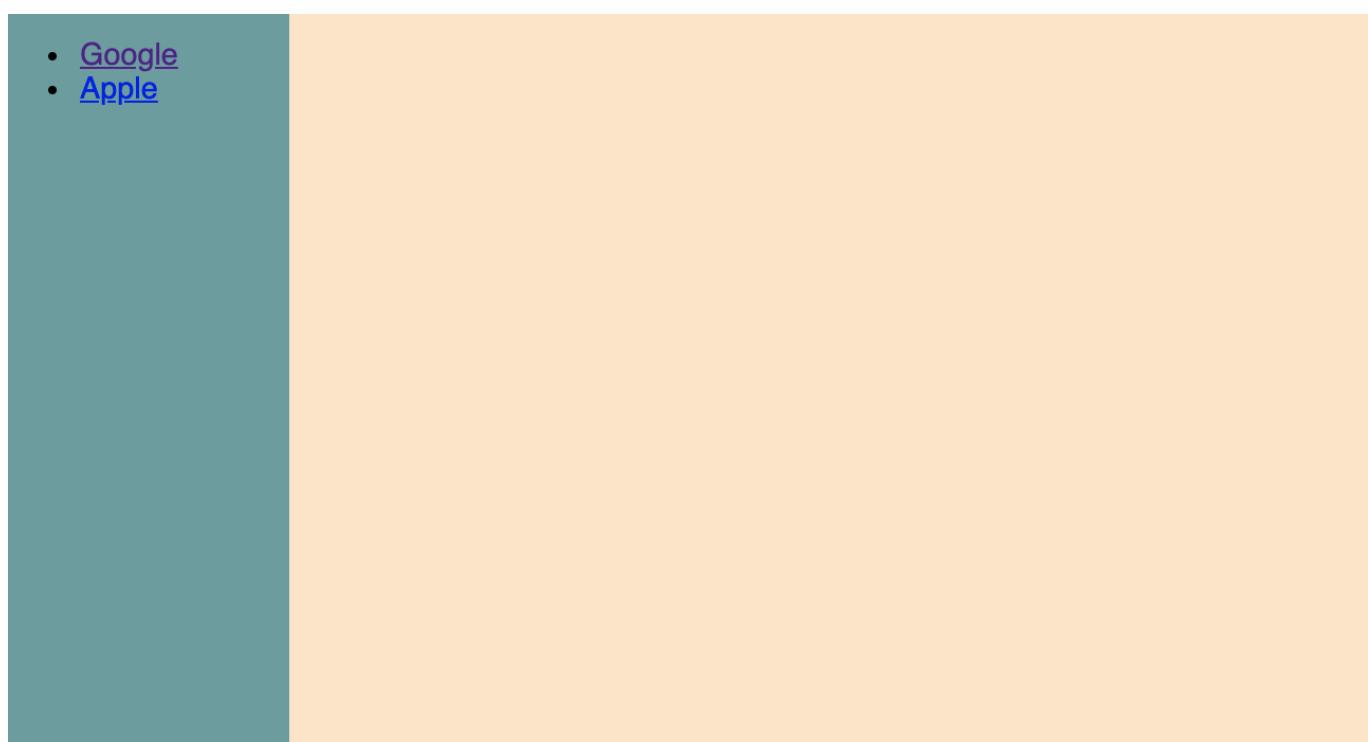
```

float: left;
width: 150px;
height: 100vh;
background: cadetblue;
}

.content {
  float: left;
  height: 100vh;
  background-color: bisque;
  width: calc(100% - 150px);
}

```

Bueno funciona, parece que no ha sido tan complicado ¿No? incluso si redimensionamos se adapta...



ahora nos llega un nuevo requerimiento, queremos añadir un pixel a la izquierda del menú para que se vea como un separador entre el menú y el contenido, ¿Podríamos usar *border* para ello no? Veamos que pasa:

./src/miestilo.css

```

.menu {
  float: left;
  width: 150px;
  height: 100vh;
+ border-right: 1px solid black;
}

```

Si probamos... podemos ver que aparece el borde pero... nuestra área de contenido desaparece  
¿Cóooomooo? ¿Qué ha pasado aquí?

- [Google](#)
- [Apple](#)

¿Dónde ha quedado el *div* del contenido? Si hacemos *scroll* lo podemos encontrar...

- [Google](#)
- [Apple](#)

Un sólo pixel nos ha roto todo el layout que estábamos montando, ahora en vez de 150 pixeles tenemos 151... tendríamos que actualizarlo en el *calc*.

¿Cómo podríamos solucionar esto?

Podríamos añadir en el *calc* el pixel de más:

```
.content {  
  float: left;  
  height: 100vh;  
  background-color: bisque;  
  width: calc(100% - 150px);
```

```
+ width: calc(100% - 151px)
}
```

Otra opción sería tirar de lo que vimos en la sesión anterior y usar el *box-sizing* en el menú con la opción *border-box* así tiene en cuenta el borde como parte de la caja:

```
.menu {
  float: left;
+ box-sizing: border-box;
  width: 150px;
  height: 100vh;
  border-right: 1px solid black;
}
```

Esta solución es buena mientras le metamos *border* si le cambiamos por ejemplo el *margin* esto se vuelve a desmontar 😰.

Esto nos lleva a construir castillos de naipes poco mantenibles.

En la actualidad debemos de evitar el uso de *float*, salvo en situaciones muy específicas.

## Flexbox

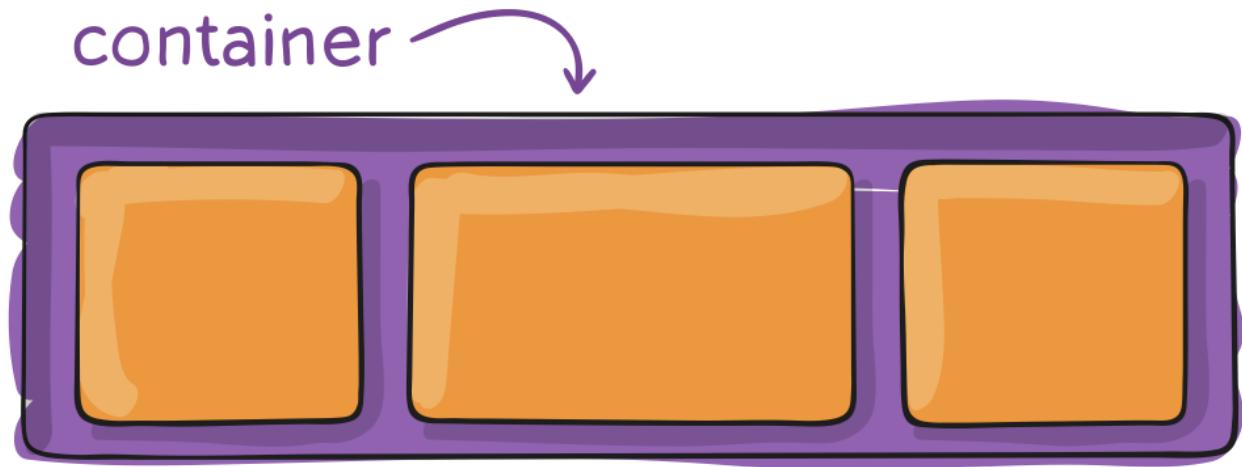
---

*Flexbox* nos permite crear layouts flexibles y responsivos organizando elementos automáticamente. Cuando nos referimos a layout no es sólo el de la aplicación completa como hemos visto antes, si no incluso un listado en el que queremos organizar unas columnas.

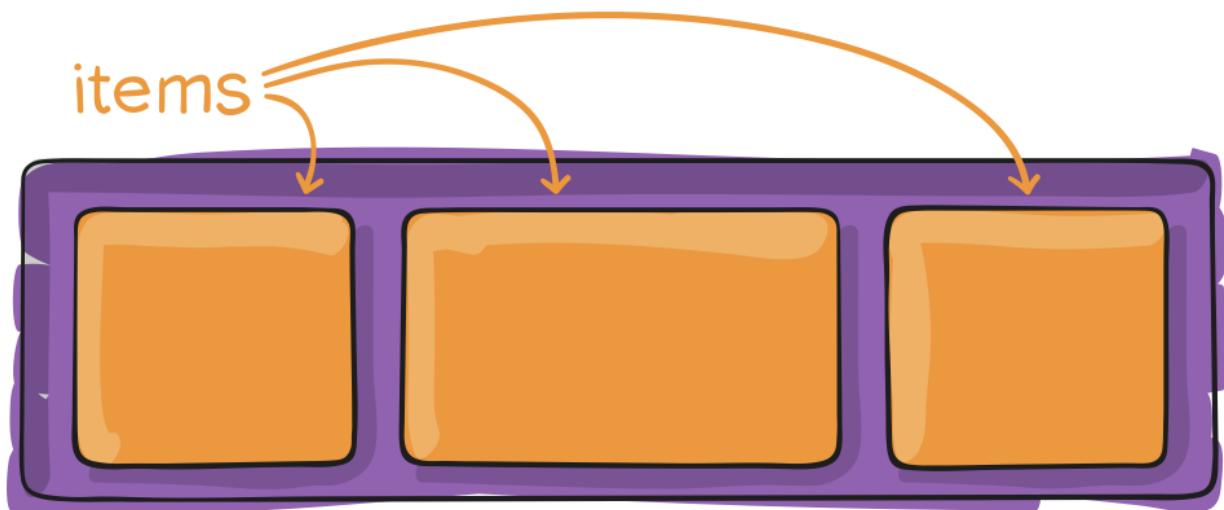
Nos organiza los elementos automáticamente con una serie definiciones por nuestra parte, esto es muy importante.

Los dos conceptos principales con los que vamos a trabajar con *flexbox* son:

- El contenedor: contiene una serie de elementos (*items*) y se encarga de decidir como se van a mostrar los *items* que tiene dentro.



- Los items: son cada uno de los elementos que queremos que distribuya el *container* (están dentro del mismo).



Un ejemplo sencillo en código:

```
<div class="container">
  <p>Item 1</p>
  <div>
    <p>Item 2</p>
    <p>Item 3</p>
  </div>
</div>
```

Cosas a tener en cuenta:

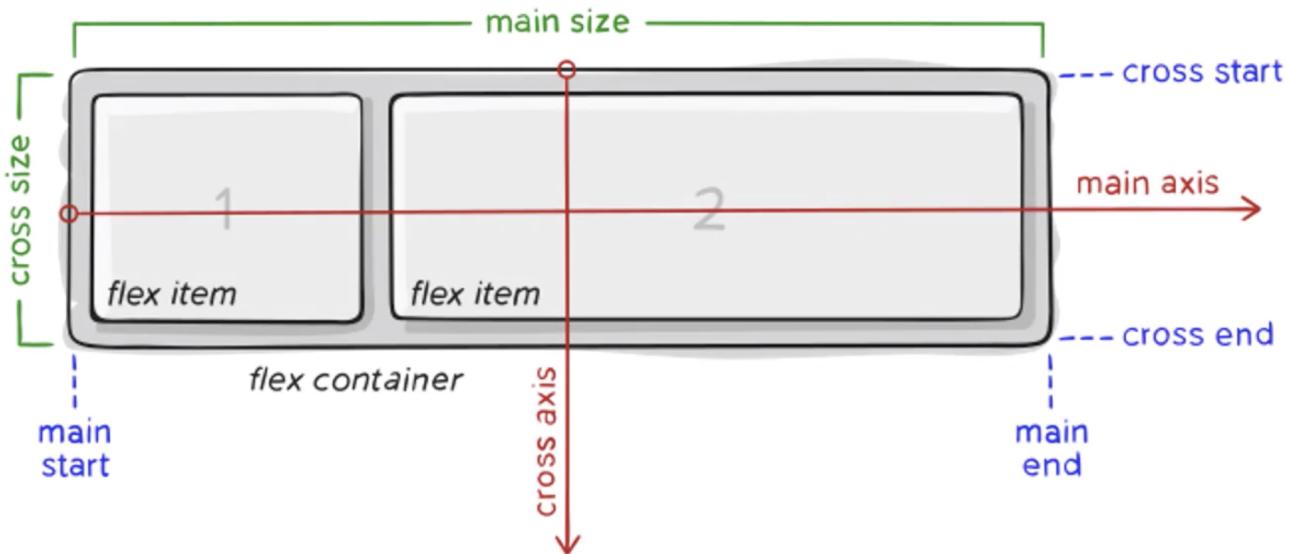
- Un contenedor *flexbox* no se encarga de decir que tamaño tiene, se ajusta al flujo normal del documento.
- Los elementos que tenemos dentro si se encarga *flexbox* de decir como lo organizamos.

## Conceptos clave

### Ejes

*Flexbox* trabaja con dos ejes:

- El eje principal definido por la propiedad *flex-direction*
- El eje secundario (cruzado / cross) que es perpendicular a este.



Cuando creamos un contenedor *flexbox* le podemos indicar cual es el eje principal, y *flexbox* es capaz de extraer cual es el eje secundario (es perpendicular al principal). Es decir si le decimos a *flexbox* que el eje principal es el eje X (horizontal) el sabe que el secundario es el eje Y (vertical). Podríamos hacer justo al contrario.

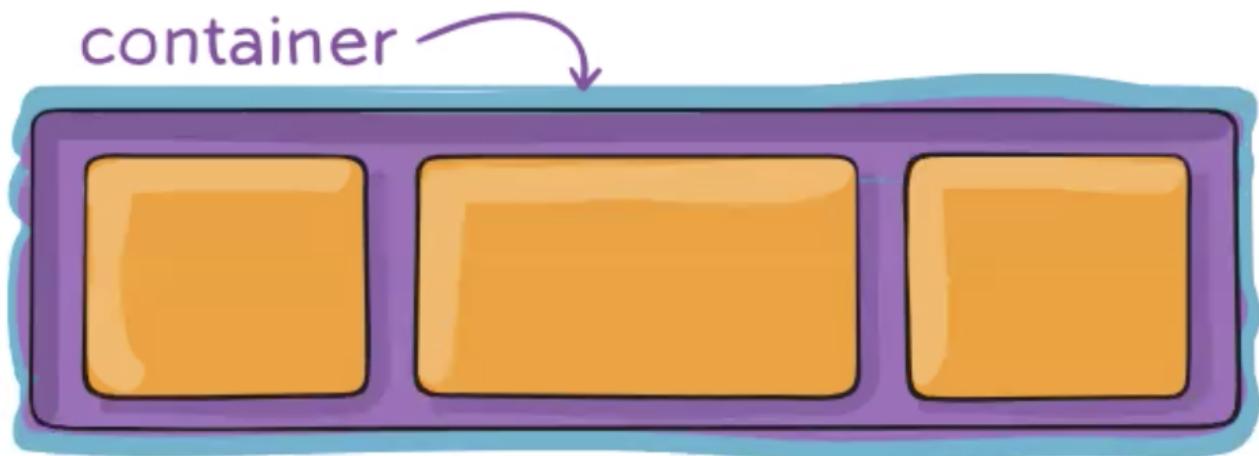
Otro tema interesante es indicarle la dirección, por ejemplo si tenemos un eje horizontal podemos decirle que pinte de izquierda a derecha los *items* o al contrario, lo mismo con el y.

### Display

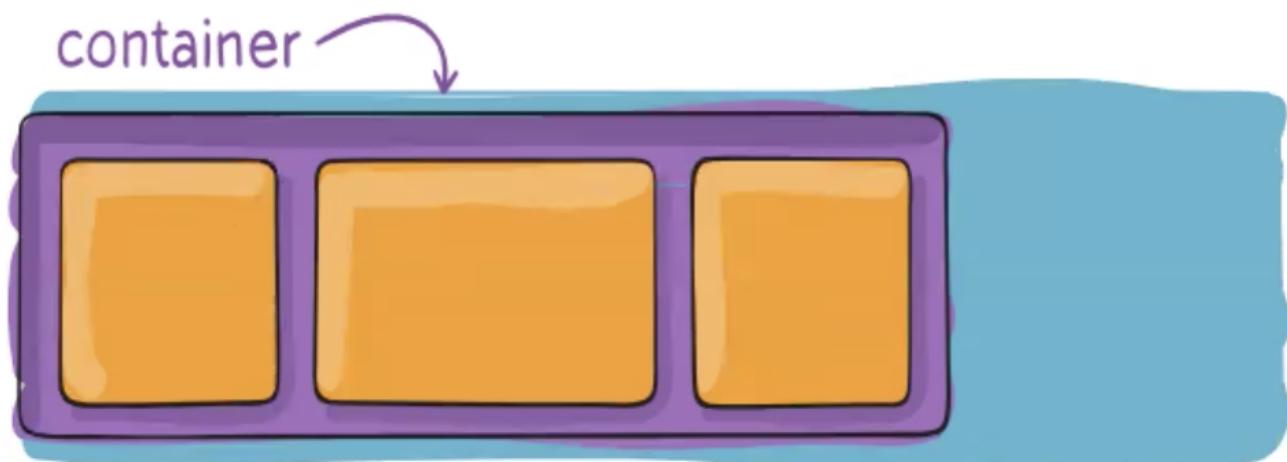
¿Cómo le indicamos al html que vamos a trabajar con *flexbox*? En el contenedor en la propiedad *display* le indicamos que va a tener el valor *flex* o *inline-flex*.

```
display: flex | inline-flex;
```

- Con *display flex* va a ocupar el 100% del espacio, como si fuera un bloque (block).



- Si es inline va a tomar el espacio mínimo para representar los elementos, como si fuera un inline.



## Direction

Una vez que ya hemos definido el contenedor (con `display flex` o `inline-flex`), le podemos indicar la dirección de nuestro eje, esto lo hacemos con la propiedad `flex-direction`.

```
flex-direction: row | row-reverse | column | column-reverse;
```

## Direcciones, eje x, eje y

- [Por defecto] Si le indicamos `flex-direction row` le decimos que nuestro eje principal va a ser el de las filas (eje x), y el secundario el vertical (eje y).
- Si le indicamos `flex-direction column` le estamos diciendo que nuestro eje principal va a ser el de las columnas (eje y) y el secundario el horizontal (eje x).

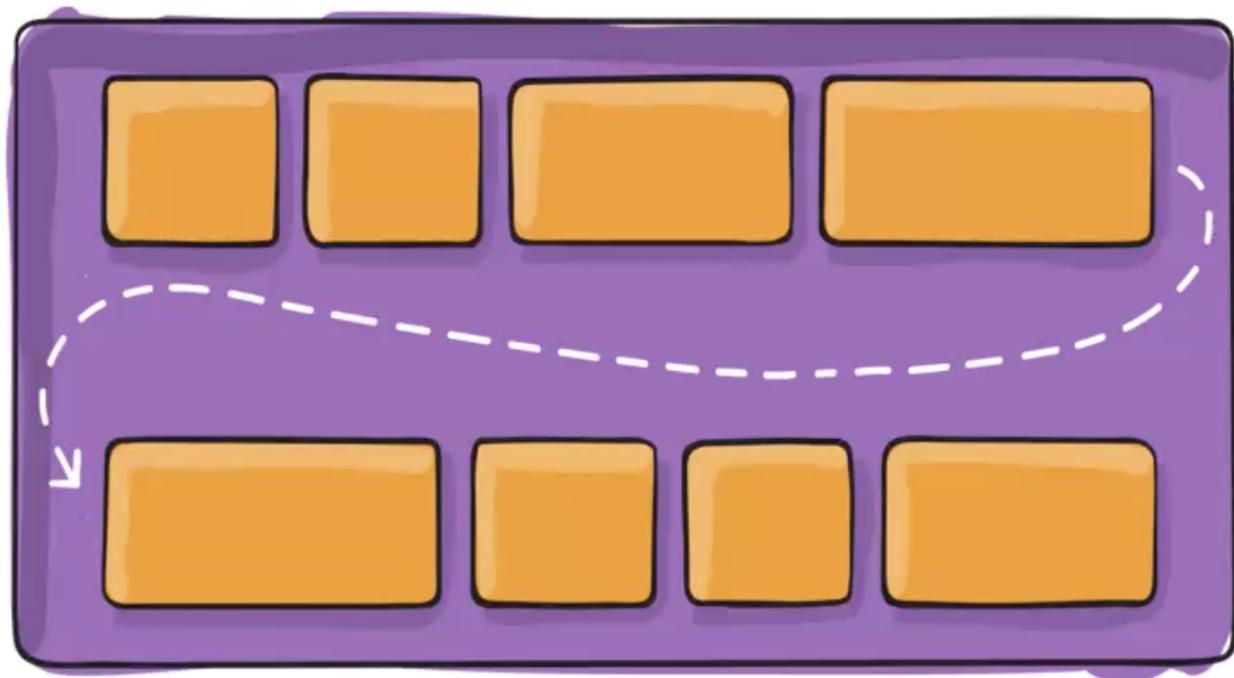
- Otras variantes: *row-reverse* va a pintar los items de derecha a izquierda, y *column-reverse* los pinta de abajo a arriba.

## Flex Wrap

Cuando creamos un contenedor y le añadimos items, nos podemos encontrar que no tengamos espacio suficiente para todos, aquí tenemos que decidir que comportamiento seguir.

Flexbox ofrece una propiedad *flex-wrap* que acepta los siguientes valores, veámoslo con un ejemplo de fila:

- [Por defecto] *nowrap*: aquí le decimos que no crees nuevas filas, que se salga el texto del contenedor, aquí decidimos si le añadimos un *overflow hidden* para que se oculte o un *overflow scroll* para que aparezca una barra de scroll. queremos romper nuestros elementos en varias filas o columnas.
- *wrap*: aquí si no entran los elementos, el contenedor va creando nuevas filas.
- *wrap-reverse*: igual que *wrap*, pero igual que crearse en orden lógico (se añaden filas abajo), los elementos wrapeados pasarían a estar arriba.



**flex-wrap: nowrap | wrap | wrap-reverse;**

Si estuviéramos en modo *columna* pasaría igual pero añadiendo nuevas columnas.

## Propiedades de alineación

### Contenedor

#### Justify-content

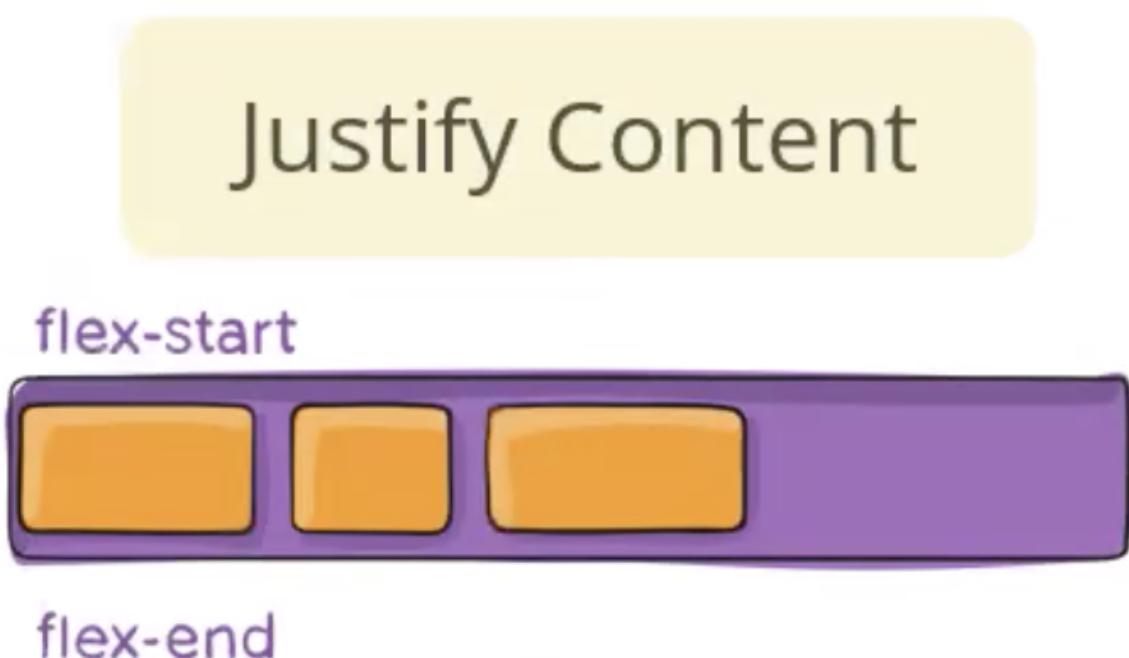
Ya sabemos que podemos organizar nuestros elementos dentro del contenedor en un eje, vamos a ver ahora como decirle a *flexbox* que nos organice esos elemento que tiene dentro, para ello tenemos la propiedad *justify*.

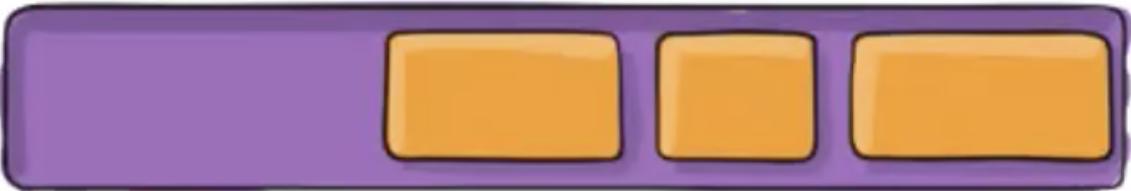
```
justify: flex-start | flex-end | center | space-between | space-around |  
space-evenly;
```

*Justify* afecta al eje principal del contenedor, opciones:

- [Por defecto] *flex-start*: organízalos al principio.
- *flex-end*: al final del eje principal.
- *center*: organízalos en el centro de eje principal:
- *space-between*: en este caso el primero y el ultimo caen al principio y al final del contenedor, sin dejar ningún espacio ni margen, y el resto a una distancia equidistante entre ellos, y reparte el espacio que sobre entre los demás elementos que tenga.
- *space-around*: en este caso todos los elementos van a tener un espacio igual alrededor de ellos, el espacio que sobra lo reparte entre todos los elementos, de forma que cada elemento tiene el mismo espacio a la izquierda y derecha, el efecto: el primero y el último elemento tienen un espacio que es como la mitad de separación entre los que hay en medio.
- *space-evenly*: desaparece el efecto del caso anterior, el espacio entre el principio y el final son iguales, reparte ese espacio.

Si por ejemplo nuestro eje principal fuera el horizontal:





center



space-between



space-around



space-evenly



Con estas propiedades podemos jugar a repartir y organizar nuestros *items* dentro del contenedor en el eje principal.

Más adelante veremos que le podemos indicar a los *items* que se repartan el espacio disponible (por ejemplo podemos decirle a un elemento que coja todo el espacio disponible, en este caso, por ejemplo, un *flex-end* no tendría sentido ya que hay un elemento que va a pillar el espacio disponible de todo el contenedor).

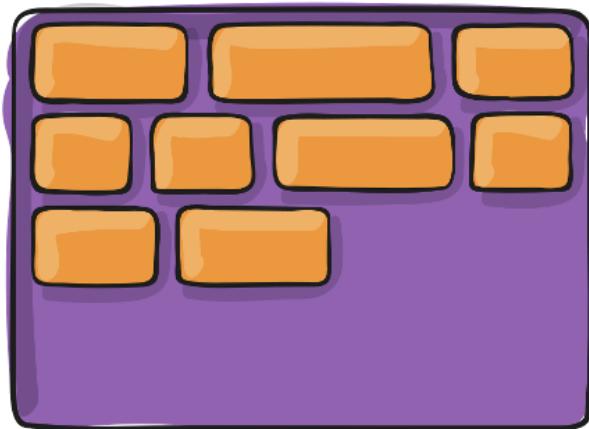
#### Align-content

Align-content afecta al eje secundario.

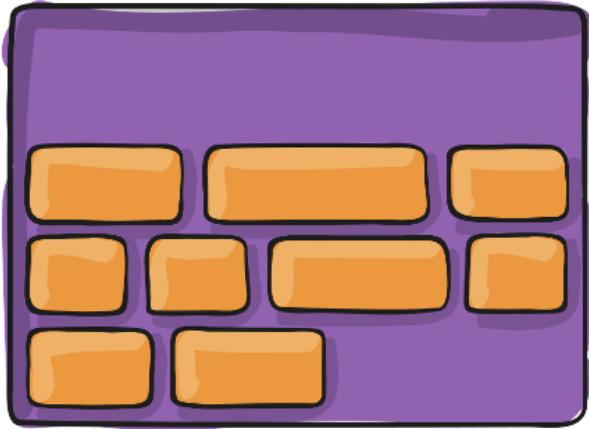
```
align-content: flex-start | flex-end | center | stretch | space-between |  
space-around;
```

En el caso de que el eje secundario fuera el vertical:

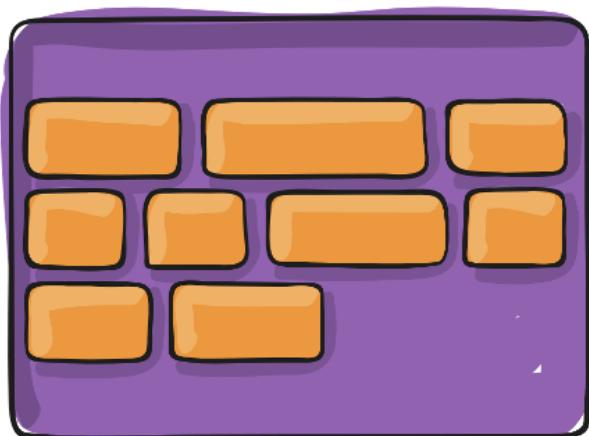
**flex-start**



**flex-end**



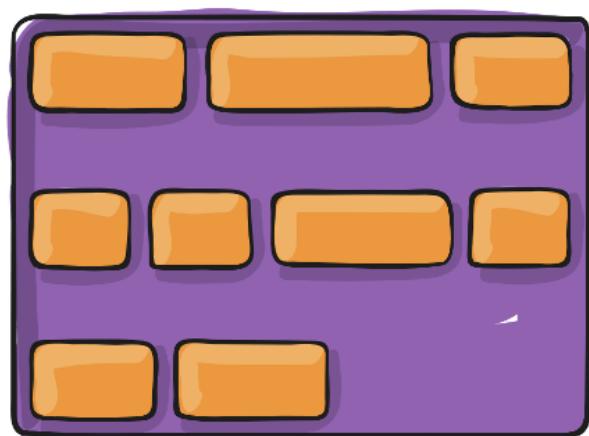
**center**



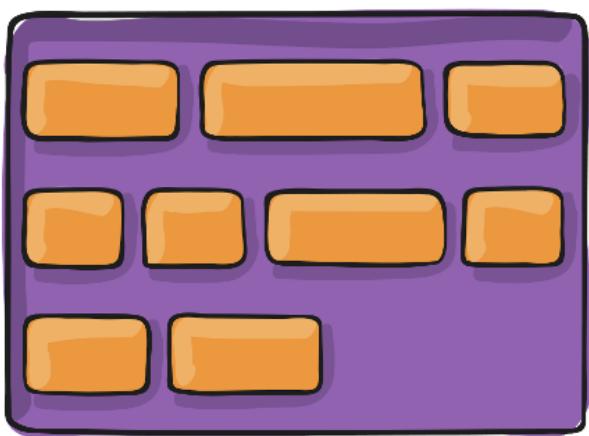
**stretch**



**space-between**



**space-around**



Valores (suponiendo que el eje secundario es el vertical):

- **flex-start:** arranca colocando los elemento en la parte superior del contenedor.
- **flex-end:** al contrario que el anterior este los coloca en la parte inferior.
- **center:** centra todo el contenido.
- **stretch:** intenta aprovechar todo el espacio disponible haciendo más grandes los items.

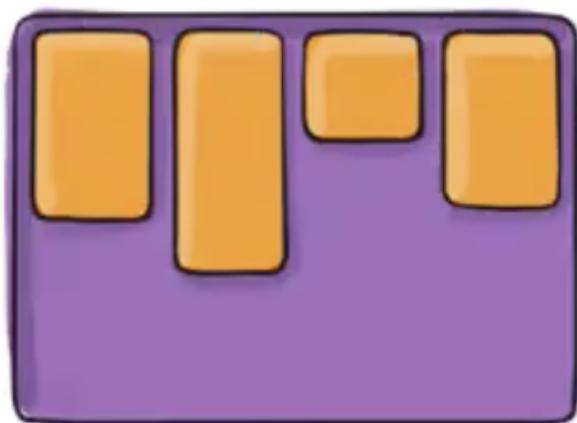
- *space-between*: parecido a *space-between* de justify pero aplicándolo al eje secundario.
- *space-around\_between*: parecido a *space-around* de justify pero aplicándolo al eje secundario.

## Align-items

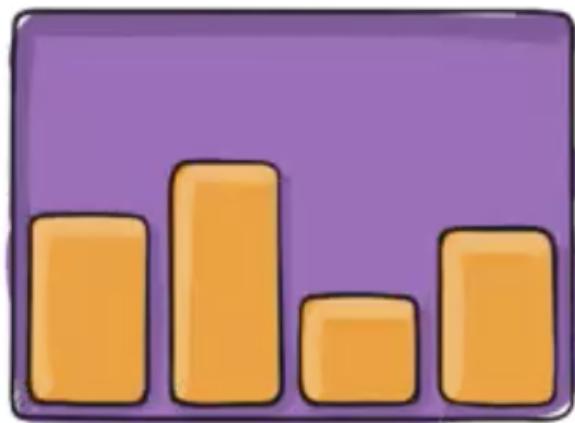
Si nos fijamos en la figura de align-content del apartado anterior (ejemplo eje principal horizontal), todos los elementos tienen el mismo alto, esto no tiene porque ser así ¿Qué hacemos si tienen tamaños distintos? podemos tirar de *align-items*

```
align-items: flex-start | flex-end | center | stretch | baseline;
```

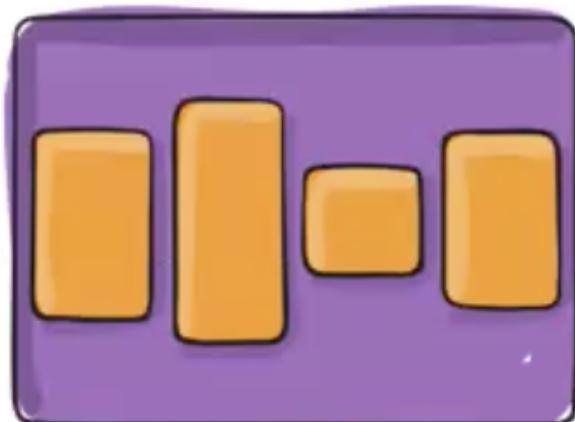
**flex-start**



**flex-end**



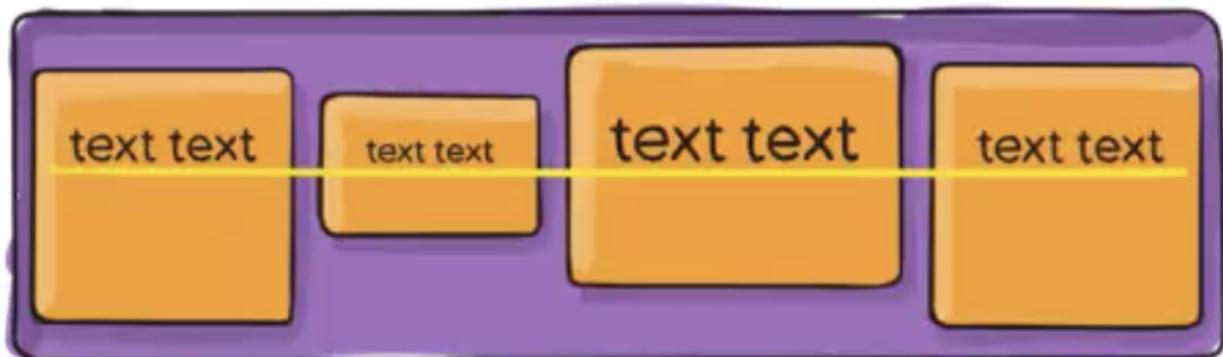
**center**



**stretch**



**baseline**



Valores (siguiendo ejemplo eje secundario vertical):

- **flex-start:** Que todos los elementos los muestre al principio cada uno tenga su alto.
- **flex-end:** Que todos los elementos los muestra alineado al final y que cada uno tenga su tamaño.
- **center:** Que todos los elemento aparezcan centrados.

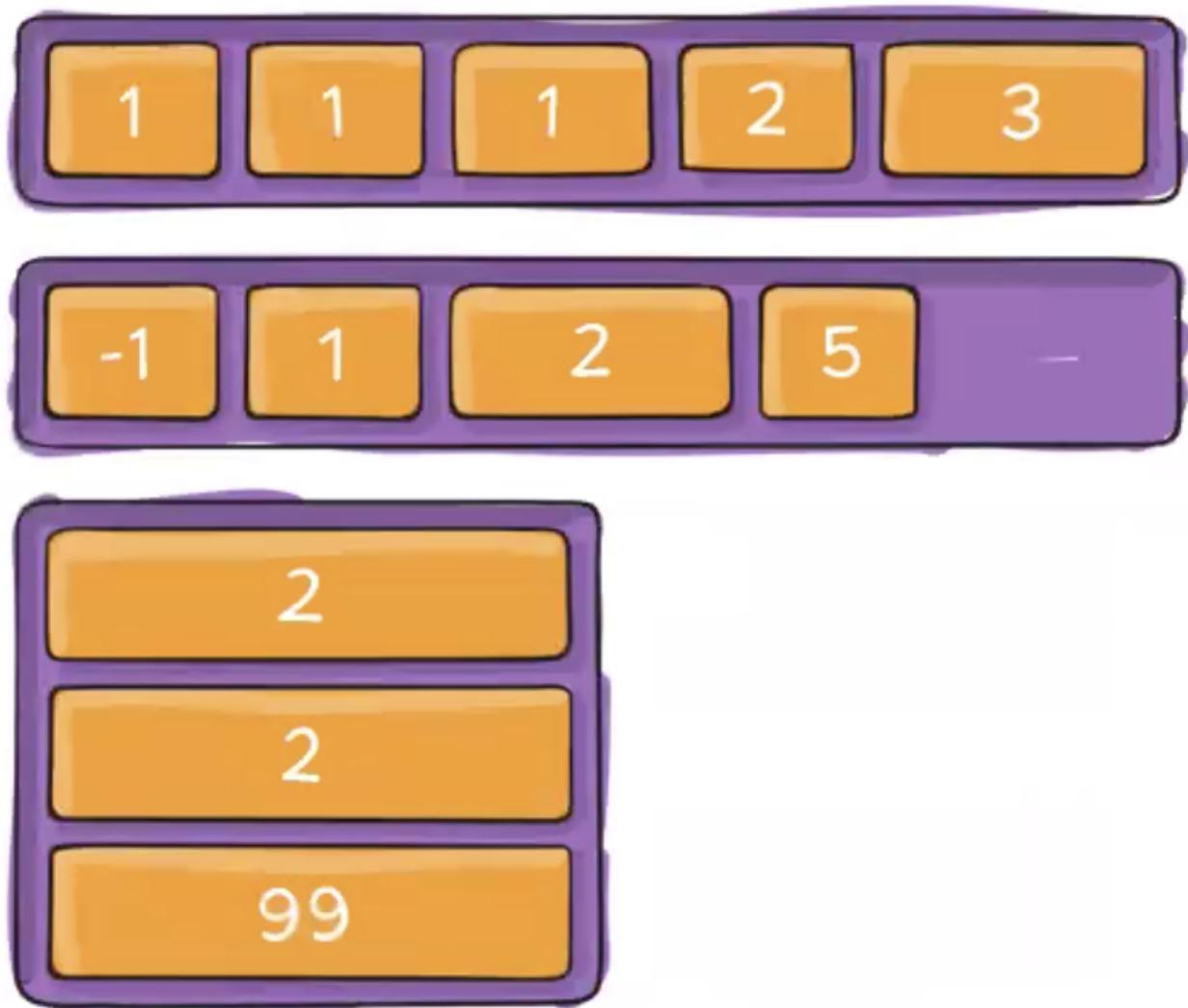
- *stretch*: Que todos los elementos intenten ocupar el máximo espacio vertical disponible.
- *baseline*: nos permite que nos centre en base a la primera línea de texto, así todos los texto se muestran alineados.

## Propiedades para los hijos (items)

Además de poder definir propiedades a nivel de contenedor, podemos definir comportamiento para cada *item*.

## Order

Podemos indicarle a un *item* un orden, esto nos permite cambiar el orden de los elementos sin tener que mover elementos en el HTML, añadiendo unos pesos.



Del ejemplo:

- Todos los que están con peso 1 se muestran primero, y al haber empate si se usa para esos items el orden de aparición en el html.
- Después irían colocándose el resto de menos a mayor peso.

- Si usamos un número negativo, se pondría por delante del peso 1.

Si estuvieramos trabajando con un `order row-reverse` el orden lo aplicaría al contrario.

## Grow Shrink and Basis

Este es muy importante, puede resultar un poco raro de entender.

Con `flex-basis` le indicamos a ese ítem cuál es el tamaño que tiene que pillar antes de poder crecer o decrecer (`grow-> crecer, shrink -> decrecer`).

En `Basis` le decimos cuál es su punto de partida, todo es relacionado respecto a sus hermanos, `grow`, `shrink` y `basis` tiene un shorthand `flex` en que por ejemplo aplicando un uno lo aplicamos a las tres propiedades a la vez, que ocurre si todos los elementos tienen los mismos valores (por ejemplo 1,1,1) entonces se estiran para ocupar el espacio disponible en la misma medida.

En el segundo caso en la imagen, si le decimos que tome un `flex-grow` de dos, le estamos indicando que crezca el doble de lo que crezcan los hermanos, lo mismo pasaría con el `shrink` pero para cuando tienen que encoger.

Por ejemplo puedo partir de una configuración de `layout` tal que:

- De primeras ocupa menos espacio que los hermanos.
- Conforme crezcas crece el doble que los hermanos.
- Cuando encojas, decrece el doble que los hermanos.

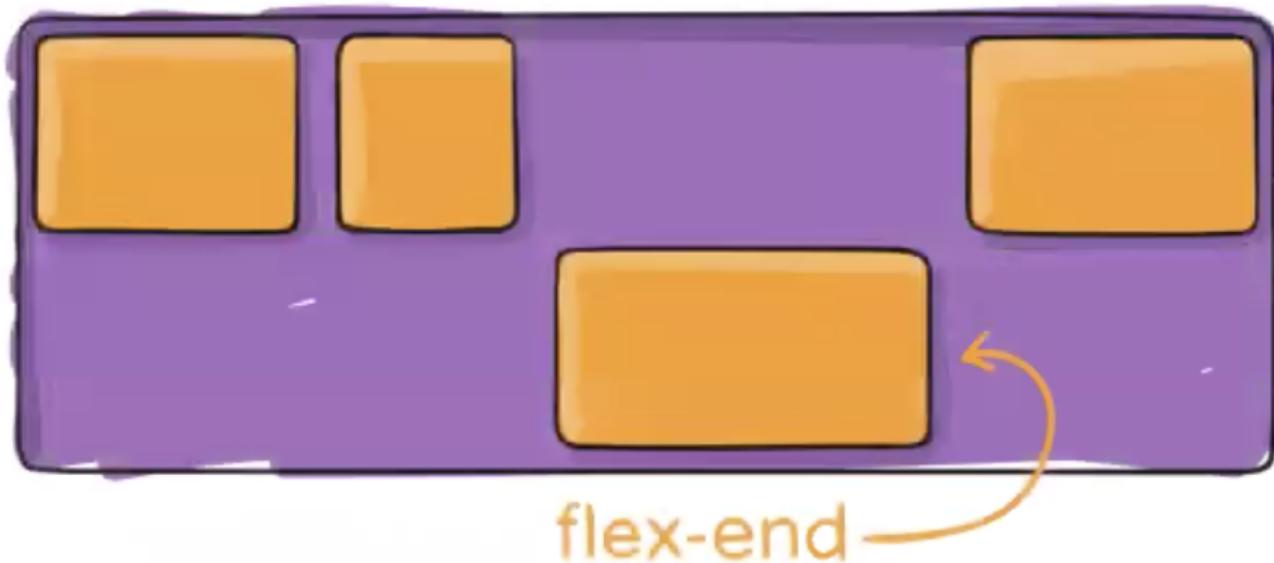
Esto lo entenderemos del todo cuando nos pongamos con los ejemplos.



## Align Self

Nos permite sacar un elemento específico de su sitio en el `layout` y por ejemplo ponerlo al final del todo, pero ojo, el resto de hermanos no ocupan su sitio.

# flex-start



## Flexbox a base de ejemplos

---

### Menú y container

Vamos a continuar con el codesandbox que habíamos creado previamente, partimos del ejemplo del sidebar + cuerpo, el ejemplo que se nos quedaba un poco cojo si intentábamos crearlo usando sólo `div` y `align`.

Vamos a borrar lo que teníamos dentro del body y arrancamos de nuevo:

`./index.html`

```
<body>
  <div class="my-flex-container">
    <div class="menu"></div>
    <div class="content"></div>
  </div>
</body>
```

Ponemos un contenedor `flex` y tenemos dentro dos ítems, uno para el menú y otro para el contenido.

En la parte de CSS borramos todo y vamos manos a la obra:

Definimos unos márgenes para el body (como lo teníamos antes):

`./src/miestilo.css`

```
body {  
    margin: 0px;  
    padding: 0px;  
}
```

Vamos ahora a definir el contenedor *flex*:

- Le indicamos que es un contenedor *flex* con la propiedad *display: flex*
- Aunque el valor por defecto valdría, vamos a indicarle explicitamente el *flex-direction* que en este caso será el *row* ya que vamos a colocar las filas en el eje horizontal, de izquierda a derecha, es decir distribuimos los *items* en una fila, primero el menú y después el contenido.
- Y le vamos a indicar que el contenedor ocupe todo el espacio con las propiedades ViewPort Width y ViewPort Height (esto no es de *flexbox*, es de *css*).

./src/miestilo.css

```
body {  
    margin: 0px;  
    padding: 0px;  
}  
  
+ .my-flex-container {  
+     display: flex;  
+     flex-direction: row;  
+     width: 100vw;  
+     height: 100vh;  
+ }
```

A la clase menú, vamos a indicarle los siguientes atributos:

- Le damos un ancho en pixeles.
- Le damos un color de fondo (el mismo que se usaba antes).
- De momento no le introducimos nada específico de *flexbox*.

./src/miestilo.css

```
.my-flex-container: {  
    display: flex;  
    flex-direction: row;  
    width: 100vw;  
    height: 100vh;  
}  
  
+ .menu {  
+     width: 150px;  
+     background-color: cadetblue;  
+ }
```

Vamos ahora a por el área de content:

- Le damos el mismo color que tenía antes.
- Le indicamos que tome todo el espacio posible ¿Cómo hacemos esto? Estableciendo el valor `flex-grow` a 1, de esta manera, al no tener ningún hermano que tenga `flex-grow` este elemento intentará tomar todo el espacio disponible.

`./src/miestilo.css`

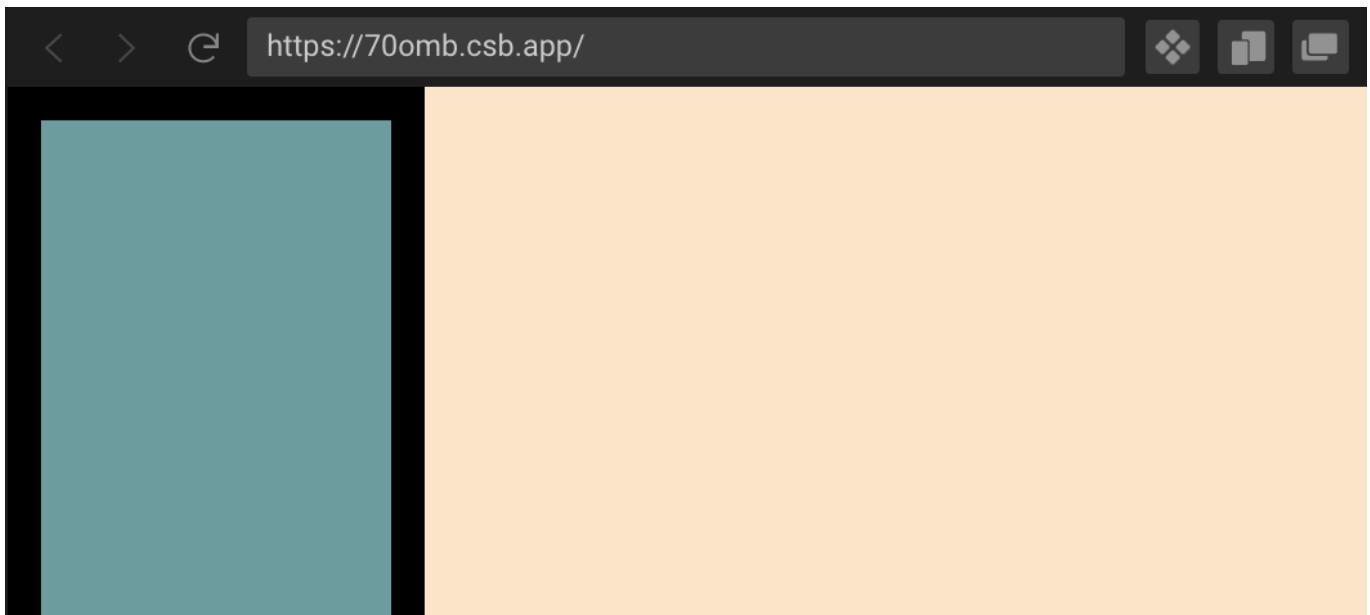
```
.menu {  
    width: 150px;  
    background-color: cadetblue;  
}  
  
+ .content {  
+   background-color: bisque;  
+   flex-grow: 1;  
+ }
```

Si vemos el resultado en el navegador podemos apreciar que tenemos el mismo comportamiento que antes.

Vamos a ver que pasa si le añadimos un borde al menú, ¿Se romperá el *layout* como en la aproximación anterior? La respuesta es no, podemos variar los valores como queramos que se sigue respetando el área del menú y la del body.

`./src/miestilo.css`

```
.menu {  
    width: 150px;  
    background-color: cadetblue;  
+ border: 15px solid black;  
}
```



Le podemos añadir un margen al menú, lo que queramos y ya no se rompe, no tenemos que ir con cálculos, etc...

## Flex-direction

Vamos a limpiar el body y añadimos tres elementos:

`./src/index.html`

```
<body>
+  <div>
+   <div>
+     <h1>Elemento 1</h1>
+   </div>
+   <div>
+     <h1>Elemento 2</h1>
+   </div>
+   <div>
+     <h1>Elemento 3</h1>
+   </div>
+ </div>
</body>
```

Y vamos a indicarle al *div* principal (el que hace de contenedor) que va a tener un clase que se va a llamar *my-flex-container*.

```
<body>
-  <div>
+ <div class="my-flex-container">
  <div>
    <h1>Elemento 1</h1>
```

En el css borramos lo que teníamos, y nos ponemos manos a la obra.

Dejamos el body con margin y padding 0.

./src/miestilo.css

```
body {  
    margin: 0px;  
    padding: 0px;  
}
```

Vamos poner a un color cada contenedor (esto no tiene que ver con *flexbox*). Creamos un clase que se va a llamar mi *flex container*:

- Vamos a seleccionar el primer hijo que tenga ese *container* (el primer *div*).
- Vamos a darle a ese *item* el color *darkmagenta*.

./src/miestilo.css

```
body {  
    margin: 0px;  
    padding: 0px;  
}  
  
+ .my-flex-container div:nth-child(1) {  
+   background-color: darkmagenta;  
+ }
```

Hacemos lo mismo con el resto y le damos a cada uno colores distintos.

```
body {  
    margin: 0px;  
    padding: 0px;  
}  
  
.my-flex-container div:nth-child(1) {  
  background-color: darkmagenta;  
}  
  
+ .my-flex-container div:nth-child(2) {  
+   background-color: darkolivegreen;  
+ }  
  
+ .my-flex-container div:nth-child(3) {  
+   background-color: darkred;  
+ }
```

Vamos ahora a indicar a todos los *div* que están dentro de ese container que tengan un *padding* de 5px y que el texto lo pinte en color blanco.

```
body {  
    margin: 0px;  
    padding: 0px;  
}  
  
+ .my-flex-container div {  
+   padding: 5px;  
+   color: white;  
+ }
```

Hasta aquí todo bien, tenemos tres elemento *div*, que por defecto son *block* con lo que tienden a coger el 100% del espacio disponible y caen uno debajo del otro.

Vamos a convertir nuestro contenedor en un contenedor flex-box:

- Le indicamos que su display es *flex*.
- Por defecto el valor de *flex-direction* es *row* aunque no lo indiquemos, pero para que quede más claro lo añadimos.
- 

./src/miestilo.css

```
body {  
    margin: 0px;  
    padding: 0px;  
}  
  
+ .my-flex-container {  
+   display: flex;  
+   flex-direction: row;  
+ }
```

Ya con esto nos aparece cada *div item* alineado en una fila.

Para ver cuento ocupa, vamos añadirle un borde al contenedor flex:

./src/miestilo.css

```
.my-flex-container {  
+ border: 5px solid black;  
  display: flex;  
  flex-direction: row;  
}
```

Fíjate que el contenedor ocupa el 100% del ancho disponible, y los items sólo ocupan el espacio suficiente para mostrar el texto de cada elemento, repartidos en el eje horizontal.



Por probar podemos cambiar el *flex-direction* de la clase `.my-flex-container` a *column*

`./src/miestilo.css`

```
.my-flex-container {  
  border: 5px solid black;  
  display: flex;  
  - flex-direction: row;  
  + flex-direction: column;  
}
```



Vamos a seguir jugando, ¿Te acuerdas de ese flag que parecía raro para pintar *del reverse*? Vamos a probarlo, cambiamos ahora el *flex-direction* a *row-reverse* ¿Qué va a pasar?

- Que va a empezar a pintar en el extremo derecho del contenedor.
- Que el elemento 1 se pinta en el extremo derecho, y después va el 2 y el 3.

`./src/miestilo.css`

```
.my-flex-container {  
  border: 5px solid black;  
  display: flex;
```

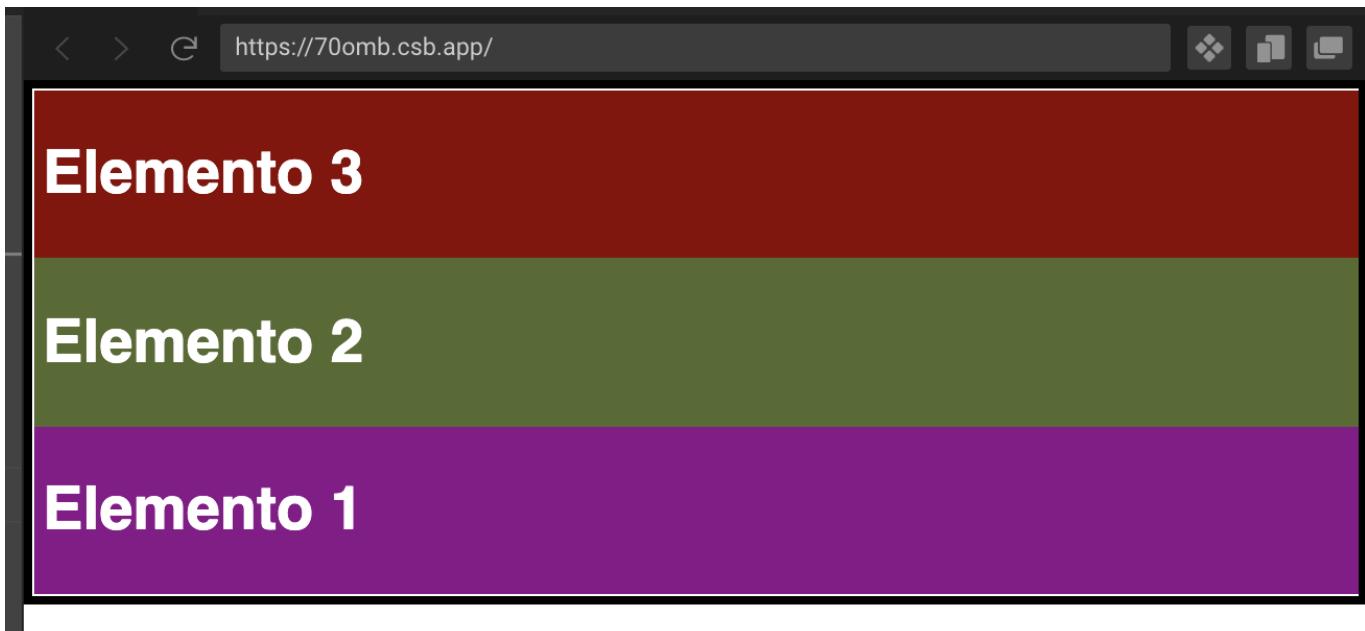
```
- flex-direction: column;  
+ flex-direction: row-reverse;  
}
```



También puedes probar con *column-reverse*:

```
./src/miestilo.css
```

```
.my-flex-container {  
  border: 5px solid black;  
  display: flex;  
- flex-direction: row-reverse;  
+ flex-direction: column-reverse;  
}
```



## Flex-wrap

Tomamos como punto de partida el ejemplo anterior, fijaros que en nuestro caso los elemento sólo ocupan el espacio indispensable para pintarse, por eso no ocupan todo el contenedor.

Vamos a volver a establecer el *flex-direction* en nuestro *flex-container* a row:

```
.my-flex-container {  
    border: 5px solid black;  
    display: flex;  
    - flex-direction: column-reverse;  
    + flex-direction: row;  
}
```

¿Qué pasaría si cada *item* tuviera un ancho considerable? Por ejemplo vamos a darle 500 píxeles de ancho:

./src/miestilo.css

```
.my-flex-container {  
    border: 5px solid black;  
    display: flex;  
    flex-direction: row;  
    + width: 400px;  
}
```

Pues que si la ventana tiene de ancho menos de 1200 píxeles, el ancho de los elementos se adaptará y tendrá menos de 400 píxeles de ancho.

![Si no hay espacio suficiente el contenedor ajusta el ancho]  
(./content/noson400.png)

Incluso si vamos a un tamaño más pequeño vemos que ya se nos pueden llegar a montar elementos o no mostrarse bien, aquí podríamos jugar con la propiedad que vimos en el capítulo anterior `_overflow_` lo podemos poner a `_hidden_` o `_scroll_`

./src/miestilo.css

```
```diff  
.my-flex-container {  
    border: 5px solid black;  
    display: flex;  
    flex-direction: row;  
    width: 400px;  
+ overflow: scroll;  
}
```

¿Y si queremos que si o si tome ese espacio de 400 píxeles? Aquí es donde entra en juego la propiedad `flex-wrap` (que por defecto está en `nowrap`), podemos indicarle al contenedor `flex`, que en caso de que no quepan los elementos lo va a ir desplazando a una fila inferior.

./src/miestilo.css

```
.my-flex-container {  
  border: 5px solid black;  
  display: flex;  
  flex-direction: row;  
  width: 400px;  
  - overflow: scroll;  
  + flex-wrap: wrap;  
}
```

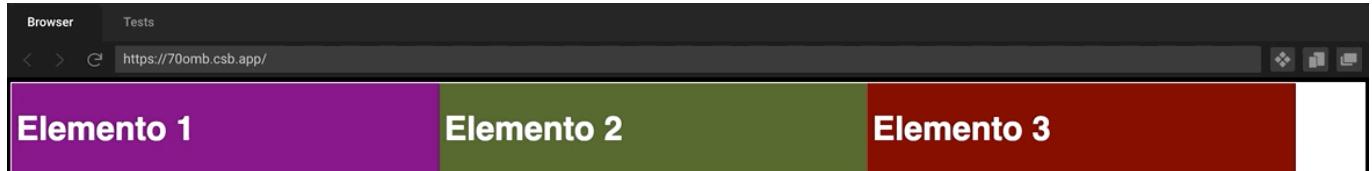
Fijate ahora que mientras haya espacio el contenedor muestra todos los elementos en una fila, y si empezamos a encoger el ancho, llega un momento en el que empieza a meter los items en las siguiente fila:



Si lo hacemos muy pequeño podemos ver que se nos queda todo en el eje vertical, esto cuando hacemos diseño responsivo se usa mucho, por ejemplo tengo un diseño para escritorio y en cuanto paso a móvil van cayendo los *items* en vertical.

También tenemos la opción con el *flex-wrap: flex-reverse*, así si no cabe un elemento se lleva las filas hacia arriba.

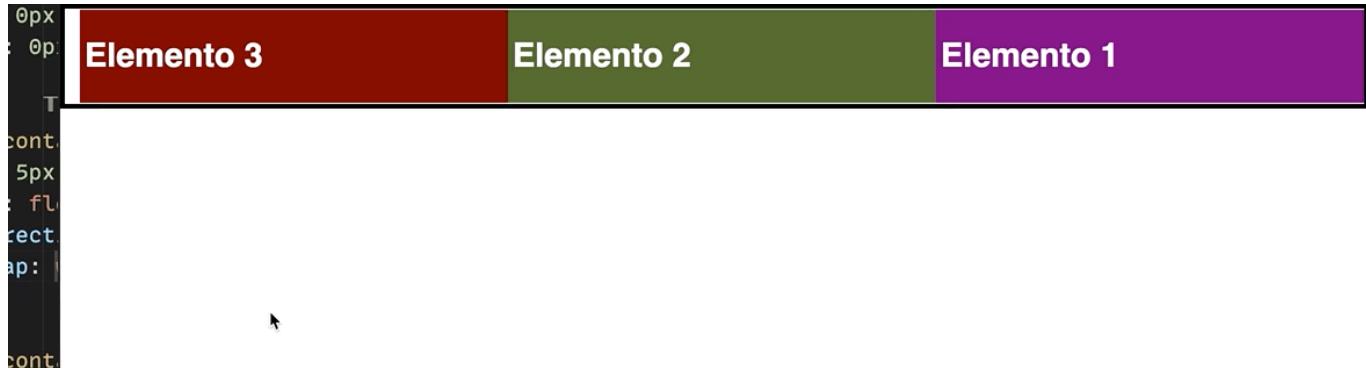
```
.my-flex-container {  
  border: 5px solid black;  
  display: flex;  
  flex-direction: row;  
  width: 400px;  
  - flex-wrap: wrap;  
  + flex-wrap: wrap-reverse;  
}
```



Ahora podemos empezar a poner combinaciones "raras"

¿Qué pasa si combino *row-reverse* con *wrap-reverse*? Que se nos justifica a la derecha, y se van apilando los elementos que no caben, ¿Qué elemento se va para arriba? el elemento 3 Ya que tenemos el elemento 3.

```
.my-flex-container {  
    border: 5px solid black;  
    display: flex;  
    - flex-direction: row;  
    + flex-direction: row-reverse;  
    width: 400px;  
    + flex-wrap: wrap-reverse;  
}
```



Pero si tuvieramos el *flex-wrap wrap*...

```
.my-flex-container {  
    border: 5px solid black;  
    display: flex;  
    flex-direction: row-reverse;  
    width: 400px;  
    - flex-wrap: wrap-reverse;  
    + flex-wrap: wrap;  
}
```

Veríamos como ahora el 3 cae abajo, después el 2... y acabaría ordenado del 1 al 3 en vez del 3 al 1.

Por eso es muy importante tener muy claro cuales son las propiedades y a que afecta, si no empezamos con la lotería de ir probando a lo loco hasta que damos con lo que queremos.

Un tema muy interesante que incorporan las *devtools* de flexbox es que los contenedores *\_flex-* los marcan con una etiqueta, y que en la propiedad *display flex*, aparece un ícono en el que si pinchamos tienen una guía para ir cambiando de forma visual.

Screenshot of the Chrome DevTools Elements tab showing the DOM tree and the Styles panel.

```

r; camera; encrypted-media; geolocation; gyroscope;
hid; microphone; midi; payment; usb; vr; xr-spatial-
tracking" sandbox="allow-forms allow-modals allow-
popups allow-presentation allow-same-origin allow-s
cripts" src="https://70omb.csb.app/" title="affecti
onate-roentgen-70omb" id="sandbox-preview" class="s
c-fcmNTy eXNXqu" style="opacity: 1; z-index: 1; bac
kground-color: white; pointer-events: initial;">
  <!DOCTYPE html>
  <html>
    <head>...</head>
    <body>
      <div class="my-flex-container" flex="flex"> ...
        <div>...
      </div>...
    </body>
</html>

```

The Styles panel shows the following CSS rules:

```

.my-flex-container {
  border: 5px solid black;
  display: flex;
  flex-direction: row-reverse;
  flex-wrap: wrap;
}

.my-flex-container {
  border: 5px solid black;
  display: flex;
  flex-direction: row-reverse;
  flex-wrap: wrap;
}

```

Screenshot of the Chrome DevTools Styles panel showing the flexbox configuration for the .my-flex-container class.

**flex-direction: row-reverse**

**flex-wrap: wrap**

Open flexbox editor

**align-content: normal**

**justify-content: normal**

**align-items: normal**

## Flex-flow

Flex-flow es la forma corta (shorthand) para *flex-direction* y *flex-wrap*, podemos decirle de una tacada que queremos el *flex-direction* row y el *wrap* normal.

```
.my-flex-container {
  border: 5px solid black;
```

```
display: flex;
- flex-direction: row-reverse;
- flex-wrap: wrap;
+ flex-flow: row wrap
}
```

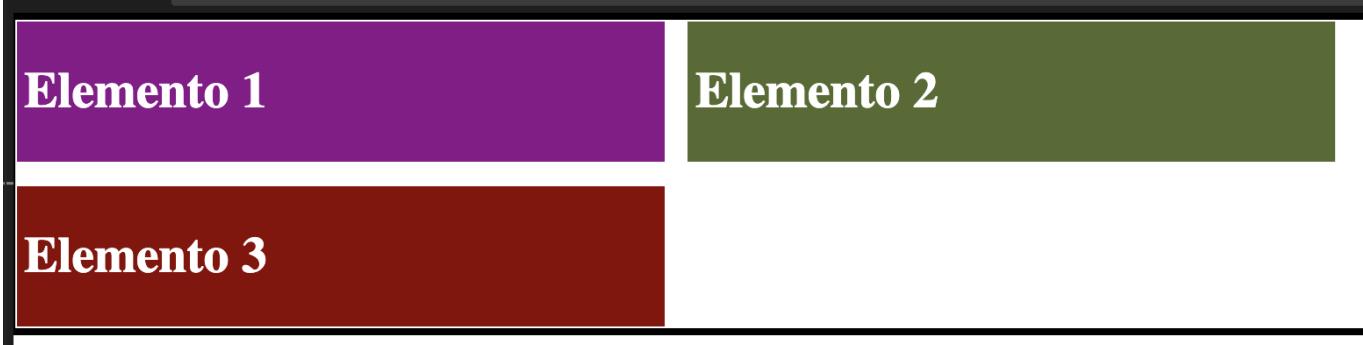
¿Que usar? Aquí depende un poco de como estéis más cómodos y que acordéis en vuestro equipo.

## Gap

Una propiedad que hace relativamente poco ha incorporado al estándar de *flex-box* es *gap*, esta propiedad si la teníamos disponible en CSS Grid, nos permite añadir un espacio entre cada *item*, sin tener que ir añadiendo *margins* ni nada de eso.

En el ejemplo anterior podríamos poner:

```
.my-flex-container {
  border: 5px solid black;
  display: flex;
  flex-flow: row wrap;
+ gap: 15px;
}
```



## Items con tamaños diferentes

Hasta ahora hemos estado trabajando con *items* que tenían el mismo ancho, esto no siempre es así, veamos el siguiente ejemplo:

En el HTML borramos el contenido del *body* y añadimos el siguiente:

```
./index.html
```

```
<body>
+ <div class="my-flex-container">
+ <div>Esto es un texto: 123</div>
+ <div>Esto es un texto: 123456</div>
+ <div>Esto es un texto: 1236546546</div>
```

```

+ <div>Esto es un texto: 12365465465465</div>
+ <div>texto: 123</div>
+ <div>Esto es un texto: 1234645654</div>
+ <div>Esto es un texto: 12</div>
+ <div>Esto es un texto: 123</div>
+ <div>texto: 123</div>
+ <div>texto: 123645065406540650</div>
+ </div>
</body>

```

En el css, borramos todos y ponemos lo siguiente:

- Mantenemos el *body* con marge cero y padding 0.
- Al *flex container* le añadimos un *border* para poder chequear visualmente el tamaño.
- A cada *item* le quitamos el ancho fijo, y añadimos un color así como un *border*.

./src/miestilo.css

```

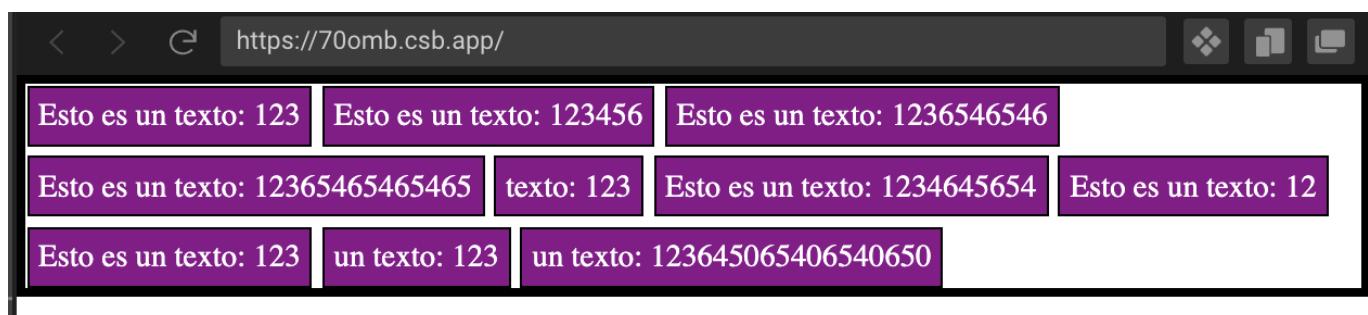
body {
  margin: 0px;
  padding: 0px;
}

.my-flex-container {
  border: 5px solid black;
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  gap: 5px;
}

.my-flex-container div {
  box-sizing: border-box;
  border: 1px solid black;
  padding: 5px;
  color: white;
  background-color: darkmagenta;
}

```

Ahora tenemos *items* de diferente tamaño, con lo que el *layout* ya no se ve todo cuadrado.



## Justify content

Para no meter demasiado ruido vamos a dejar el *html* en tres cajas, borramos el *body* y ponemos lo siguiente:

*./index.html*

```
<body>
  <div id="flex-container">
    <div class="box" id="box1">BOX 1</div>
    <div class="box" id="box2">BOX 1</div>
    <div class="box" id="box3">BOX 1</div>
  </div>
</body>
```

Limpiamos el CSS, y volvemos a empezar:

*./src/miestilo.css*

```
body {
  margin: 0px;
  padding: 0px;
}
```

Un tema interesante, el *body* también puede tener un *display flex*

Y aplicamos los siguiente estilos:

Primero el *flex-container* que esta vez lo acotamos por id del elemento:

- Le añadimos el *display flex*.
- Le decimos que el *flex-direction* sea *row*.
- Le añadimos un border para ver de manera fácil el espacio que ocupa.

```
body {
  margin: 0px;
  padding: 0px;
}

+ #flex-container {
+   display: flex;
+   flex-direction: row;
+   border: 3px solid black;
+ }
```

Le vamos a indicar ahora que todas las cajas que vamos a pintar (los items hijo), tengan el color *cadetblue*, y le vamos a dar un *padding*, y un *border* para diferenciar hasta donde llega cada caja.

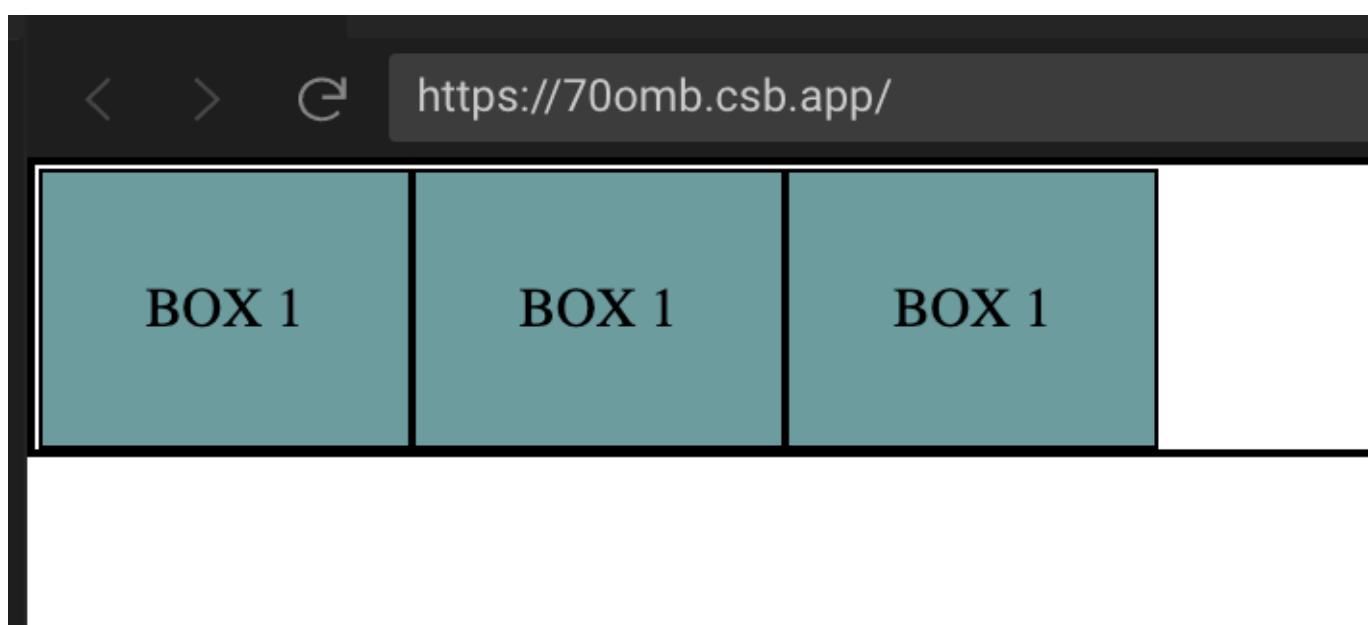
```

#flex-container {
  display: flex;
  flex-direction: row;
  border: 3px solid black;;
}

+ .box {
+ border: 1px solid black;
+ background-color: cadetblue;
+ padding: 30px;
+ }

```

Con esto ya tenemos las tres cajas, pintándose, de momento nada nuevo:



Fijaros que nos sobra espacio, podríamos empezar a jugar con *justify-content*, para que los justificara a un lado, u otro, o los centrara.

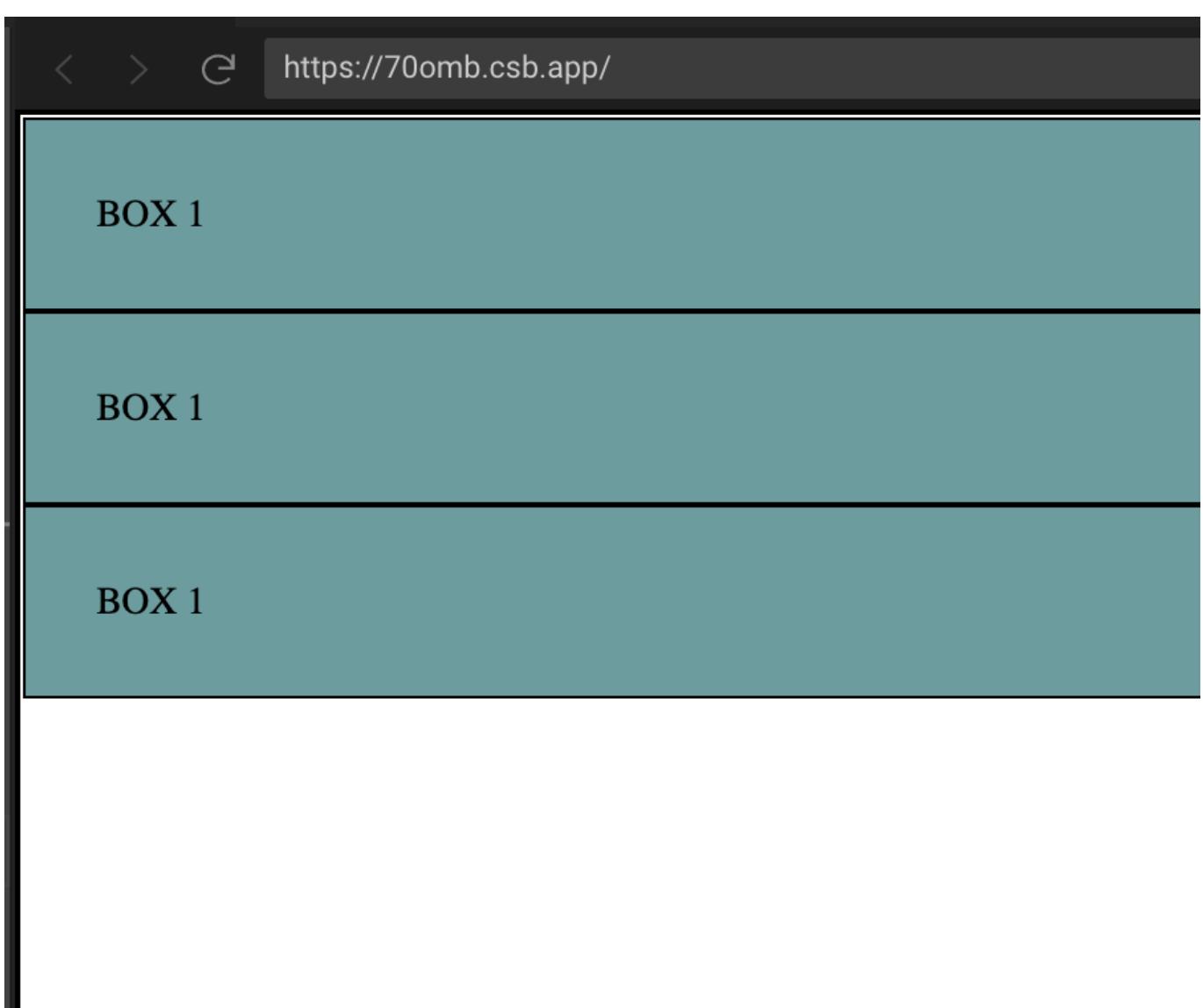
Vamos ahora a añadirle más estilado el *flex-container*:

- Le decimos que tome todo el ancho disponible (*height: 100vh*)
- Para evitar que salga scroll, le decimos que el *box-sizing* sea *border-box*
- El flex direction esta vez le vamos a decir que sea *column*
- Y el *justify content flex-start*

```

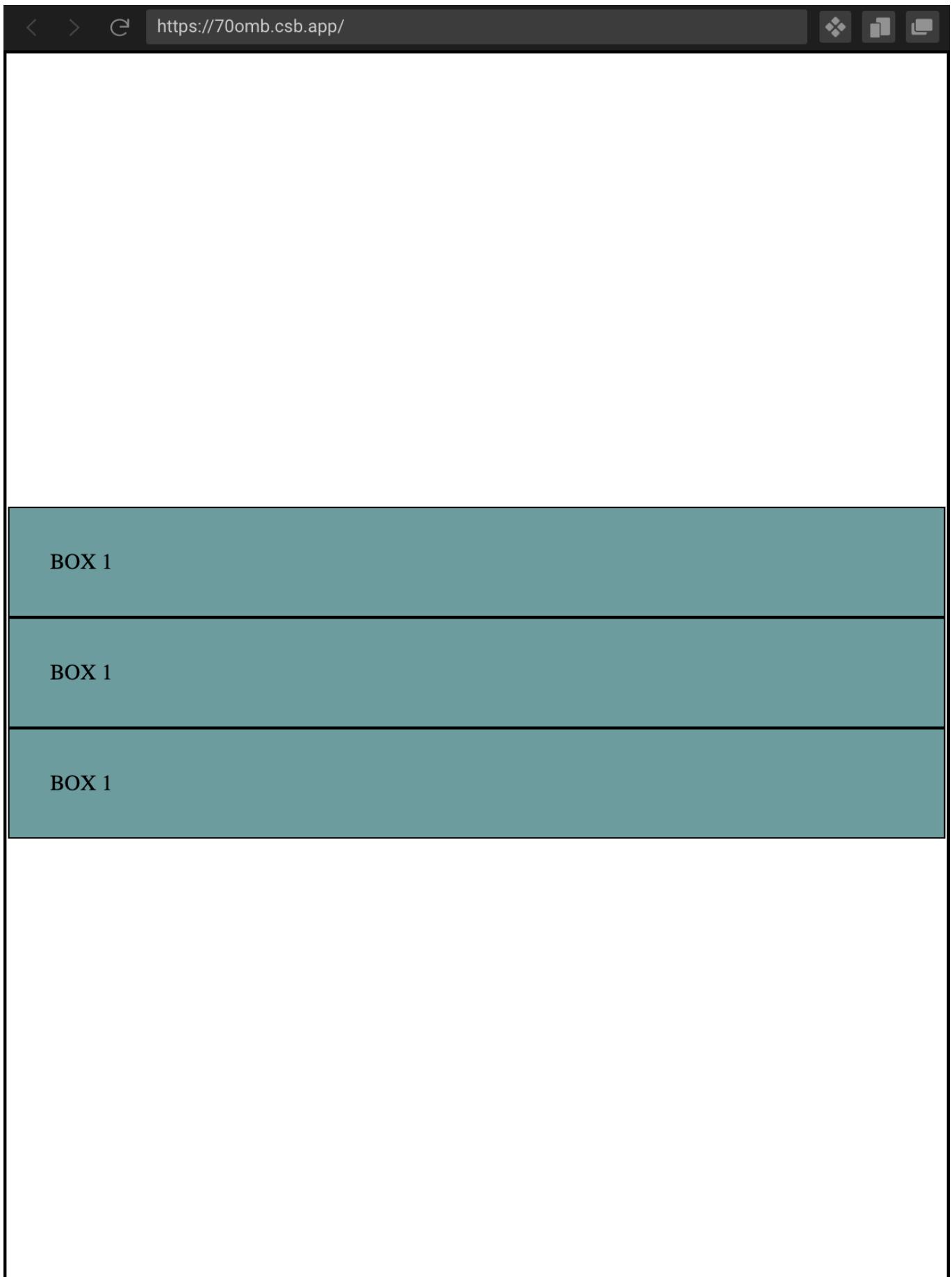
#flex-container {
+ box-sizing: border-box;
+ height: 100vh;
  display: flex;
- flex-direction: row;
+ flex-direction: column;
  border: 3px solid black;
+ justify-content: flex-start;
}

```



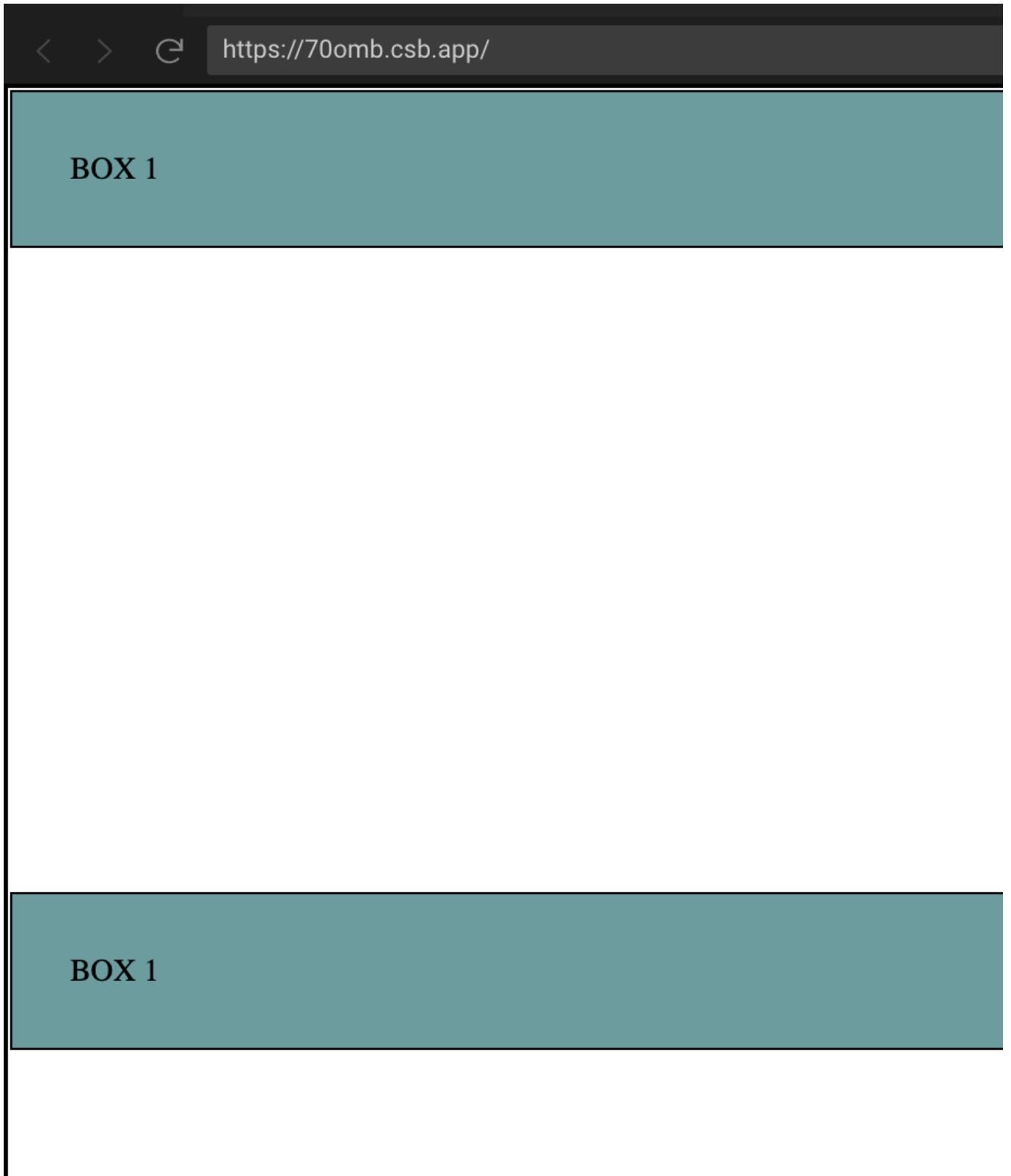
Si cambiamos el valor de *justify-content* a *center* podemos ver como sale centrado verticalmente:

```
#flex-container {  
  box-sizing: border-box;  
  height: 100vh;  
  display: flex;  
  flex-direction: column;  
  border: 3px solid black;  
  - justify-content: flex-start;  
  + justify-content: center;  
}
```



Y si aplicamos *space-between* podemos ver como se distribuyen en el espacio disponible:

```
#flex-container {  
  box-sizing: border-box;  
  height: 100vh;  
  display: flex;  
  flex-direction: column;  
  border: 3px solid black;  
  - justify-content: center;  
  + justify-content: space-between;  
}
```



## BOX 1

Console

8

Problems

0

./index.html

```
<div id="flex-container">
    <div class="box" id="box1">BOX 1</div>
    <div class="box" id="box2">BOX 1</div>
    <div class="box" id="box3">BOX 1</div>
+     <div class="box" id="box4">BOX 1</div>
</div>
```

Podemos añadir un nuevo elemento al HTML y comprobar que el espacio se reparte.

Si cambiamos el *flex-direction* a *row* podemos ver como lo reparte horizontalmente.

./src/miestilo.css

```
#flex-container {
    box-sizing: border-box;
    height: 100vh;
    display: flex;
-   flex-direction: column;
+   flex-direction: row;
    border: 3px solid black;
    justify-content: space-between;
}
```

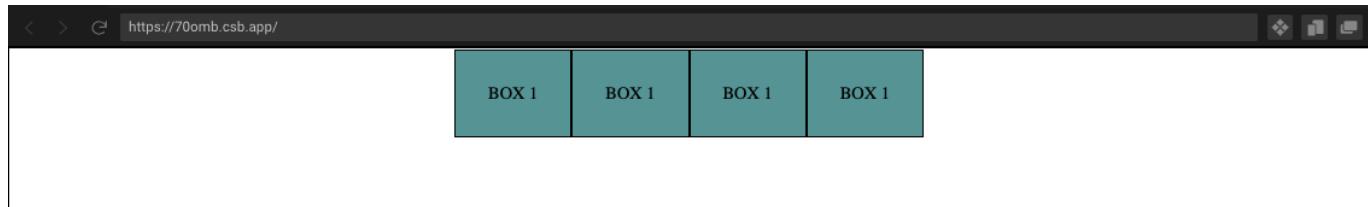
A tener en cuenta:

- Si después jugamos a que los *items* ocupen el espacio disponible, no lo vamos a poder repartir y no van a tener efecto poner esas propiedades.
- Ahora mismo las cajas en modo *row* ocupan todo el alto porque se lo hemos indicado en la propiedad *height* del container (y por defecto el *flexbox container* tiene la propiedad *align-items stretch*), si no el contenedor ocuparía el espacio del contenido. Una prueba que puedes hacer es dejar el *height 100vh* y añadir la propiedad *align-items* con el valor *flex-start*, de esta forma no saldrían estirados, si no que se mostraría cada *item* arriba del todo y con su tamaño original.

```
#flex-container {  
  box-sizing: border-box;  
  height: 100vh;  
  display: flex;  
  flex-direction: row;  
  border: 3px solid black;  
  justify-content: space-between;  
  + align-items: flex-start;  
}
```

Vamos ahora a centrar el contenido en eje horizontal y vertical, tenemos el *flex-direction row*, si elegimos *justify-content center* se alinea en el eje horizontal:

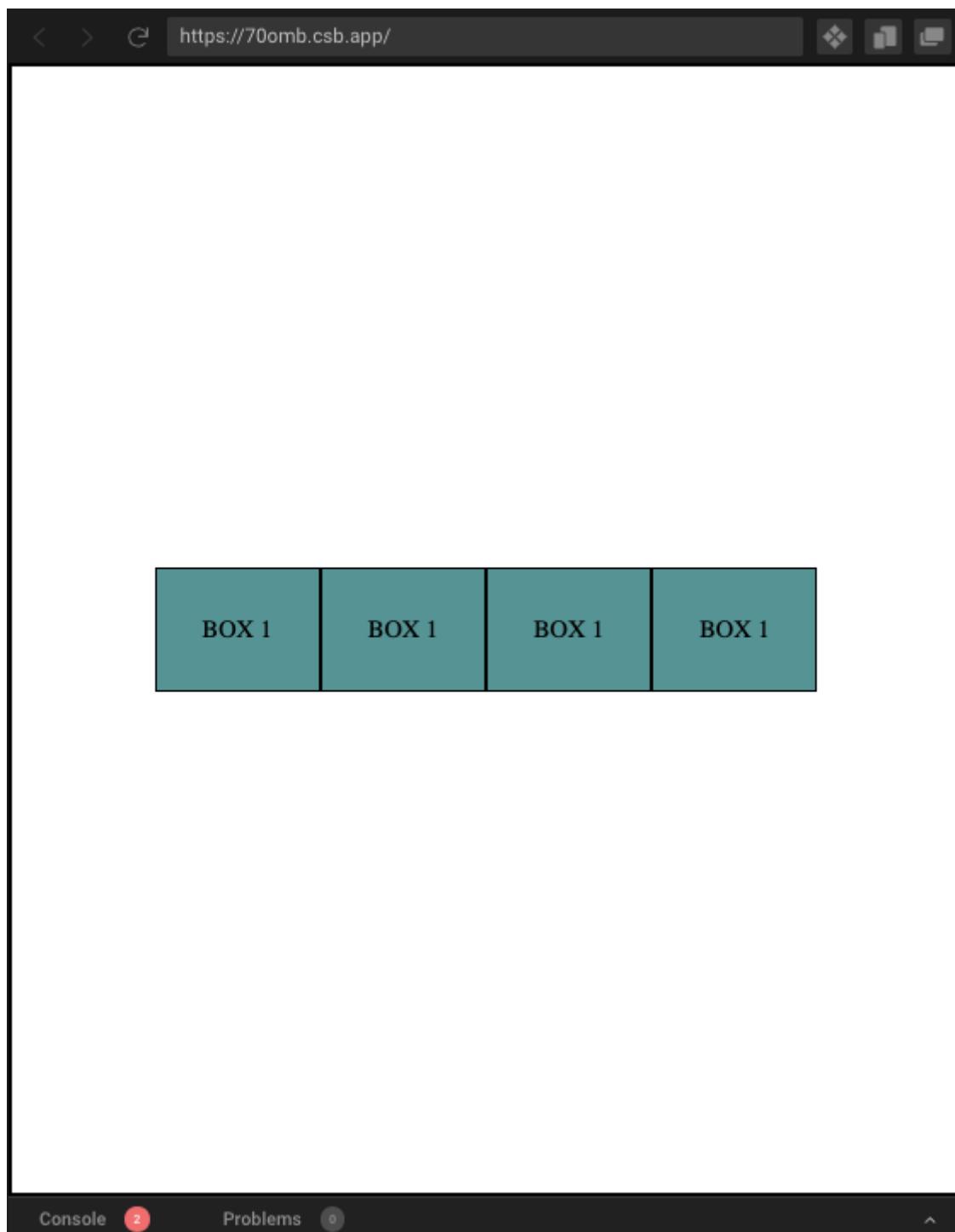
```
#flex-container {  
  box-sizing: border-box;  
  height: 100vh;  
  display: flex;  
  flex-direction: row;  
  border: 3px solid black;  
  - justify-content: space-between;  
  + justify-content: center;  
  align-items: flex-start;  
}
```



Y si queremos centrarlo en el eje secundario modificamos el valor de *align-items* a *center*

```
#flex-container {  
  box-sizing: border-box;
```

```
height: 100vh;  
display: flex;  
flex-direction: row;  
border: 3px solid black;  
justify-content: center;  
- align-items: flex-start;  
+ align-items: center;  
}
```



Así podríamos estar jugando con las diferentes combinaciones entre eje principal y secundario.

Vamos ahora a jugar con *Baseline*, va a modificar en el HTML el primer *div* e introducir un *h1*, en el segundo un *h3*

*index.html*

```
<div id="flex-container">
-     <div class="box" id="box1">BOX 1</div>
+     <div class="box" id="box1"><h1>BOX 1</h1></div>
-     <div class="box" id="box2">BOX 1</div>
+     <div class="box" id="box2"><h3>BOX 2</h3></div>
     <div class="box" id="box3">BOX 1</div>
     <div class="box" id="box4">BOX 1</div>
</div>
```

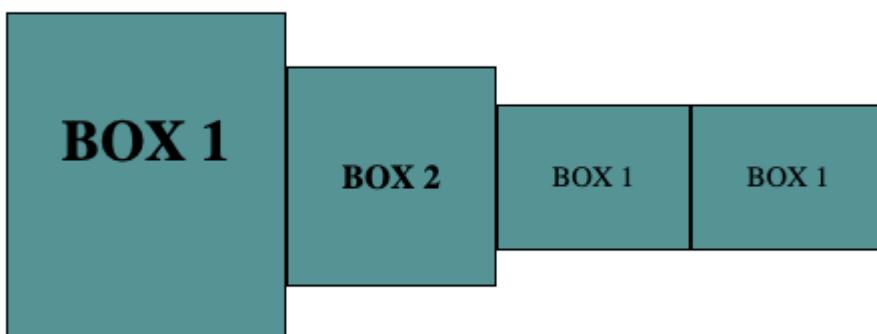
Y a la primera caja le vamos a meter un *padding* de 70 píxeles para que se vea más la diferencia.

*./src/miestilo.css*

```
.box {
    border: 1px solid black;
    background-color: cadetblue;
    padding: 30px;
}

+ #box1 {
+   padding-bottom: 70px;
+ }
```

¿Qué aspecto tiene esto?

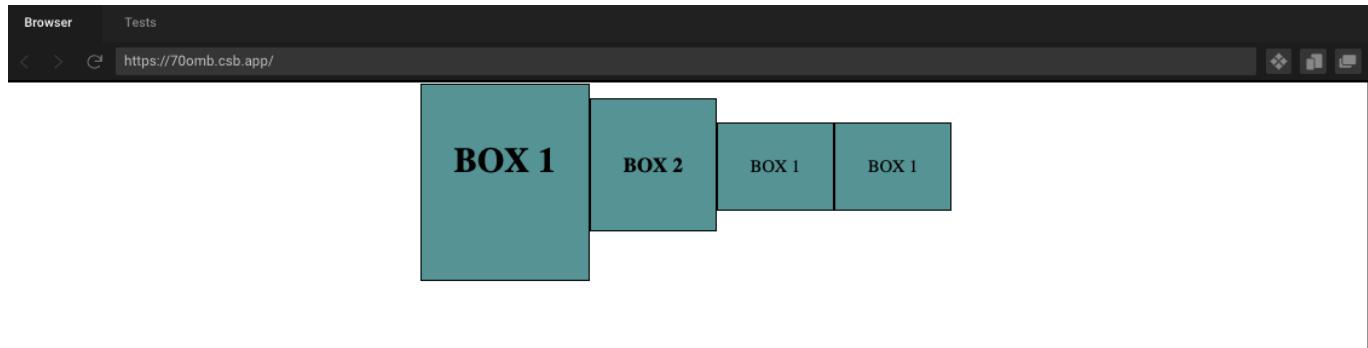


Si os fijáis los textos no salen alineados ¿Qué podemos hacer? vamos decirle que queremos centrarlos por la primera linea de texto.

```
#flex-container {
    box-sizing: border-box;
    height: 100vh;
```

```
display: flex;
flex-direction: row;
border: 3px solid black;
justify-content: center;
- align-items: center;
+ align-items: baseline;
}
```

Ahora tenemos todas las cajas centradas en la primera línea de texto:



¿Y si queremos tener más de una línea? Vamos a añadir un *flex-wrap*, es buena idea tener *direction* y *wrap* cerca por si un día queremos pasarlo a *shorthand flow*.

./src/miestilo.css

```
#flex-container {
  box-sizing: border-box;
  height: 100vh;
  display: flex;
  flex-direction: row;
+ flex-wrap: wrap;
  border: 3px solid black;
  justify-content: center;
  align-items: baseline;
}
```

Vamos también a darle un ancho más grande a las cajas

./src/miestilo.css

```
.box {
+ width: 150px;
  border: 1px solid black;
  background-color: cadetblue;
  padding: 30px;
}
```

Y vamos a crear otras dos cajas para forzar el *wrap*.

./index.html

```
<body>
  <div id="flex-container">
    <div class="box" id="box1"><h1>BOX 1</h1></div>
    <div class="box" id="box2"><h3>BOX 2</h3></div>
    <div class="box" id="box3">BOX 3</div>
    <div class="box" id="box4">BOX 4</div>
+    <div class="box" id="box3">BOX 5</div>
+    <div class="box" id="box4">BOX 6</div>
  </div>
</body>
```

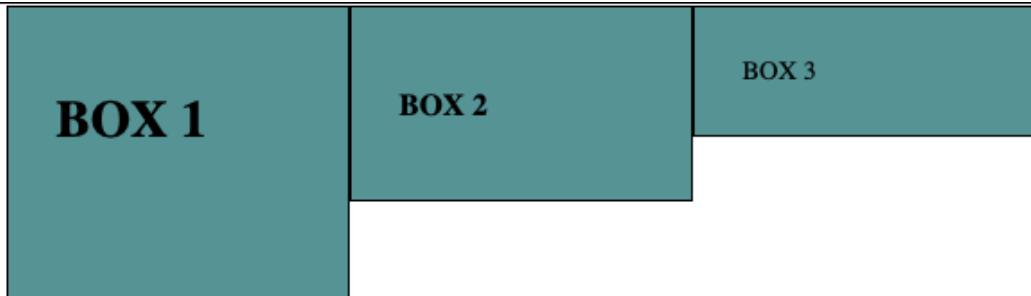
El aspecto que tenemos

### Cajas repartidas en dos líneas

¿Que pasa si ahora alineamos en el *flex start* del eje secundario? Que los elementos se alinearían al top de cada fila (tenemos dos filas en este caso)

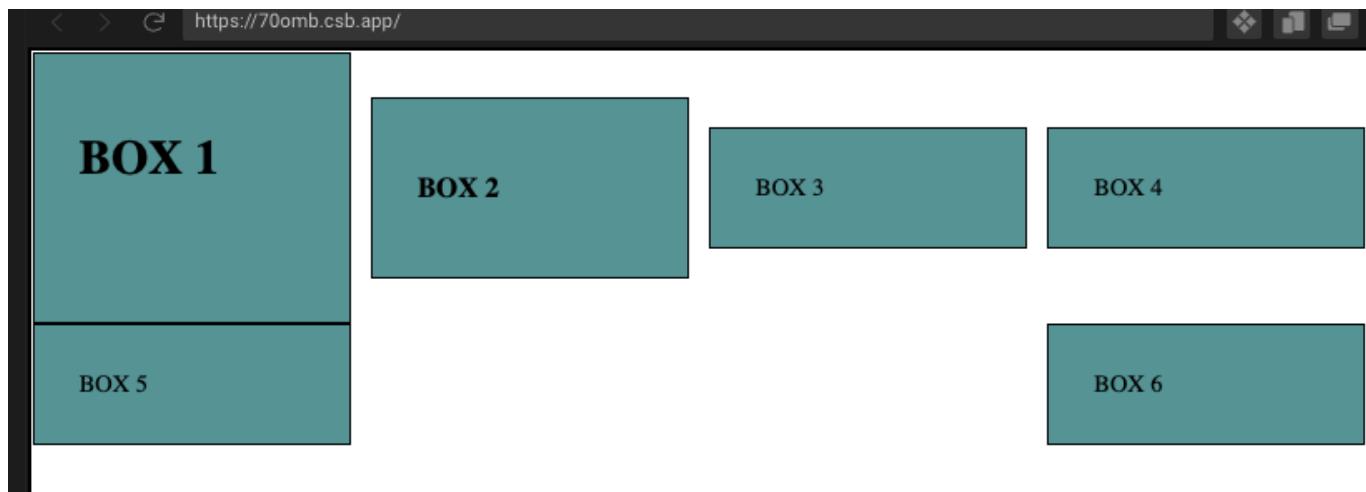
```
#flex-container {
  box-sizing: border-box;
  height: 100vh;
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  border: 3px solid black;
  justify-content: center;
- align-items: baseline;
+ align-items: flex-start;

}
```



Si jugamos ahora con el *align-content* podemos encontrarnos que en algunos casos puede entrar en conflicto con *align-items*

```
#flex-container {  
  box-sizing: border-box;  
  height: 100vh;  
  display: flex;  
  flex-direction: row;  
  flex-wrap: wrap;  
  border: 3px solid black;  
  - justify-content: center;  
  + justify-content: space-between;  
  + align-content: flex-start;  
  - align-items: flex-start;  
  + align-items: center;  
}
```



La primera fila ocuparía todo lo que tiene que ocupar (el tamaño del mayor elemento de esta fila), con *align-content* le decimos que los items arriba en la fila, y cada fila ocupa lo que ocupe el tamaño del más grande que está dentro, si quieres, puedes probar a darle a *box5* un *padding-top* de 200px.

```
#box1 {  
  padding-bottom: 70px;  
}  
  
+ #box5 {  
+   padding-top: 200px;  
+ }
```

Y dentro de cada fila organizamos los *items* con *align*.

Si queremos que ahora todos los elementos tengan el máximo de la fila podemos hacer:

```
#flex-container {  
  box-sizing: border-box;  
  height: 100vh;  
  display: flex;  
  flex-direction: row;  
  flex-wrap: wrap;  
  border: 3px solid black;  
  justify-content: space-between;  
-  align-items: center;  
+  align-items: stretch;  
-  align-content: flex-start;  
+  align-content: stretch;  
}
```

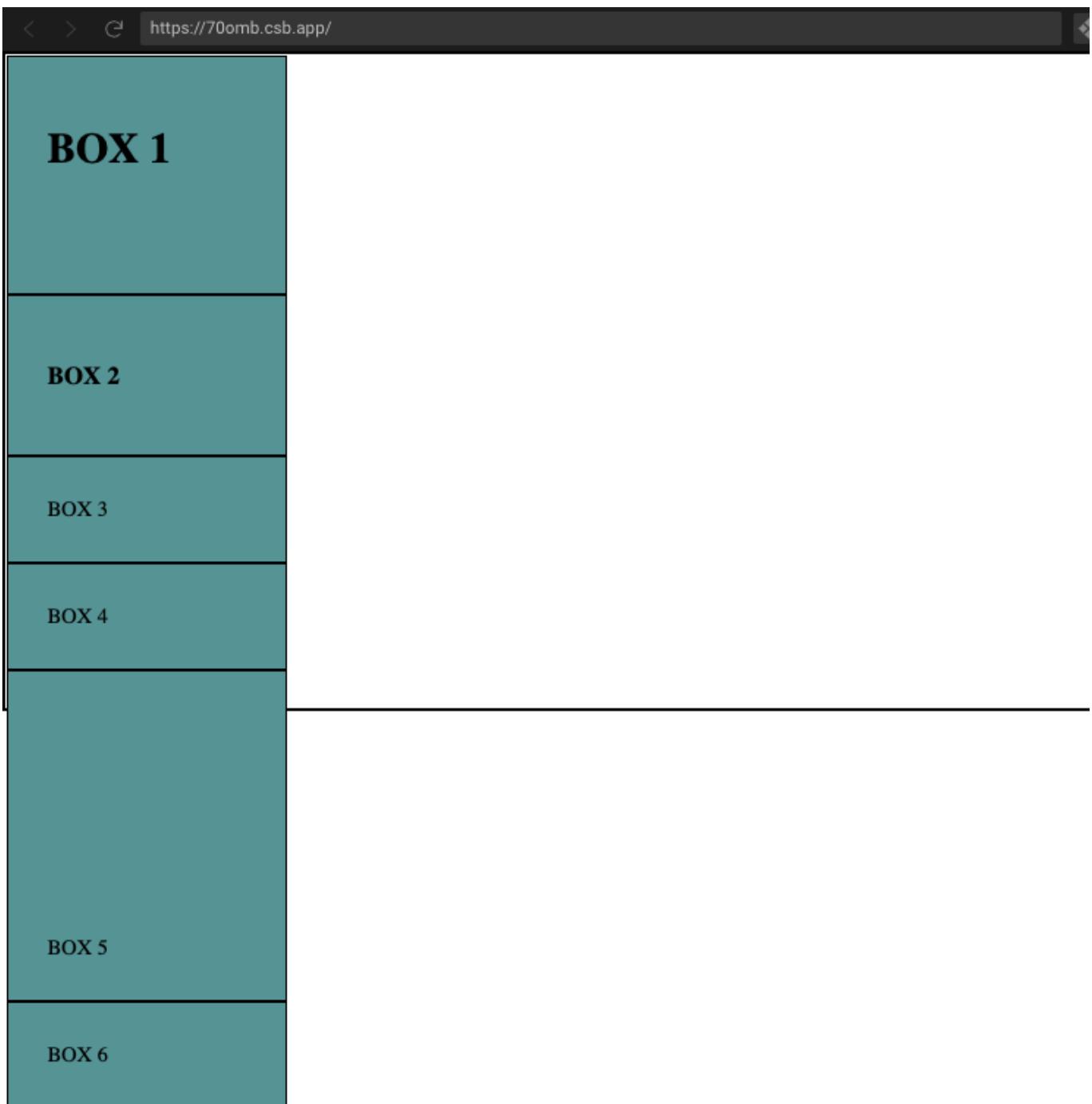
Vamos a montar ejemplos con *flex-direction column*

Vamos a modificar el *flex-container*

- Le indicamos que el *flex-direction* es *column*

- Le damos un height fijo de 500px.
- En el *flex-wrap* le indicamos que no queremos *wrap*
- Al *justify-content* le asignamos *flex-start*

```
- #flex-container {  
-   box-sizing: border-box;  
-   height: 100vh;  
-   display: flex;  
-   flex-direction: row;  
-   flex-wrap: wrap;  
-   border: 3px solid black;  
-   justify-content: space-between;  
-   align-items: stretch;  
-   align-content: stretch;  
- }  
  
+ #flex-container {  
+   box-sizing: border-box;  
+   height: 500px;  
+   display: flex;  
+   flex-direction: column;  
+   flex-wrap: nowrap;  
+   border: 3px solid black;  
+   justify-content: flex-start;  
+ }
```



Como vemos, no puede representar los items en el espacio que le proponemos, podríamos plantearnos jugar con la propiedad *overflow* y decirle que esconda los elementos que no caben o que añada un *scroll* (pruebalo 😊)

Vamos a eliminar tres items del HTML para poder jugar de manera fácil:

./index.html

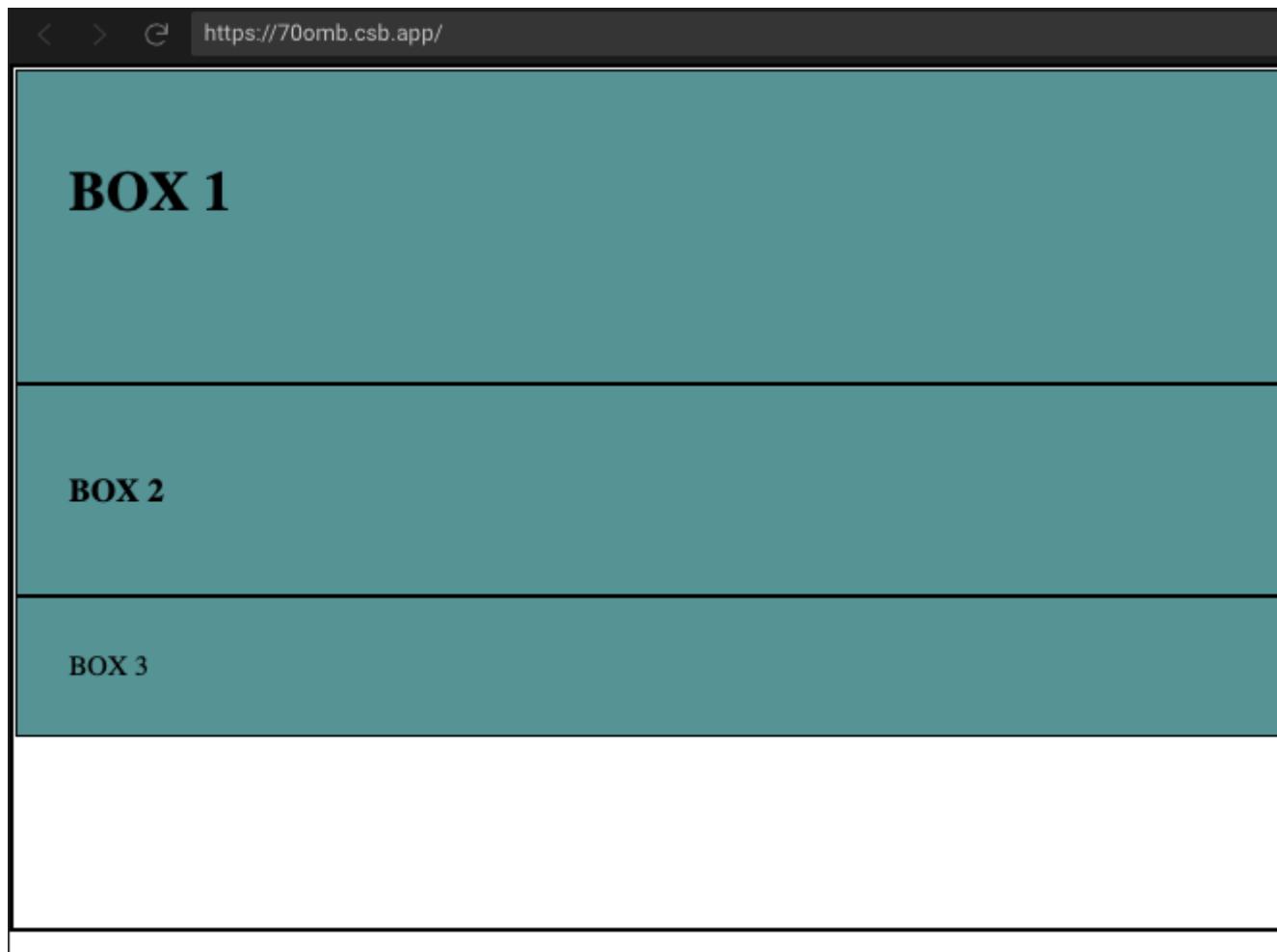
```
<body>
  <div id="flex-container">
    <div class="box" id="box1"><h1>BOX 1</h1></div>
    <div class="box" id="box2"><h3>BOX 2</h3></div>
    <div class="box" id="box3">BOX 3</div>
    -   <div class="box" id="box3">BOX 5</div>
```

```
-     <div class="box" id="box4">BOX 6</div>
  </div>
</body>
```

Vamos ahora a quitar el ancho de las cajas (lo tenemos limitado a 150 pixeles)

./src/miestilo.css

```
.box {
-   width: 150px;
  border: 1px solid black;
  background-color: cadetblue;
  padding: 30px;
}
```

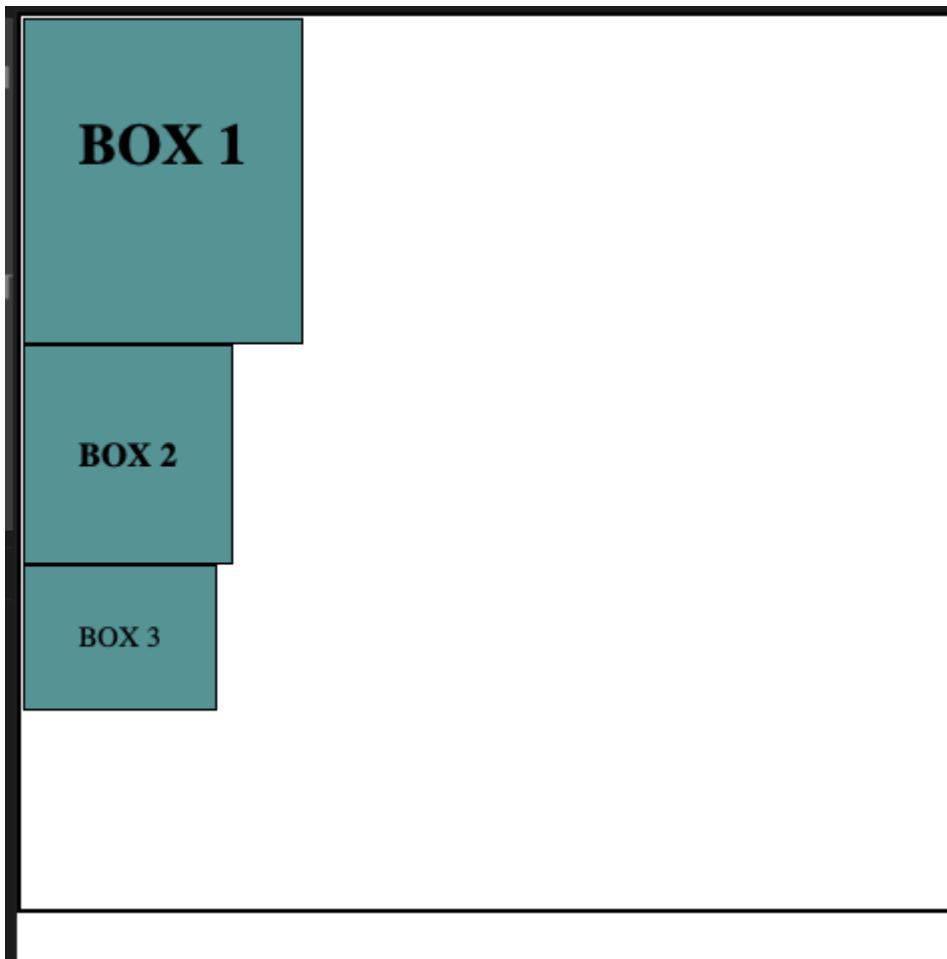


Ahora mismo el eje secundario lo tenemos definido como *stretch*: y ocupa todo el ancho posible, vamos a cambiar el *align-items* a *flex-start*

```
#flex-container {
  box-sizing: border-box;
  height: 500px;
  display: flex;
```

```
flex-direction: column;  
flex-wrap: nowrap;  
border: 3px solid black;  
justify-content: flex-start;  
+ align-items: flex-start;  
}
```

Fíjate como ahora cada caja tiene su tamaño:



Si ahora en *align-items* le indicáramos *align-items center* nos centraría los elementos en el eje secundario (en este caso el horizontal).

```
#flex-container {  
  box-sizing: border-box;  
  display: flex;  
  height: 500px;  
  flex-direction: column;  
  flex-wrap: nowrap;  
  border: 3px solid black;  
  justify-content: flex-start;  
  + align-items: center;  
}
```

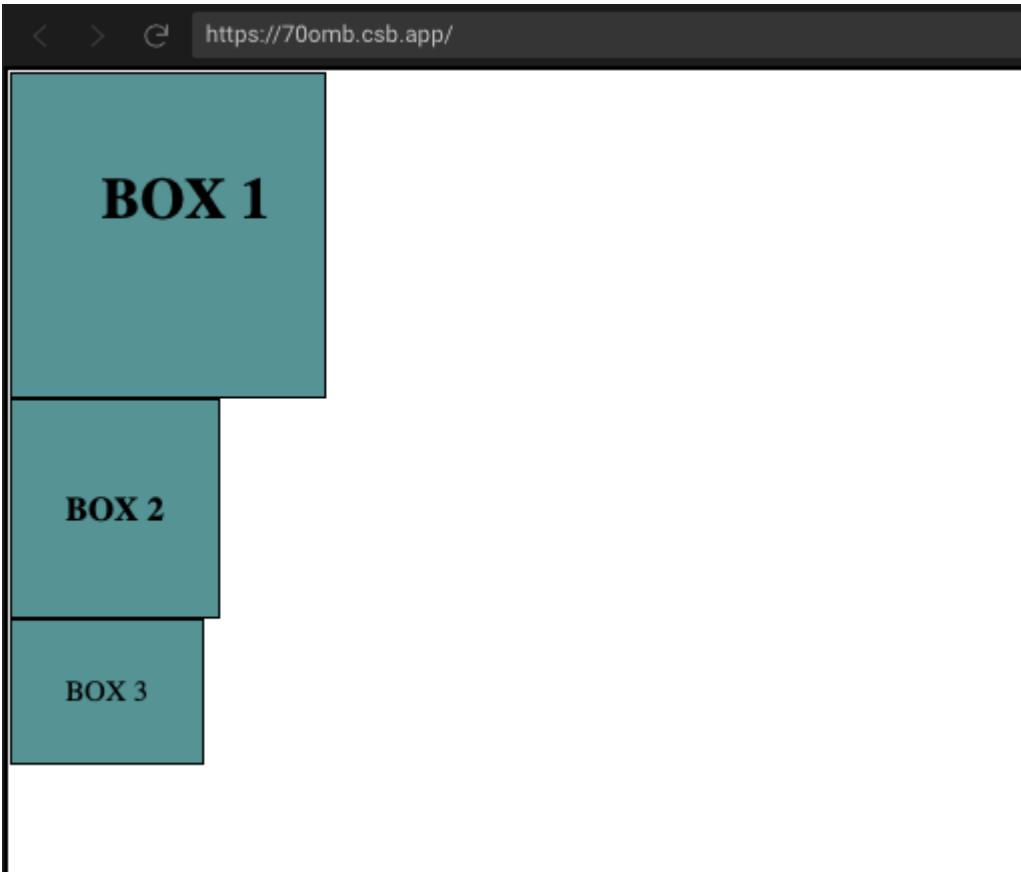
Una cosa que tenemos que tener en cuenta es que para el *flex-direction column*, el *align-items baseline* no lo tiene en cuenta, si quieras podemos probar esto:

```
#flex-container {  
    box-sizing: border-box;  
    display: flex;  
    height: 500px;  
    flex-direction: column;  
    flex-wrap: nowrap;  
    border: 3px solid black;  
    justify-content: flex-start;  
    - align-items: center;  
    + align-items: baseline;  
}
```

Y para que se vea más claro a *box1* le metemos un *padding left* para que tenga un ancho distinto al resto de cajas:

```
#box1 {  
    + padding-left: 50px;  
    padding-bottom: 70px;  
}
```

Esto quedaría de la siguiente manera (fíjate que *box 1* no esta alineado con el texto de *box2*)



Aquí puedes jugar asignando *flex-end* y ver el resultado.

Vamos a ver ahora como se crean columnas nuevas con el eje vertical,

Volvemos a añadir las cajas que eliminamos anteriormente

*./index.html*

```
<body>
  <div id="flex-container">
    <div class="box" id="box1"><h1>BOX 1</h1></div>
    <div class="box" id="box2"><h3>BOX 2</h3></div>
    <div class="box" id="box3">BOX 3</div>
    <div class="box" id="box4">BOX 4</div>
+    <div class="box" id="box5">BOX 5</div>
+    <div class="box" id="box6">BOX 6</div>
  </div>
</body>
```

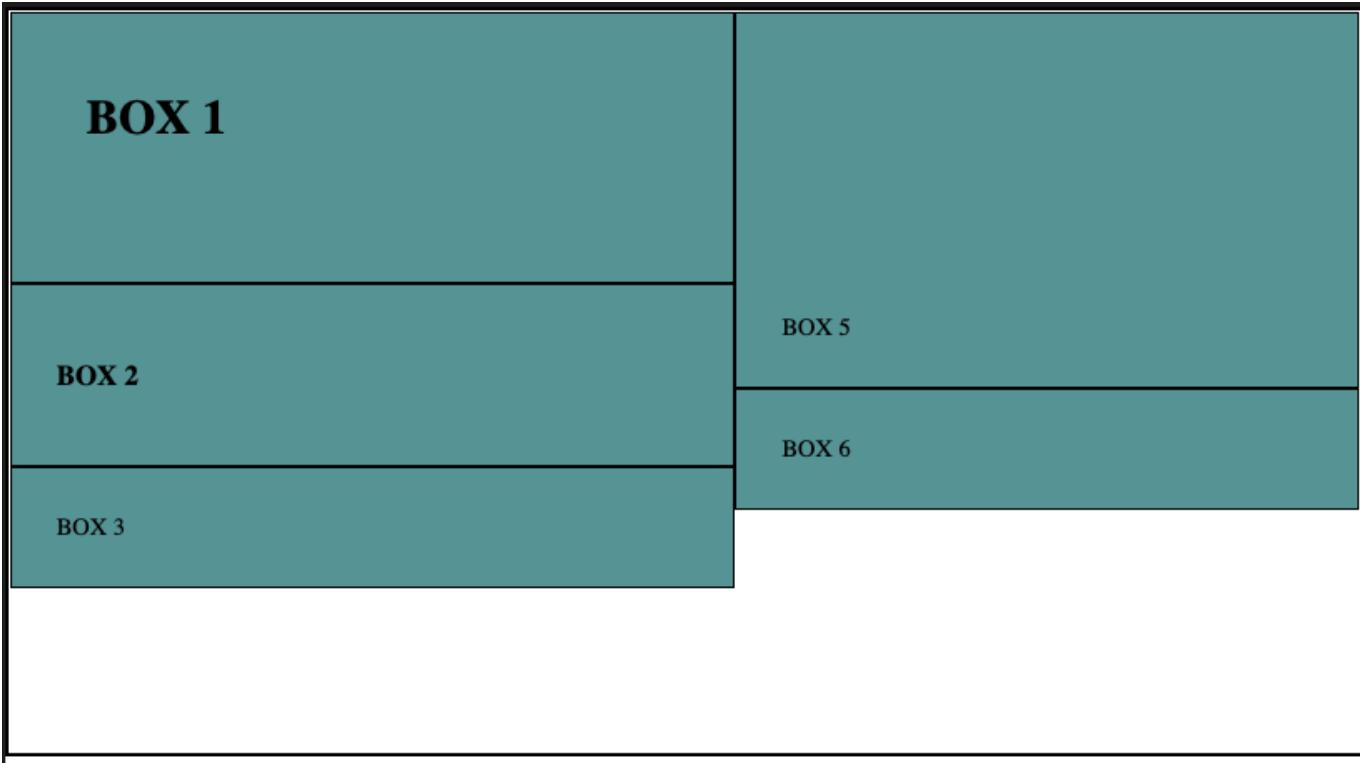
Vamos a hacer algunas modificaciones en el container:

```
#flex-container {
  box-sizing: border-box;
  display: flex;
  height: 500px;
  flex-direction: column;
  flex-wrap: nowrap;
  border: 3px solid black;
  justify-content: flex-start;
-  align-items: baseline;
}
```

Ya tenemos los seis items ocupando todo el ancho posible.

Para crear nuevas columnas le asignamos a la propiedad *flex-wrap* el valor *wrap*

```
#flex-container {
  box-sizing: border-box;
  display: flex;
  height: 500px;
  flex-direction: column;
-  flex-wrap: nowrap;
+  flex-wrap: wrap;
  border: 3px solid black;
  justify-content: flex-start;
}
```



Aquí podríamos jugar con valores tales como *wrap-reverse*.

Si te fijas en el eje secundario (el horizontal) los elementos toman todo el valor que pueden, si queremos que sólo ocupen el valor necesario, podemos cambiar el valor de la propiedad *align-items* que por defecto está a *stretch*, lo ponemos como *flex-start*

```
#flex-container {  
    box-sizing: border-box;  
    display: flex;  
    height: 500px;  
    flex-direction: column;  
    flex-wrap: wrap;  
    border: 3px solid black;  
    justify-content: flex-start;  
    + align-items: flex-start;  
}
```



Podemos jugar con varios valores de *align-items* por ejemplo *center*.

```
#flex-container {  
  box-sizing: border-box;  
  display: flex;  
  height: 500px;  
  flex-direction: column;  
  flex-wrap: wrap;  
  border: 3px solid black;  
  justify-content: flex-start;  
  - align-items: flex-start;  
  + align-items: center;  
}
```

Y *align-content* por defecto esta como *stretch*, así tenemos dos columnas de igual tamaño que intenta ocupar todo, vamos a decirle que no coja el tamaño máximo, vamos a darle un *align-content flex-start*.

```
#flex-container {  
  box-sizing: border-box;  
  display: flex;  
  height: 500px;  
  flex-direction: column;  
  flex-wrap: wrap;  
  border: 3px solid black;  
  justify-content: flex-start;  
  align-items: center;  
}
```

```
+ align-content: flex-start;  
}
```



Y si quisiera repartir el alto para todas las cajas (eje principal):

```
#flex-container {  
  box-sizing: border-box;  
  display: flex;  
  height: 500px;  
  flex-direction: column;  
  flex-wrap: wrap;  
  border: 3px solid black;  
  - justify-content: flex-start;  
  + justify-content: space-between;  
  align-items: center;  
  align-content: flex-start;  
}
```

Otro *setting* interesante: *justify-content space-around* fíjate que se queda con la mitad del espacio el primero y el último, mira la diferencia con *space-evenly*.

Fijaos que con pocos cambios todo lo que podemos hacer con un *layout*, ¿Te imaginas tener que maquetar algo así sin utilizar *flex-box*?

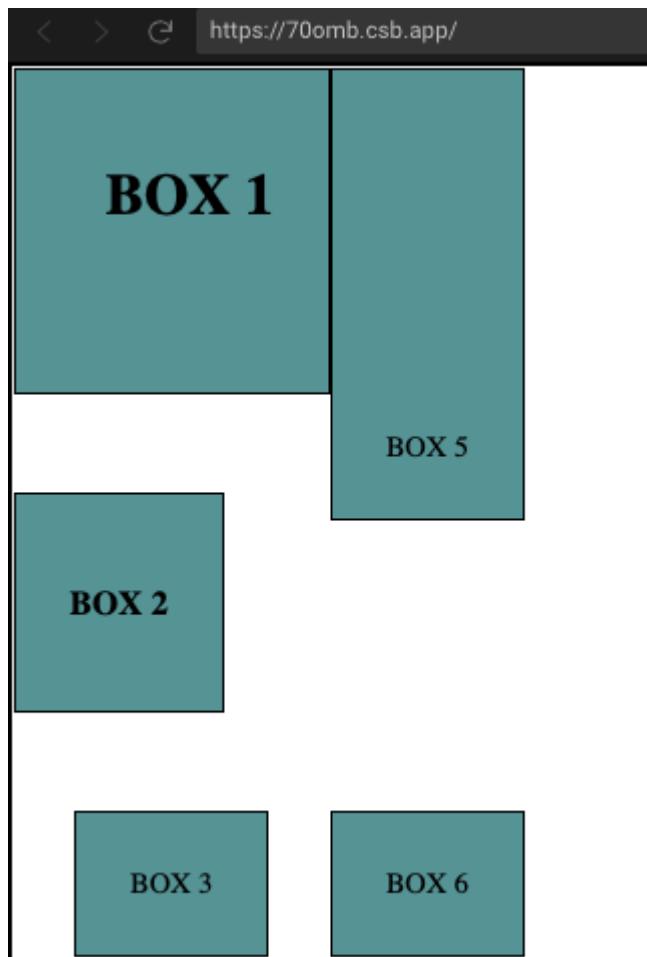
Un tema interesante también es la accesibilidad, podemos definir el orden del HTML en un sentido para que un lector lo lea en ese sentido, y sin embargo cambiar el orden de visualización utilizando las propiedades de *flex-box*: por ejemplo tenemos un menú y una parte principal, de cara al lector podemos

colocar el contenedor del menú a la izquierda, y si queremos podemos posicionarlo visualmente a la derecha podemos usar un *row-reverse*.

Hasta ahora hemos visto como alinear todos los elementos de una manera u otra, pero... ¿y si queremos alinear un elemento en concreto de otra forma?... para eso en un *item* podemos usar *align-self*, vamos a definirlo para el elemento #box2

```
#box1 {  
  padding-left: 50px;  
  padding-bottom: 70px;  
}  
  
+ #box2 {  
+   align-self: flex-start;  
+ }
```

Fíjate que interesante que box2 cobra vida propia y se alinea a la izquierda.



## Flex-grow y Flex-shrink

Limpiamos el HTML y el css.

El contenido nuevo del body:

```
<body>
+ <div id="flex-container">
+   <div class="box" id="box1">BOX 1</div>
+   <div class="box" id="box2">BOX 2</div>
+   <div class="box" id="box3">BOX 3</div>
+ </div>
</body>
```

El contenido de arranque del css:

- El *body* con los márgenes a cero.
- El *flex-container* al que le añadimos un *border* además del display *flex*
- Una clase *box* en la que le damos un *padding* de *\_50px* y color negro.

```
body {
  margin: 0px;
  padding: 0px;
}

#flex-container {
  border: 3px solid black;
  display: flex;
  flex-direction: row;
}

.box {
  padding: 50px;
  border: 1px solid black;
}
```

En este caso, cada elemento tiene su tamaño y nos sobra espacio par el contenedor, si ponemos ahora un *display inlineflex* el tamaño del contenedor se ajustaría al tamaño de las cajas:

```
#flex-container {
  border: 3px solid black;
- display: flex;
+ display: inline-flex;

  flex-direction: row;
}
```

¿Qué pasa si ahora le meto un *justify-content space-between* ?

```
#flex-container {  
    border: 3px solid black;  
    display: inline-flex;  
    + justify-content: space-between;  
    flex-direction: row;  
}
```

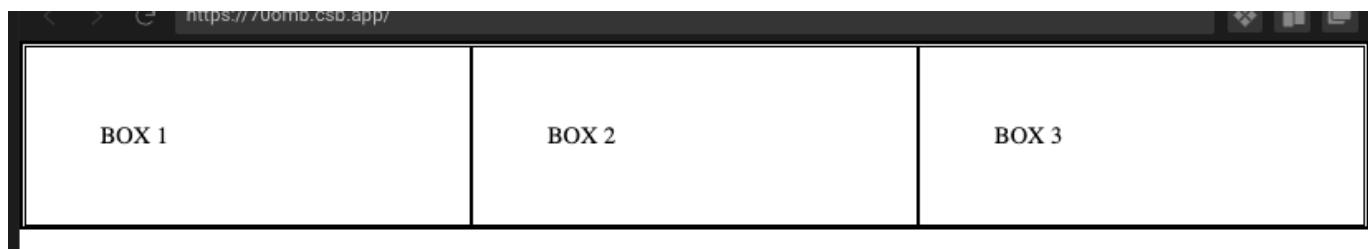
En este caso no pasa nada, porque al haber hecho el *display inline-flex* no lo queda espacio material al contenedor para hacer un *space-between*.

Volvemos a restaurar el *display flex* y eliminamos el *justify-content* así volvemos a tener espacio y vamos a jugar a espaciar los items.

```
#flex-container {  
    border: 3px solid black;  
    - display: inline-flex;  
    + display: flex;  
    - justify-content: space-between;  
    flex-direction: row;  
}
```

Vamos a indicarle que todos nuestros box tienen un *flex 1*: así pues todos los items ocupan el mismo espacio, el espacio del contenedor en vez de repartirlo donde toque, se lo mete a los *items*:

```
.box {  
    + flex: 1;  
    padding: 50px;  
    border: 1px solid black;  
}
```



En este caso pasa como el anterior si quiero jugar con el *justify-content* no puedo ya que no queda espacio disponible.

Si miramos las *devtools* puedes apreciar que la propiedad *flex:1*, en realidad establece:

- *flex-grow* a 1.
- *flex-shrink* a 1.
- *flex-basis* a 0%.

Styles Computed Layout Event Li

Filter

```
element.style {
}

.box {
    flex: 1 1 0%;
    flex-grow: 1;
    flex-shrink: 1;
    flex-basis: 0%;

    padding: 50px;
    border-width: 1px;
    border-style: solid;
    border-color: black;
    border-image: initial;
}
```

Vamos a jugar ahora con los *ids* de cada caja, así cada uno tendrá un comportamiento distinto.

Primero eliminamos el *flex: 1* de *.box*

*./src/miestilo.css*

```
.box {
    flex: 1;
    padding: 50px;
    border: 1px solid black;
}
```

Y vamos a añadirlos para cada *id* de los *divs* de caja que hemos definido en el *html*, y le vamos a decir:

- Que *box2* va tener un *flex-grow* de 1, si chequeamos ahora vemos que *box2* toma todo el espacio libre disponible (seguimos...).
- Ahora vamos a decirle que *box1* también tiene un *flex* de 1, así *box1* y *box2* se reparten el espacio disponible (si le pusiéramos un dos, no tomaría el doble, si no su tamaño más el doble del espacio disponible).

```
#box1 {
    flex-grow: 1;
}

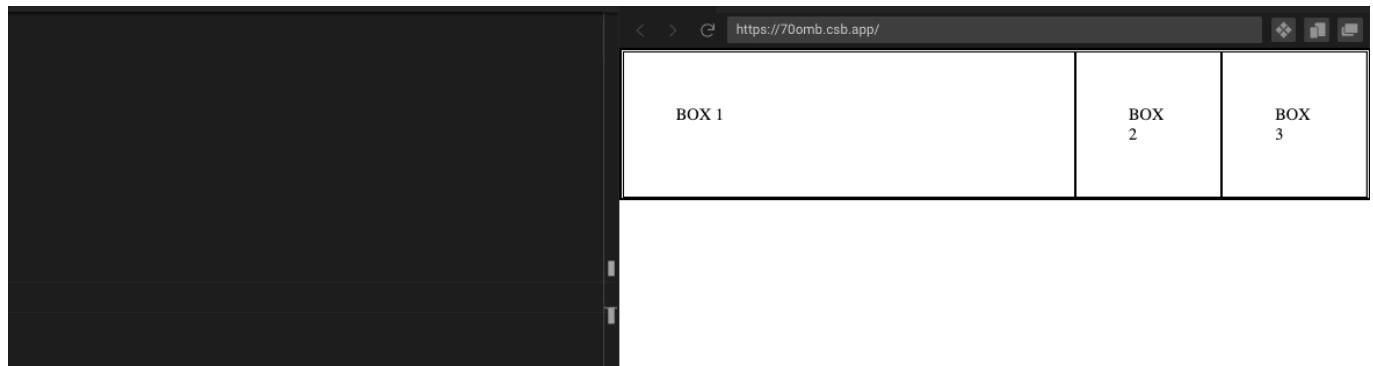
#box2 {
    flex-grow: 1;
}
```

Otra opción interesante es decirle de cuánto mínimo parte el elemento y a partir de ahí crece, tocando el ejemplo anterior:

- A *box1* le damos un factor de crecimiento 1, y un *flex-basis* de 400px
- A *box2* le damos un factor de crecimiento grande 100

```
#box1 {  
+ flex-basis: 400px;  
  flex-grow: 1;  
}  
  
#box2 {  
- flex-grow: 1;  
+ flex-grow: 100;  
}
```

Fíjate aquí que para un tamaño de ventana pequeño *box1* ocupa más que *box2*, ya que su *flex-basis* es de 400 píxeles, el *box1* no deja crecer a nadie hasta que llega a su *basis*, a medida que crecemos *box2* va creciendo más rápido.



Vamos a ver ahora *flex-shrink* el comportamiento es parecido al de *flex-grow* pero en vez de cubrir la capacidad de crecer, cubre la de encoger.

Vamos a modificar los estilos de las dos cajas:

- Eliminamos los *flex-grow*
- Le damos a *box1* y *box2* un *basis* de 200px
- Le damos a *box1* un *flex-shrink* de 1
- Le damos a *box2* un *flex-shrink* de 2 (este encogería más rápido, liberaría más espacio)

```
#box1 {  
- flex-basis: 400px;  
+ flex-basis: 200px;  
- flex-grow: 1;  
+ flex-shrink: 1;  
}
```

```
#box2 {  
+ flex-basis: 200px;  
- flex-grow: 100;  
+ flex-shrink: 2;  
}
```

Con esta configuración, si hay espacio extra no se ocupa (no hay *flex-grow*), pero si se quedan sin espacio, va tomando el espacio disponible de *box2* a más velocidad que de *box1*

Todos estos principios aplican igual si nos ponemos con *flex-direction column*.

Vamos a probar a hacer una ventana con el siguiente layout:

- Tenemos una cabecera de la página (*header*).
- Tenemos un área principal de contenido (*main*).
- Tenemos un pie de página (*footer*).

Borramos el contenido del *body* en el *html* y metemos el siguiente.

```
<body>  
+ <header>Mi cabecera</header>  
+ <main>Mi contenido principal</main>  
+ <footer>Footer</footer>  
</body>
```

¿Cómo puedo estilar esto?

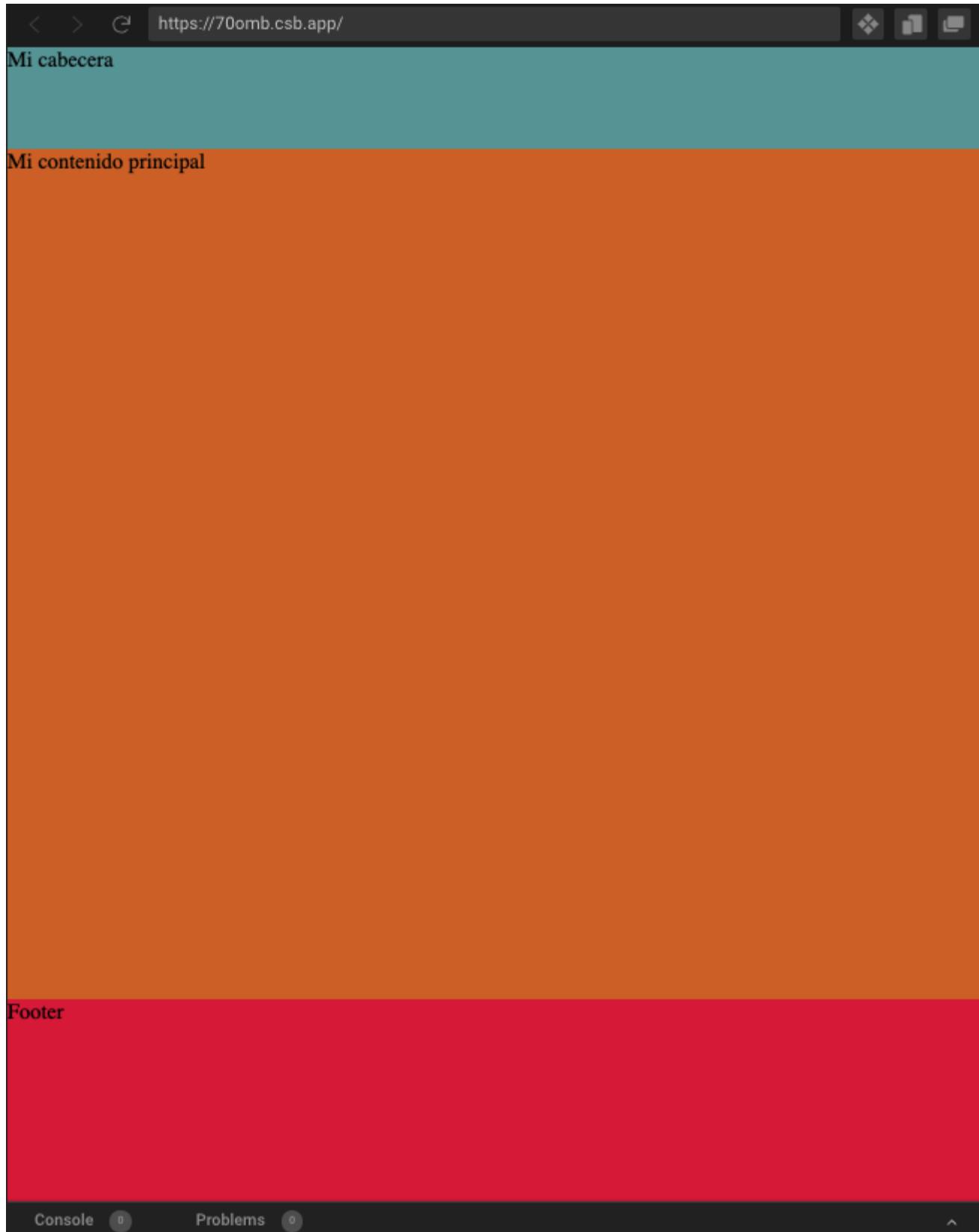
Vamos a por el fichero de estilos:

- Borramos lo que hay y sólo dejamos el *\_body*:
- En el caso del *body* lo añadimos como un contenedor *flex* y le decimos que va a trabajar en modo *columna*, también le decimos que ocupe todo el espacio vertical de la ventana.
- En la cabecera la damos un alto estándar, y el color *cadetblue*.
- En el *main* le indicamos que ocupe todo el espacio que tenga disponible y la damos el color *chocolate*.
- El *footer* lo dejamos tal cual (ocupará el espacio que ocupe su contenido), y le damos el color *crimson*

```
body {  
    margin: 0px;  
    padding: 0px;  
+ height: 100vh;  
+ display: flex;  
+ flex-direction: column;  
}  
  
+ header {  
+ height: 75px;
```

```
+ background-color: cadetblue  
+ }  
  
+ main {  
+ background-color: chocolate;  
+ flex: 1;  
+ }  
  
+ footer {  
+ height: 150px;  
+ background-color: crimson;  
+ }
```

Con esto tengo un *layout* de cabecera/cuerpo/pie de página, en el que el cuerpo mira de tener todo el espacio disponible:



Además si quisiéramos que *header* y *footer* fueran siempre visibles, aunque el *main* tenga mucho contenido, podríamos hacer lo siguiente:

```
main {  
  background-color: chocolate;  
  flex: 1;  
+ overflow-y: auto;  
}
```

Otro tema interesante, ya tengo en el *body* un *flex container*, ... ahora en el *main* puedo anidar otro *flex container* y por ejemplo crearme una zona para el menú lateral, y otro para el contenido:

```
<body>
  <header>Mi cabecera</header>
  -  <main>Mi contenido principal</main>
  +  <main>
  +    <div class="menu">
  +      Mi menu
  +    </div>
  +    <div class="contenido">
  +      Contenido
  +    </div>
  +  </main>
  <footer>Footer</footer>
</body>
```

Puedo definir en el css

- Un contenedor *flex* en el *main*.
- Al menú le damos un ancho fijo y un color *olivedrab*
- Al contenido le decimos que pille todo el espacio disponible.

```
main {
  background-color: chocolate;
  flex: 1;
+ display:flex;
+ flex-direction: row;
}
+
+ .menu {
+   background-color: olivedrab;
+   width:150px;
+ }
+
+ .contenido {
+   flex:1;
+ }
```

## Flex order

Volvemos a Limpiar el HTML y el css.

En el HTML ponemos tres cajas.

*./index.html*

```

<body>
  <div id="flex-container">
    <div class="box" id="box1">BOX 1</div>
    <div class="box" id="box2">BOX 2</div>
    <div class="box" id="box3">BOX 3</div>
  </div>
</body>

```

En el css dejamos los estilos del *body* (sin márgenes), y añadimos un *flex-container* con *display flex*, y con *flex-direction row* y *nowrap*, y le damos un estilado a las cajas.

*./src/miestilo.css*

```

body {
  margin: 0px;
  padding: 0px;
}

#flex-container {
  border: 3px solid black;
  display: flex;
  flex-direction: row;
  flex-wrap: nowrap;
}

.box {
  padding: 50px;
  border: 1px solid black;
}

```

Vamos a jugar con la propiedad *order*, para ello vamos a jugar con los *ids* de cada caja, estilamos:

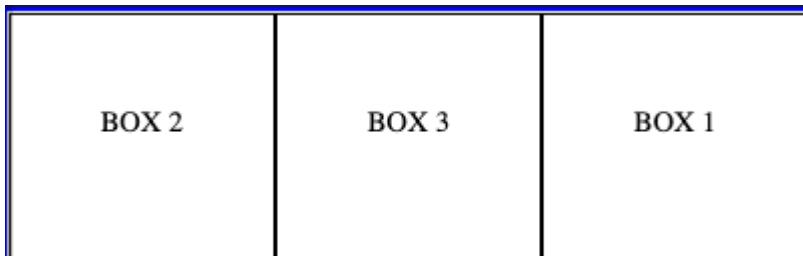
```

.box {
  padding: 50px;
  border: 1px solid black;
}

+ #box1 {
+   order: 3;
+ }

```

Al hacer esto, *box1* se va al final del todo.



Si no indicamos nada, cada *item* tiene *order 0*, así que para poner un item al principio sin tocar el resto de valores por defecto tendríamos que poner *-1*.

Esto se puede combinar con *row-reverse*...

## Anexo - Recursos flexbox

---

Guía:

[A complete guide to Flexbox](#)

Juegos

[Flexbox Froggy](#)

[Flex box defense](#)

[Flex box zombies](#)