
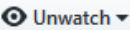




#8

CSS Avanzado – Clase anterior + Layouts

Refresquemos





 [ada-frontend-septimageracion](#) / **clase-07**
forked from [fmontalenti/clase-06](#)

 Unwatch 1  Star 0  Fork 1

[Code](#) [Pull requests](#) 0 [Projects](#) 0 [Wiki](#) [Insights](#) [Settings](#)


No description, website, or topics provided. [Edit](#)




[Manage topics](#)



 **11** commits  **2** branches  **0** releases  **2** contributors

Branch: **master** [New pull request](#) [Create new file](#) [Upload files](#) [Find File](#) [Clone or download](#)

This branch is 6 commits ahead of [fmontalenti:master](#). [Pull request](#) [Compare](#)

 **FMGordillo** Update README.md Latest commit [2f839e3](#) 13 minutes ago

 README.md	Update README.md	13 minutes ago
 clase-07.pdf	Clase agregada	14 minutes ago
 index.html	FIX: Ultimo, olvide borrar los tags de mas	a day ago

 **README.md** 

Clase 07

[Ver clase](#)

display...

- block
- inline
- none
- visibility: hidden;
 - Afecta la distribución, pero el elemento no se ve más

Position (clase 09)

- `static` (por defecto)
- `relative`
- `fixed`
- `absolute`
- `sticky`

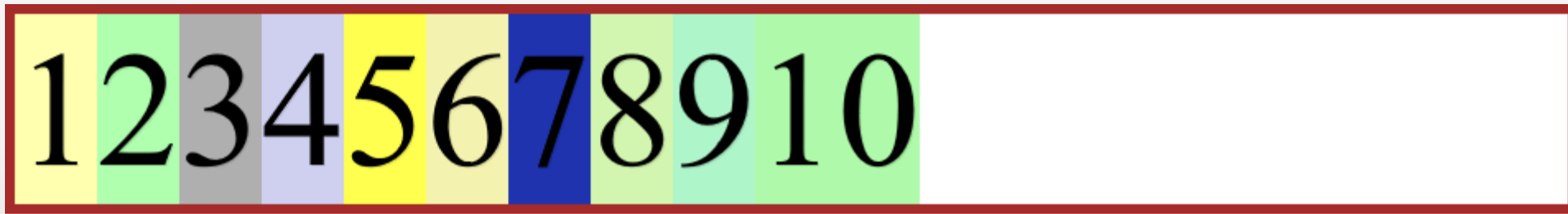
display: flex;

```
.container {  
  display: flex;  
  border: 6px solid brown;  
}
```

Flex

- Resuelve problemas que “display: float” no
 - Lo vemos la clase 09
- Distribuye el espacio libre de manera “inteligente” (sin hacer cálculos)
- Las propiedades se dividen en dos tipos:
 - De contenedores
 - De items

```
display: flex;
```



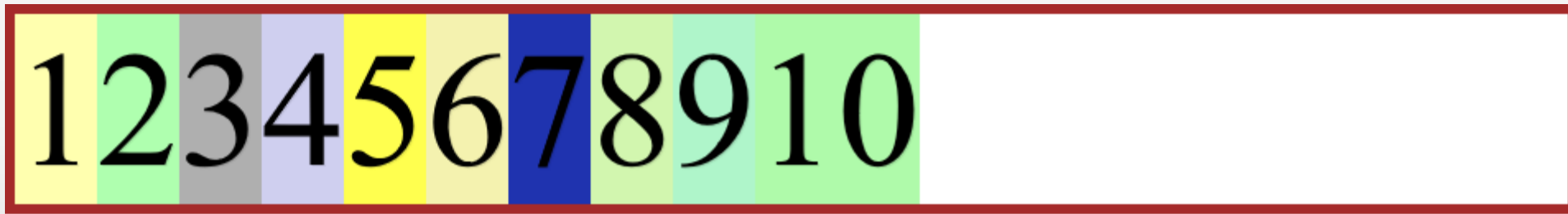
Onda “display: block;” pero de Flex

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



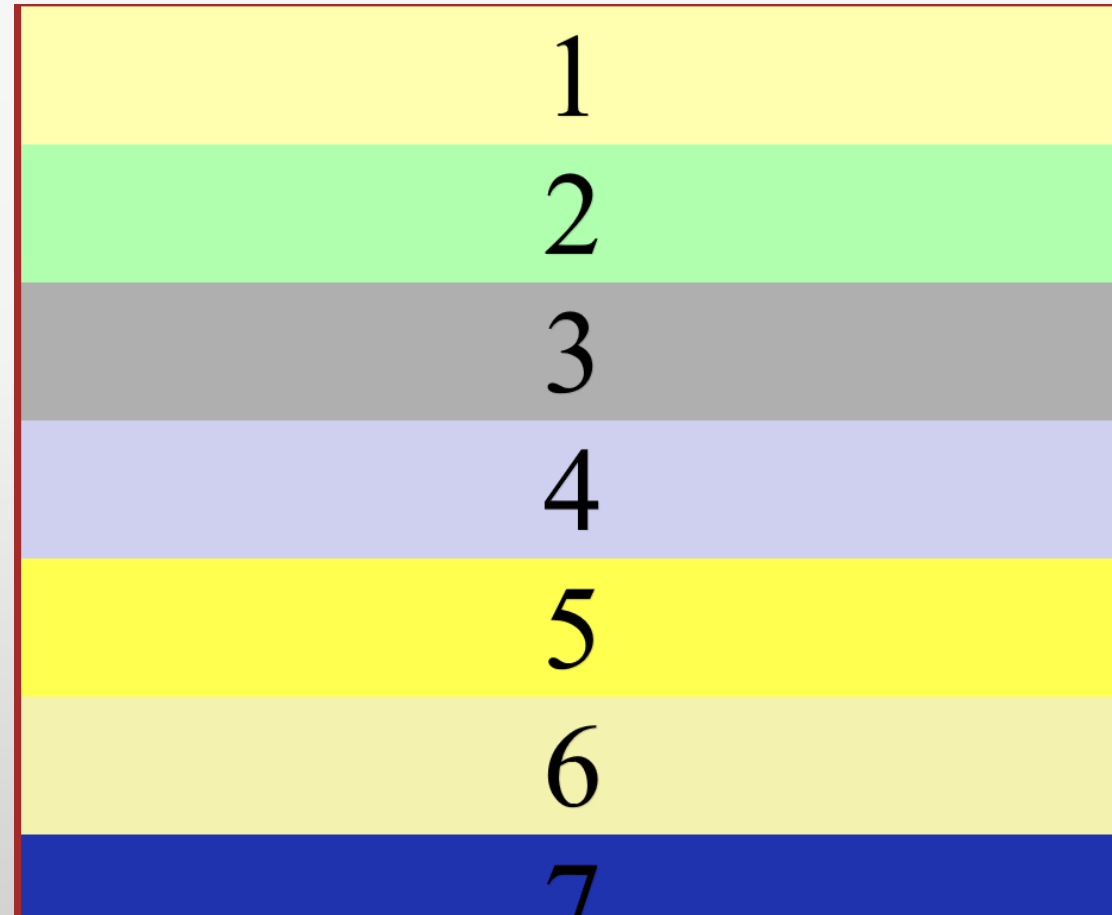
Onda “display: inline;” pero de Flex


```
flex-direction: row;
```



Por defecto, al poner “display: flex;”

```
flex-direction: column;
```



Partamos de...

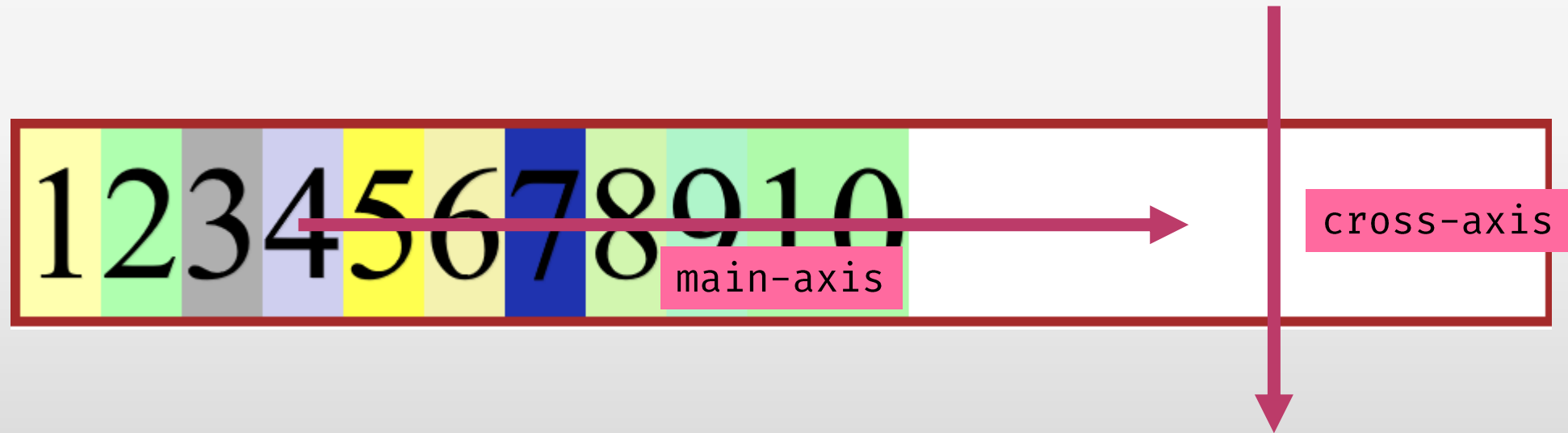
```
/* Vamos a trabajar aca */  
.container {  
  display: flex;  
  flex-direction: row; /* por defecto */  
  border: 6px solid ■ brown;  
}
```

Partamos de...

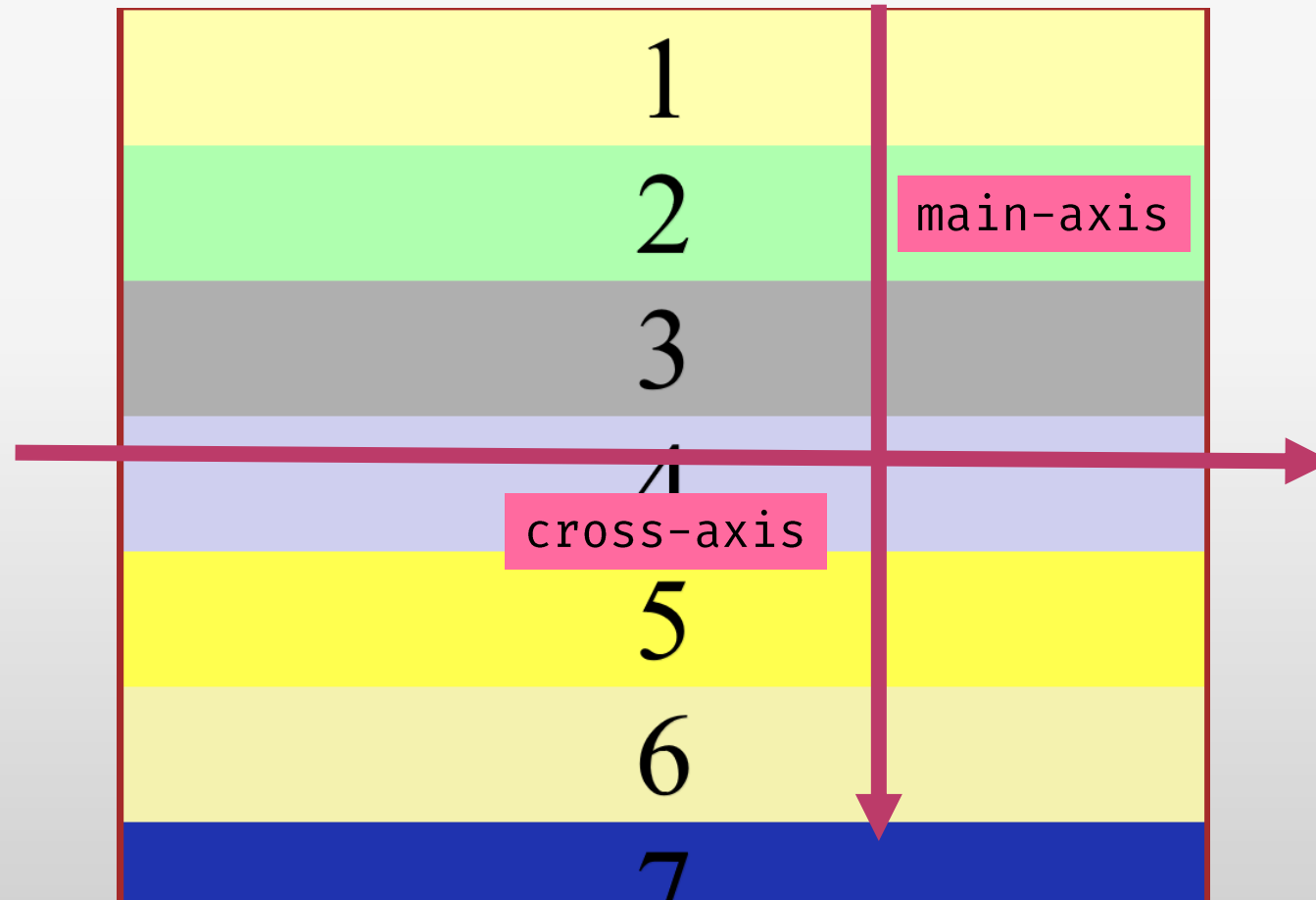
¿Qué más hay que saber?

- `main-axis` y `cross-axis`:
 - Son importantes al saber cómo vamos a orientar nuestro contenido
- `flex-direction`:
 - Si está en `row`, el `main-axis` será de izq a der
 - Si está en `column`, el `main-axis` será de arriba hacia abajo

```
flex-direction: row;
```



```
flex-direction: column;
```



¿Y por qué me importa?

- `justify-content` alinea elementos sobre main-axis
- `align-items` alinea sobre cross-axis (similar al de arriba)
- `align-content` alinea elementos sobre cross-axis, pero en grupos

Partamos de...

```
/* Vamos a trabajar aca */  
.container {  
  display: flex;  
  flex-direction: row; /* por defecto */  
  border: 6px solid ■ brown;  
}
```

justify-content

- Flex-start (por defecto)
- Flex-end
- Center
- Space-between
- Space-around
- ...

```
justify-content: 'center'
```



12345678910

`justify-content: 'flex-end'`



12345678910

A horizontal rectangular container with a dark red border. Inside, the numbers 1 through 10 are displayed. The first six numbers (1-6) are in a white area on the left. The remaining four numbers (7-10) are in a row of four colored boxes on the right: 7 is in a blue box, 8 is in a light green box, 9 is in a light blue box, and 10 is in a light green box. This illustrates the 'flex-end' justify-content property where items are aligned to the right side of the container.

justify-content: 'space-between'



align-items

- Stretch (por defecto)
- flex-start
- flex-end
- center
- baseline

```
align-items: 'center'
```

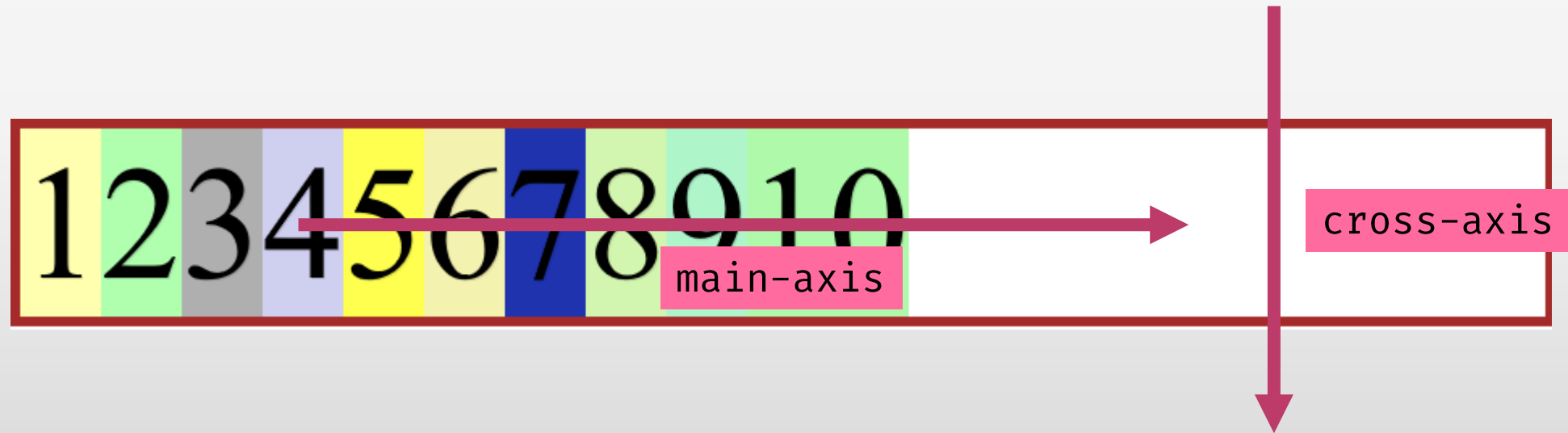


?????

¿Y por qué me importa?

- `justify-content` alinea elementos sobre main-axis
- `align-items` **alinea sobre cross-axis** (similar al de arriba)
- `align-content` alinea elementos sobre cross-axis, pero en grupos


```
flex-direction: row;
```



align-items

- Si seguimos usando “flex-direction: row”, debemos:
 - Darle más espacio al contenedor
- ¿Para qué?
 - Así las cajitas “tienen espacio” verticalmente (en el cross-axis), y así podemos alinearlas

align-items

```
.container {  
  display: flex;  
  flex-direction: row; /* por defecto */  
  height: 400px;  
  border: 6px solid brown;  
}
```

12345678910

align-items: 'center'

```
.container {  
  display: flex;  
  flex-direction: row; /* por defecto */  
  height: 400px;  
  align-items: center;  
  border: 6px solid brown;  
}
```

```
align-items: 'center'
```

12345678910

align-items: 'stretch' (por defecto)

12345678910



`align-items: 'flex-end'`

1 2 3 4 5 6 7 8 9 10

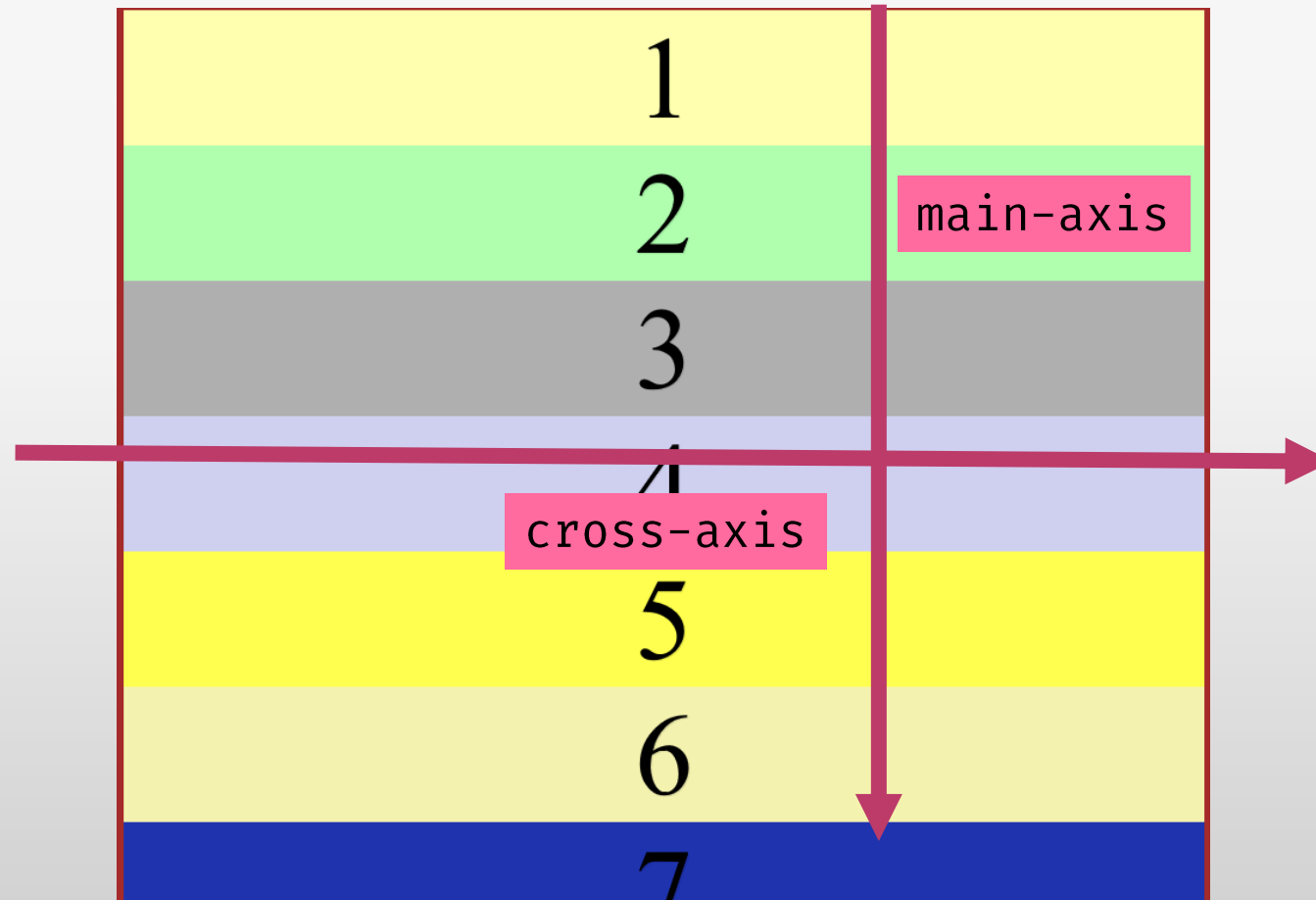
¿Y qué pasa si...?

```
.container {  
  display: flex;  
  flex-direction: column;  
  border: 6px solid brown;  
}
```

Pista: main-axis y cross-axis van a cambiar.

Por ende, va a cambiar la forma en que usamos “align-items” y “justify-content”

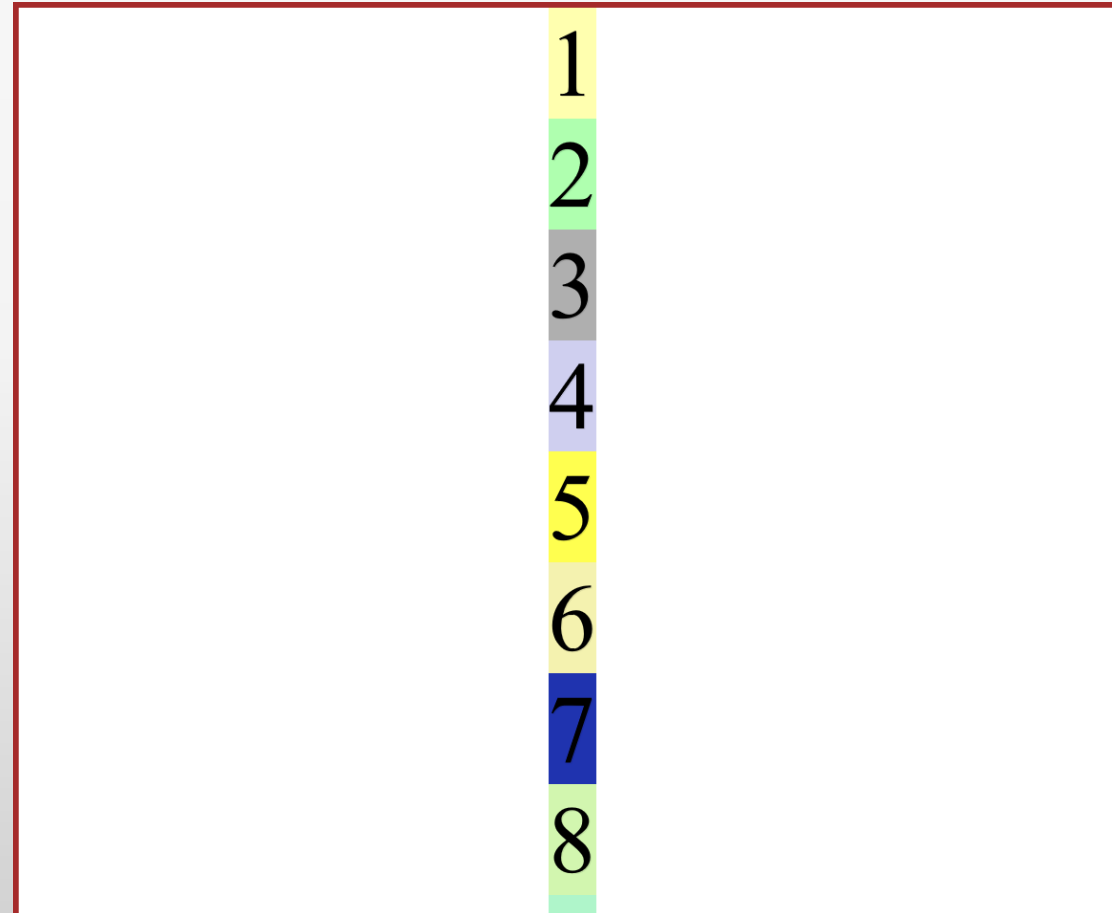

```
flex-direction: column;
```



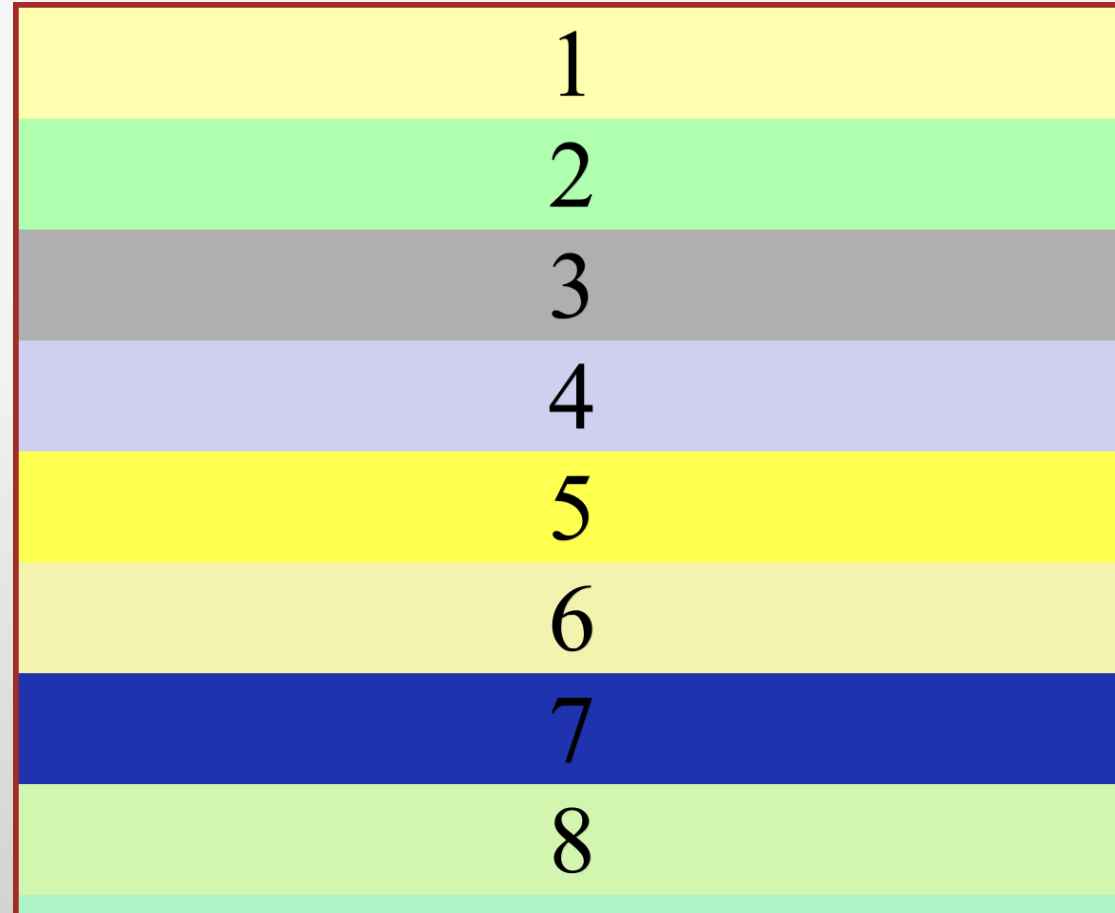
align-items: 'center'

```
.container {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  border: 6px solid brown;  
}
```

```
align-items: 'center'
```



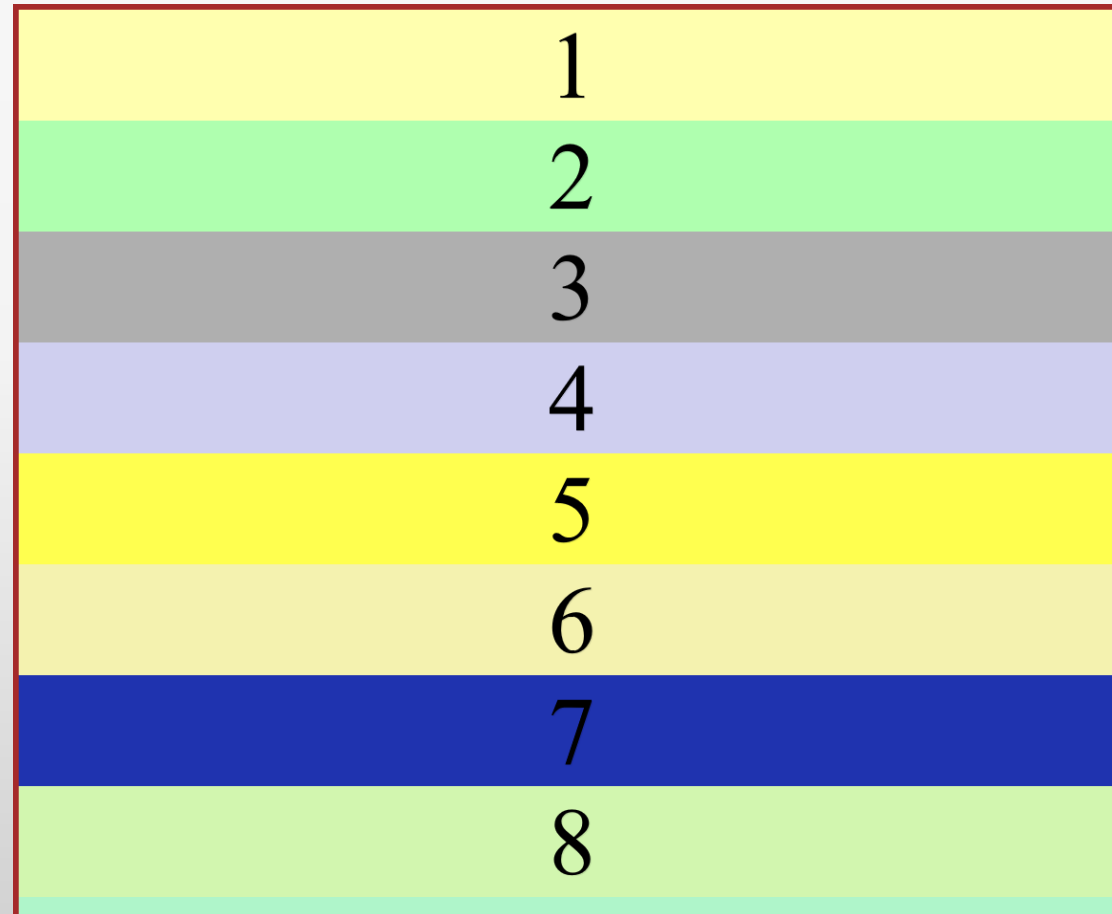
`align-items: 'stretch' (default)`



justify-content: 'center'

```
.container {  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  border: 6px solid brown;  
}
```

`justify-content: 'center'`



?????

¿Y por qué me importa?

- `justify-content` alinea elementos sobre main-axis
- `align-items` alinea sobre cross-axis (similar al de arriba)
- `align-content` alinea elementos sobre cross-axis, pero en grupos

justify-content

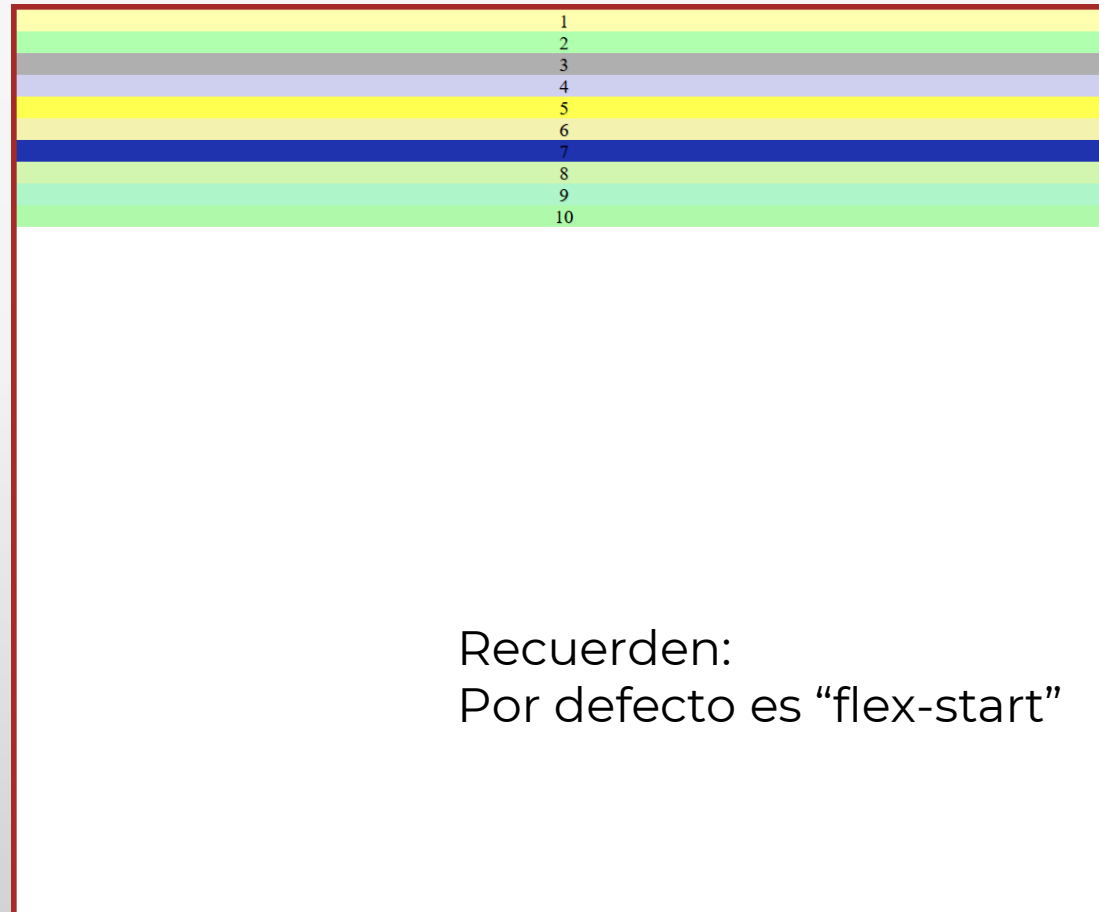
- Si seguimos usando “flex-direction: column”, debemos:
 - Usar el 100% del alto de pantalla
 - Hacer más chicas las cajitas
- ¿Para qué?
 - Así vemos cómo se alinean las cajitas en el “main-axis” (verticalmente). Si están todas juntas, no podemos alinearlas, ya que **no tienen espacio**

justify-content

```
.container {  
  display: flex;  
  flex-direction: column;  
  height: 100vh;  
  border: 6px solid brown;  
}
```

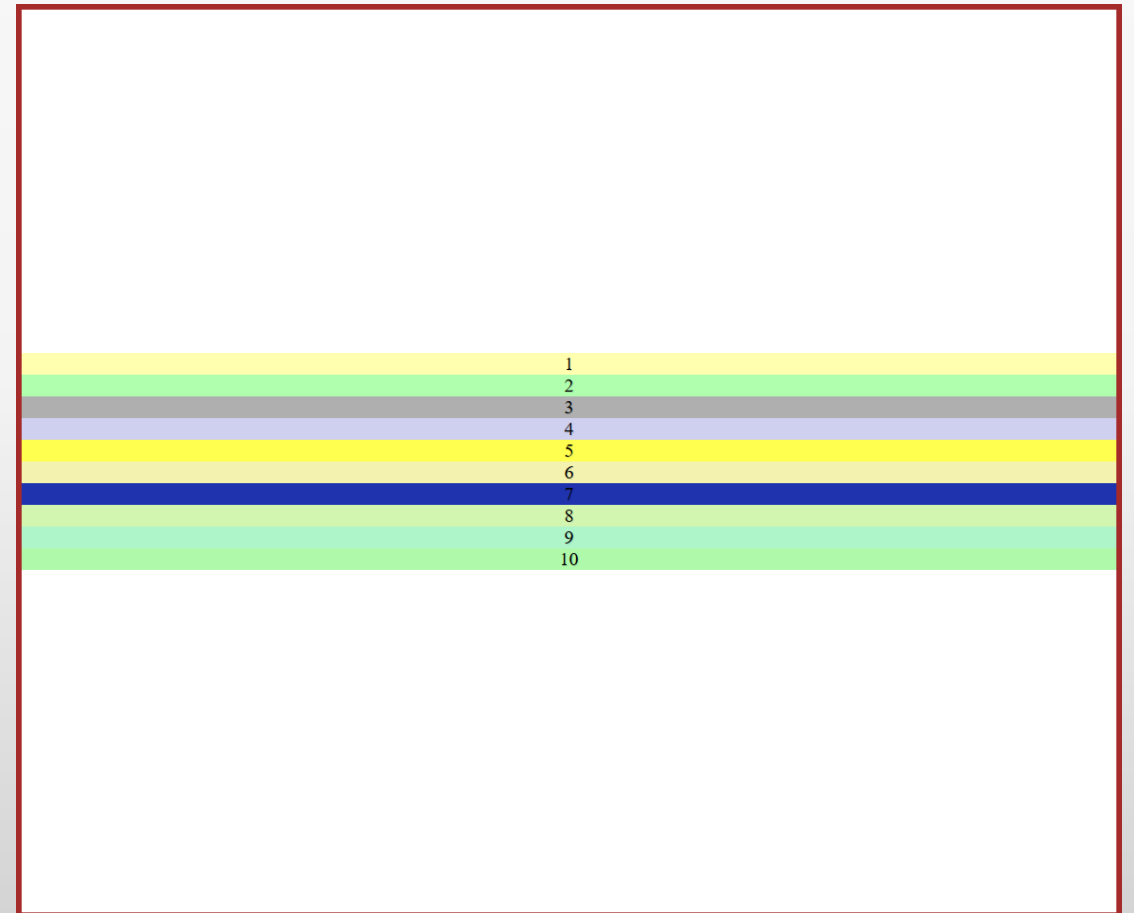
```
.box {  
  text-shadow: 1px 1px 0 rgba(0, 0, 0, 0.1);  
  font-size: 20px;  
  text-align: center;  
}
```

justify-content

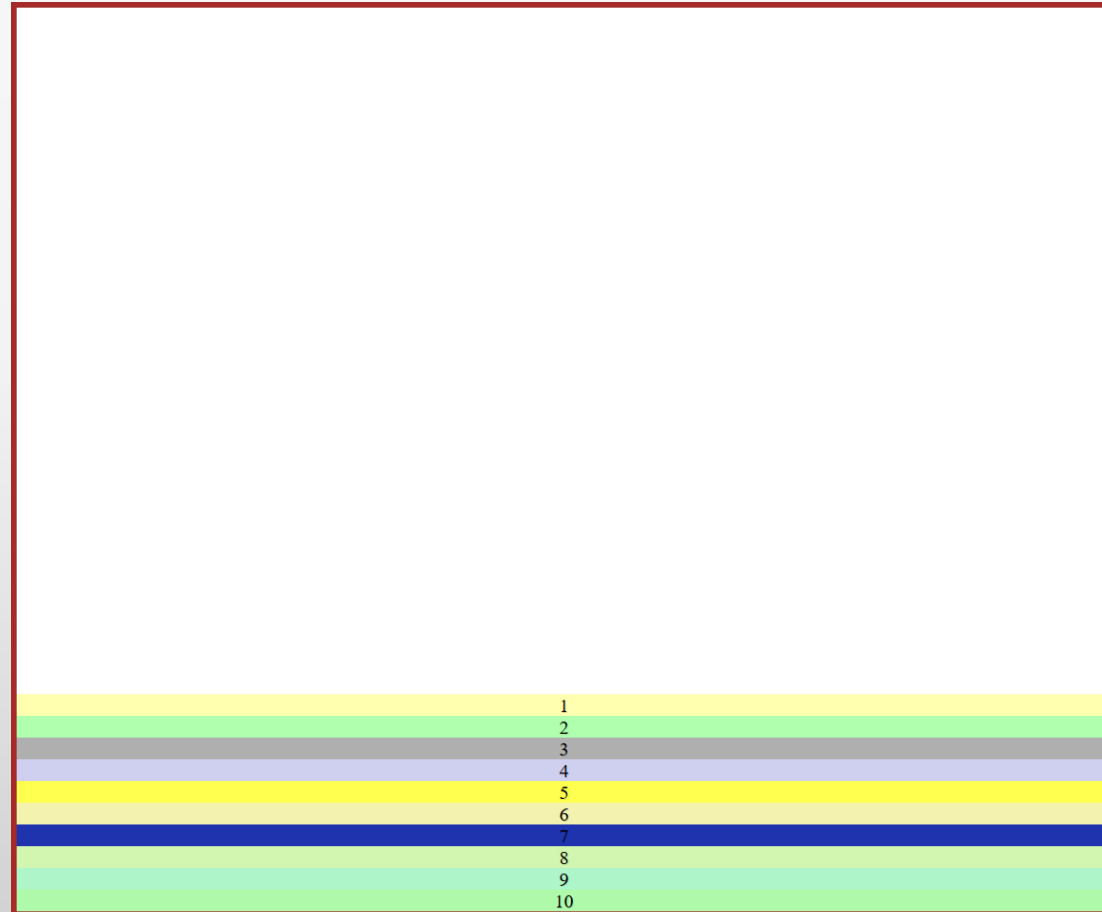


justify-content: 'center'

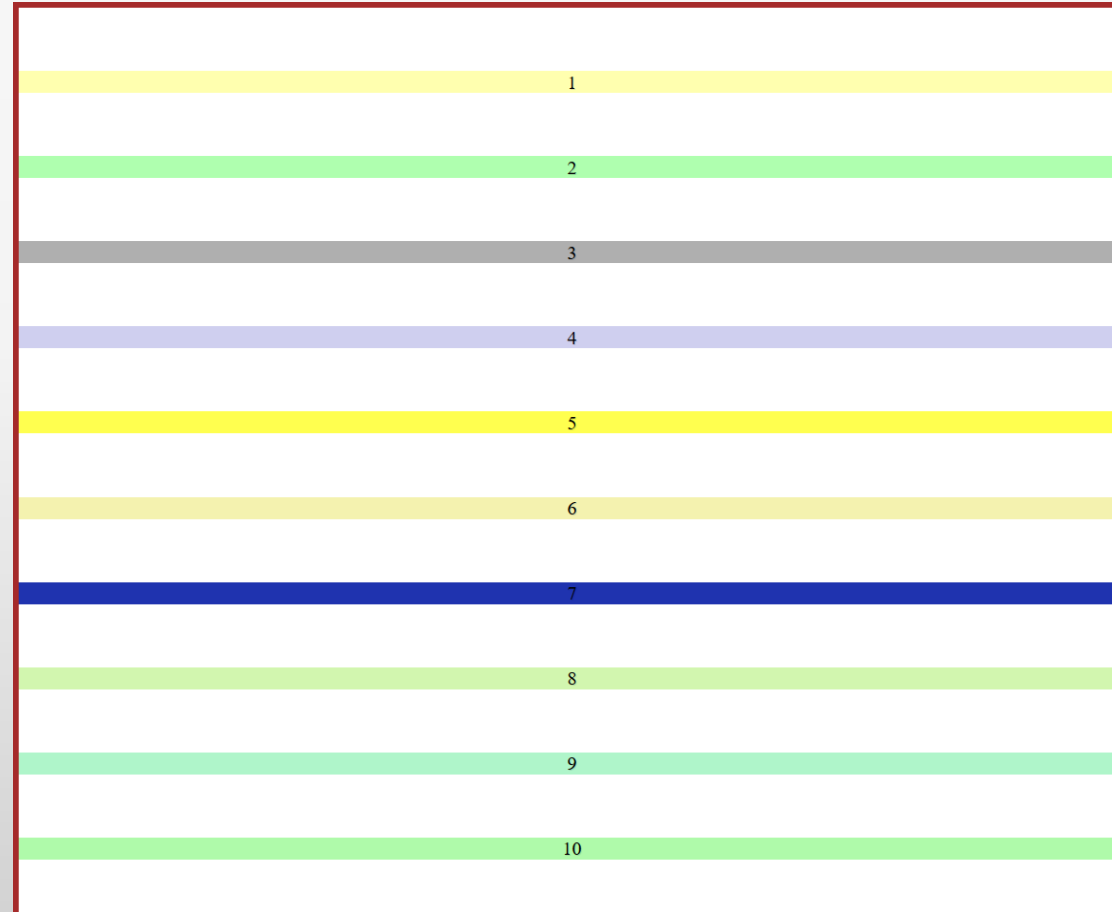
```
.container {  
  display: flex;  
  flex-direction: column;  
  height: 100vh;  
  justify-content: center;  
  border: 6px solid brown;  
}
```



`justify-content: 'flex-end'`



justify-content: 'space-evenly'



Repasando...

- Todo lo que vimos fueron propiedades de **contenedores**
(aquellos que tienen “display: flex”)
 - Consideraciones con los “axis” (ejes)
 - Alineación por axis

Items de Flex

- Son los “hijos inmediatos” de un “display: flex”

```
... <div class="container">
  ... <div class="box box1">1</div>
  ... <div class="box box2">2</div>
  ... <div class="box box3">3</div>
  ... <div class="box box4">4</div>
  ... <div class="box box5">5</div>
  ... <div class="box box6">6</div>
  ... <div class="box box7">7</div>
  ... <div class="box box8">8</div>
  ... <div class="box box9">9</div>
  ... <div class="box box10">10</div>
... </div>
```

Volvamos al punto de partida...

```
.container {  
  display: flex;  
  /* Da igual ponerlo o no */  
  /* flex-direction: row; */  
  border: 6px solid brown;  
}
```

```
.box {  
  text-shadow: 1px 1px 0px rgba(0, 0, 0, 0.1);  
  font-size: 100px;  
  text-align: center;  
}
```


Flex

- Distribuye el espacio libre de manera “inteligente” (sin hacer cálculos)
 - Agranda elementos, distribuye “el espacio” sin usar, y lo reparte de forma equitativa
 - No usamos “width” o “height” para distribuir elementos

Antes de eso, existía... float

Texto a la izquierda

Texto a la derecha

```
<div>
  <div style="float: left; background-color: red;">
    <span style="font-size: 24px;">Texto a la izquierda</span>
  </div>
  <div style="float: right; background-color: blue;">
    <span style="font-size: 24px;">Texto a la derecha</span>
  </div>
</div>
```

Antes de eso, existía... ???

flex

- Flex usa “espacios **sin unidades**” (ni px, ni cm, ¡ni nada!)
- La propiedad “flex” es una abreviación de:
 - `flex-grow`: En cuántos “espacios” la caja se distribuirá
 - Si todas las cajas tienen `flex-grow: 1`; el espacio se distribuye equitativamente
 - `flex-shrink`: En cuántos “espacios” la caja cederá, en caso de no haber más espacio
 - `flex-basis`: Define el espacio por defecto, antes de distribuir el espacio vacío

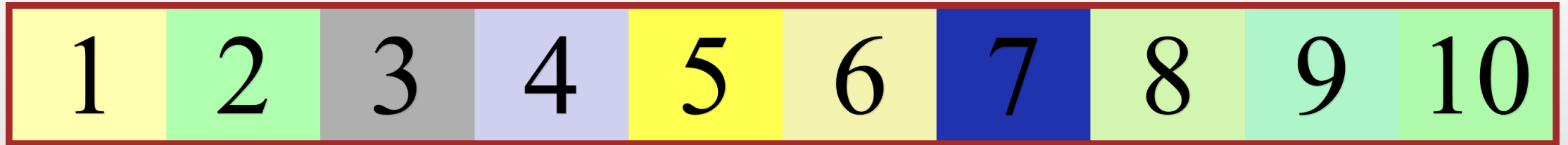
flex

- Podemos especificar uno por uno
- O mejor:
 - `flex: <flex-grow> <flex-shrink> <flex-basis>`
 - Por defecto: `flex: 0 1 auto;`

flex: 1;

```
.box {  
  text-shadow: 1px 1px 0 □ rgba(0, 0, 0, 0.1);  
  font-size: 100px;  
  text-align: center;  
  flex: 1;  
}
```

```
flex: 1;
```



flex: 1;

- ¿Y qué pasa si queremos que un elemento use más espacio que los demás?
- Solo le ponemos a **ese elemento** un flex mayor

```
.box {  
  text-shadow: 1px 1px 0 rgba(0, 0, 0, 0.1);  
  font-size: 100px;  
  text-align: center;  
  flex: 1;  
}  
  
.box1 {  
  flex: 3;  
  background-color: #ffffaf;  
}
```



```
flex: 3;
```

