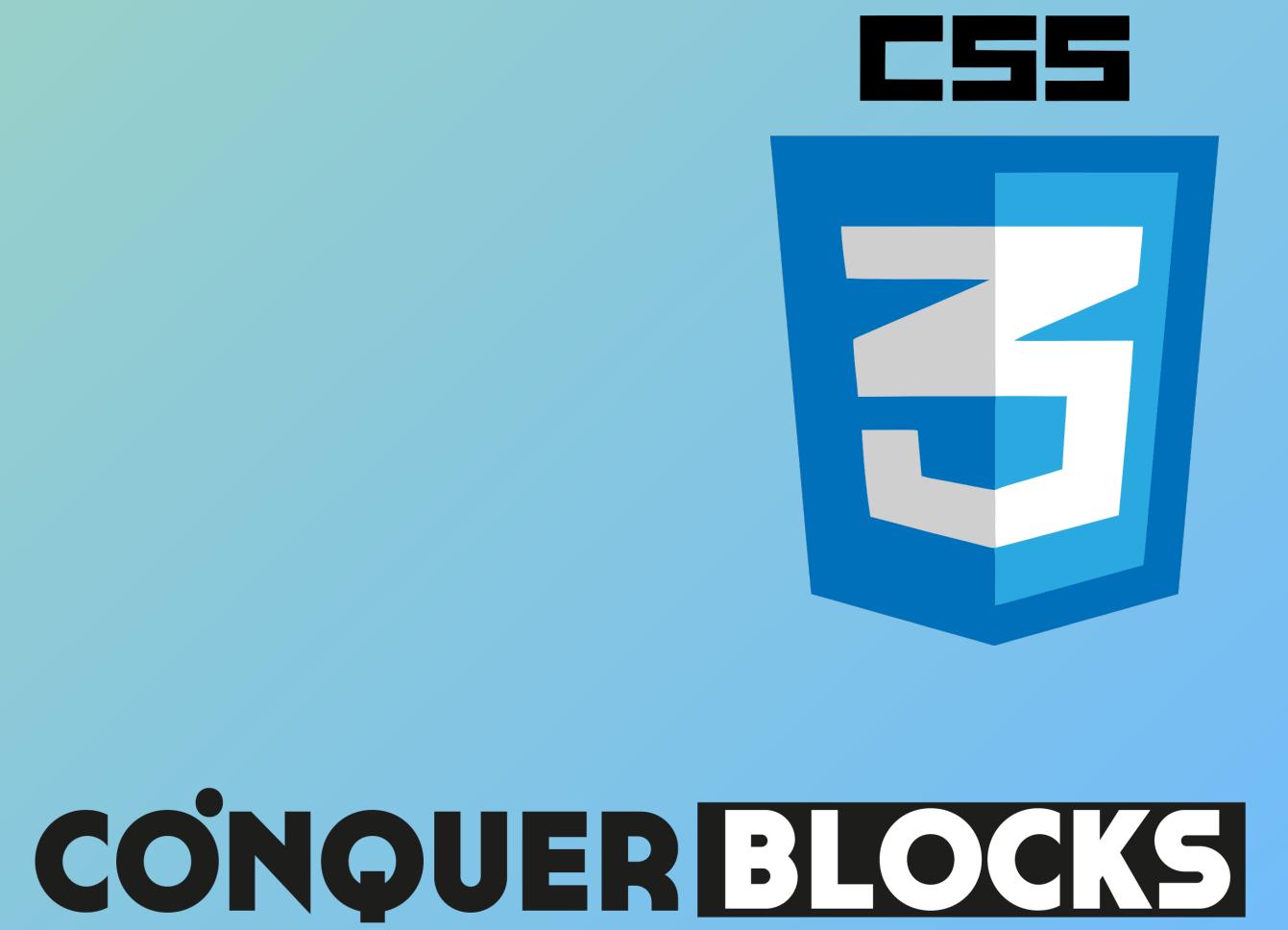


# {css}

Clase 19



CONQUER BLOCKS

# Clase 19

<índice>

## Clase 20: Flexbox

Flex direction

---

Align items

---

Justify content

---

Flex wrap

---

Gap

---

Align self

---

Order y Flexibilidad

---

# **inline vs block vs inline-block**

# Flexbox

La propiedad `display` de CSS permite modificar el comportamiento de un elemento HTML, cambiándolo al que le indiquemos, como por ejemplo `inline` o `block` (u otro de los que veremos más adelante)

# Flexbox

Tipo de caja	Características
block	Se apila en vertical. Ocupa todo el ancho disponible de su etiqueta contenedora.
Versión en línea	
inline	Se coloca en horizontal. Se adapta al ancho de su contenido. Ignora <code>width</code> o <code>height</code> .
inline-block	Combinación de los dos anteriores. Se comporta como <code>inline</code> pero no ignora <code>width</code> o <code>height</code> .

# display

## Block and Inline elements

### BLOCK

Se extienden ocupando todo el espacio disponible para ellos.

### INLINE

Son como palabras en una oración: se separan por medio de un espacio en blanco entre ellos.

# display

## **Block and inline**

El comportamiento de los elementos en bloque y en línea es fundamental para CSS y el hecho de que un documento html marcado correctamente será legible por defecto

# display

## Flujo normal

Este diseño se conoce como “Diseño de bloque y en línea” o Flujo normal” porque esta es la forma en la que los elementos se presentan si no les hacemos nada más.

# display

## Display con dos valores inner-outer

Definen dos valores: externo de visualización e  
interno de visualización.

# display

Un elemento que tengan los atributos de bloque podremos establecerle: ancho, alto, padding, margin etc...

# Flexbox

# Flexbox

En CSS, inicialmente se utilizaba el posicionamiento (static, relative, absolute...), los elementos en línea o en bloque (y derivados) o la propiedad float para realizar maquetaciones

# Flexbox

Flex (también llamado 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 de una página web.

# Flexbox

Para empezar a utilizar flex lo primero que debemos hacer es conocer algunos de los elementos básicos de este nuevo esquema, que son los siguientes

# Flexbox



# Flexbox

- **Contenedor:** Es el elemento padre que tendrá en su interior cada uno de los ítems flexibles. Observa que al contrario que muchas otras estructuras CSS, por norma general, en Flex establecemos las propiedades al elemento padre.
- **Ítem:** Cada uno de los hijos que tendrá el contenedor en su interior.

# Flexbox

- **Eje principal:** Los contenedores flexibles tendrán una orientación principal específica. Por defecto, el eje principal del contenedor flex es en horizontal (en 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).

# Flexbox

```
<div class="container"> ← Flex container →  
  <div class="item item-1">1</div> ← Flex items →  
  <div class="item item-2">2</div>  
  <div class="item item-3">3</div>  
</div>
```

# Flexbox

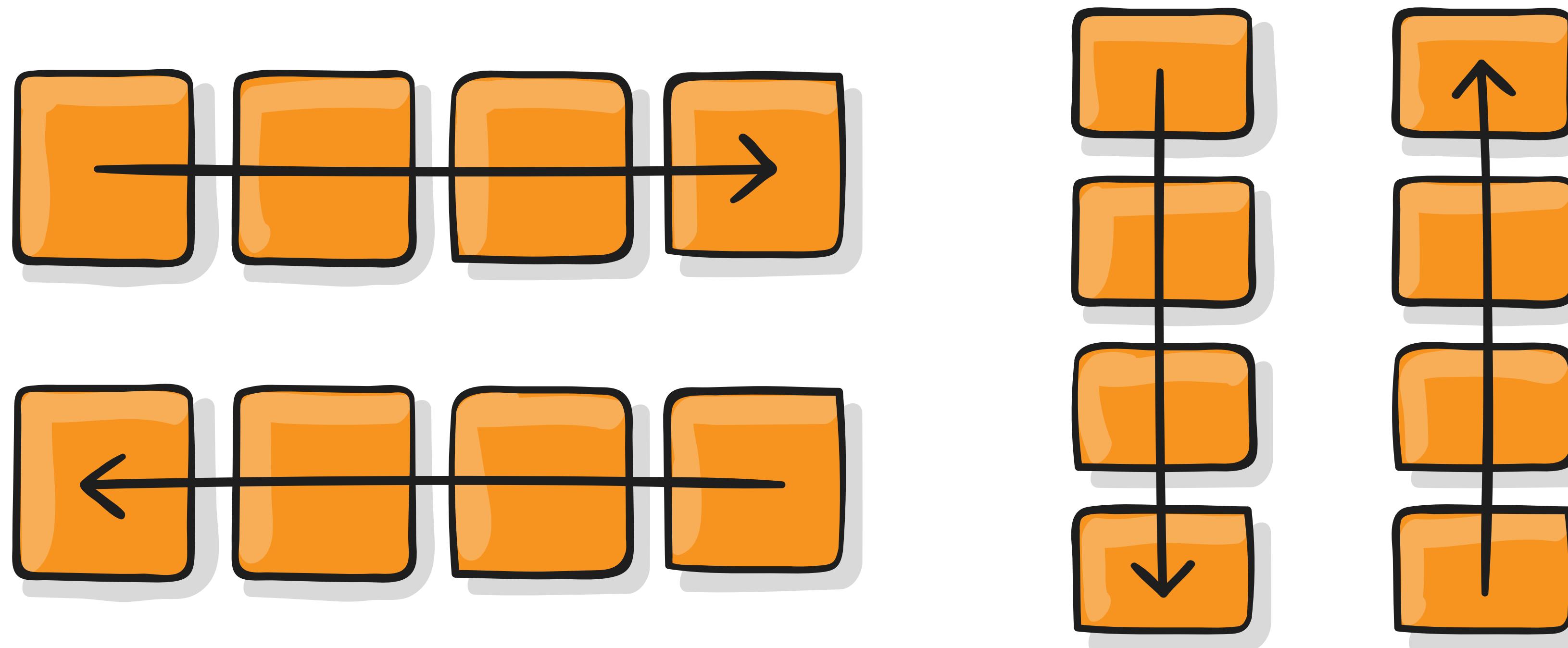
Ahora vamos a ver qué podemos hacer con esos items y controlar cómo se forman

# Dirección de los ejes flex-direction

# flex-direction

Existen dos propiedades principales para manipular la dirección y comportamiento de los ítems a lo largo del eje principal del contenedor.

# Posicionamiento absoluto

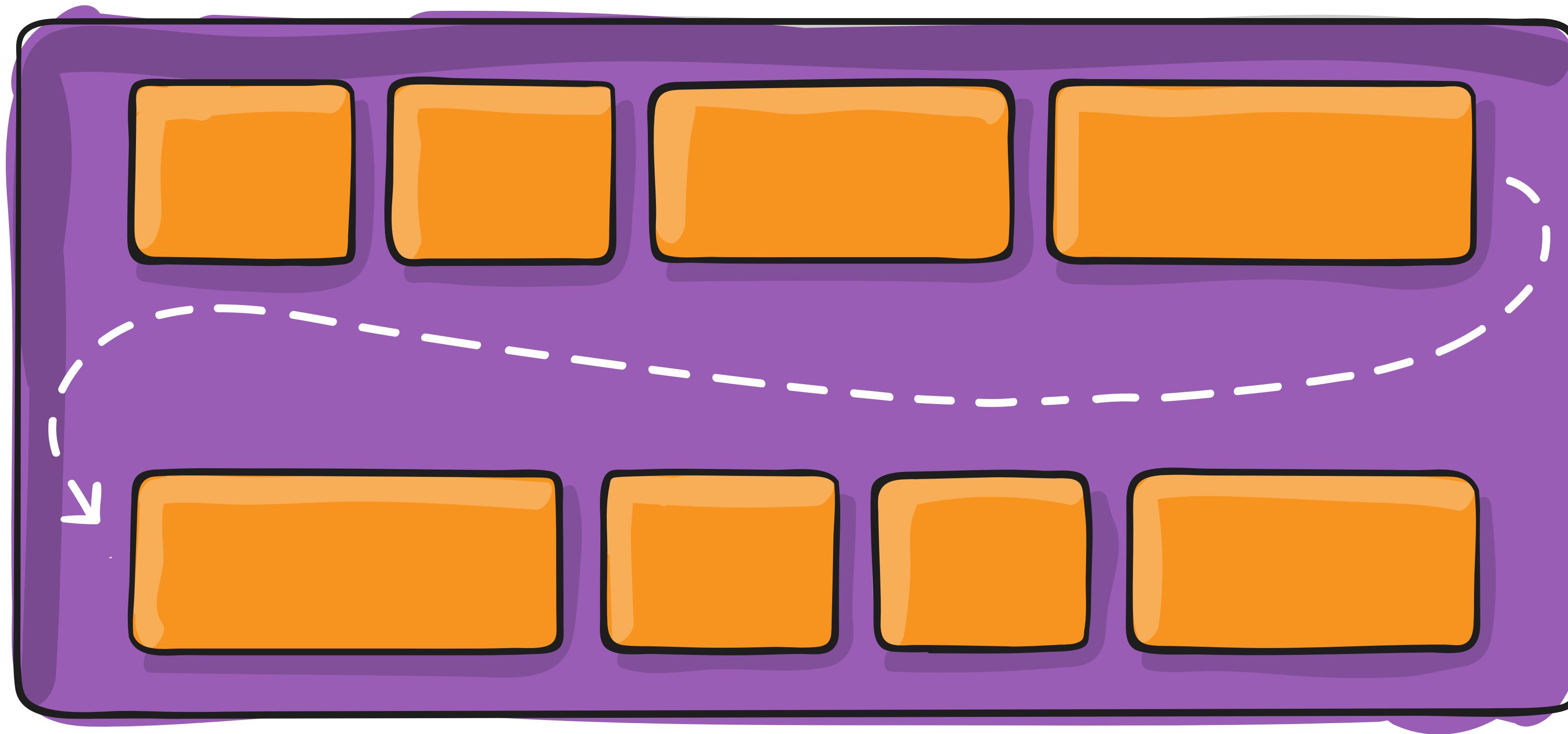


# Contenedor multilínea flex-wrap

# flex-wrap

¿Qué pasa si el contenido ocupa más del ancho?

# flex-wrap



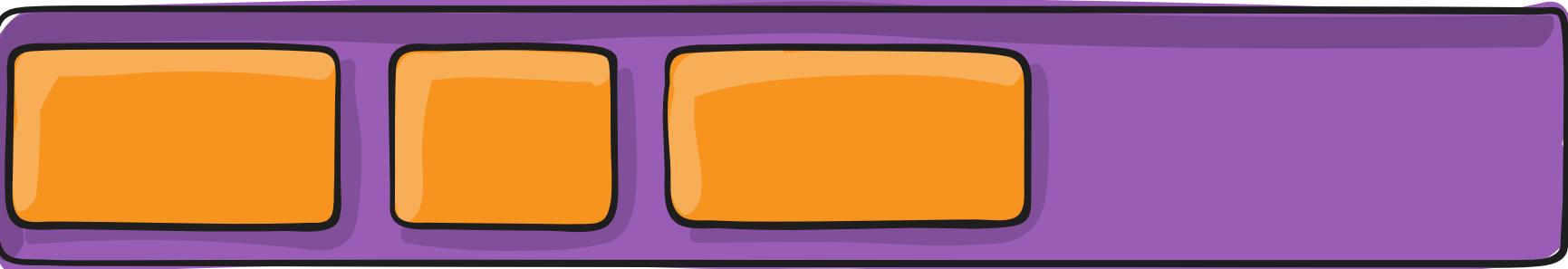
# Huecos gap

# Gap

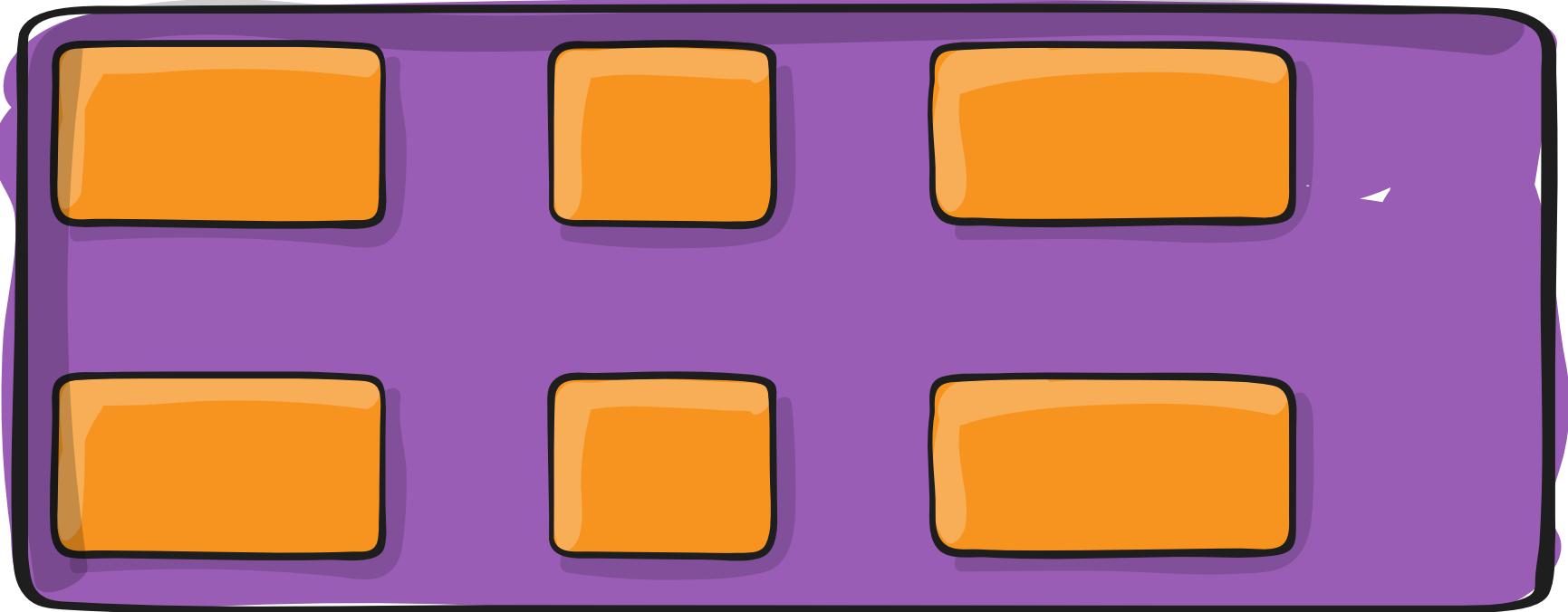
Separación entre los items

# Gap

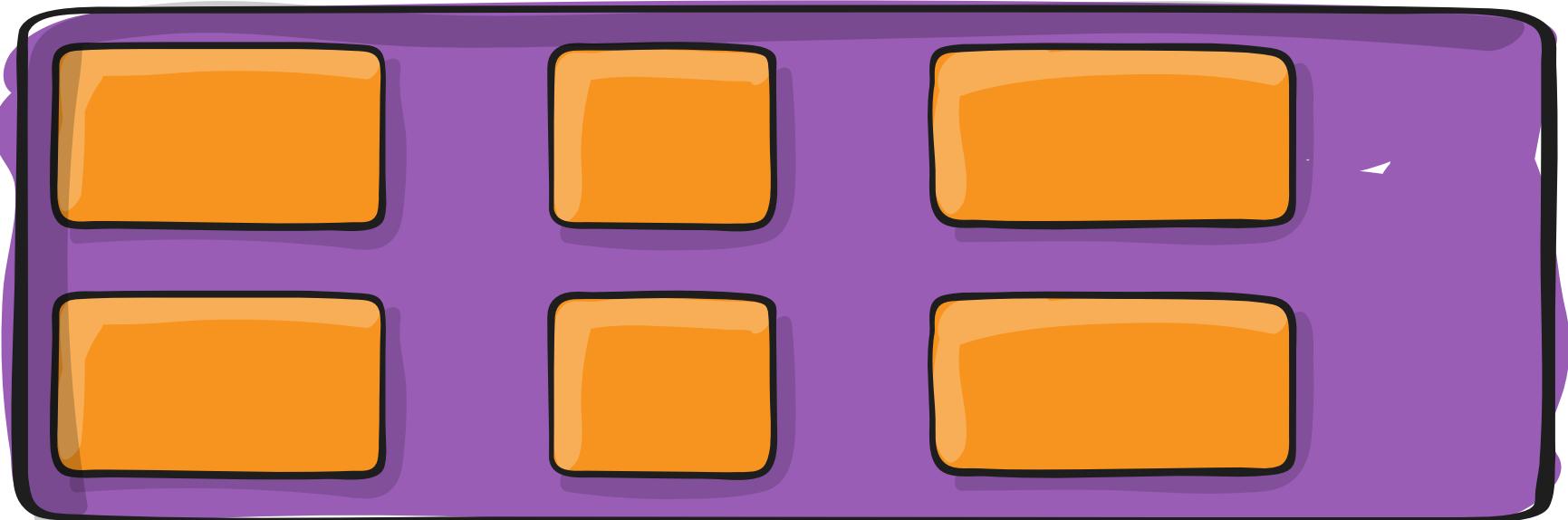
gap: 10px



gap: 30px



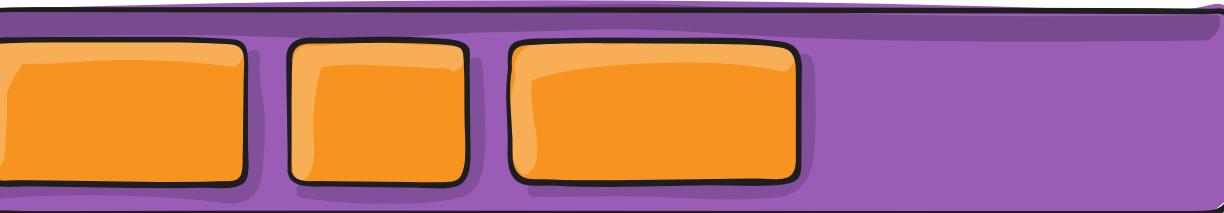
gap: 10px 30px



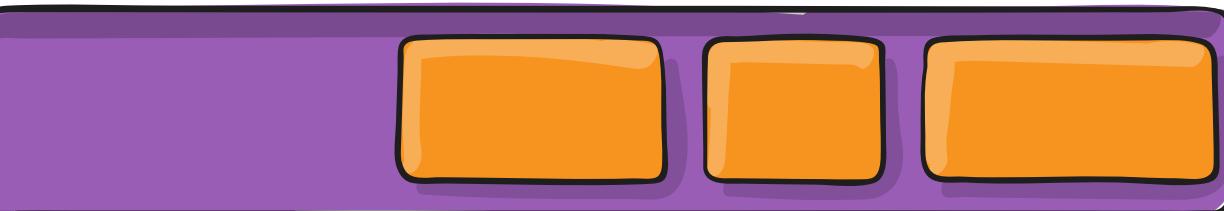
Alineación eje-x  
justify-content

# justify-content

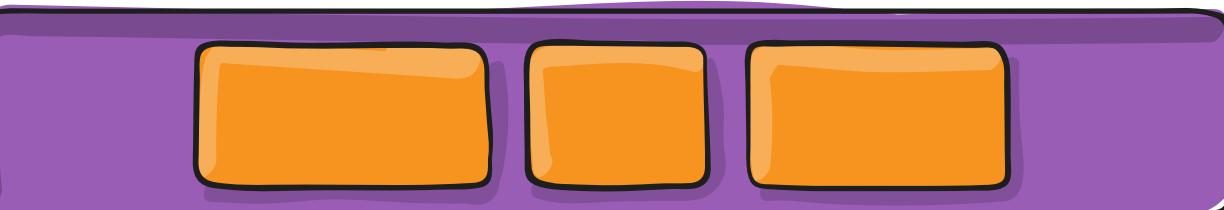
flex-start



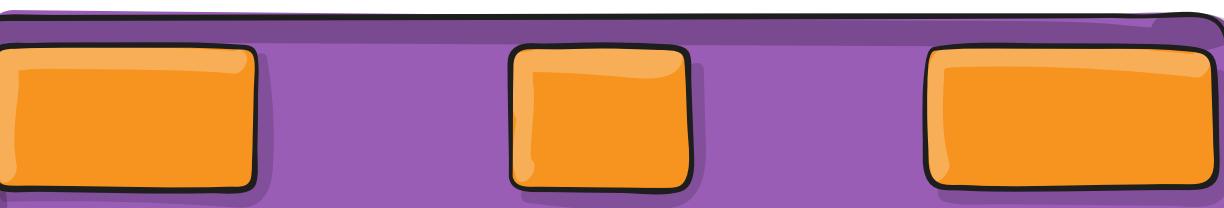
flex-end



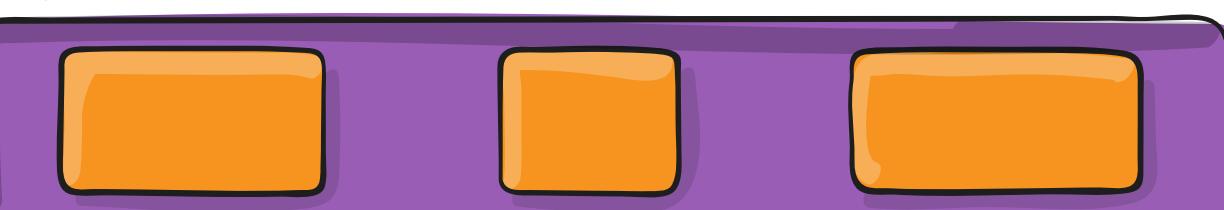
center



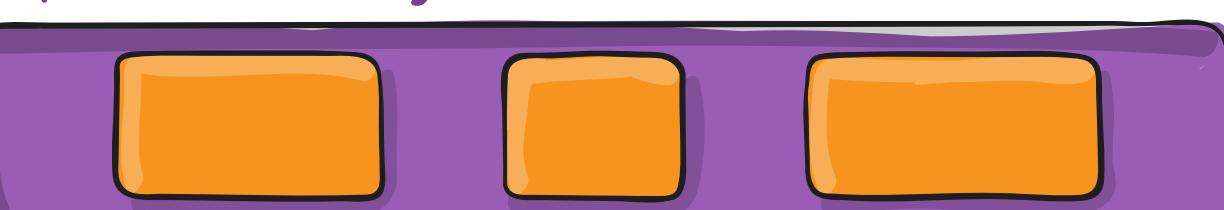
space-between



space-around



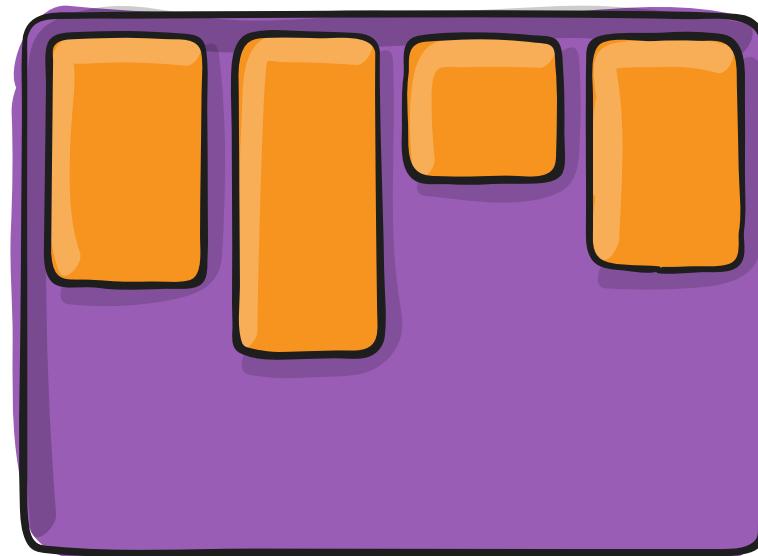
space-evenly



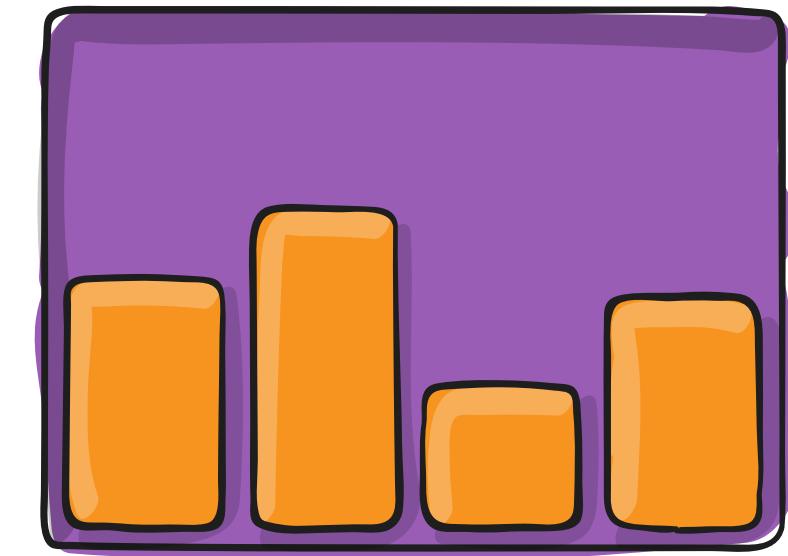
# Alineación eje-y align-items

# align-items

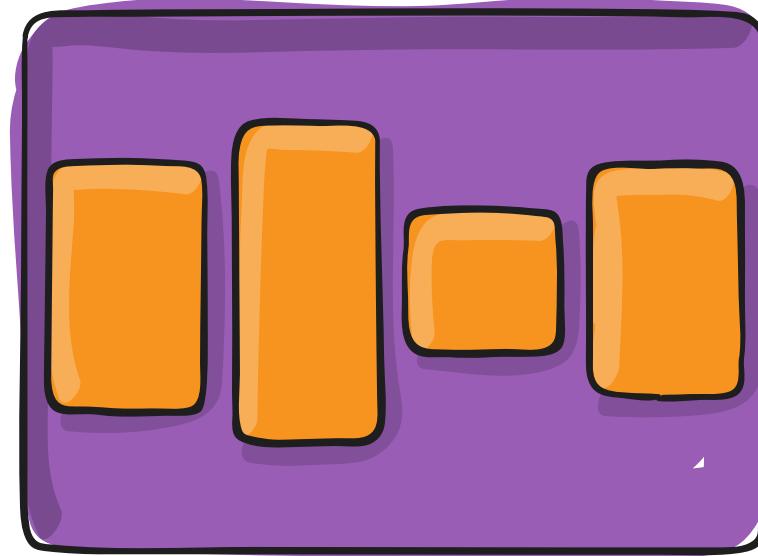
flex-start



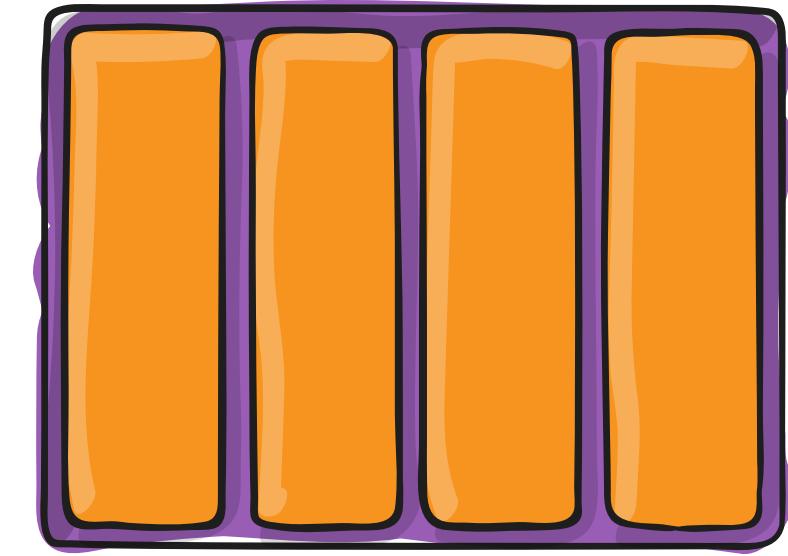
flex-end



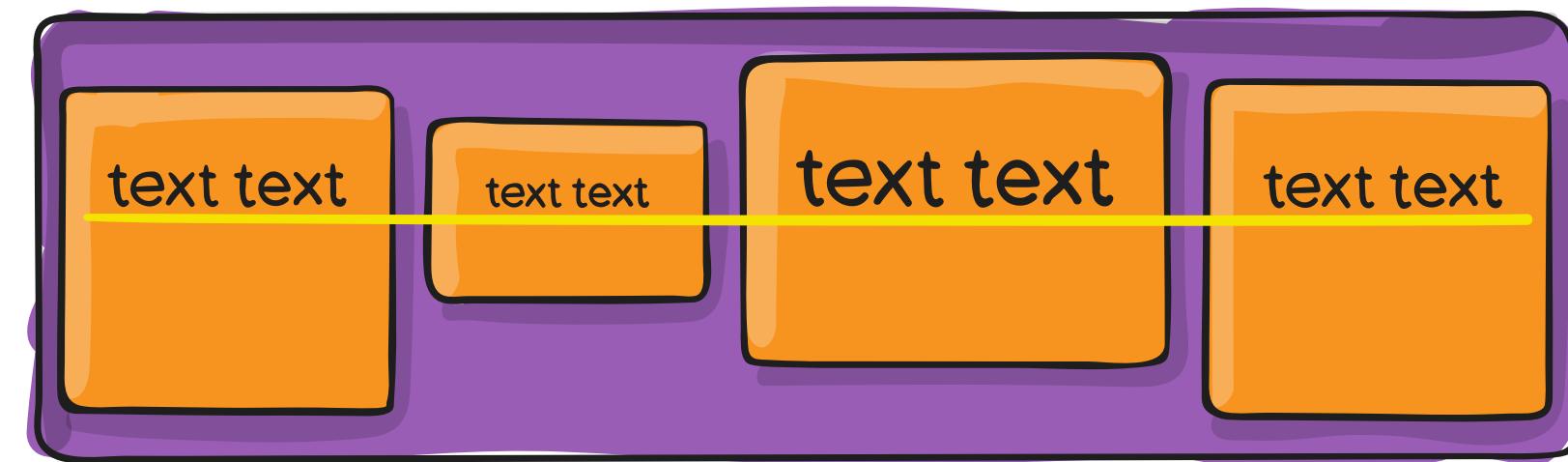
center



stretch

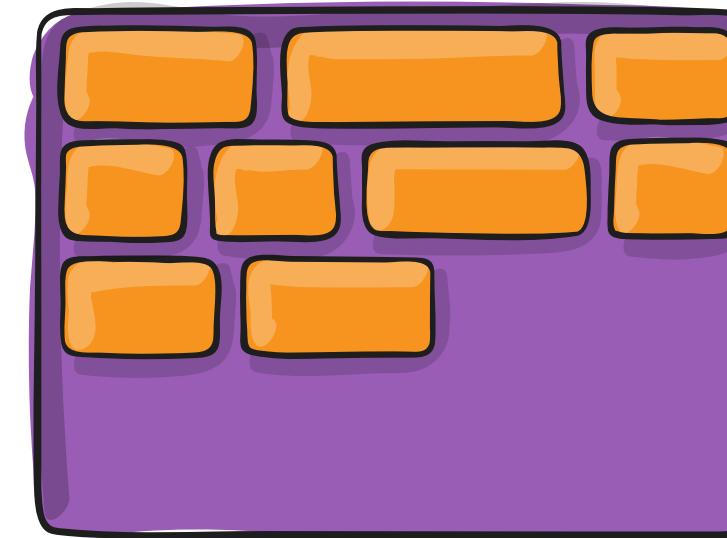


baseline

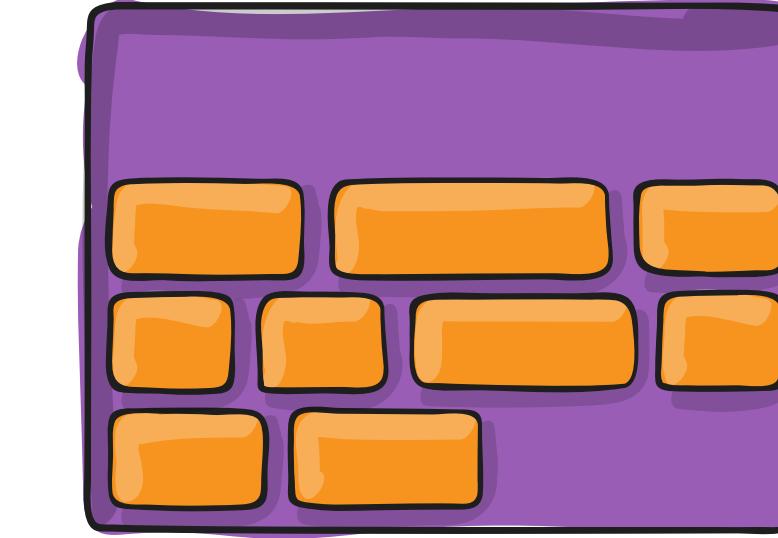


# Alineación contenedor multilinea align-content

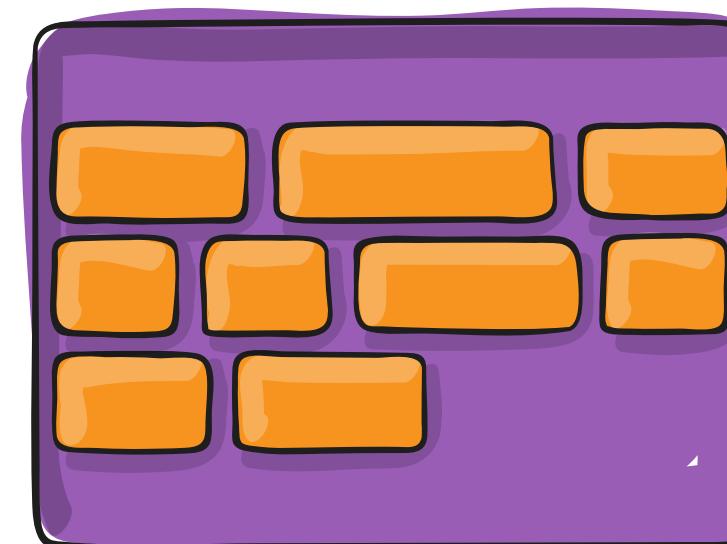
**flex-start**



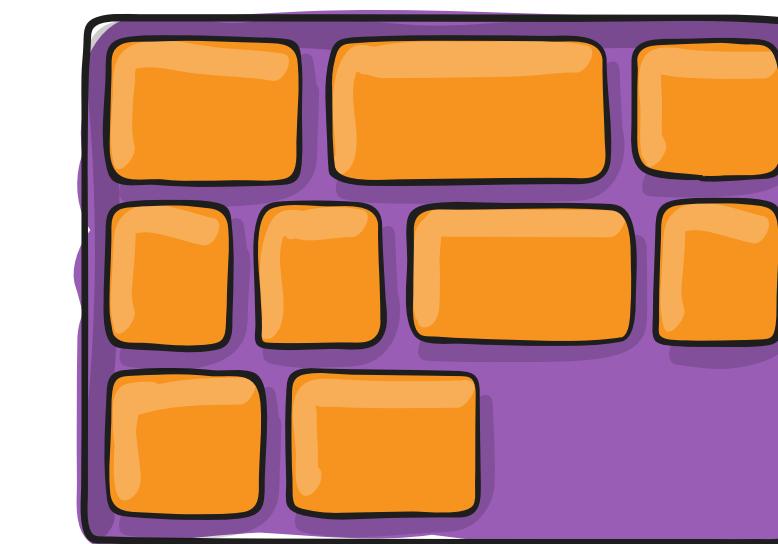
**flex-end**



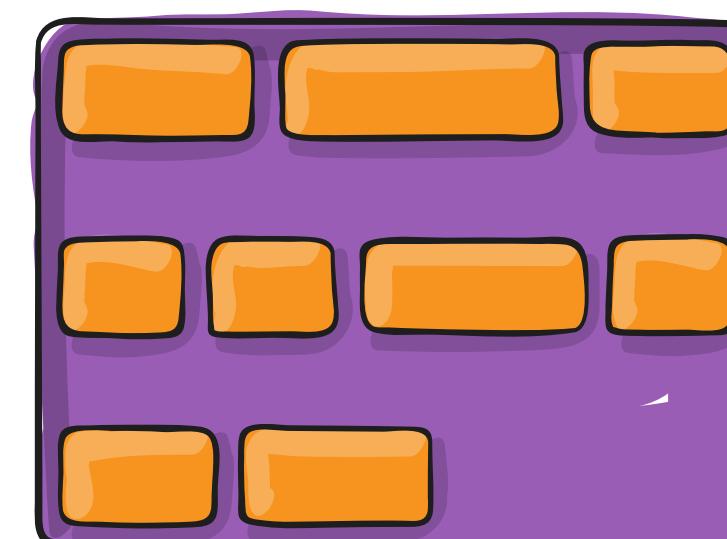
**center**



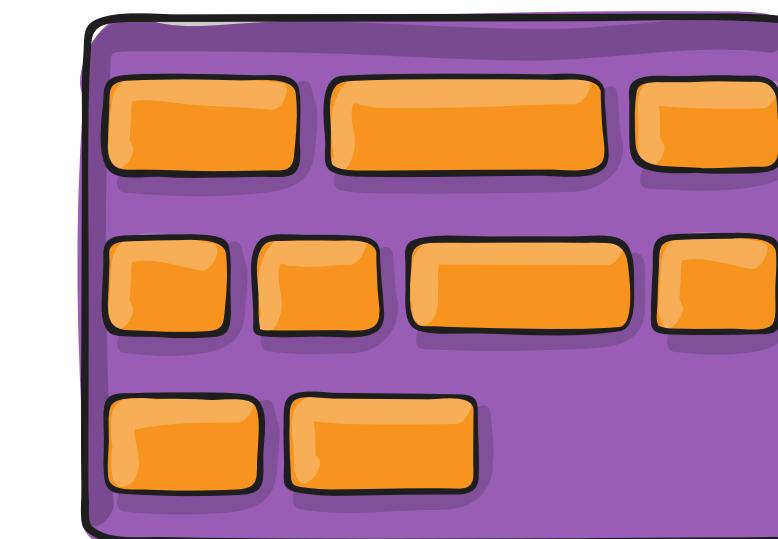
**stretch**



**space-between**



**space-around**



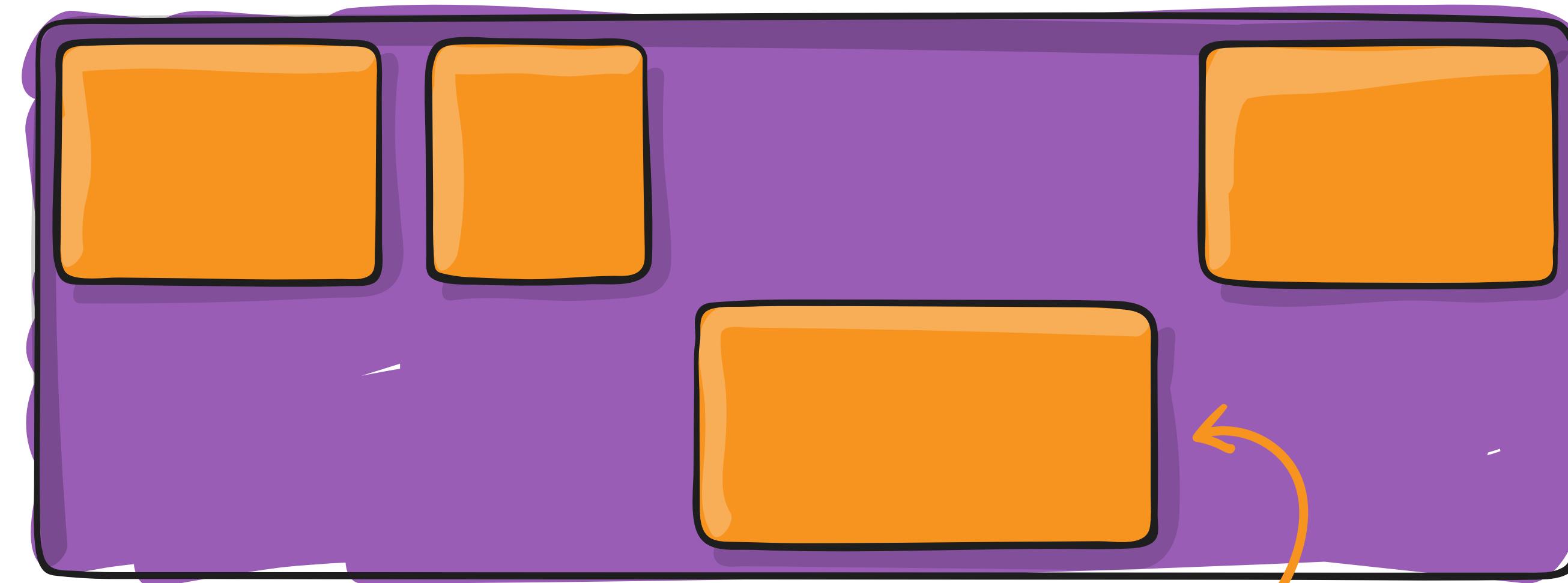
# Alineación específica align-self

# align-self

Esta propiedad es de los hijos  
no del contenedor

# align-self

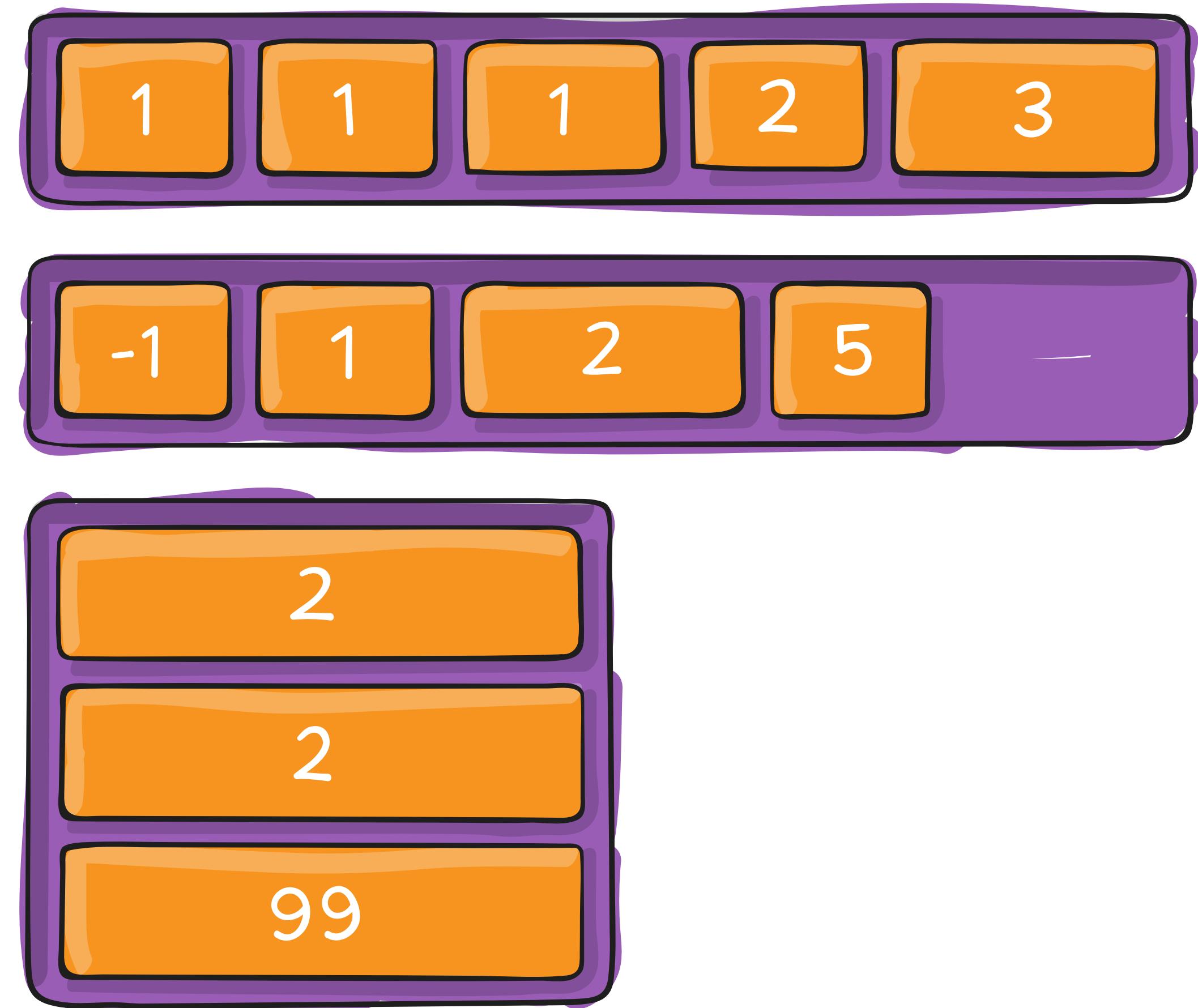
flex-start



flex-end

# Orden de elementos order

Esta propiedad es de los hijos  
no del contenedor



# **Flexibilidad**

## **flex-basis**

Esta propiedad es de los hijos  
no del contenedor

En primer lugar, tenemos la propiedad flex-basis, que define el tamaño base por defecto que tendrán los ítems antes de aplicarle una cierta distribución de espacio.

También se establece el tamaño antes de  
repartir el espacio disponible

# Flexibilidad flex-grow

Esta propiedad es de los hijos  
no del contenedor

La propiedad flex-grow actúa en situaciones donde:

- Hay un flex-basis definido.
- Los ítems cubren el tamaño total del contenedor flex padre.

En esas situaciones, la propiedad flex-grow indica el factor de crecimiento de los ítems en el caso de que no tengan un ancho o alto específico. Modifiquemos el ejemplo anterior y añadamos el siguiente fragmento de código

Por omisión, todos los elementos tienen un `flex-grow: 0` definido, de modo que no hay factor de crecimiento en el elemento, y tendrá el tamaño definido por la propiedad `flex-basis` que explicamos anteriormente.

Sin embargo, si colocamos un `flex-grow: 1` al primer hijo (ejemplo anterior), este crecerá hasta que la suma de los hijos ocupen el 100% del contenedor, mientras que el resto de hijos tendrá el tamaño base definido por `flex-basis`.

En resumen, flex-grow establece un factor de crecimiento observando en conjunto el resto de elementos, y veremos que actua siempre y cuando la suma del espacio de los elementos hijos no superen el 100% del contenedor padre.

# Flexibilidad flex-shrink

Esta propiedad es de los hijos  
no del contenedor

Por otro lado, tenemos la propiedad **flex-shrink** que es la opuesta a la propiedad **flex-grow**. Mientras que la anterior indica un factor de crecimiento, **flex-shrink** hace justo lo contrario: aplica un factor de decrecimiento.

En esas situaciones, la propiedad flex-grow indica el factor de crecimiento de los ítems en el caso de que no tengan un ancho o alto específico. Modifiquemos el ejemplo anterior y añadamos el siguiente fragmento de código

La propiedad **flex-shrink** actúa en situaciones donde:

- Hay un **flex-basis** definido.
- Los ítems no cubren el tamaño total del contenedor **flex** padre.

Sin embargo, si colocamos un `flex-grow: 1` al primer hijo (ejemplo anterior), este crecerá hasta que la suma de los hijos ocupen el 100% del contenedor, mientras que el resto de hijos tendrá el tamaño base definido por `flex-basis`.

En resumen, flex-grow establece un factor de crecimiento observando en conjunto el resto de elementos, y veremos que actua siempre y cuando la suma del espacio de los elementos hijos no superen el 100% del contenedor padre.

# Apilamiento con z-index

# **z-index**

Cuando modificamos la posición de las cajas,  
puede ocurrir apilamiento entre los diferentes  
elementos

# **z-index**

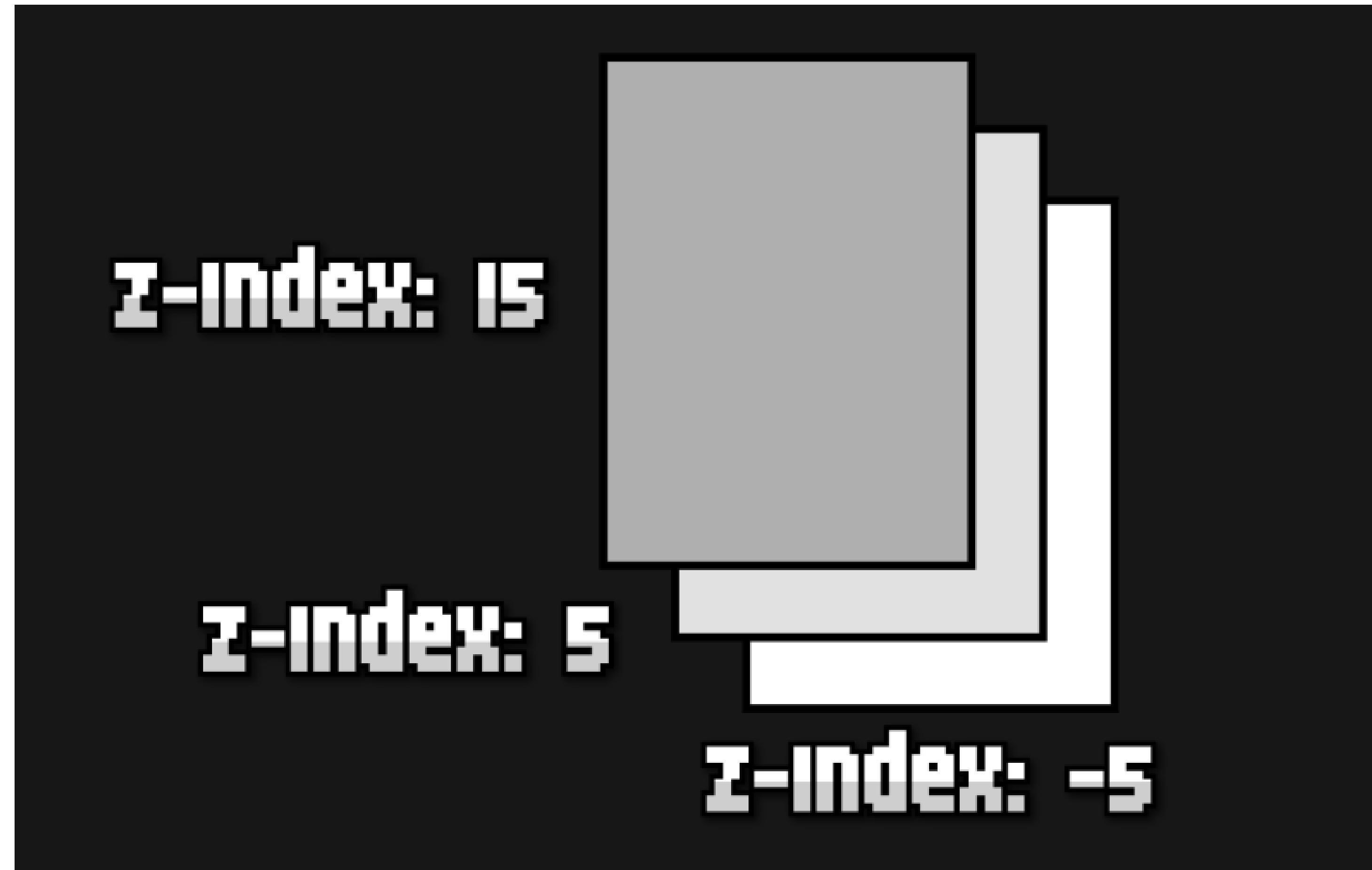
## **z-index**

establece el nivel de profundidad en el que está un elemento sobre los demás. De esta forma, podemos hacer que un elemento se coloque encima o debajo de otro.

# **z-index**

Su funcionamiento es muy sencillo, sólo hay que indicar un número que representará el nivel de profundidad del elemento. Los elementos un número más alto estarán por encima de otros con un número más bajo, que permanecerán ocultos detrás de los primeros.

# **z-index**



# z-index

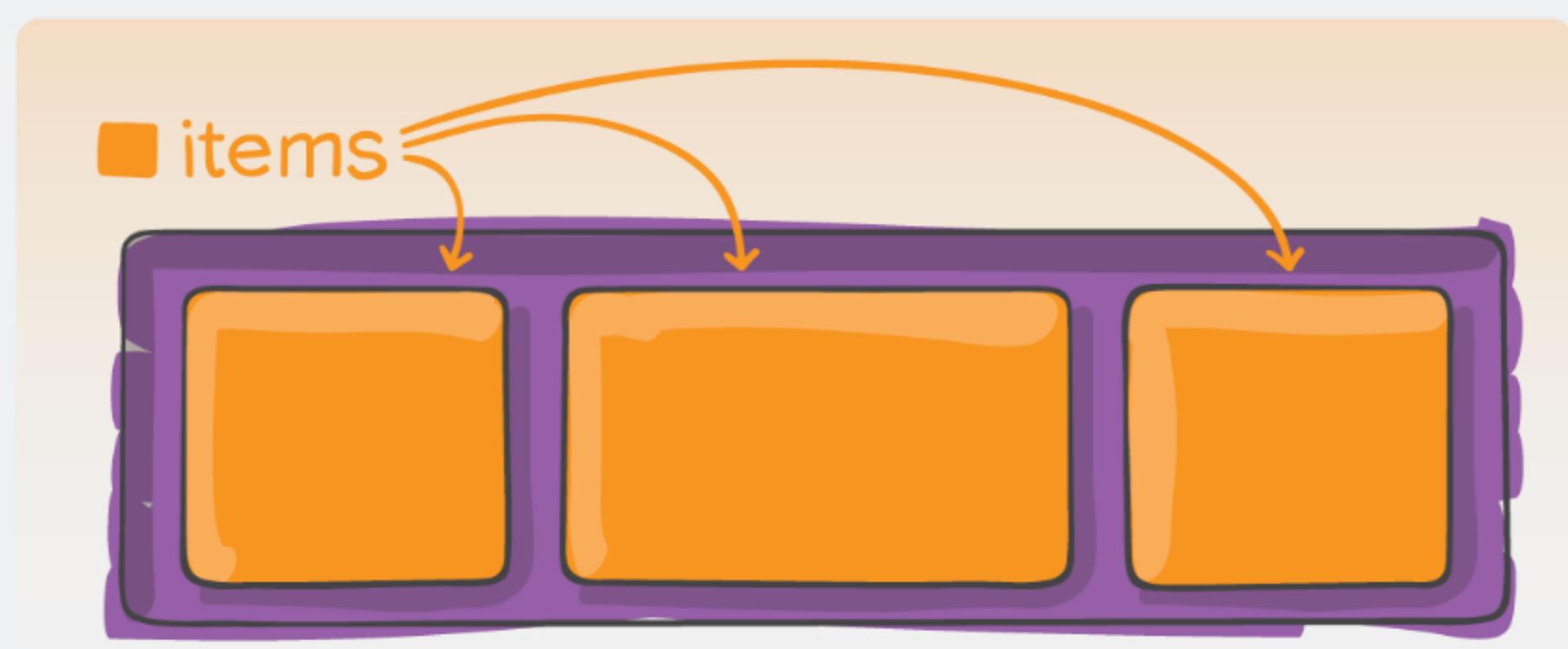
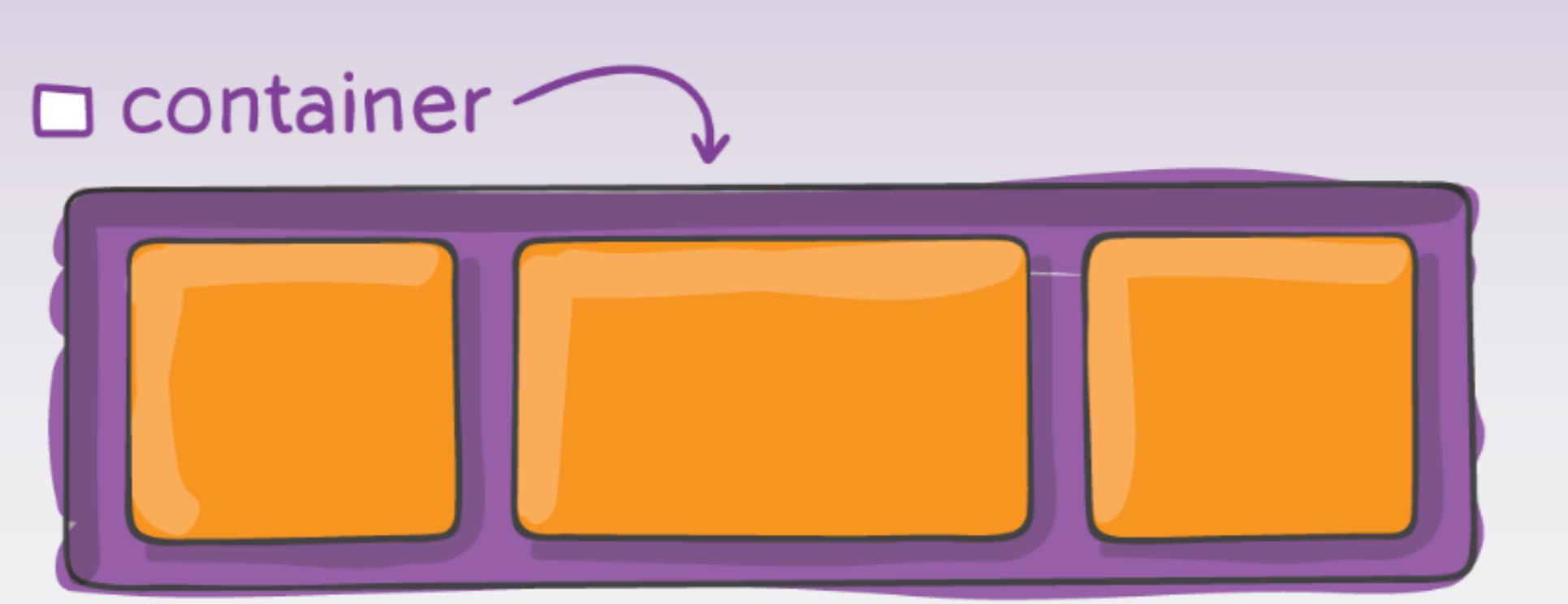
**Nota:** Los niveles z-index, así como las propiedades top, left, bottom y right no funcionan con elementos que estén utilizando posicionamiento estático. **Deben tener un tipo de posicionamiento diferente a estático.**

# z-index

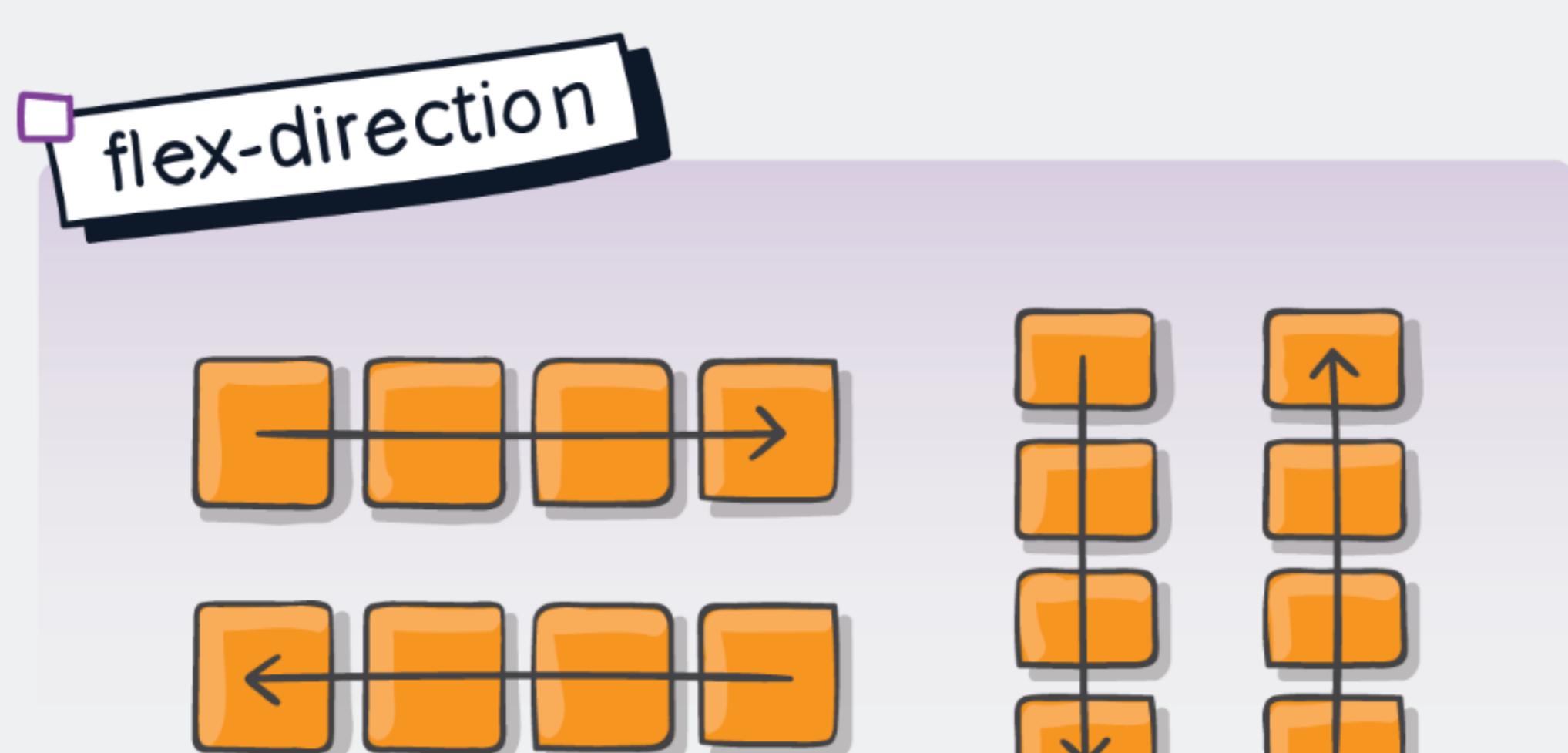
**Nota:** Los niveles z-index, así como las propiedades top, left, bottom y right no funcionan con elementos que estén utilizando posicionamiento estático. **Deben tener un tipo de posicionamiento diferente a estático.**

# CSS Flexbox

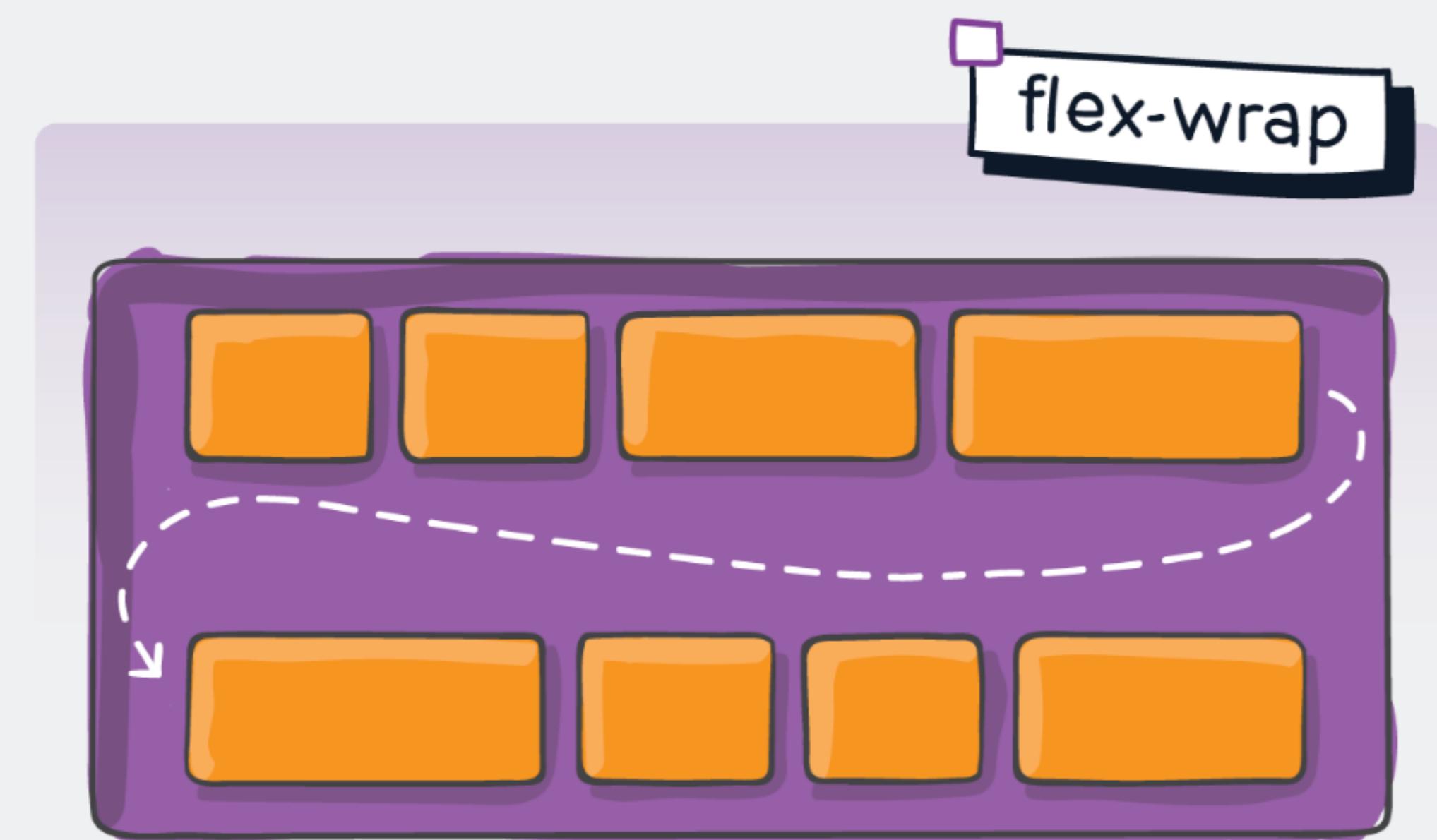
a guide from



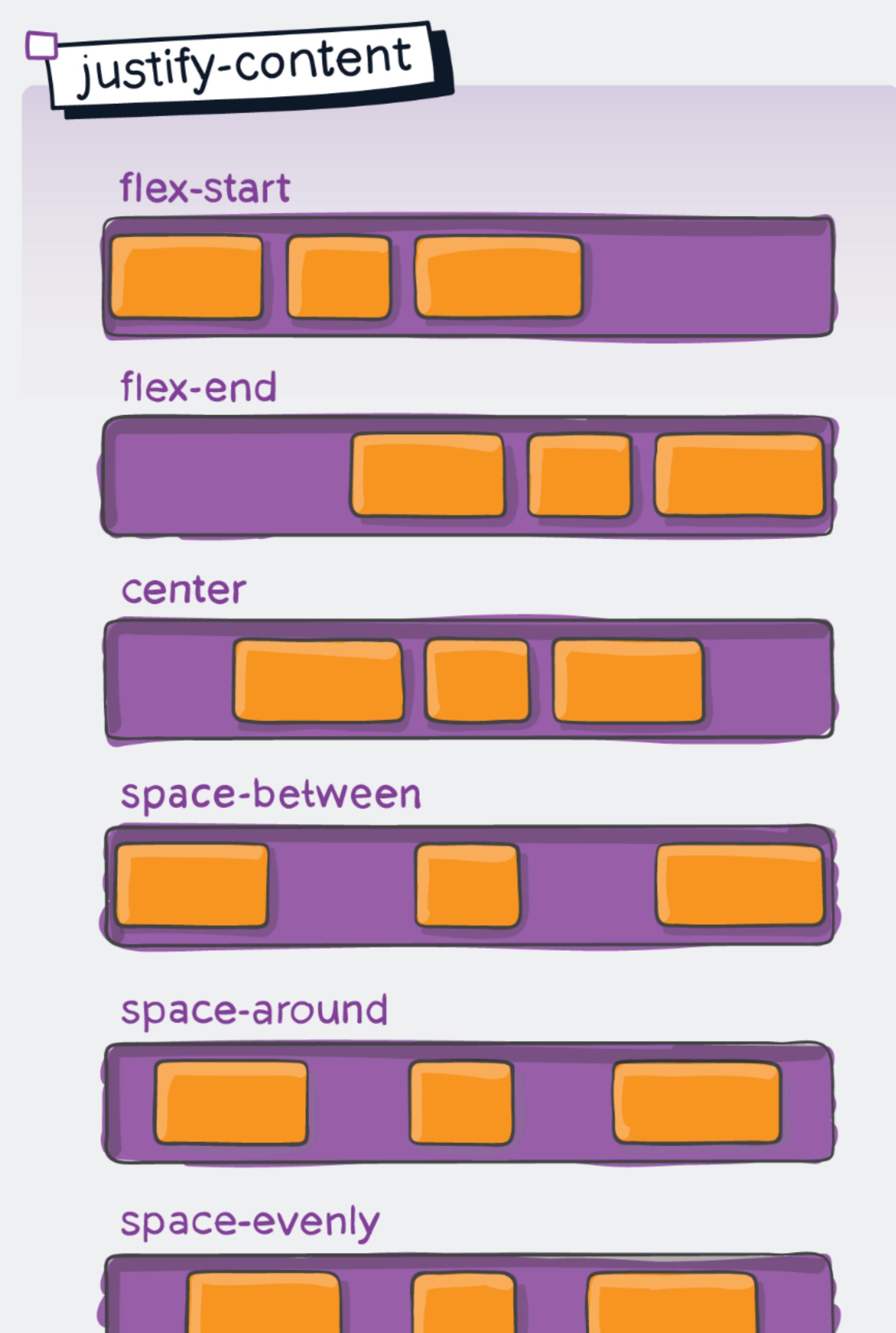
```
.container {
  display: flex; /* or inline-flex */
}
```



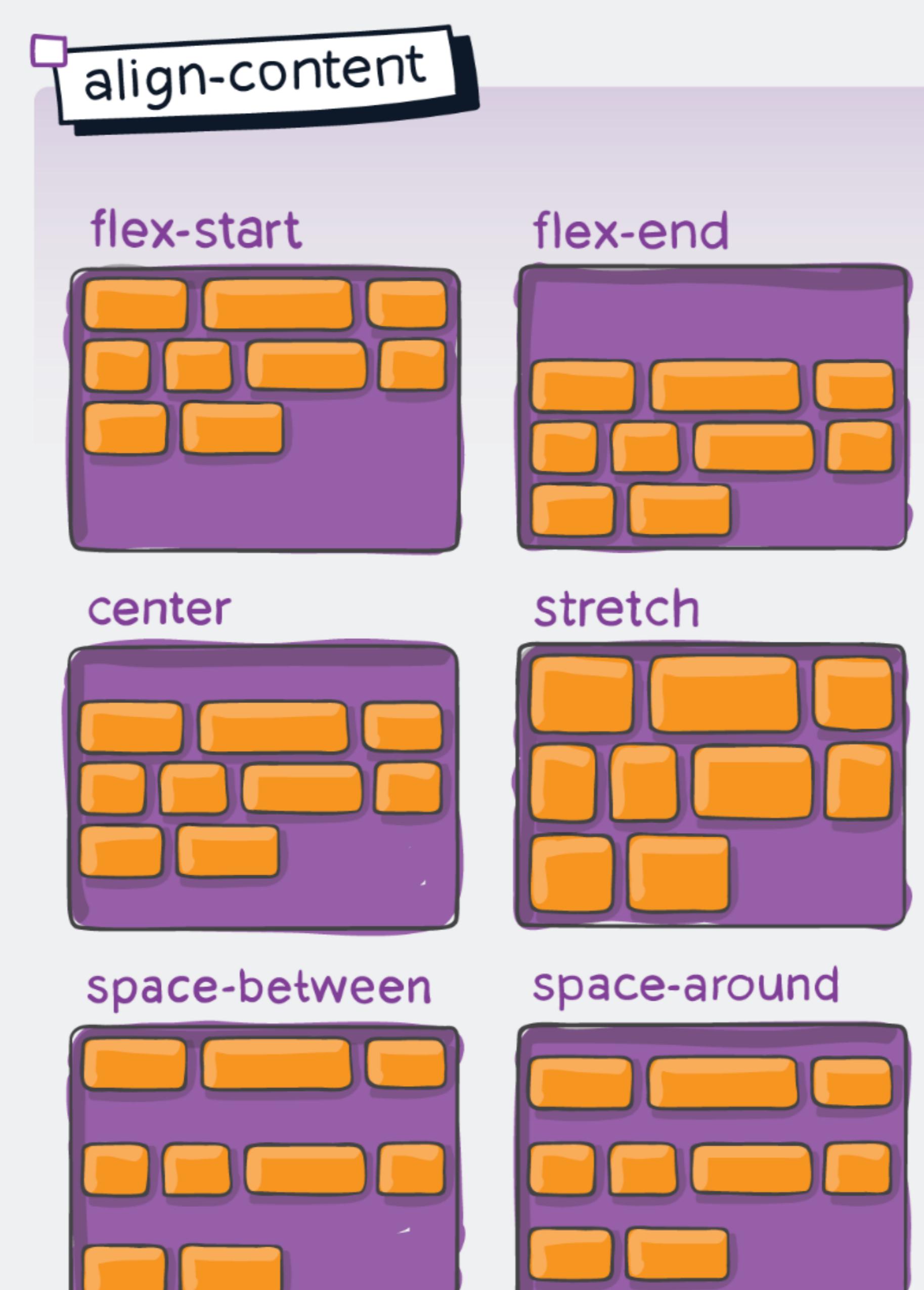
```
.container {
  flex-direction: row | row-reverse |
  column | column-reverse;
}
```



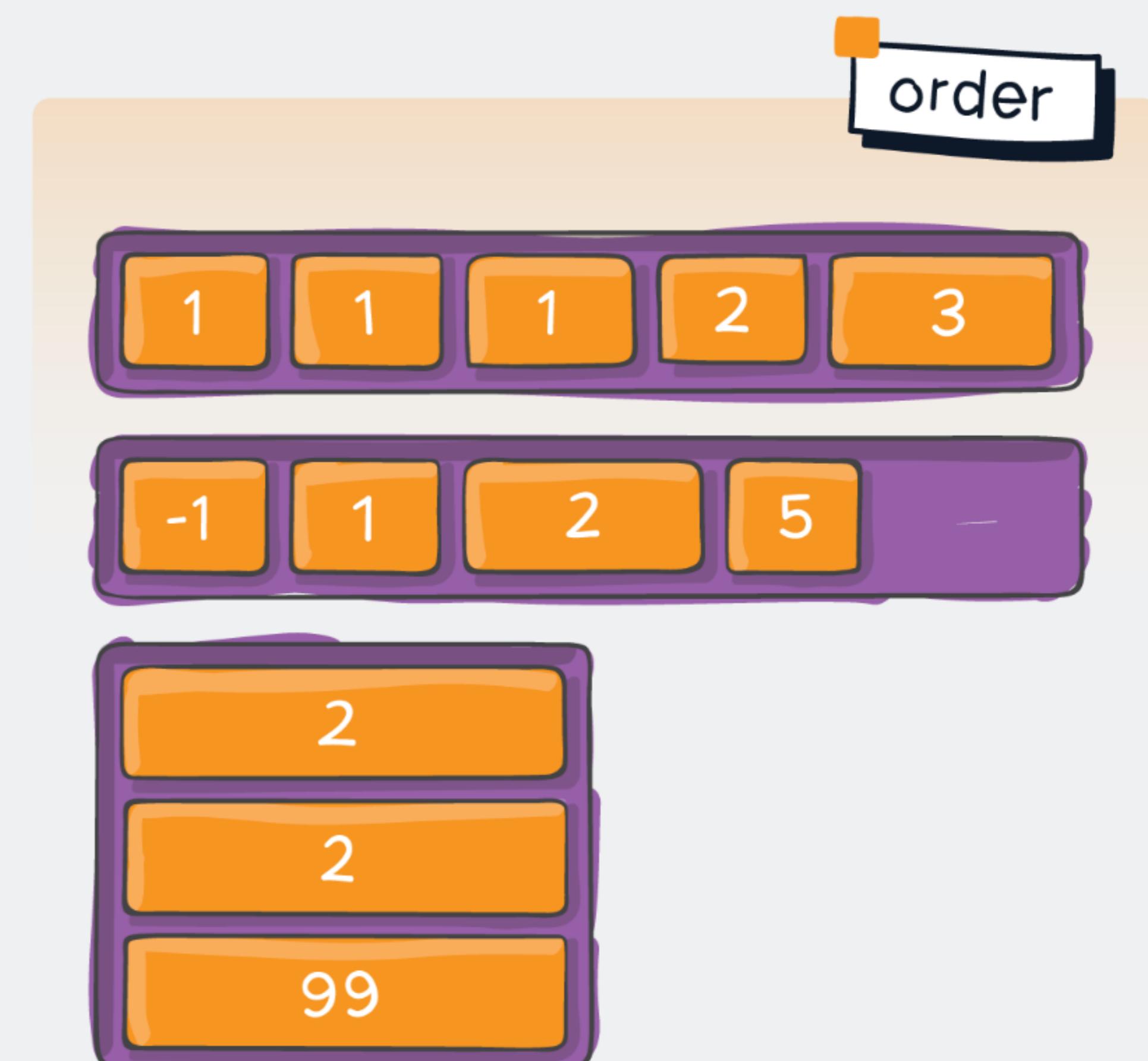
```
.container {
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```



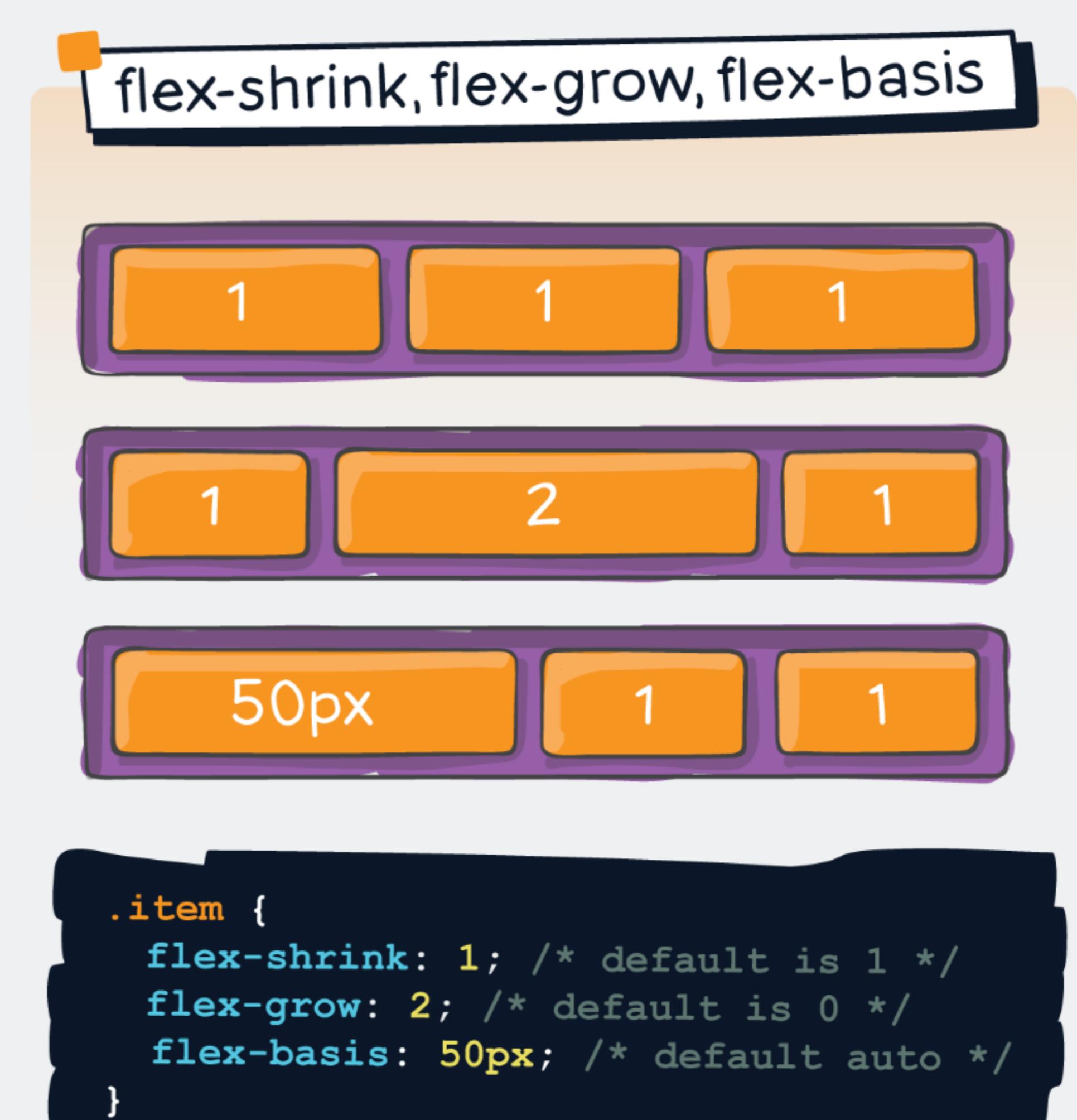
```
.container {
  justify-content: flex-start | flex-end |
  center | space-between | space-around |
  space-evenly;
}
```



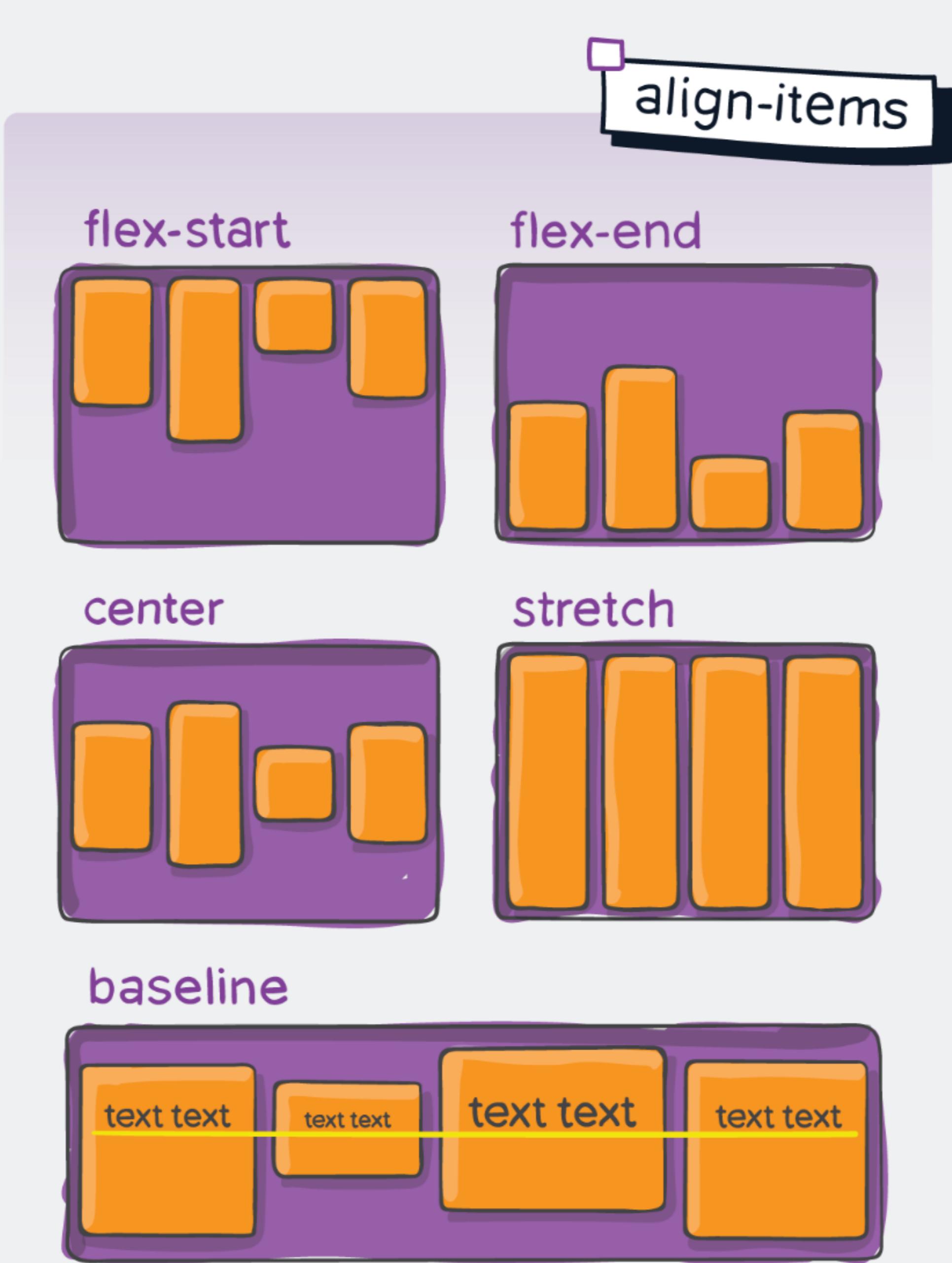
```
.container {
  align-content: flex-start | flex-end |
  center | space-between | space-around |
  space-evenly | stretch;
}
```



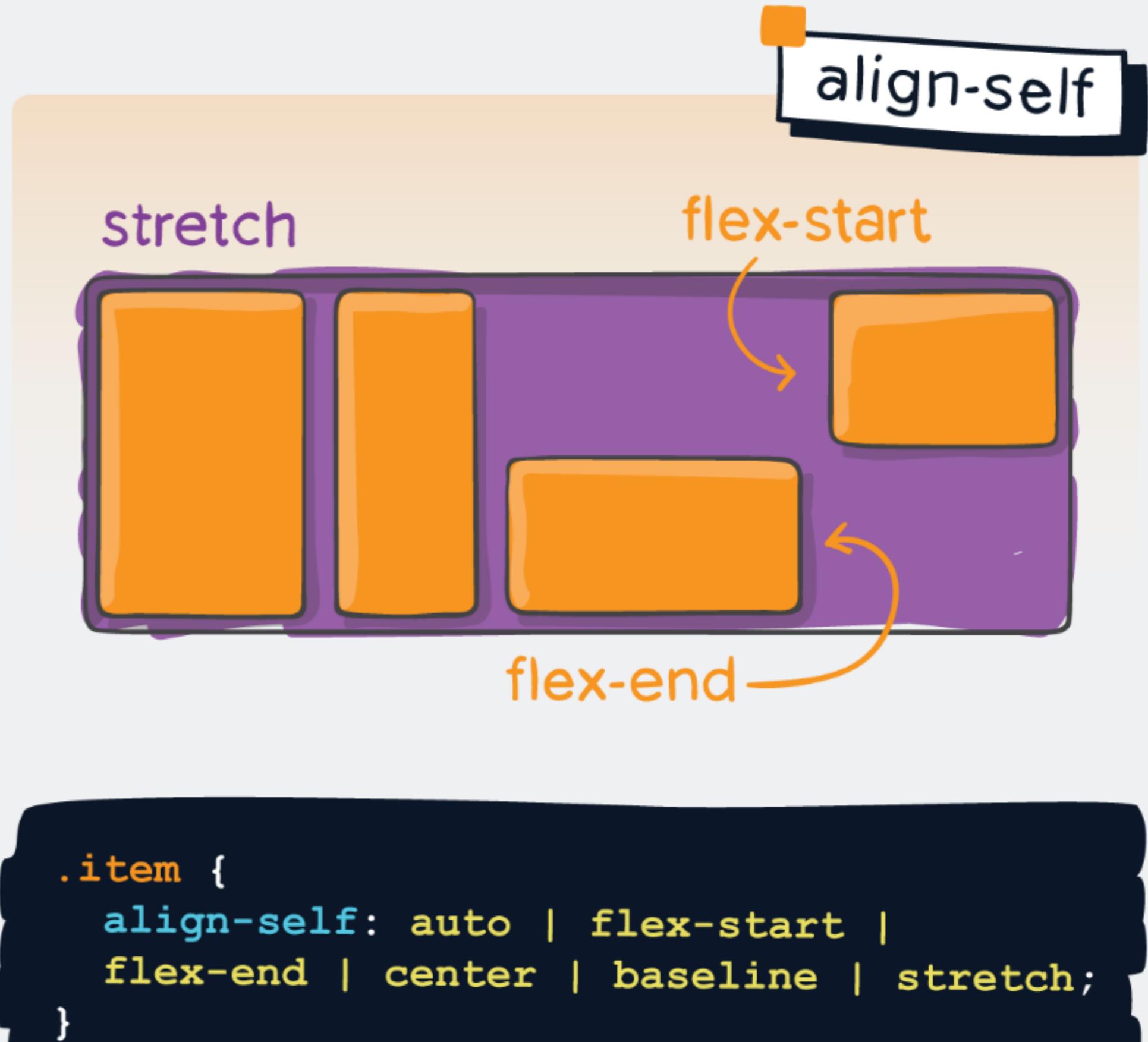
```
.item {
  order: 5; /* default is 0 */
}
```



```
.item {
  flex-shrink: 1; /* default is 1 */
  flex-grow: 2; /* default is 0 */
  flex-basis: 50px; /* default auto */
}
```



```
.container {
  align-items: stretch | flex-start |
  flex-end | center | baseline;
}
```



```
.item {
  align-self: auto | flex-start |
  flex-end | center | baseline | stretch;
}
```

# <Despedida>

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**

**CONQUERBLOCKS**