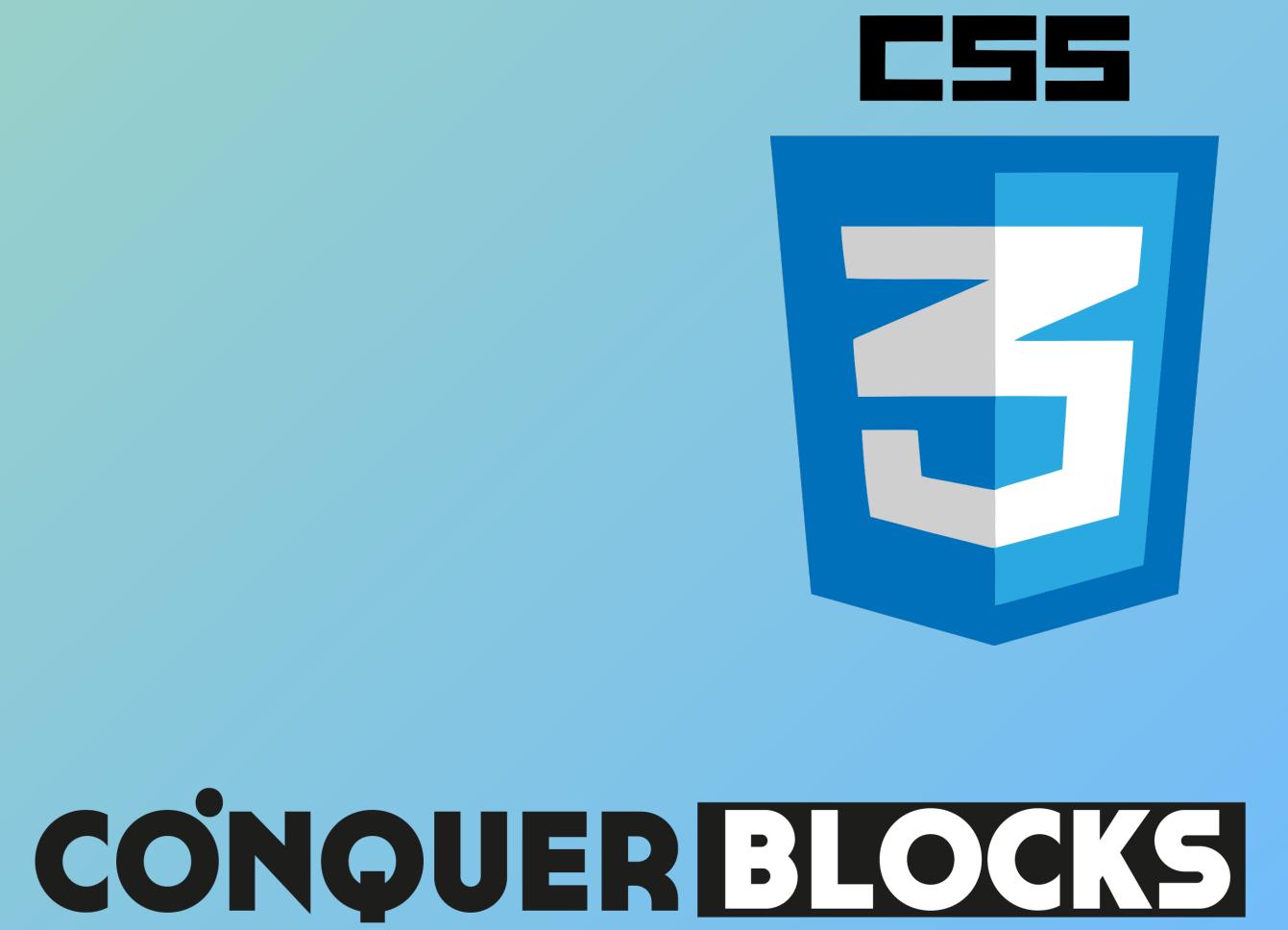


# {css}

Clase 24



# Clase 24

<índice>

## Posicionamiento Grid Capítulo 2

Recordemos lo visto

---

Resolvamos los ejemplos de la clase anterior

---

Función min-max

---

Auto-fill

---

Auto-fit

---

Alineación

---

# **Recordemos lo visto**

# Función minmax()

# minmax

La función minmax() se puede utilizar como valor para definir rangos flexibles de celda.

Funciona de la siguiente forma

# minmax

## Función

```
minmax( SIZE min, SIZE max)
```

## Descripción

Define un rango entre `min` y `max`.

## minmax

Si establecemos un rango, por ejemplo, grid-template-column: minmax(200px, 500px), estaremos indicando que la celda de columna indicada, tendrá un tamaño de 500px, salvo que redimensionemos la ventana del navegador y la hagamos más pequeña, en cuyo caso, el tamaño de la celda podría ir disminuyendo hasta 200px, medida en la cuál se quedaría como mínimo.

# minmax

```
<div class="container">
  <div class="item item-1">Item 1</div>
  <div class="item item-2">Item 2</div>
  <div class="item item-3">Item 3</div>
  <div class="item item-4">Item 4</div>
</div>

<style>
.container {
  display: grid;
  grid-template-columns: repeat(2, minmax(400px, 600px));
  grid-template-rows: repeat(2, 1fr);
  gap: 5px;
}

.item {
  background: black;
  color: white;
  padding: 1em;
}
</style>
```

Comprobarás que las celdas se hacen más pequeñas hasta un punto en el que se alcanza el mínimo.

# minmax

## Demo

# Auto-fill

# Auto-fill

En la función repeat() es posible utilizar las palabras claves auto-fill o auto-fit para indicar al navegador que queremos que rellene o ajuste el contenedor grid con múltiples elementos hijos dependiendo del tamaño del viewport

## Auto-fill

Es decir, si utilizamos `repeat(auto-fill, minmax(300px, 1fr))` el navegador se va a encargar de que los elementos hijos con el tamaño mínimo quepan en la primera fila, y los que no quepan, se desplacen a las siguientes filas del grid, de modo que se aproveche lo mejor posible el contenedor

# Auto-fill

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));  
  background: grey;  
  gap: 10px;  
}  
  
.item {  
  background: blue;  
  color: #fff;  
  font-size: 2rem;  
}
```

# Auto-fill

## Otra forma de explicarlo

Esta palabra clave nos ayuda a decirle al navegador que inserte el número de columnas o filas que sea necesario para llenar el espacio.

Podríamos escribir la siguiente línea de código:

```
// grid repeat auto-fill  
grid-template-columns: repeat (auto-fill, minmax (150px, 1fr))
```

# Auto-fill

Lo que quiere decir la función auto-fill en este caso es que el navegador puede ubicar el número de columnas que quepan en el ancho, mientras que su ancho mínimo sea de 150px. Entonces, cuando la pantalla cambie de tamaño, el navegador modificará automáticamente el número de columnas que haya según el ancho disponible. El máximo 1fr hace que las columnas siempre tengan el mismo ancho una respecto a la otra.

// grid repeat auto-fill

grid-template-columns: repeat (auto-fill, minmax (150px, 1fr))

# Auto-fill

Luego veremos un demo

# Auto-fit

## Auto-fit

Si cambiamos el ejemplo anterior a auto-fit no veremos ninguna diferencia. Sin embargo, si por ejemplo cambiamos el valor mínimo de 300px a 50px (de modo que no llegue a cubrir la primera fila completamente), comprobaremos que mientras auto-fill va llenando la fila del grid y deja el resto del espacio libre, auto-fit ajusta el tamaño de los ítems para que cubran el tamaño máximo de la fila.

# Auto-fit

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));  
  background: grey;  
  gap: 10px;  
}  
  
.item {  
  background: blue;  
  color: #fff;  
  font-size: 2rem;  
}
```

# Auto-fit

## Otra forma de explicarlo

La diferencia entre las funciones auto-fill y auto-fit en CSS Grid es que la primera pondrá el número de columnas que quepan, sin importar el número de elementos a ubicar. Esto quiere decir que, cuando usamos auto-fill, aun si tenemos pocos elementos, este seguirá poniendo columnas aunque estén vacías si el espacio es mayor a la suma de sus anchos.

# Auto-fit

La función auto-fit, en cambio, ajusta las columnas para ocupar todo el espacio disponible, sin dejar espacio restante. Ten presente que esta función respeta los anchos mínimos, al igual que auto-fill.

# Auto-fit

Demo

# Alineación

# Alineación

Existen una serie de propiedades que se pueden utilizar para colocar y ajustar nuestra cuadrícula grid o ajustar los ítems a lo largo de ella, de forma sencilla y cómoda. Algunas de estas propiedades probablemente ya las conocerás del módulo CSS flex, sin embargo, en grid pueden tener un comportamiento diferente.

# Alineación

| Propiedad       | Valores  | Afecta a... |
|-----------------|--|-------------|
| justify-items   | start   end   center   stretch   | Elementos 1 |
| align-items     | start   end   center   stretch   | Elementos 2 |
| justify-content | start   end   center   stretch   space-around   space-between   space-evenly | Contenido 1 |
| align-content   | start   end   center   stretch   space-around   space-between   space-evenly | Contenido 2 |

# Alineación

- **justify-items:** Alinea los elementos (hijos) en horizontal (eje principal) dentro de cada celda.
- **align-items:** Alinea los elementos (hijos) en vertical (eje principal) dentro de cada celda.
- **justify-content:** Alinea el contenido (la cuadricula) en horizontal (eje secundario) en el contenedor padre.
- **align-content:** Alinea el contenido (la cuadricula) en vertical (eje secundario) en el contenedor padre.

# Alineación

La primera propiedad, **justify-items** sirve para colocar los ítems de un contenedor grid a lo largo de sus celdas correspondientes, siempre en el eje principal (por defecto, en horizontal). Los valores que puede tomar esta propiedad son los siguientes

# Alineación

| Valor          | Descripción   |
|----------------|---|
| start          | Coloca cada ítem al <b>inicio</b> de su celda en el eje principal.                                      |
| end            | Coloca cada ítem al <b>final</b> de su celda en el eje principal.                                       |
| center         | Coloca cada ítem en el <b>centro</b> de su celda en el eje principal.                                   |
| <b>stretch</b> | Hace que cada ítem <b>se estire</b> y ocupe todo el espacio disponible de su celda en el eje principal. |

# Alineación

De forma análoga, la propiedad **align-items** sirve para colocar los ítems de un contenedor grid a lo largo de sus celdas correspondientes, pero en lugar de el eje principal, las coloca en el eje secundario (por defecto, en vertical). Los valores que puede tomar son los mismos que la propiedad anterior.

# Alineación

Alineación de contenido

# Alineación

La propiedad **justify-content** permite modificar la distribución del contenido de la cuadrícula en su contenedor padre, a lo largo de su eje principal (por defecto, el horizontal). Los valores que puede tomar son los siguientes

# Alineación

| Valor          | Descripción   |
|----------------|---|
| start          | Coloca la cuadrícula en su conjunto al <b>inicio</b> del contenedor padre en su eje principal ( <a href="#">horizontal</a> ).                   |
| end            | Coloca la cuadrícula en su conjunto al <b>final</b> del contenedor padre en su eje principal ( <a href="#">horizontal</a> ).                    |
| center         | Coloca la cuadrícula en su conjunto al <b>centro</b> del contenedor padre en su eje principal ( <a href="#">horizontal</a> ).                   |
| <b>stretch</b> | <b>Estira</b> la cuadrícula ocupando todo el <b>espacio disponible</b> del contenedor padre en su eje principal ( <a href="#">horizontal</a> ). |
| space-between  | Establece <b>espacios sólo entre las celdas</b> , en su eje principal ( <a href="#">horizontal</a> ).   |
| space-around   | Establece <b>espacios alrededor de las celdas</b> , en su eje principal ( <a href="#">horizontal</a> ).   |
| space-evenly   | Idem al anterior, pero solapando los espacios, de modo que sean todos de tamaño equivalente.  |

# Alineación

De forma análoga, la propiedad **align-content** sirve para colocar el contenido de la cuadrícula en su contenedor padre, pero a lo largo de su contenedor secundario (por defecto, el vertical). Los valores que puede tomar son exactamente los mismos que la propiedad anterior.

# Alineación

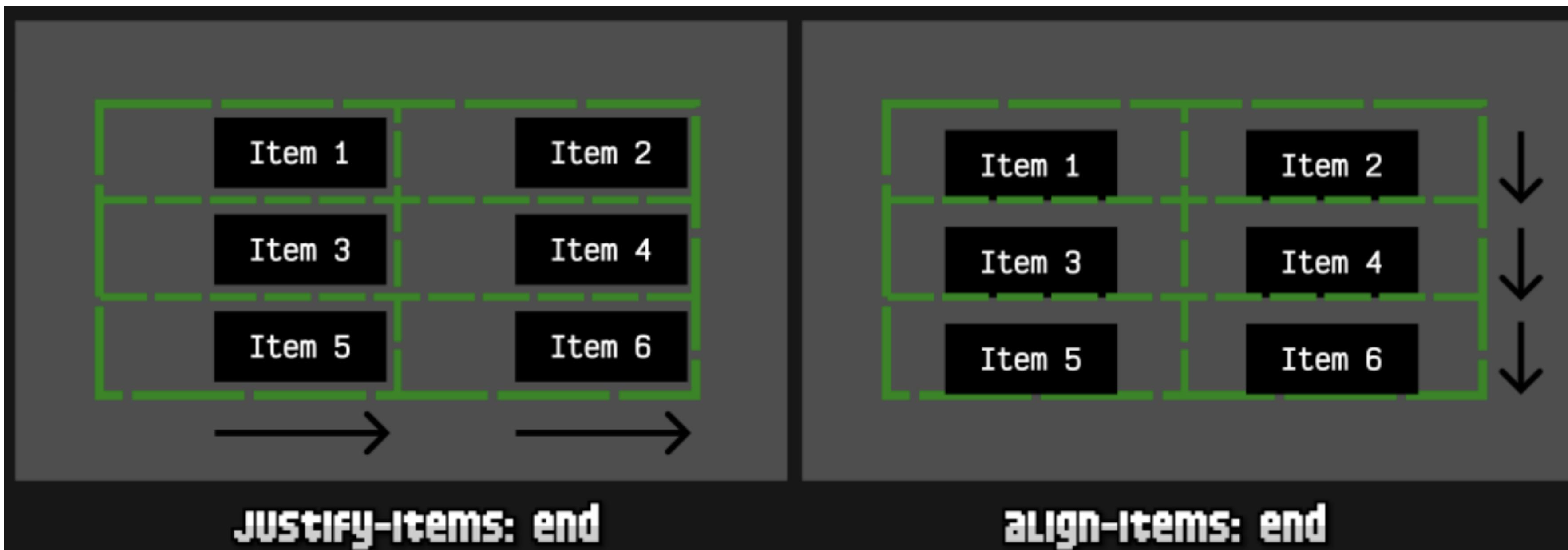
```
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
  <div class="item">Item 4</div>
  <div class="item">Item 5</div>
  <div class="item">Item 6</div>
</div>

<style>
.container {
  display: grid;
  grid-template-columns: repeat(2, 250px);
  grid-template-rows: repeat(3, 50px);
  gap: 10px;
  background: grey;
  height: 300px;

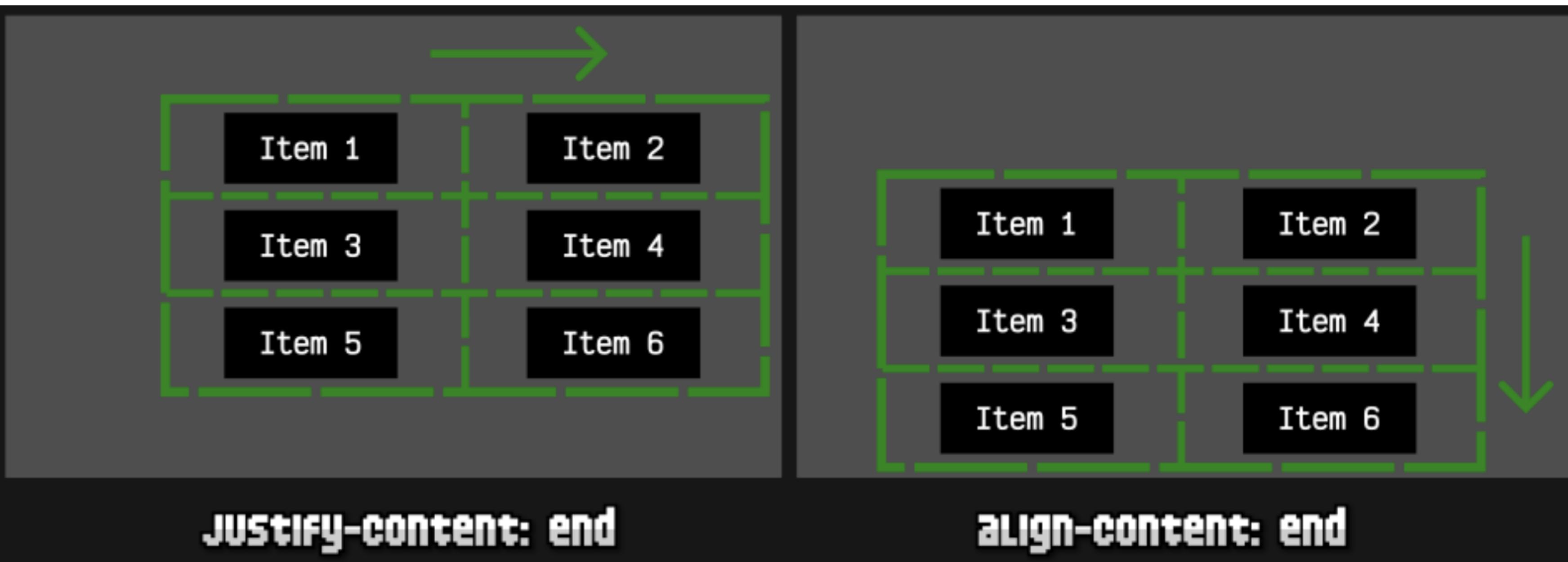
  justify-items: center;
  align-items: center;
  justify-content: center;
  align-content: center;
}

.item {
  padding: 10px;
  background: black;
  color: white;
}
</style>
```

# Alineación



# Alineación



`justify-content: end`

`align-content: end`

# Alineación

**Alineaciones específicas**  
¿os acordáis de flexbox?

# Alineación

## Alineaciones específicas

En el caso de que queramos que uno de los ítems hijos tenga una distribución diferente al resto, podemos aplicar en el elemento hijo la propiedad `justify-self` (eje principal) o `align-self` (eje secundario) sobreescribiendo su distribución general, y aplicando una específica.

# Alineación

| Propiedad    | Descripción  |
|--------------|--|
| justify-self | Altera la alineación del ítem hijo en el eje horizontal y la sobreescribe con la indicada. |
| align-self   | Altera la alineación del ítem hijo en el eje vertical y la sobreescribe con la indicada.   |

# Alineación

## Alineaciones específicas

Recuerda que estas propiedades funcionan exactamente igual que sus análogas justify-items o align-items y tienen los mismos valores, sólo que en lugar de indicarse en el elemento padre contenedor, se hace sobre un elemento hijo y repercute en dicho elemento hijo específicamente.

# Alineación

Demos

# Grid-template

Si acostumbras a utilizar estas propiedades frecuentemente, puedes utilizar la propiedad grid-template, que sirve de atajo para muchas cosas, y una de ellas, resumir en una sola propiedad los valores que tenemos en grid-template-columns y en grid-template-rows

Esta propiedad convierte en un proceso bastante cómodo el crear grids de unas dimensiones concretas de forma resumida.

En el caso de utilizar el valor `none`, las propiedades `grid-template-rows`, `grid-template-columns` y la propiedad `grid-template-areas`, se establecen a sus valores por defecto, desactivando su funcionamiento.

| Propiedad     | Valores  | Descripción                              |
|---------------|--|--|
| grid-template | <b>none   grid-template-rows / grid-template-columns</b> | Atajo para definir dimensiones del grid. |

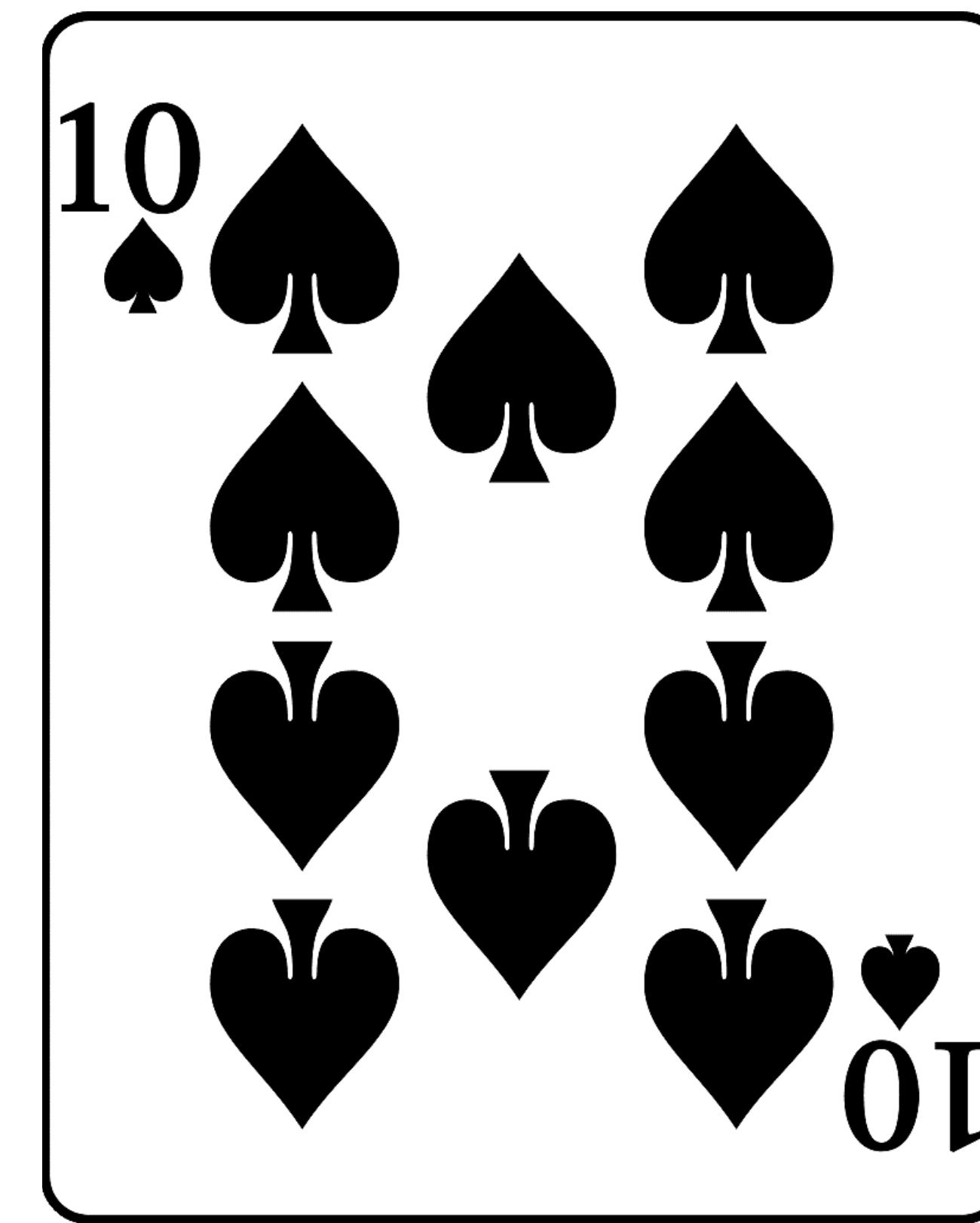
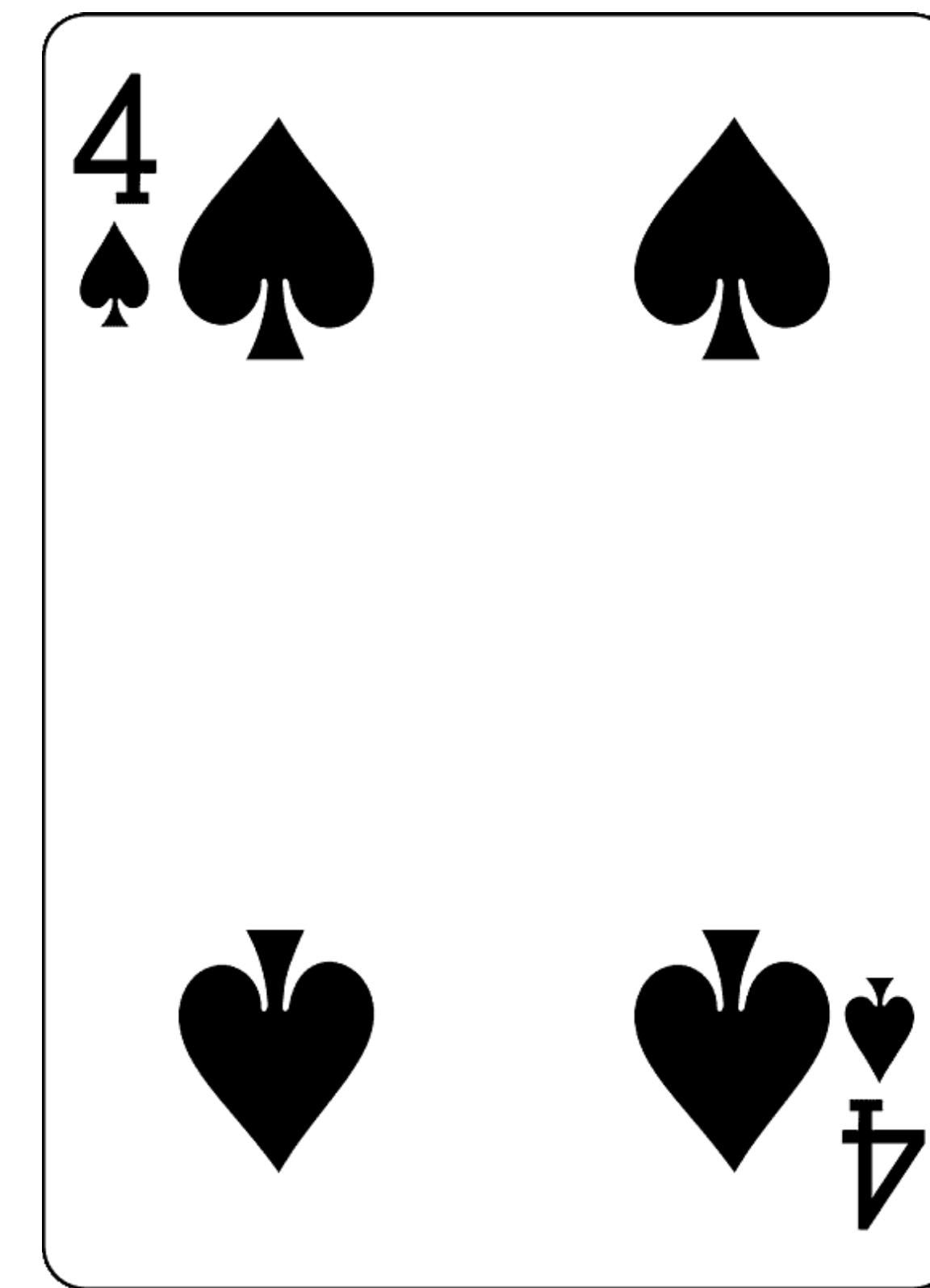
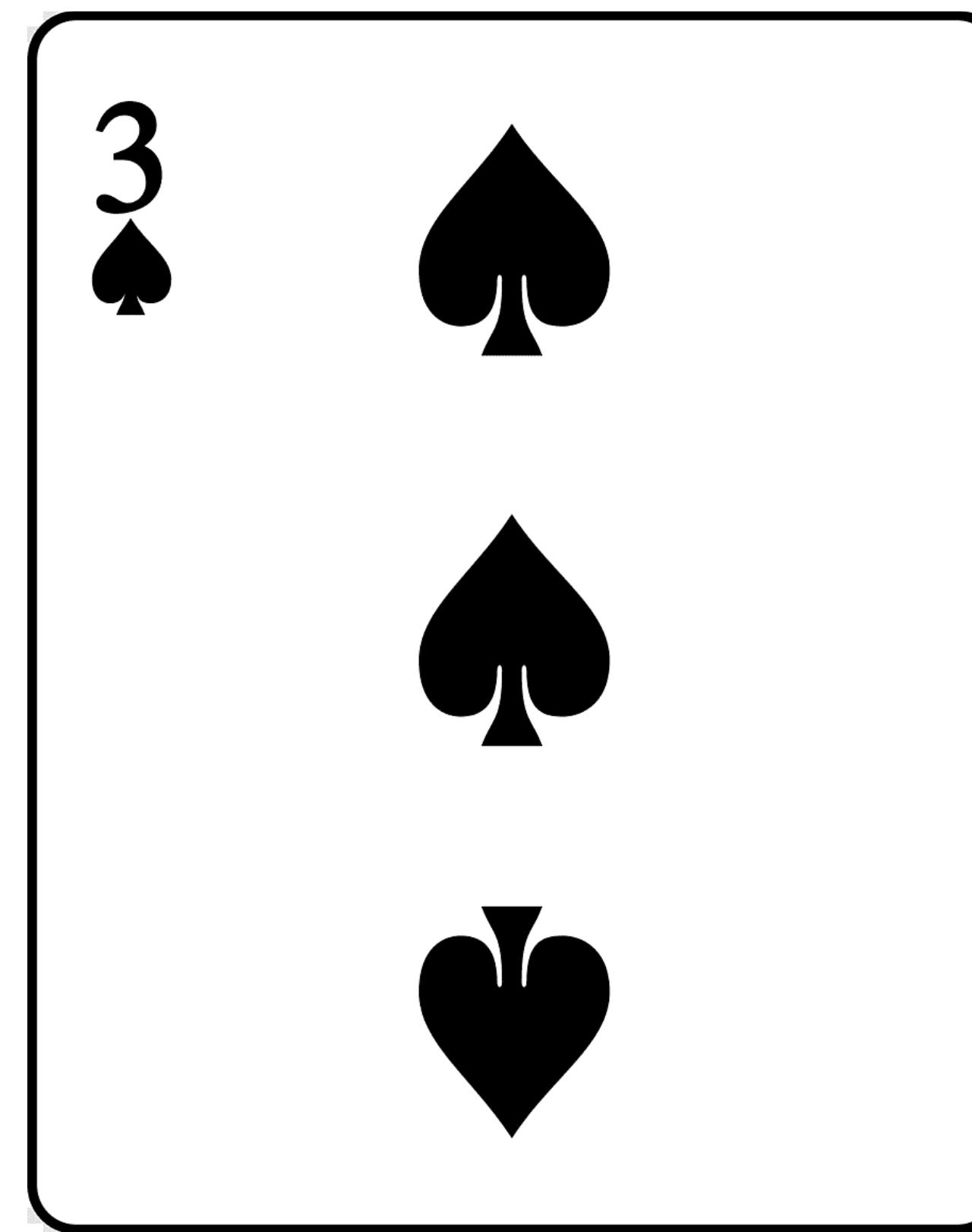
# Demo

**Esto no es todo...**

# Juego de jardín

**¿Quieres entretenerte?**

**CONQUERBLOCKS**



Tendrás que buscar información sobre: transform: rotate(180deg)

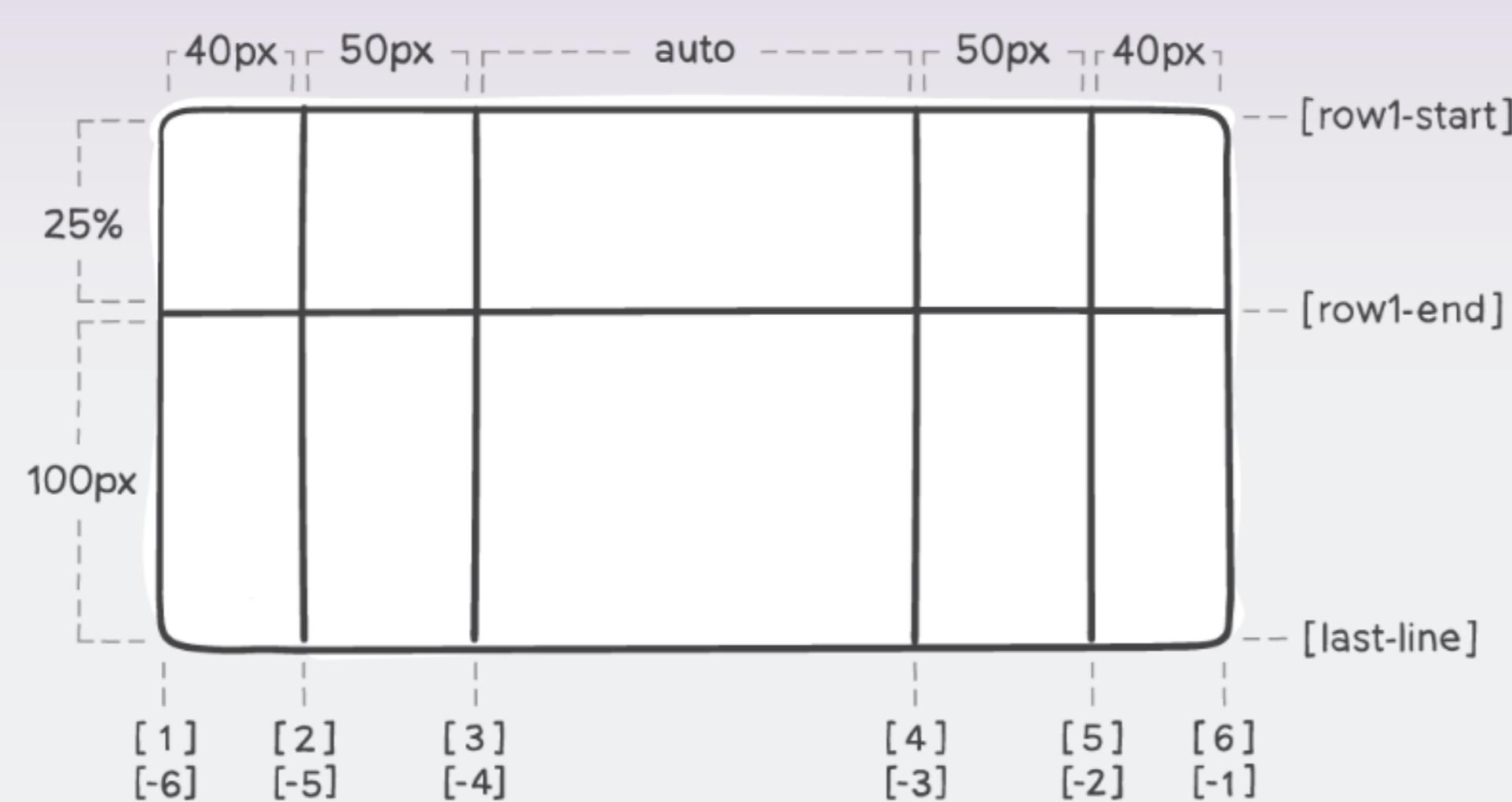
Imagen de la pica

# CSS Grid

a guide from

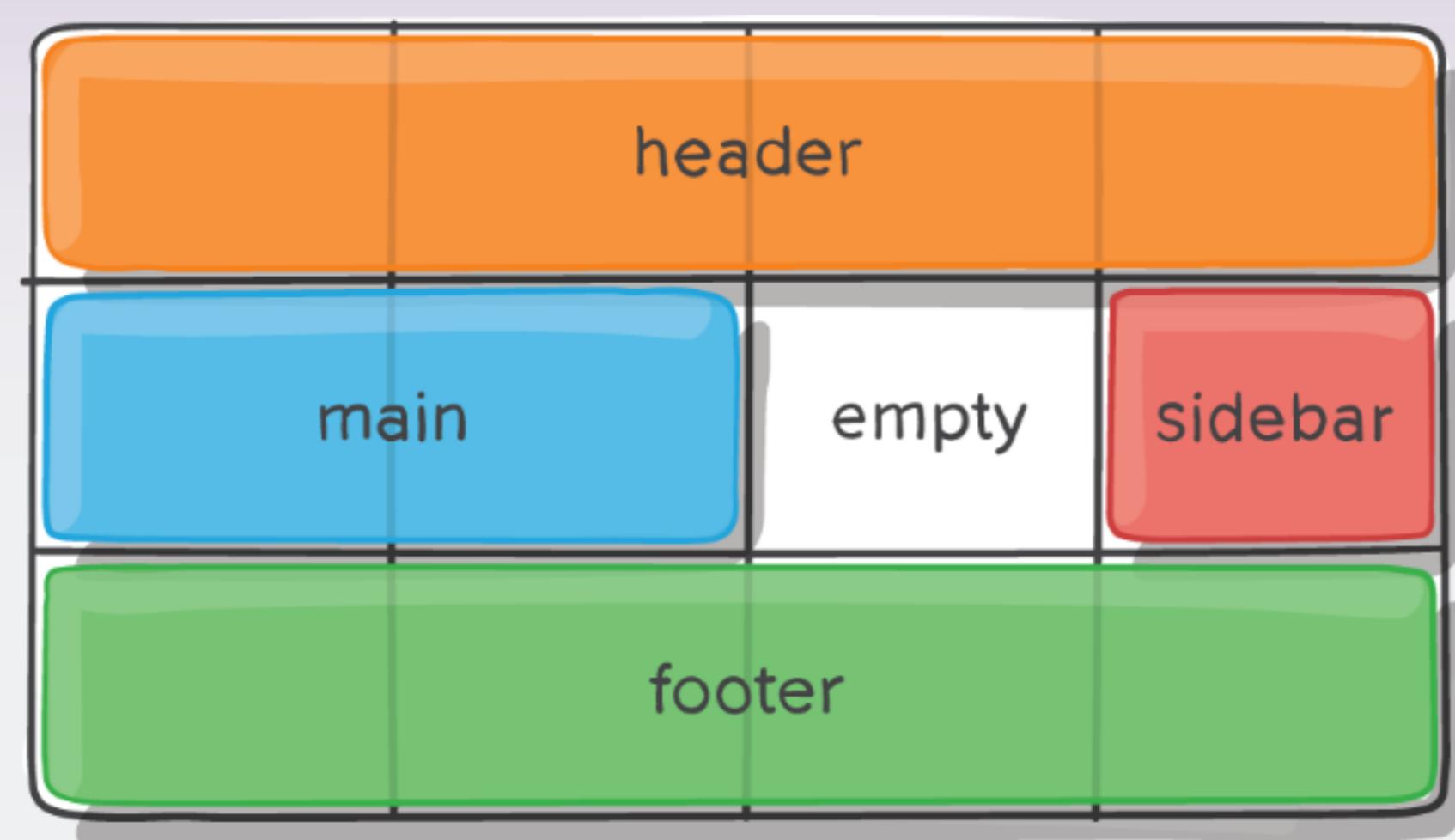


## grid-template-columns grid-template-rows



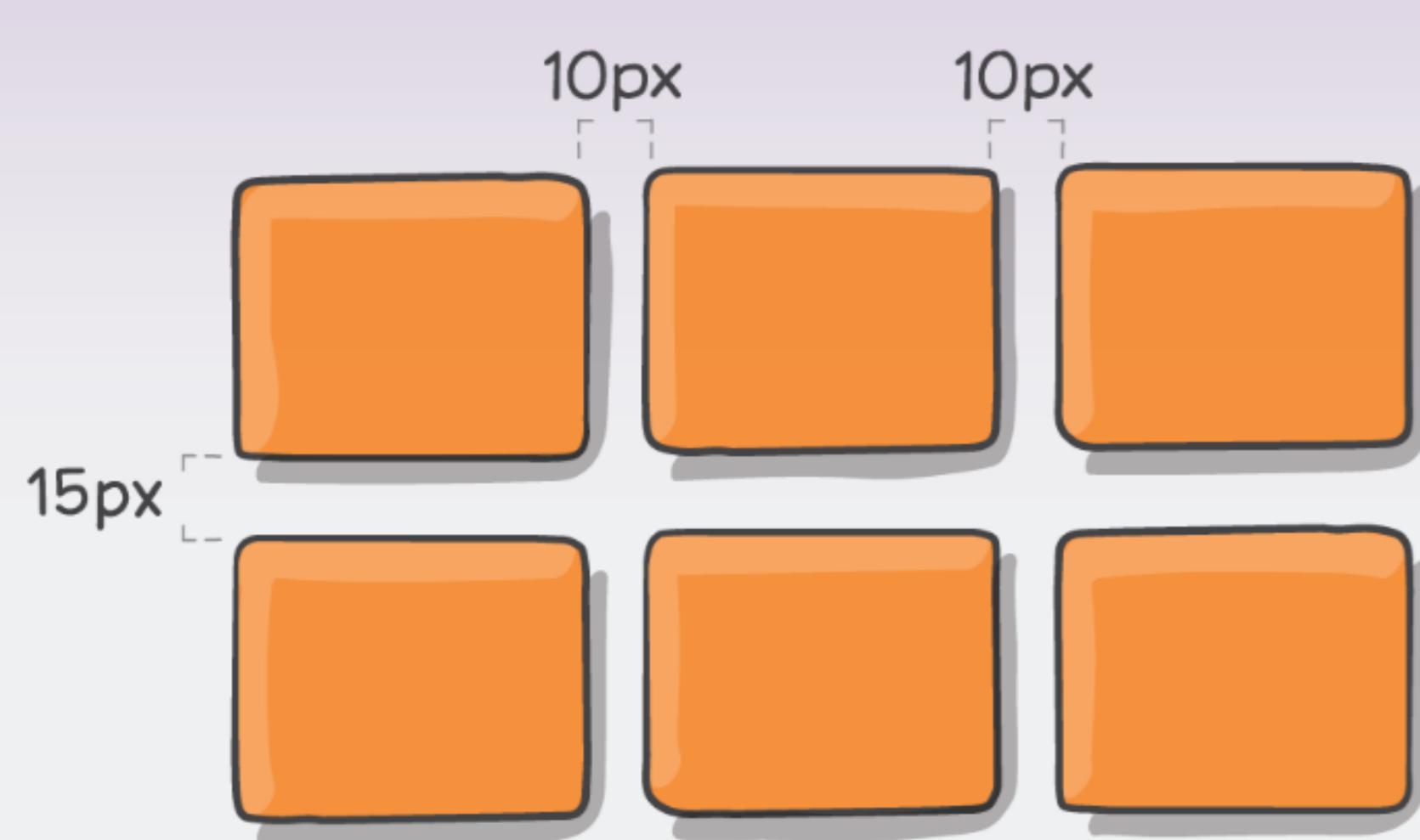
```
.container {  
  grid-template-columns: <value> | <name>;  
  grid-template-rows: <value> | <name>;  
}
```

## grid-template-areas



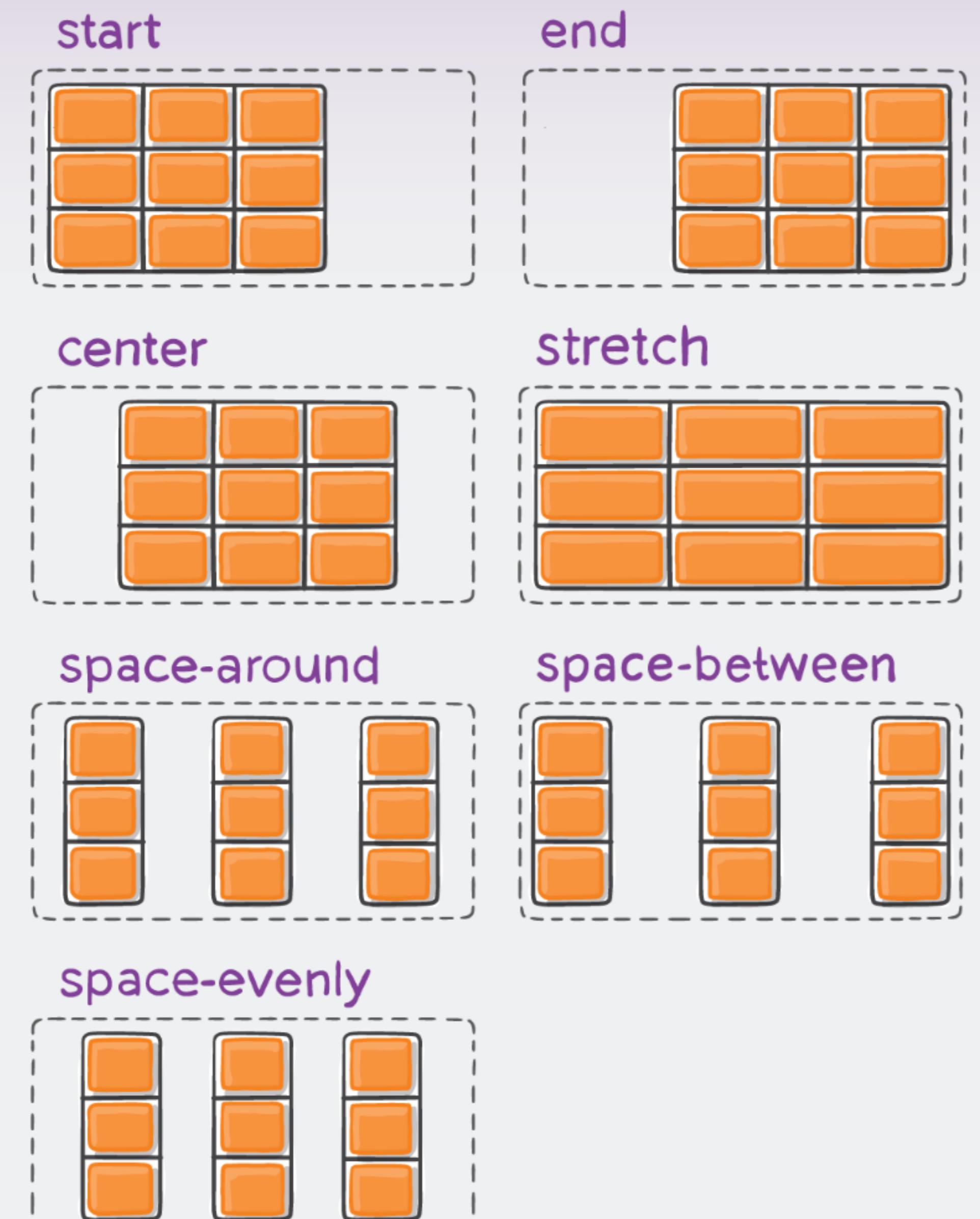
```
.container {  
  grid-template-areas: "<name> | . | none";  
}
```

## column-gap, row-gap, gap



```
.container {  
  column-gap: <value>;  
  row-gap: <value>;  
  gap: <row-gap> <column-gap>;  
}
```

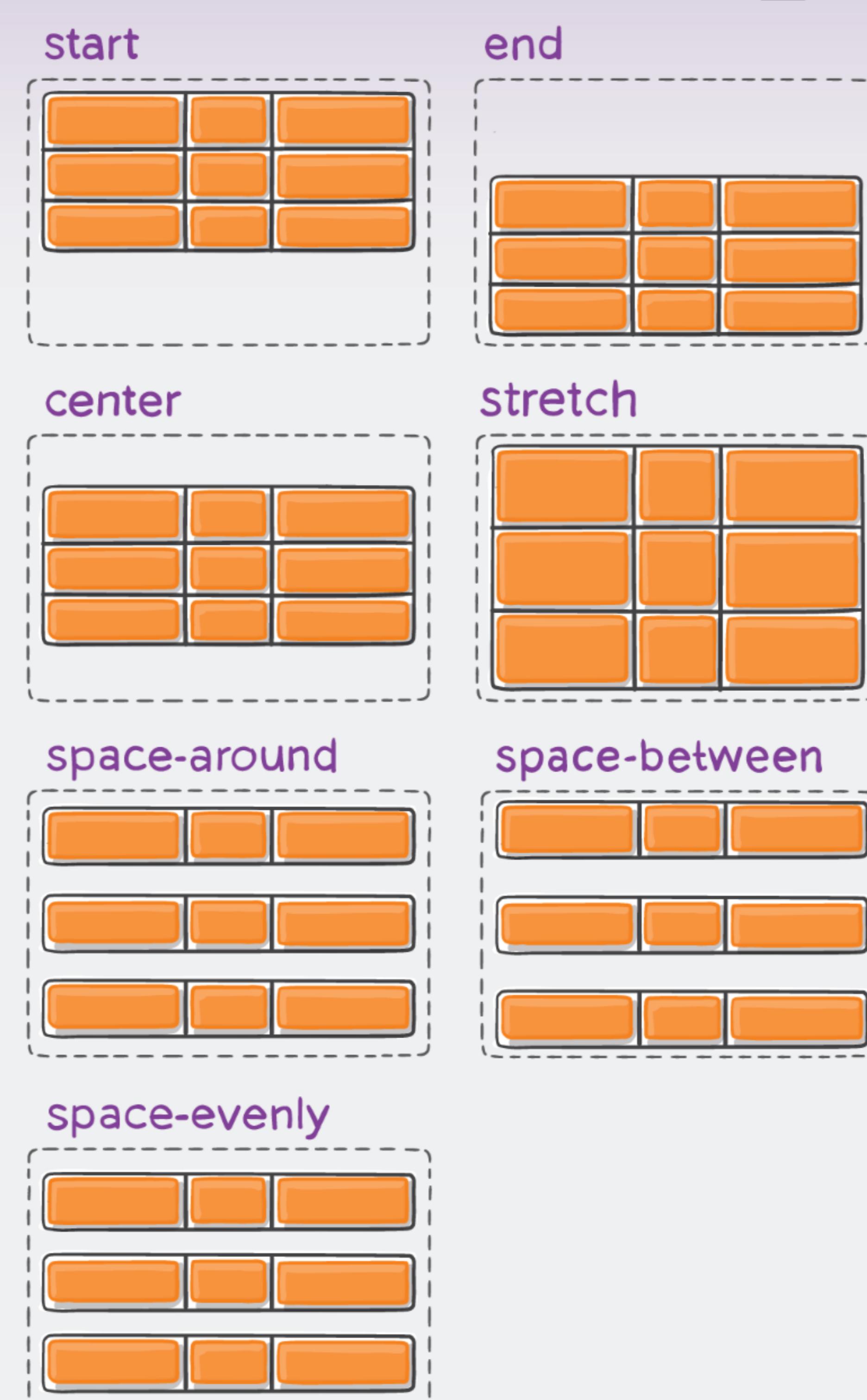
## justify-content



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

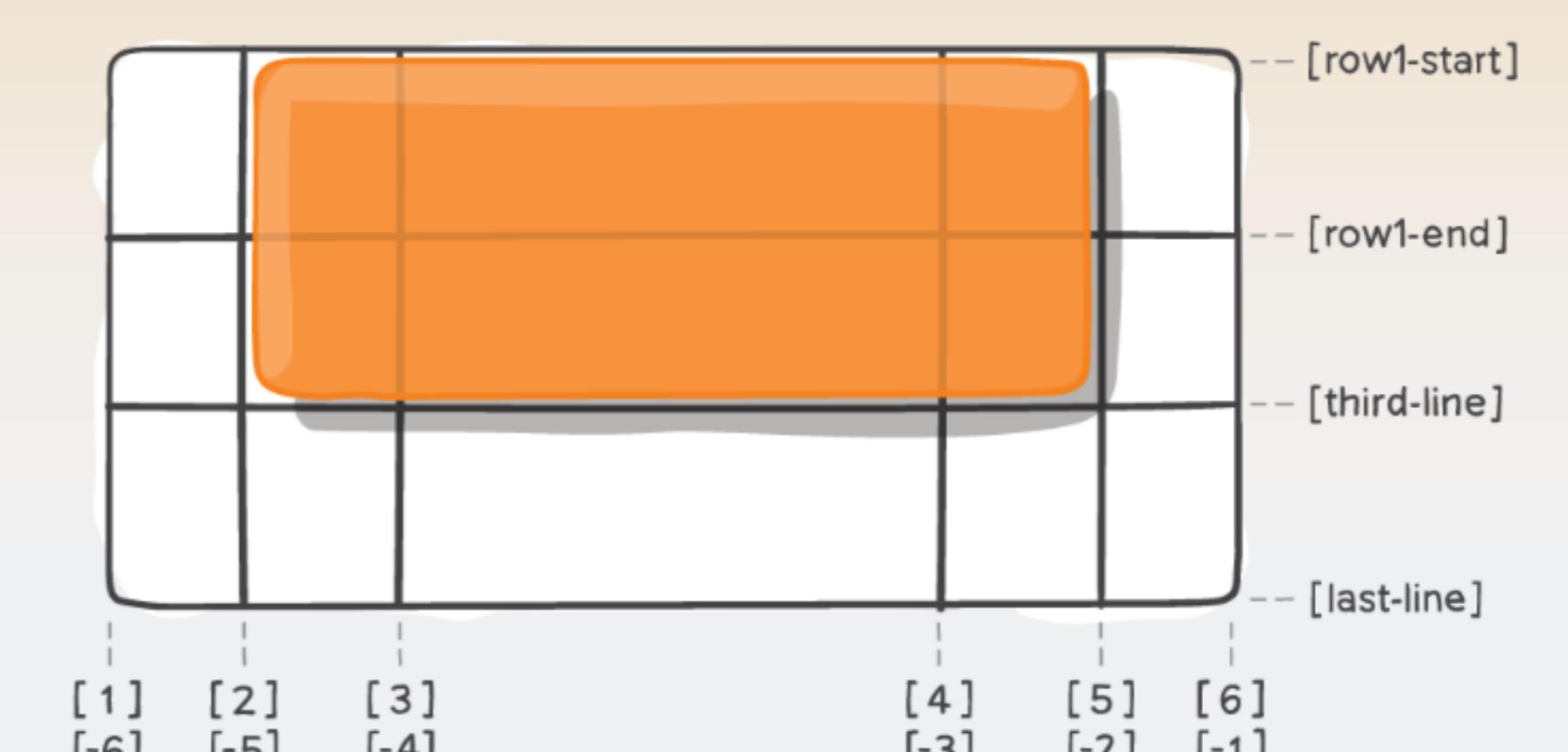
```
.container {  
  display: grid; /* or inline-grid */  
}
```

## align-content



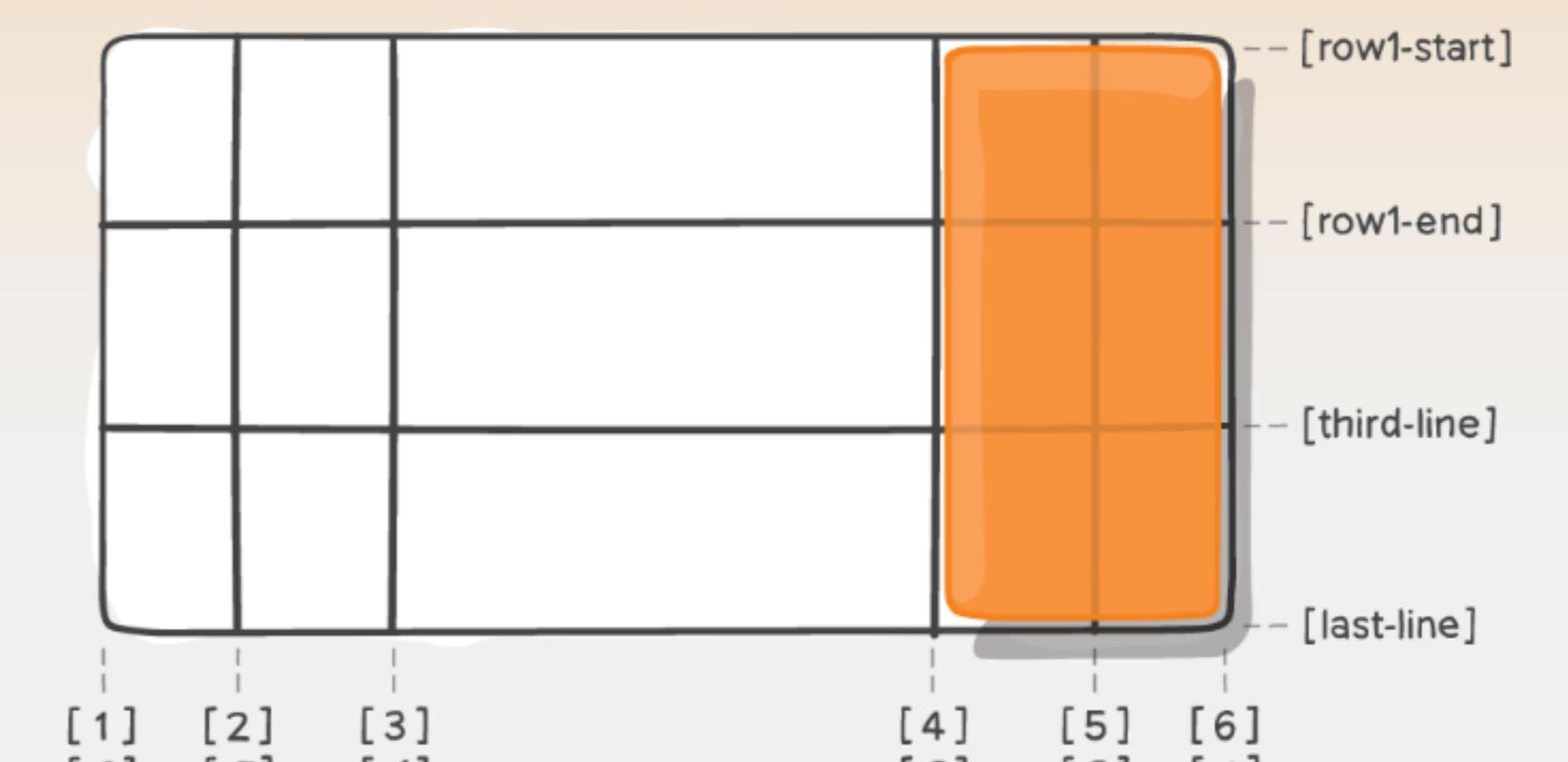
```
.container {  
  align-content: start | end | center |  
    stretch | space-around | space-between |  
    space-evenly;  
}
```

## grid-column, grid-row



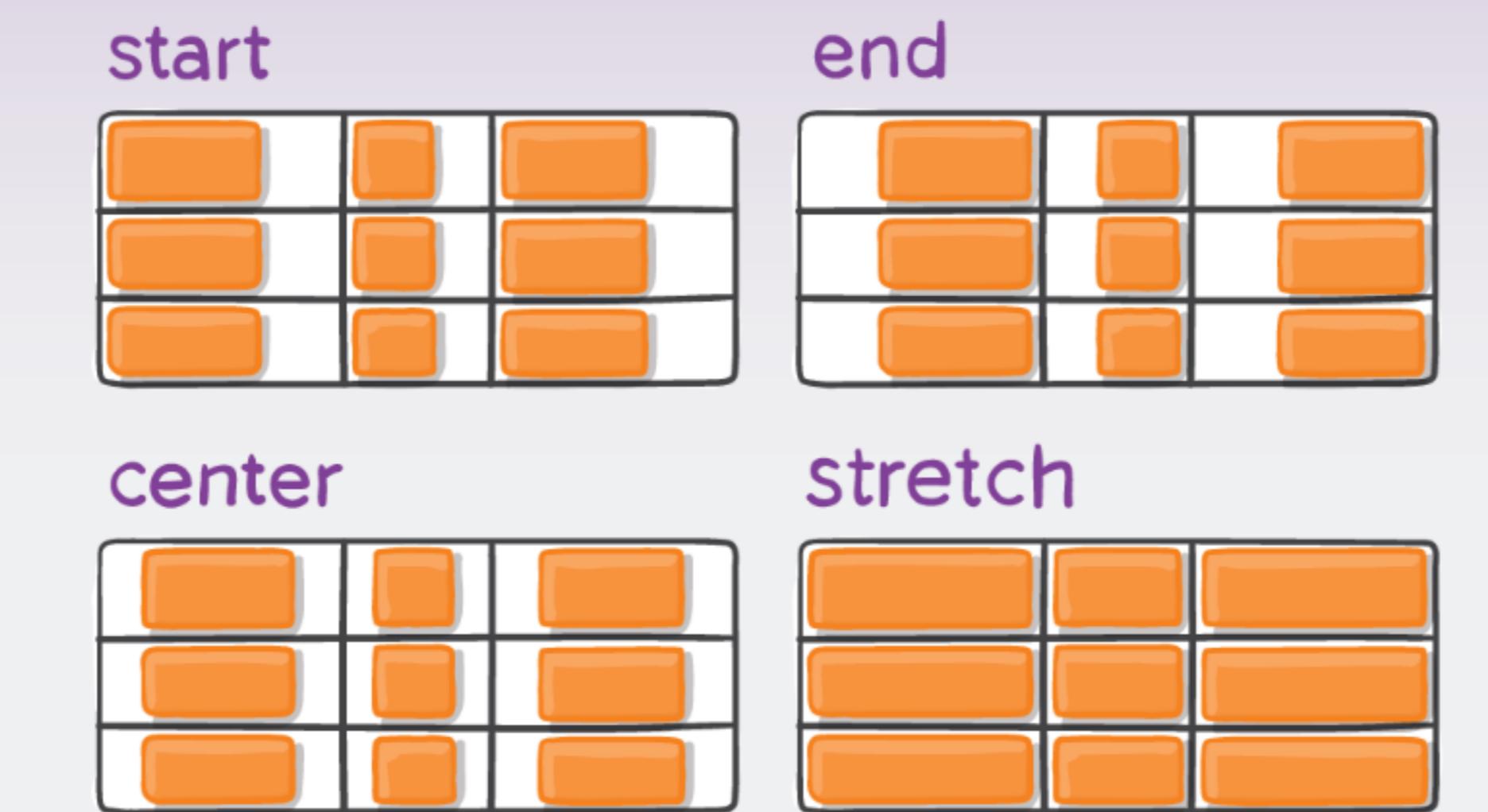
```
.item {  
  grid-column: <start-line> / <end-line> |  
    <start-line> / span <value>;  
  grid-row: <start-line> / <end-line> |  
    <start-line> / span <value>;  
}
```

## grid-area



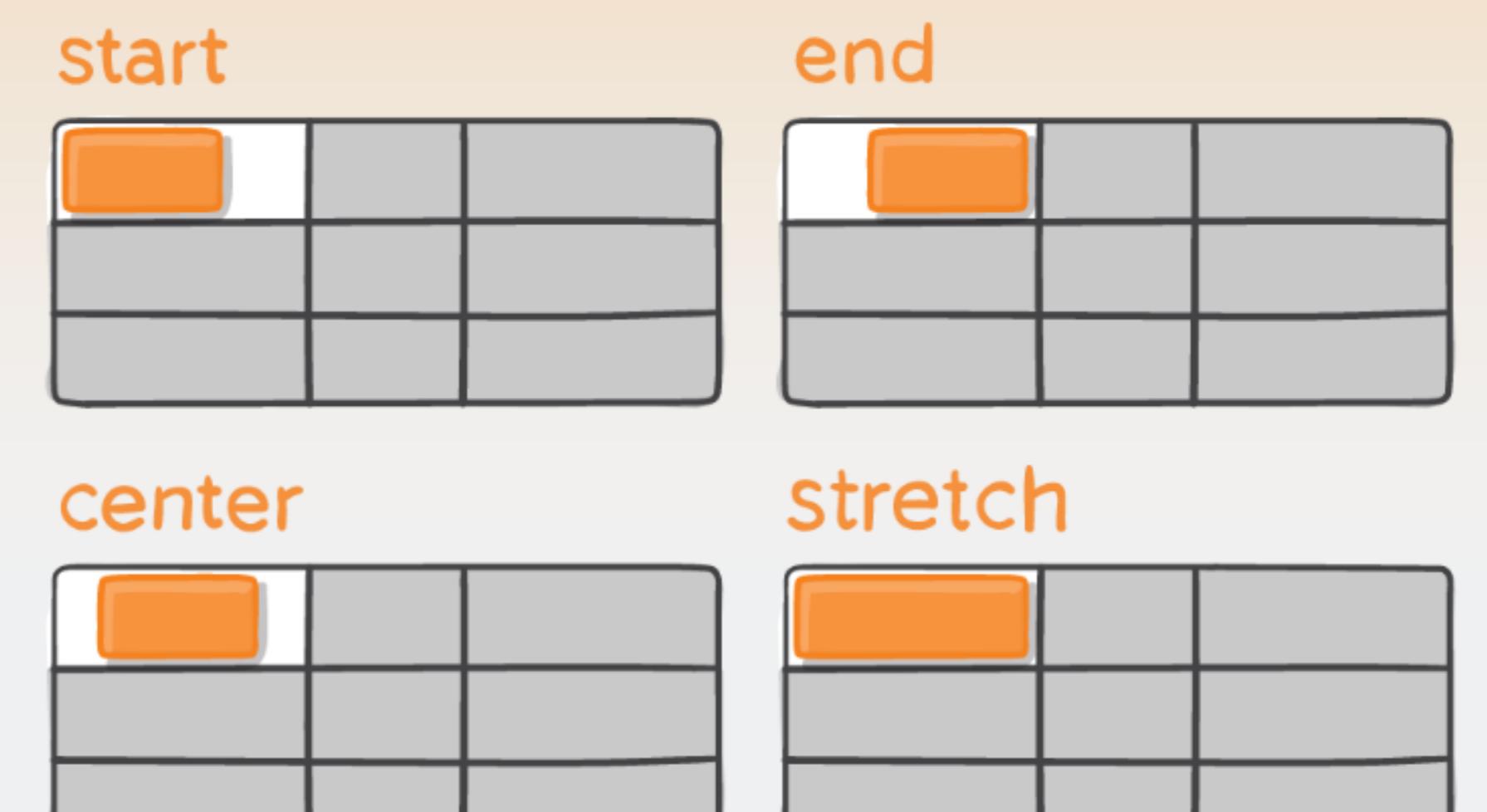
```
.item {  
  grid-area: <row-start> / <column-start> |  
    <row-end> / <column-end> | <name>;  
}
```

## justify-items



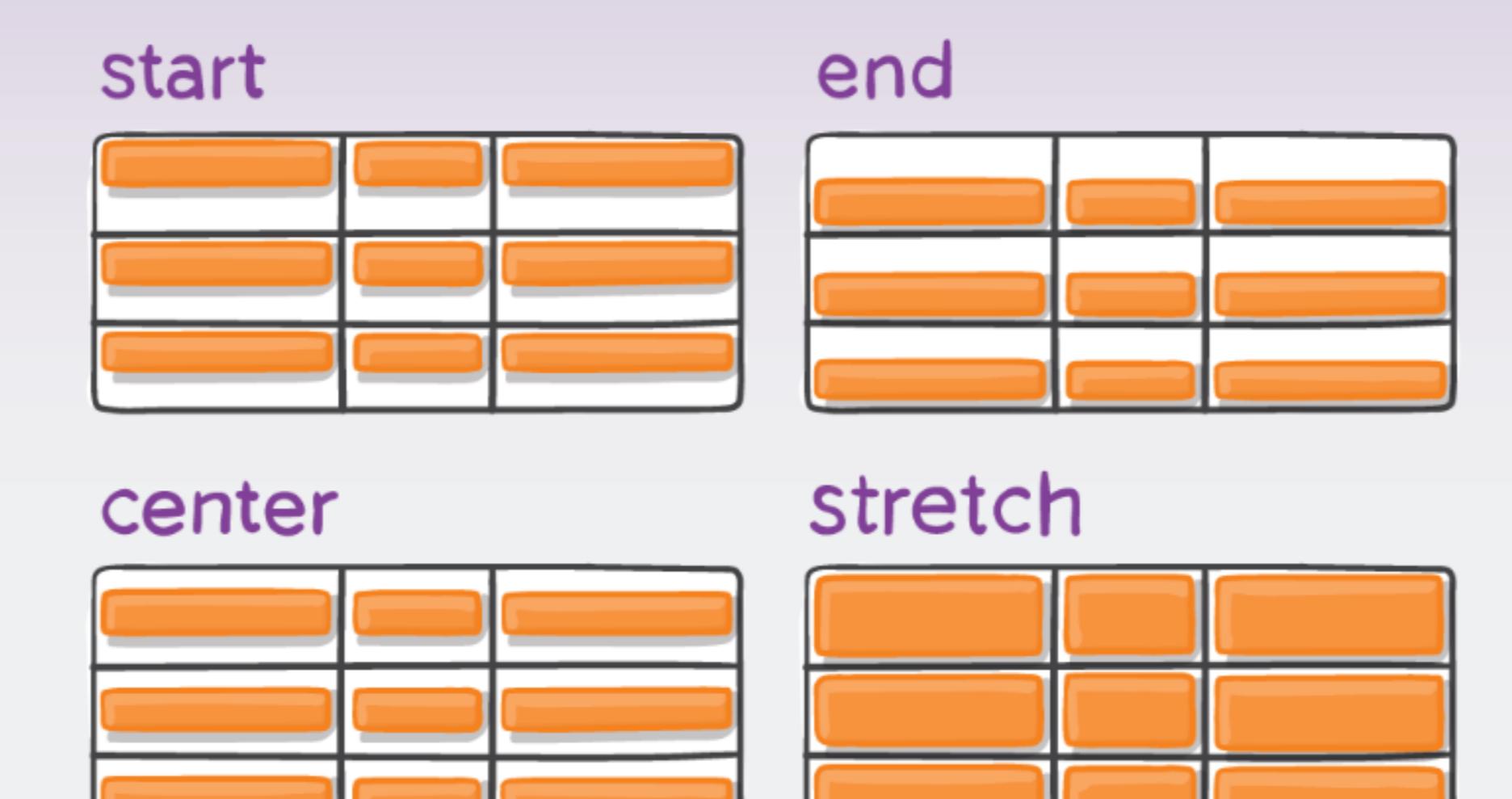
```
.container {  
  justify-items: start | end | center |  
    stretch;  
}
```

## justify-self



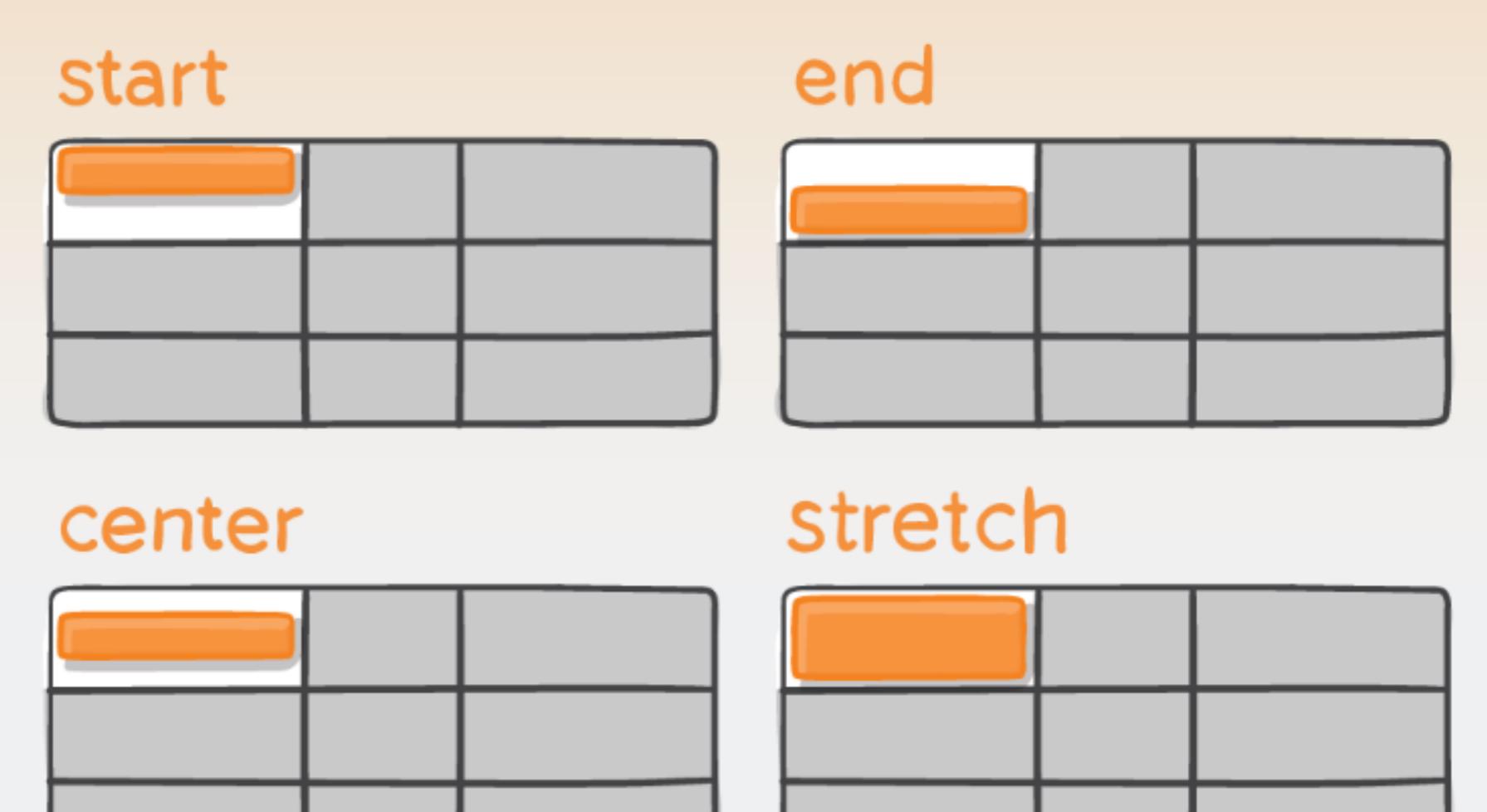
```
.item {  
  justify-self: start | end | center |  
    stretch;  
}
```

## align-items



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

## align-self



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

# Recursos

## RECURSOS

Manz: [Introducción a Grid](#)

CSS Tricks: [La guía completa de Grid](#)

# <Despedida>

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**

**CONQUERBLOCKS**