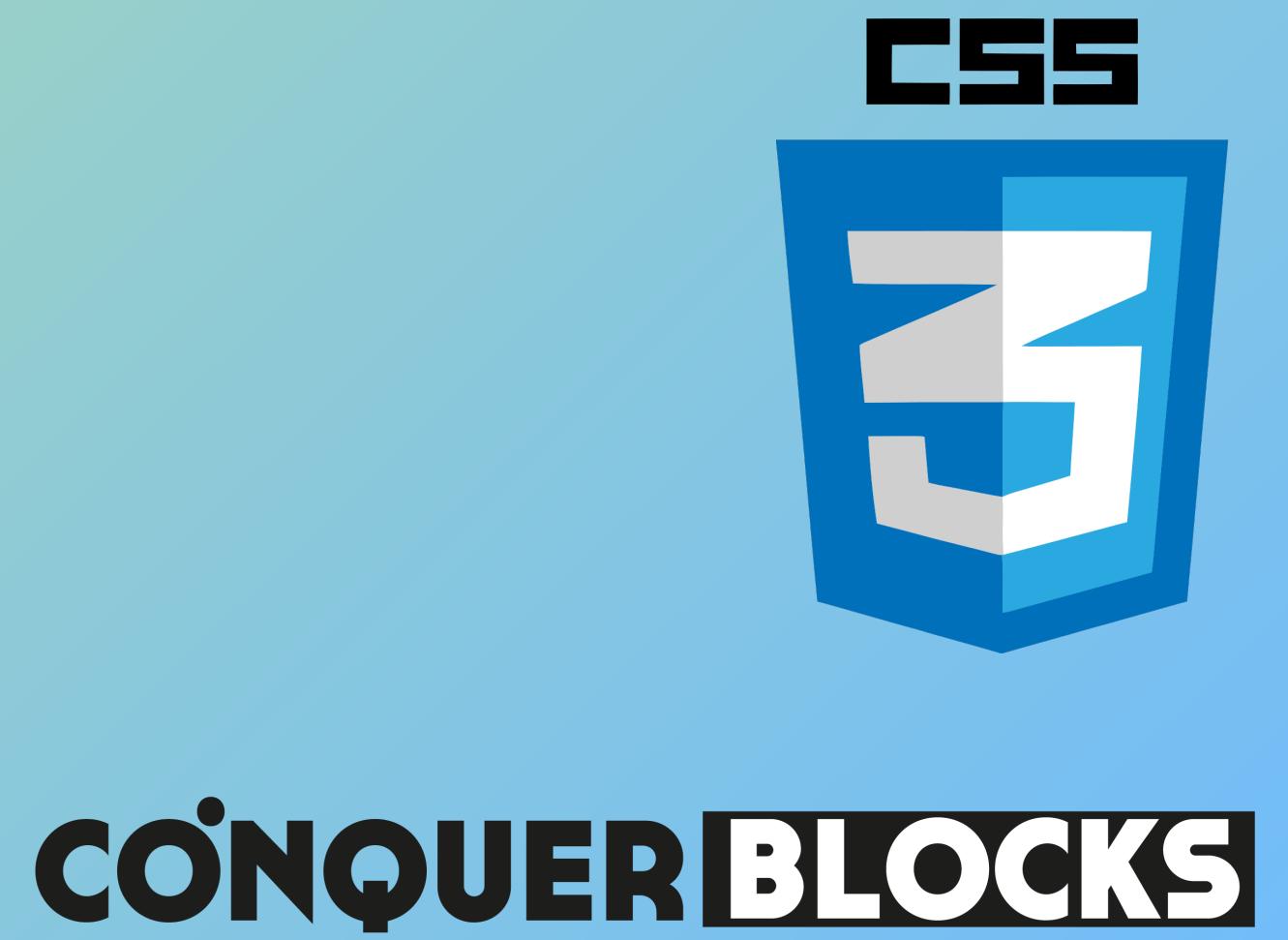


# {css}

Clase 23



# Clase 23

# <índice>

## Posicionamiento Grid Capítulo 1

Display Grid

---

Grid Template Columns y Rows

---

Gap

---

Order

---

Fr y Repeat

---

# Display Grid

# Display Grid

El sistema de elementos flexibles Flex es una gran mejora, sin embargo, está orientado a estructuras de una sola dimensión

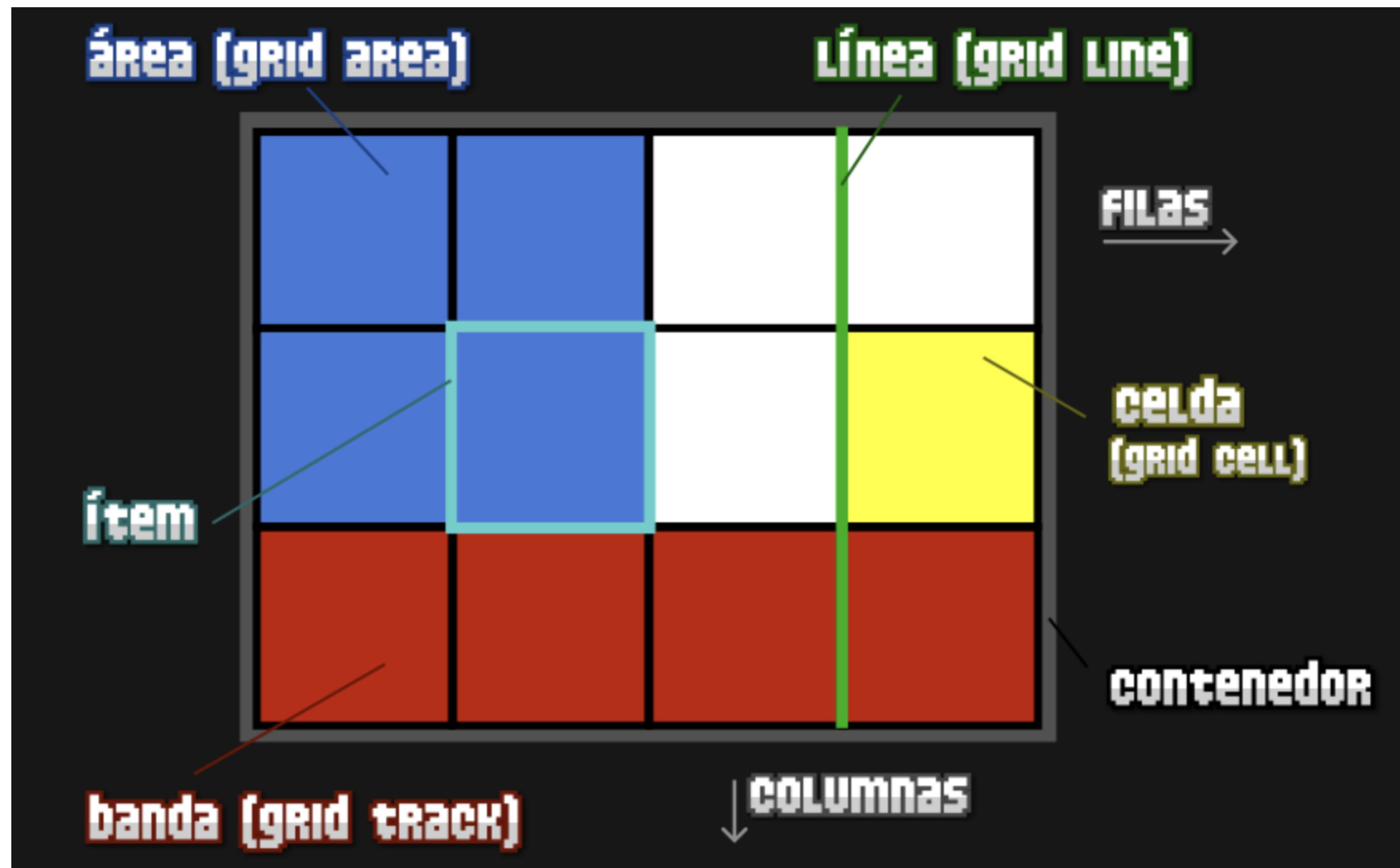
# Display Grid

Con el paso del tiempo, muchos frameworks CSS y librerías comenzaron a utilizar un sistema basado en un grid donde se definía una cuadrícula, a la que era posible darle tamaño, posición o colocación, cambiando el nombre de sus clases.

# Display Grid

**Grid** toma la filosofía (y muchas de las bases y conceptos) que se utilizan en **Flexbox**

# Display Grid



# Display Grid

- Contenedor: El elemento padre contenedor que definirá la cuadrícula o rejilla.
- Ítem: Cada uno de los hijos que contiene la cuadrícula (elemento contenedor).
- Celda (grid cell): Cada uno de los cuadritos (unidad mínima) de la cuadrícula.

# Display Grid

- Área (grid area): Región o conjunto de celdas de la cuadrícula.
- Banda (grid track): Banda horizontal o vertical de celdas de la cuadrícula.
- Línea (grid line): Separador horizontal o vertical de las celdas de la cuadrícula.

# Display Grid

```
<div class="grid"> ← contenedor →  
  <div class="item item-1">Item 1</div> ← cada uno de los ítems del grid →  
  <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>
```

# Grid Template Columns y Grid Template Rows

# Grid Template

En Grid CSS, la forma principal de definir una cuadrícula es indicar el tamaño de sus filas y sus columnas de forma explícita. Para ello, sólo tenemos que usar las propiedades CSS

`grid-template-columns`

y

`grid-template-rows`

# Grid Template

Propiedad	Valor	Descripción
grid-template-columns	[col1] [col2] ...	Establece el <b>SIZE</b> de cada columna (col 1, col 2...).
grid-template-rows	[fila1] [fila2] ...	Establece el <b>SIZE</b> de cada fila (fila 1, fila 2...).

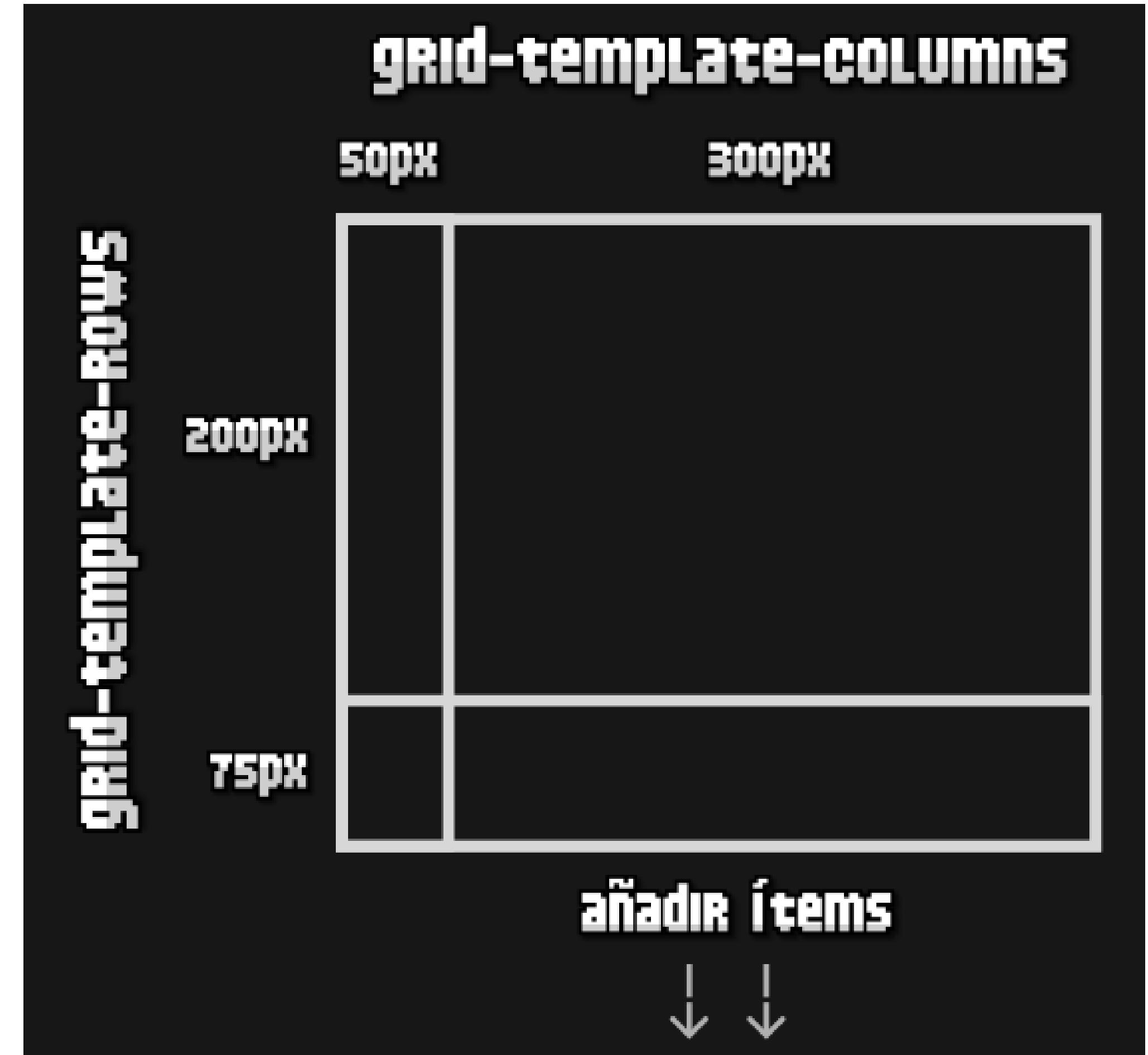
```
.grid {  
  display: grid;  
  grid-template-columns: 50px 300px;  
  grid-template-rows: 200px 75px;  
}
```

# Grid Template Columns

Con la propiedad `display: grid` definimos que queremos crear un grid, y mediante las propiedades `grid-template-columns` y `grid-template-rows` definimos los tamaños de las columnas y las filas del mismo. Esto significa que, a priori, tendríamos una cuadricula o grid de 4 celdas en total

# Grid Template Columns

```
.grid {  
  display: grid;  
  grid-template-columns: 50px 300px;  
  grid-template-rows: 200px 75px;  
}
```



# Grid Template Columns

Ahora, ten en cuenta que corre de nuestra cuenta vigilar que el número de elementos hijos en el grid es el que debería.

# Grid Template Columns

A medida que fueramos incluyendo más ítems en el grid, podríamos aumentar también el número de parámetros de la propiedad grid-template-columns y/o la propiedad grid-template-rows.

# Grid Template Columns

En caso de tener más ítems de lo que se indica en la propiedad, los ítems restantes se incluirían sin formato. De tener menos, simplemente se ocuparían los ítems implicados.

# Grid Template Columns

Unidades relativas y absolutas  
Propiedad shorthand

# Grid Template Columns

Demo

**Fr y Repeat**

## Fr y Repeat

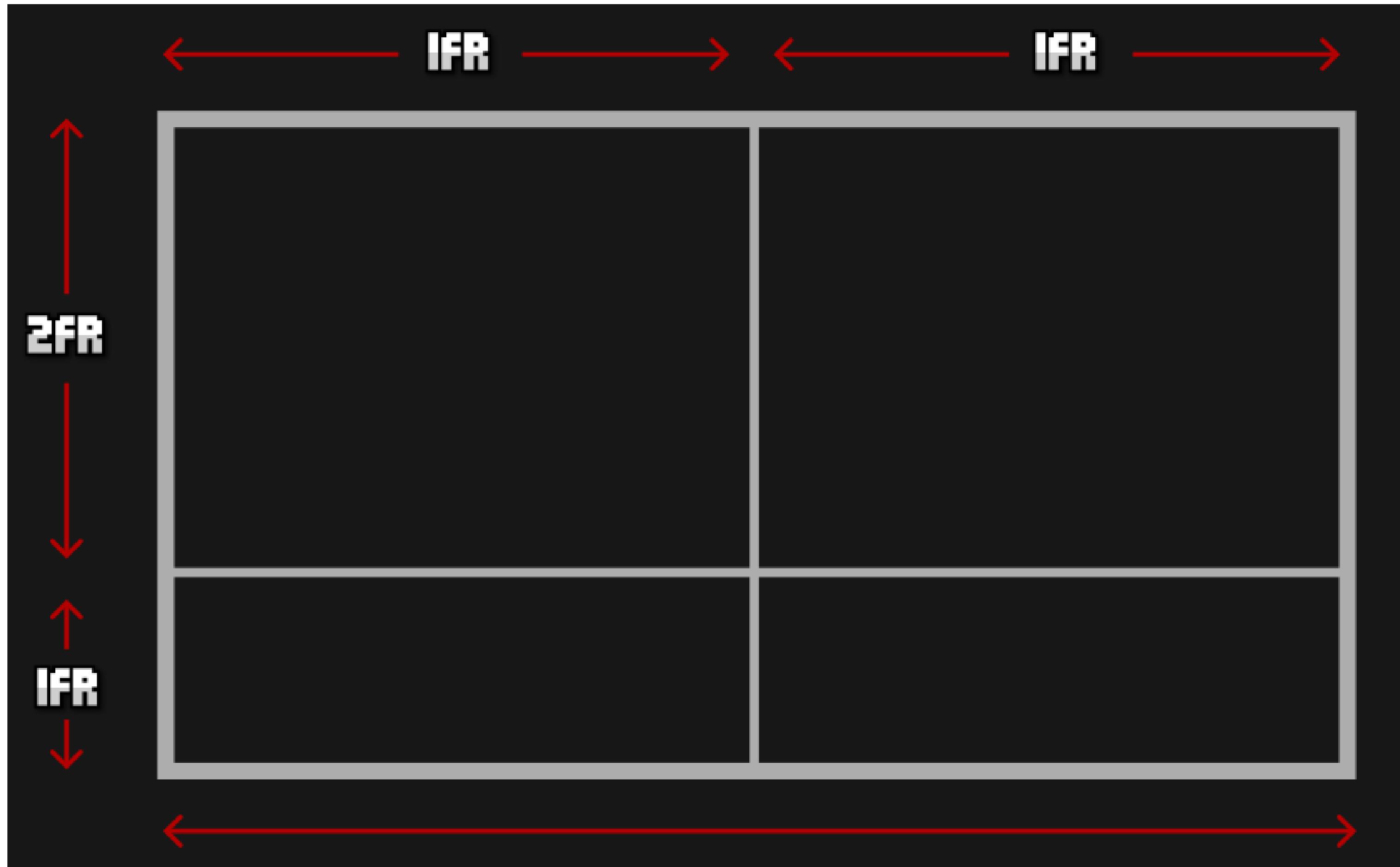
En el ejemplo anterior he utilizado píxels como unidades de las celdas de la cuadrícula, sin embargo, también podemos utilizar otras unidades: porcentajes, la palabra clave auto (que obtiene el tamaño restante) o la unidad especial de grid fr (fracción restante), que explicaremos a continuación.

# Fr y Repeat

- Dos columnas: Mismo tamaño de ancho para cada una.
- Dos filas: La primera fila ocupará el doble (2fr) que la segunda fila (1fr).

```
.grid {  
  display: grid;  
  grid-template-columns: 1fr 1fr;  
  grid-template-rows: 2fr 1fr;  
}
```

# Fr y Repeat



## Fr y Repeat

De esta forma, es muy fácil predecir el espacio que va a ocupar la cuadrícula, ya que sólo tenemos que sumar todas las unidades para saber el tamaño total, y comparar con cada columna o fila para saber como de grande o pequeña es respecto al total. Así tendremos un mejor control del espacio restante de la cuadrícula, y resultará más intuitivo calcularlo.

# Fr y Repeat

Se pueden combinar varias unidades diferentes, como por ejemplo píxeles (px), fracciones restantes (fr), porcentajes (%) y otras combinaciones similares.

# Fr y Repeat

Repeat

## Fr y Repeat

En algunos casos, en las propiedades grid-template-columns y grid-template-rows podemos necesitar indicar las mismas cantidades un número alto de veces, resultando repetitivo y molesto escribirlas varias veces. Se puede utilizar la función repeat() para indicar repetición de valores, señalando el número de veces que se repiten y el tamaño en cuestión.

# Fr y Repeat

```
.grid {  
  display: grid;  
  
  grid-template-columns: 100px repeat(4, 50px) 200px;  
  grid-template-rows: repeat(2, 1fr 2fr);  
  
  /* Equivalente a... */  
  grid-template-columns: 100px 50px 50px 50px 50px 200px;  
  grid-template-rows: 1fr 2fr 1fr 2fr;  
}
```

# Fr y Repeat

Asumiendo que tuvieramos un contenedor grid con 24 ítems hijos en el HTML, el ejemplo anterior crearía una cuadrícula con 6 columnas y 4 filas. Recuerda que en el caso de tener más ítems hijos, el patrón se seguiría repitiendo.

# Fr y Repeat

## Minmax

Si establecemos un rango, por ejemplo, `grid-template-columns: 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.

# Fr y Repeat

## Demo

```
<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>
```

**Gap**

# Gap

Por defecto, la cuadrícula tiene todas sus celdas pegadas a sus celdas contiguas. Aunque sería posible darle un margin a las celdas dentro del contenedor, existe una forma más apropiada: los huecos (gutters).

# Gap

Para especificar los huecos (espacio entre celdas) podemos utilizar las propiedades column-gap y/o row-gap. En ellas indicaremos el tamaño de dichos huecos

# Gap

Propiedad	Descripción
column-gap	Establece el <b>SIZE</b> de los huecos entre columnas ( <a href="#">líneas verticales</a> ).
row-gap	Establece el <b>SIZE</b> de los huecos entre filas ( <a href="#">líneas horizontales</a> ).

# Gap

```
.grid {  
  column-gap: 100px;  
  row-gap: 10px;  
}
```



# Order

# Order

Funciona exactamente igual que como funciona en flex. Es una propiedad mediante la cual podemos modificar y establecer un orden de los elementos mediante números que actuarán como «peso» del elemento

## Order

Por defecto, todos los elementos hijos de un contenedor flex tienen establecido un order por defecto al valor 0. Si indicamos una propiedad order con un valor numérico diferente, recolocará los ítems según dicho número, colocando antes los elementos con un número order más pequeño (incluso números negativos) y los elementos con números más altos después.

# Order

Demo

# Áreas

Los grids por áreas no son una alternativa a los grids explícitos (definidos por filas y columnas). Ambos pueden trabajar conjuntamente o por separado, según interese.

Mediante los Grids por área es posible indicar el nombre y posición concreta de cada área de una cuadrícula. Para ello utilizaremos la propiedad grid-template-areas en nuestro contenedor padre, donde debemos especificar el orden de las áreas en la cuadrícula.

Posteriormente, en cada ítem hijo, utilizamos la propiedad grid-area para indicar el nombre del área del que se trata y que el navegador pueda identificarlas:

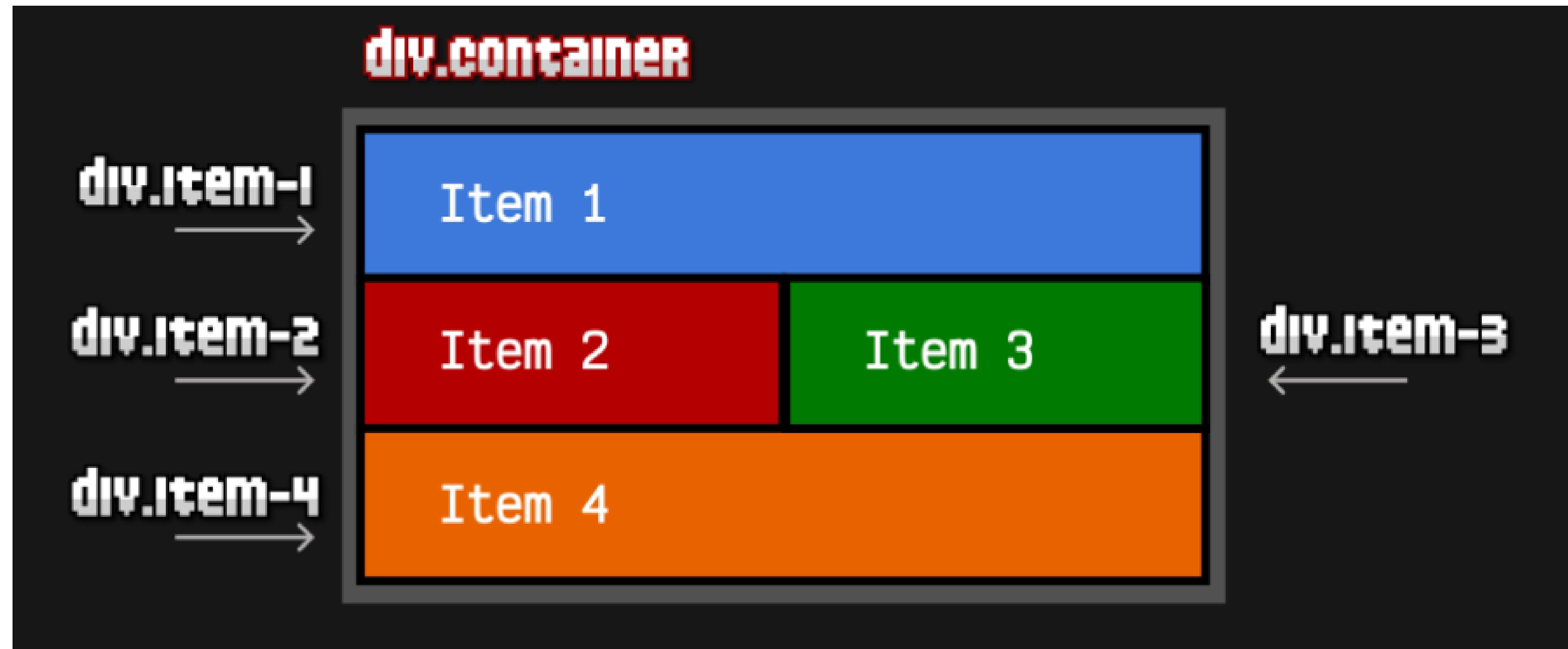
Propiedad	Descripción
grid-template-areas	Indica la disposición de las áreas en el grid. Cada texto entre comillas simboliza una fila.
grid-area	Indica el nombre del área. Se usa sobre ítems hijos del grid.

```
<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>

<style>
.container {
  display: grid;
  grid-template-areas: "head head"
                       "menu main"
                       "foot foot";
}

.item-1 { grid-area: head; background: blue; }
.item-2 { grid-area: menu; background: red; }
.item-3 { grid-area: main; background: green; }
.item-4 { grid-area: foot; background: orange; }
</style>
```

- El Item 1 sería nuestra cabecera (head), que ocuparía la primera fila (toda la parte superior).
- El Item 2 sería nuestro menú lateral (menu), que ocuparía el área izquierda del grid (debajo de la cabecera).
- El Item 3 sería nuestro contenido (main), que ocuparía el área derecha del grid (debajo de la cabecera).
- El Item 4 sería nuestro pie de cuadrícula (foot), que ocuparía la última fila (área inferior del grid).



Cada una de estas filas se definen como un donde indicaremos el nombre de un área que posteriormente definiremos en nuestro código CSS.

Cada fila puede tener ninguna o varias áreas que habría que separar por espacio. A continuación veremos algunos ejemplos de los valores que podemos indicar en esta propiedad y su significado

Valores	Descripción
<code>none</code>	Indica que no se creará ninguna plantilla de áreas.
<code>"head"</code>	Indica que se creará una fila de una columna con el área head.
<code>"head menu"</code>	Indica que se creará una fila de 2 columnas con el área head en una y el área menu en otra.
<code>"head head"</code>	Indica que se creará una fila de 2 columnas con el área head ocupando ambas.
<code>".."</code>	Indica que se colocará una celda sin nombre ( <a href="#">nula</a> ) en esta posición.

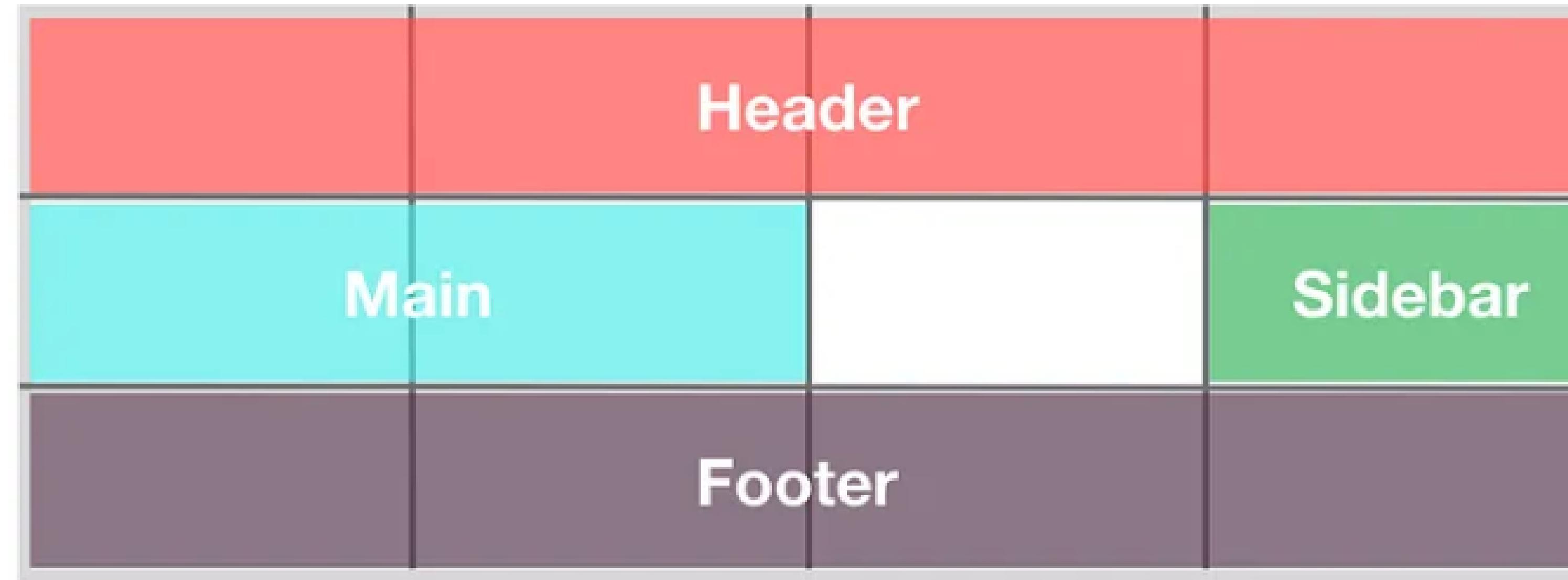
# Demo

# Ejemplos

1	2	3
4	5	6
7		

1	2
3	4
5	6

50px	100px



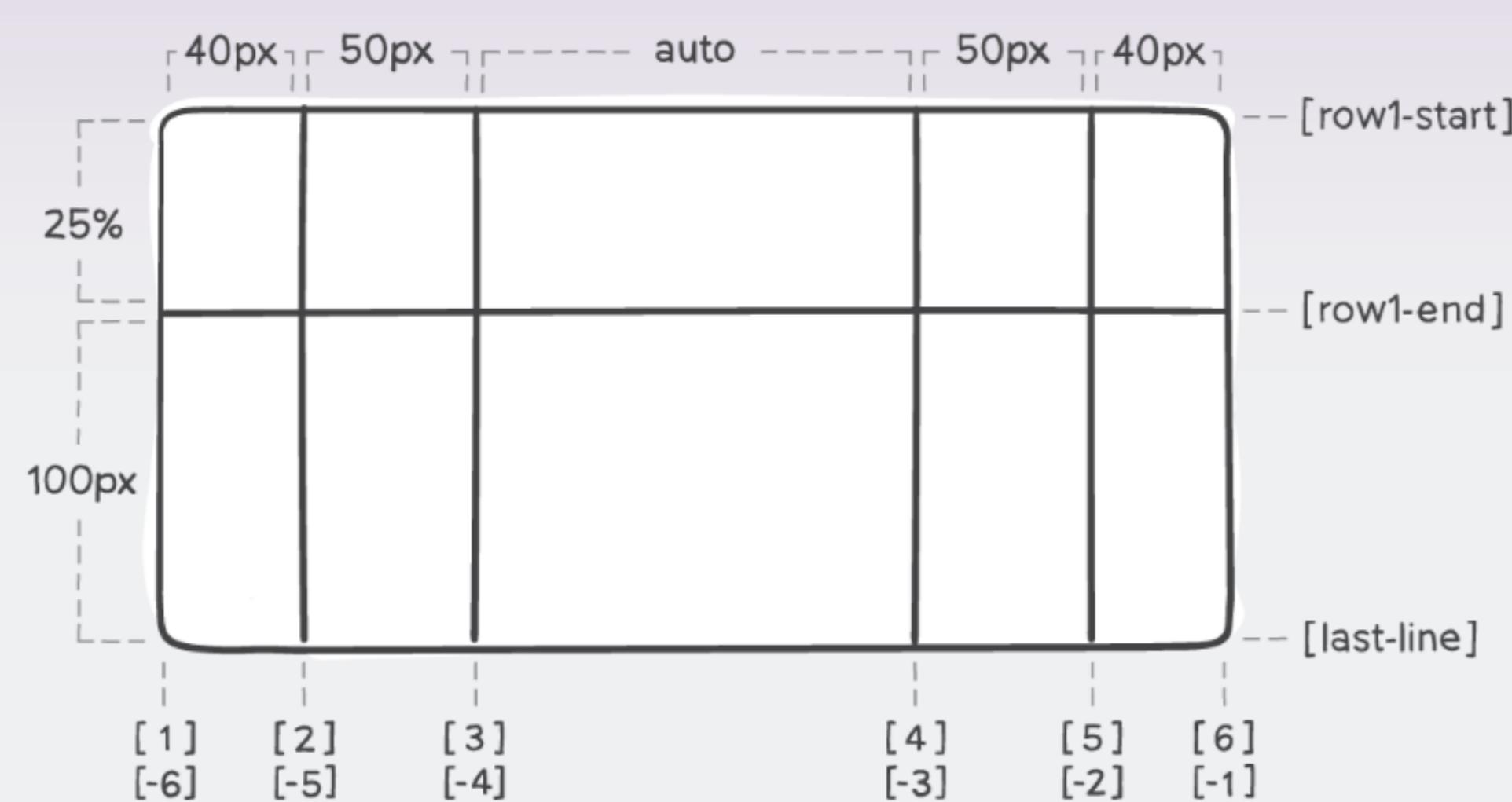
Demo

# CSS Grid

a guide from

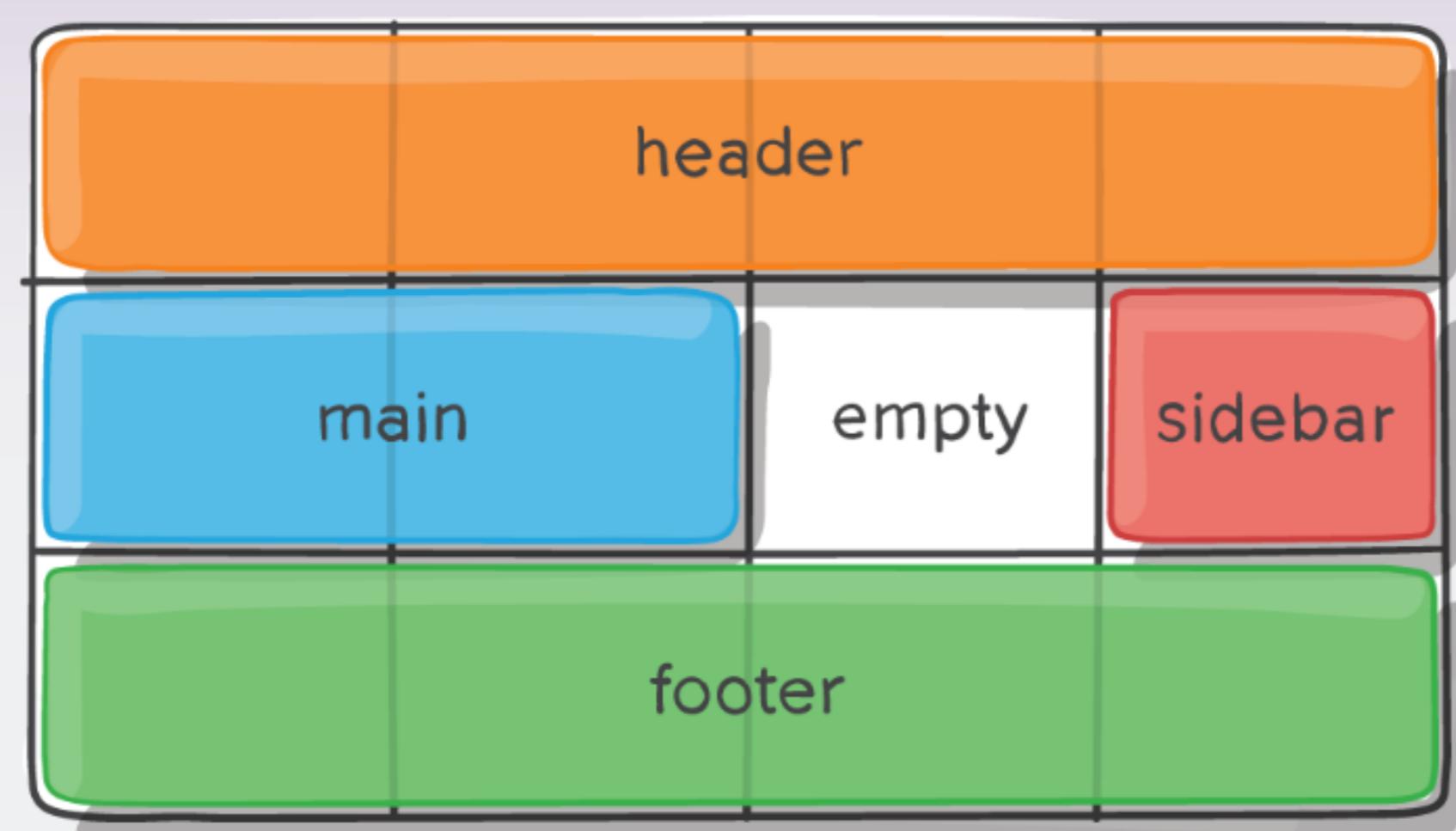


## grid-template-columns grid-template-rows



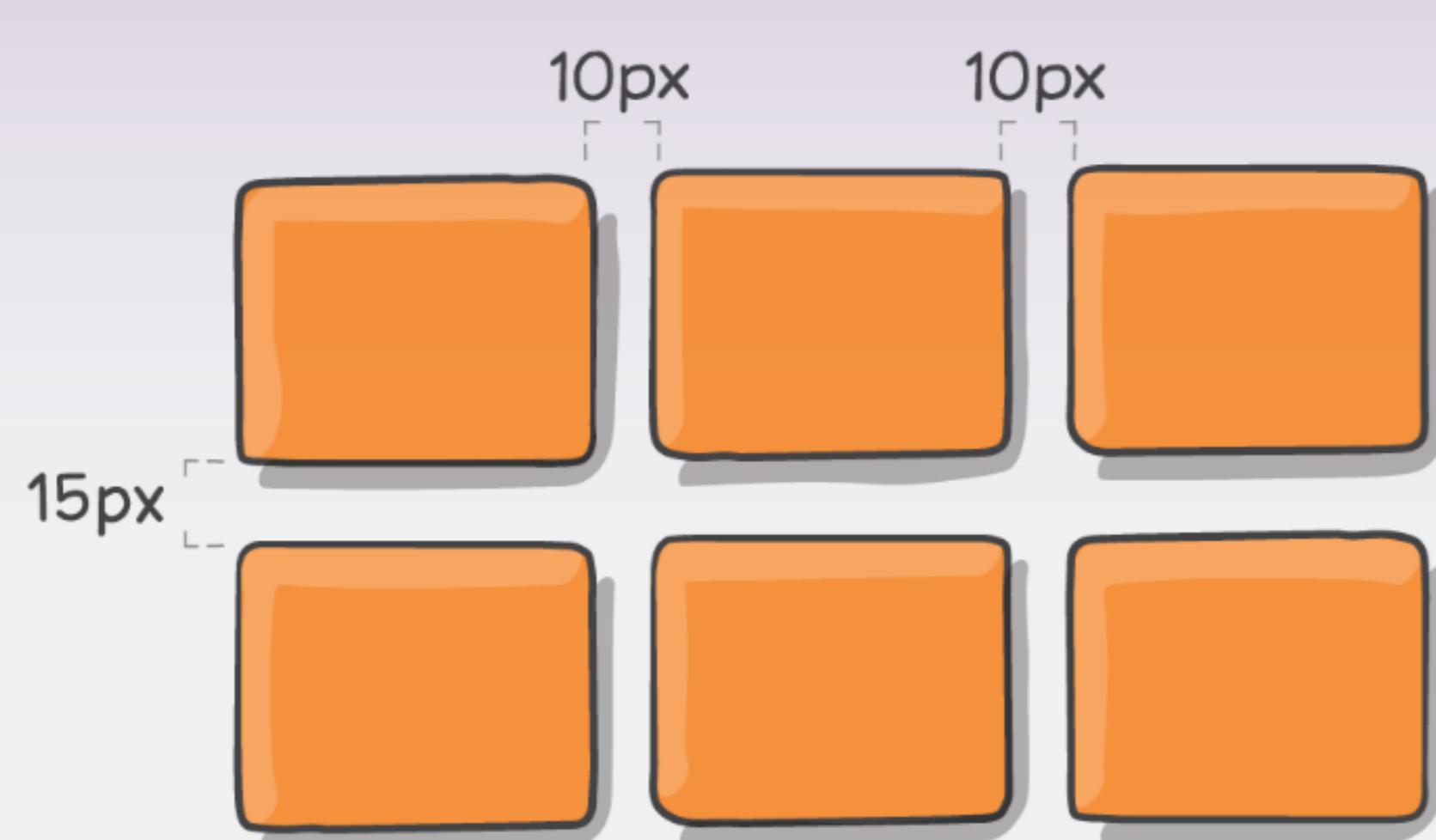
```
.container {  
  grid-template-columns: <value> | <name>;  
  grid-template-rows: <value> | <name>;  
}
```

## grid-template-areas



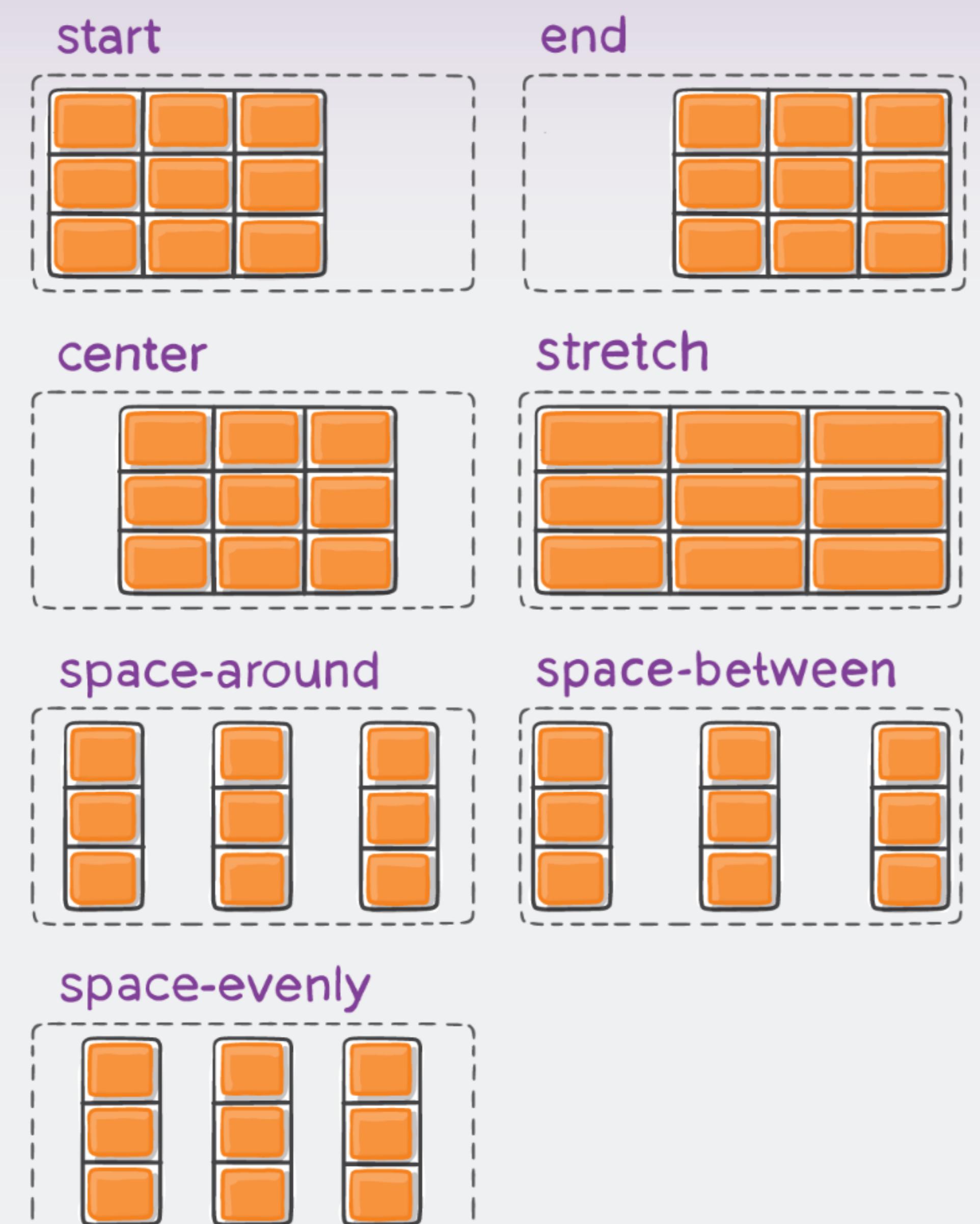
```
.container {  
  grid-template-areas: "<name> | . | none";  
}
```

## column-gap, row-gap, gap



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

## justify-content



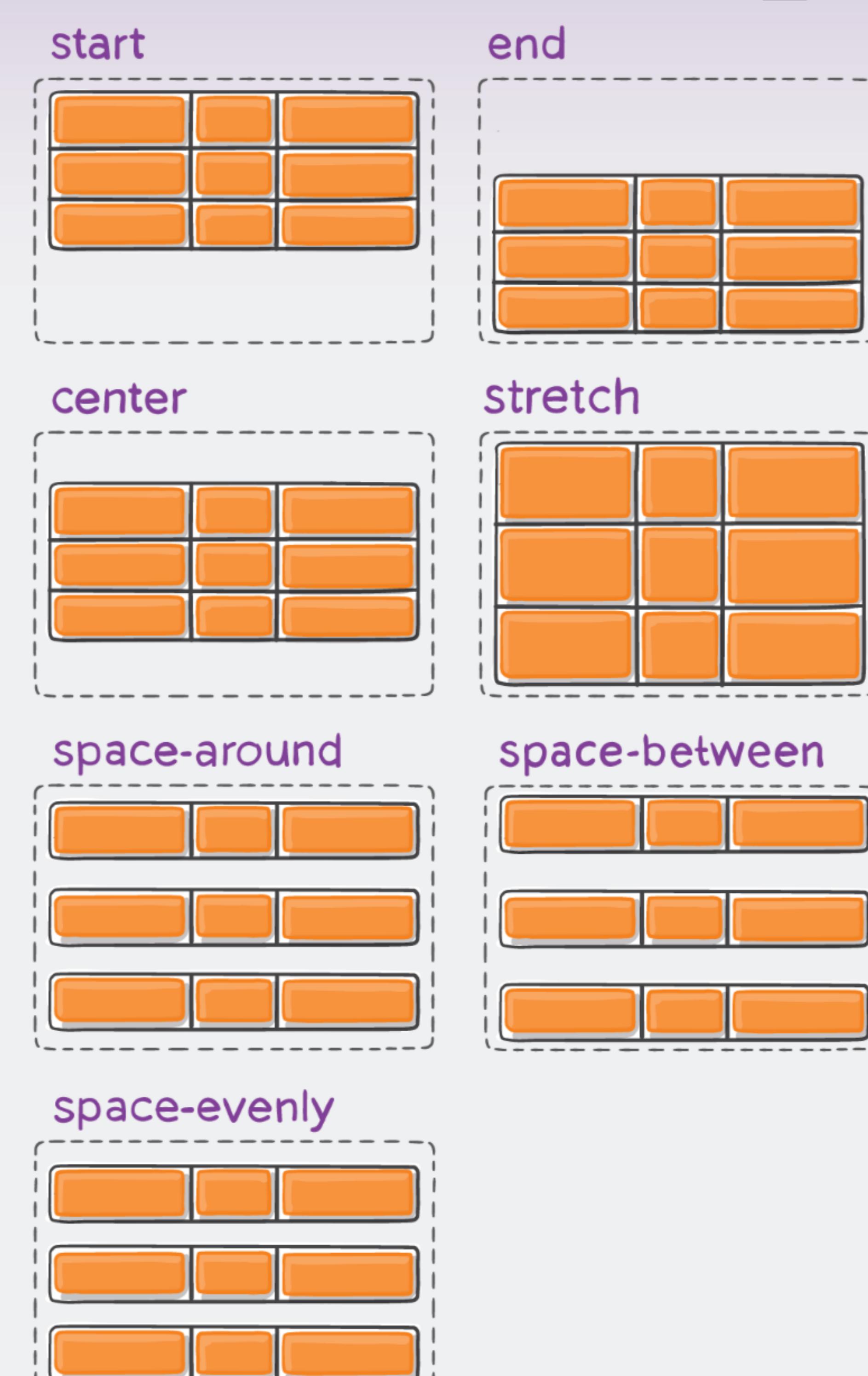
```
.container {  
  justify-content: start | end | center |  
  stretch | space-around | space-between |  
  space-evenly;  
}
```



# CSS Grid

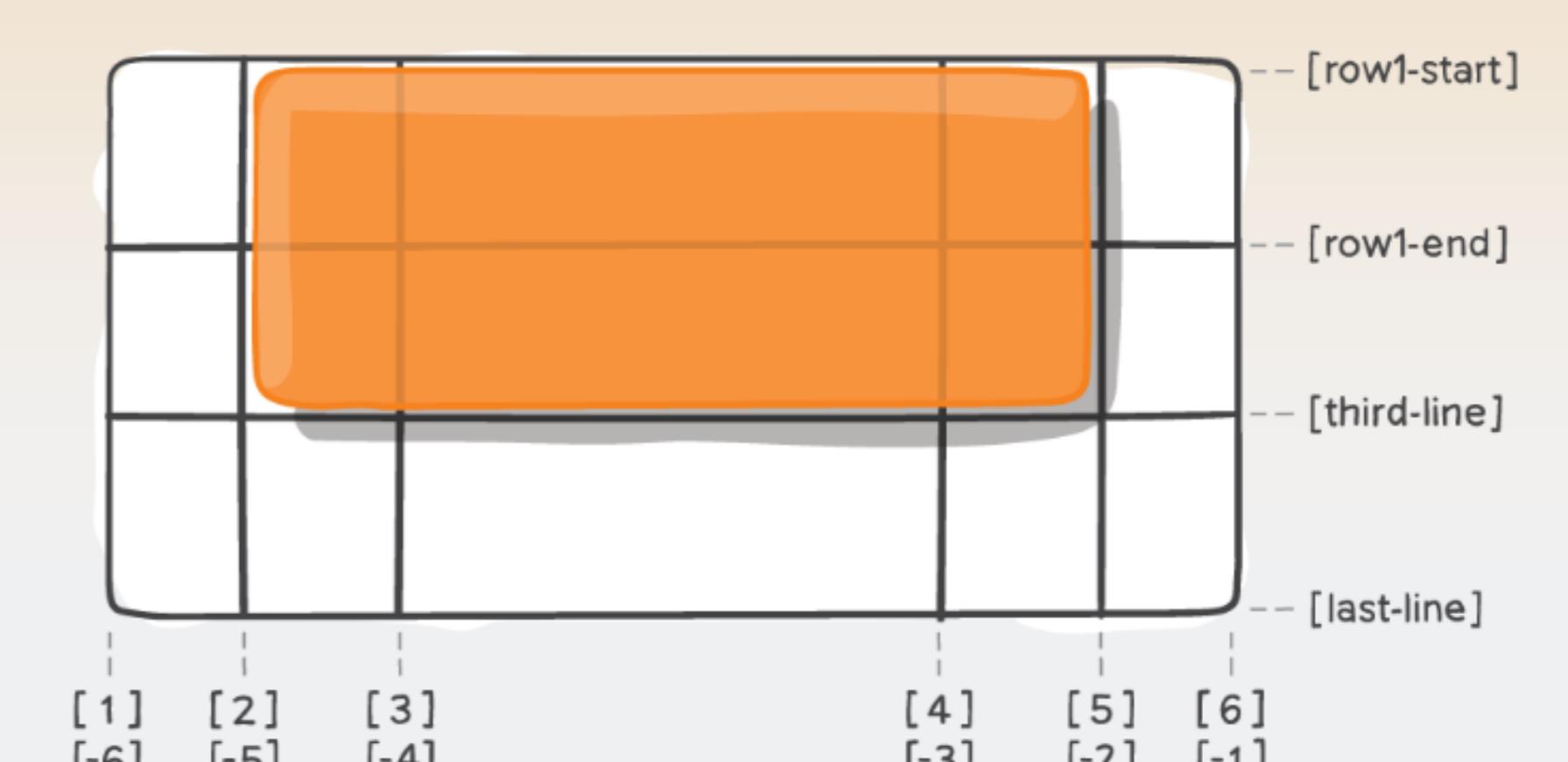
```
.container {  
  display: grid; /* or inline-grid */  
}
```

## align-content



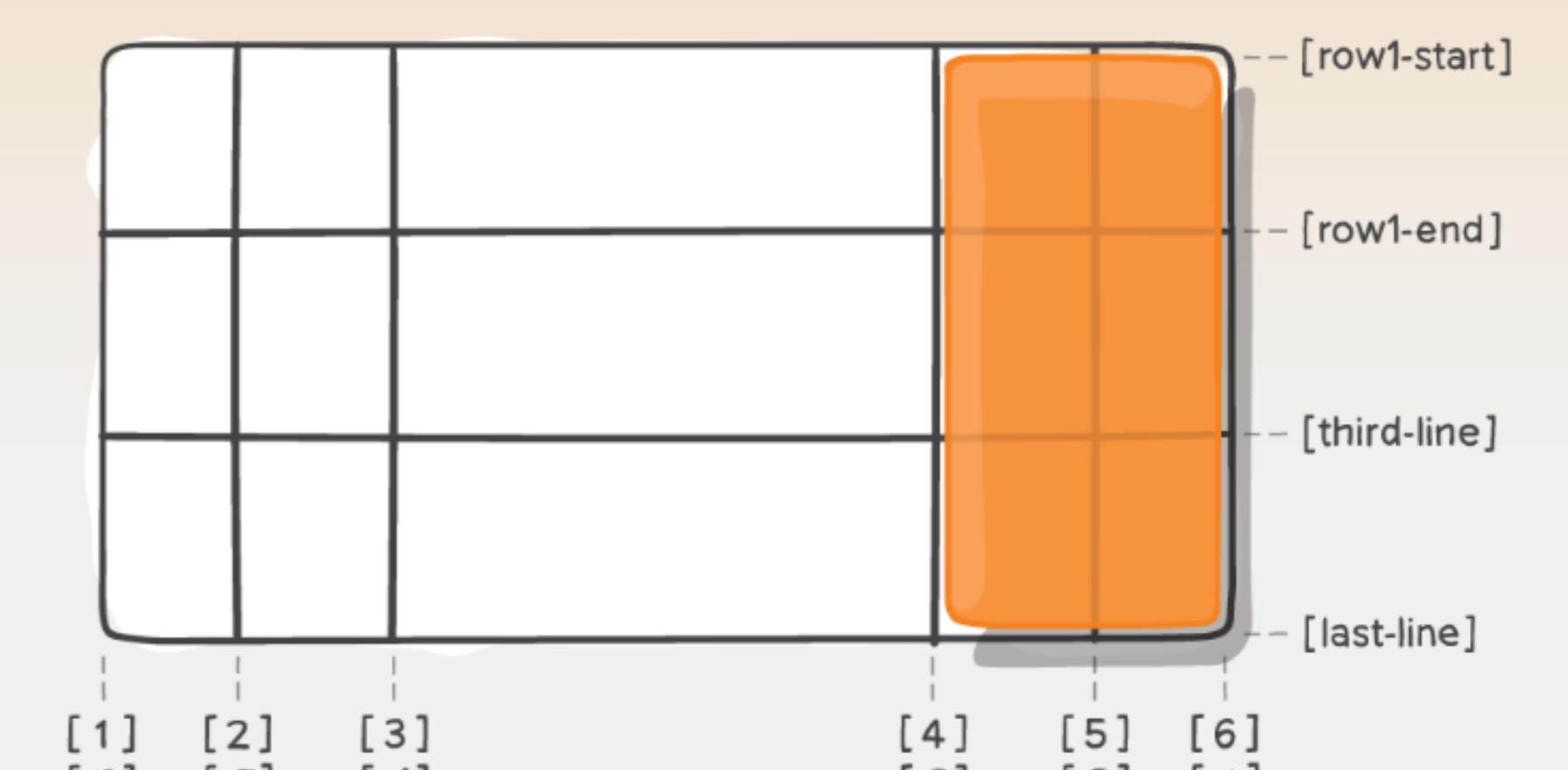
```
.container {  
  align-content: start | end | center |  
  stretch | space-around | space-between |  
  space-evenly;  
}
```

## grid-column, grid-row



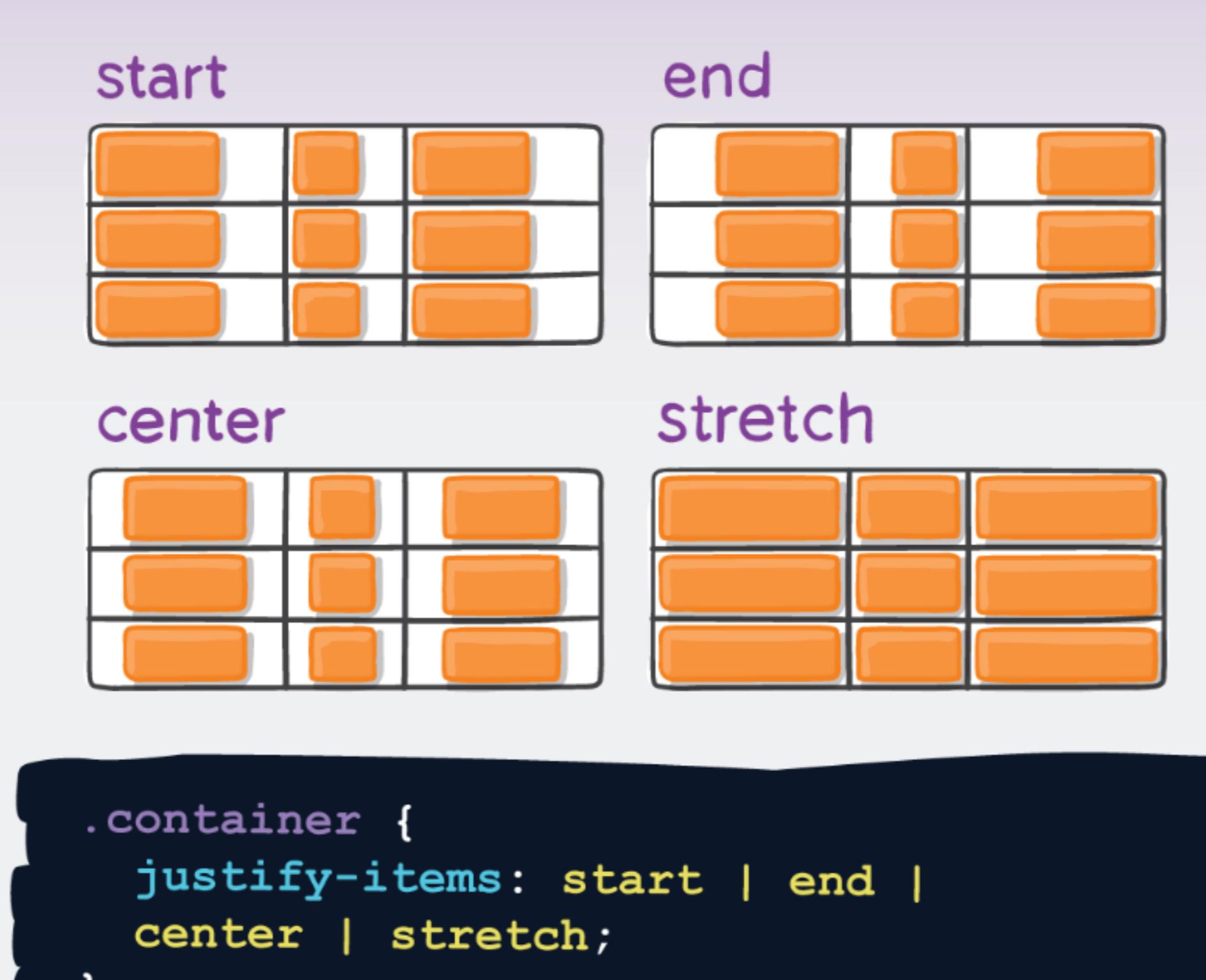
```
.item {  
  grid-column: <start-line> / <end-line> |  
  <start-line> / span <value>;  
  grid-row: <start-line> / <end-line> |  
  <start-line> / span <value>;  
}
```

## grid-area



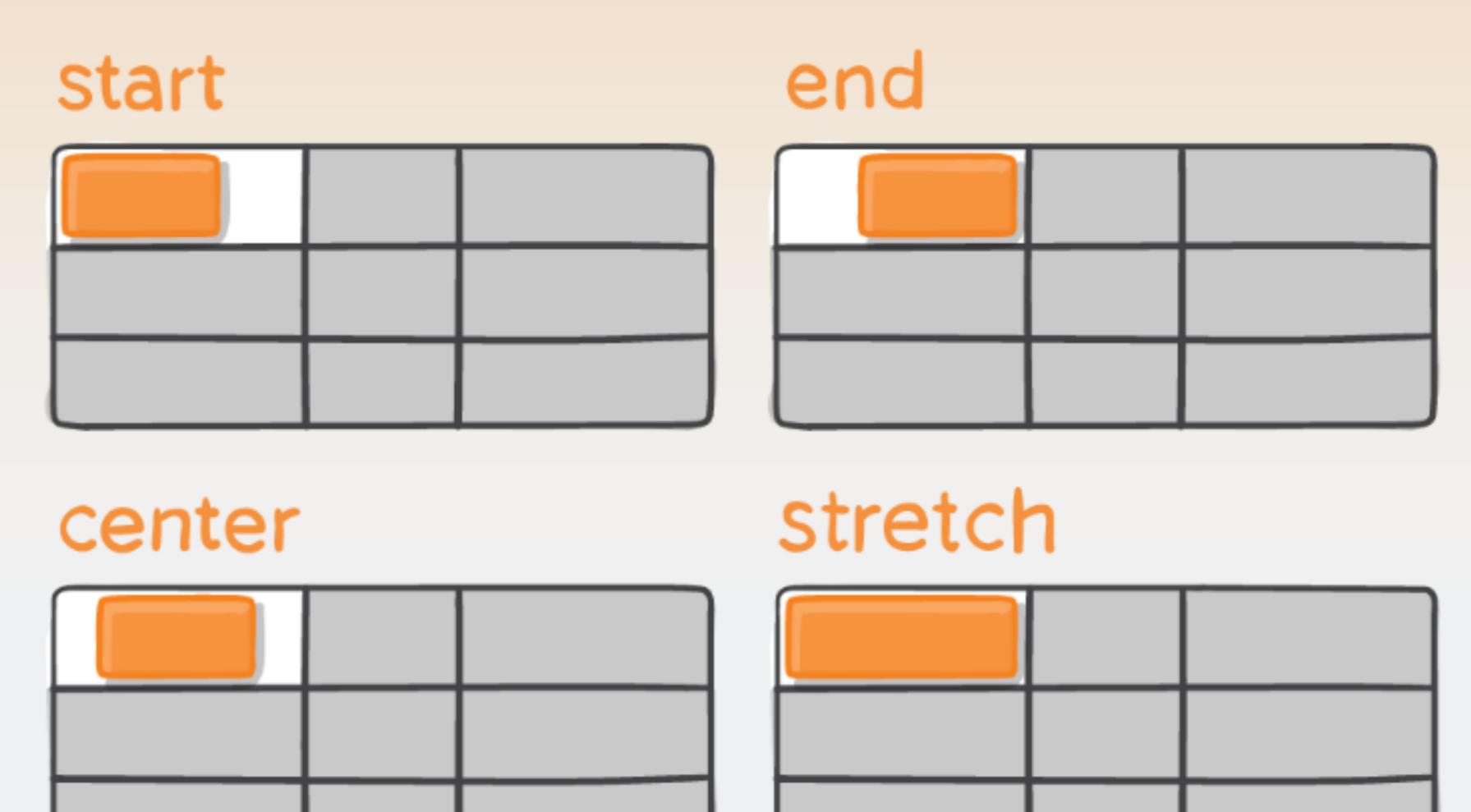
```
.item {  
  grid-area: <row-start> / <column-start>  
  / <row-end> / <column-end> | <name>;  
}
```

## justify-items



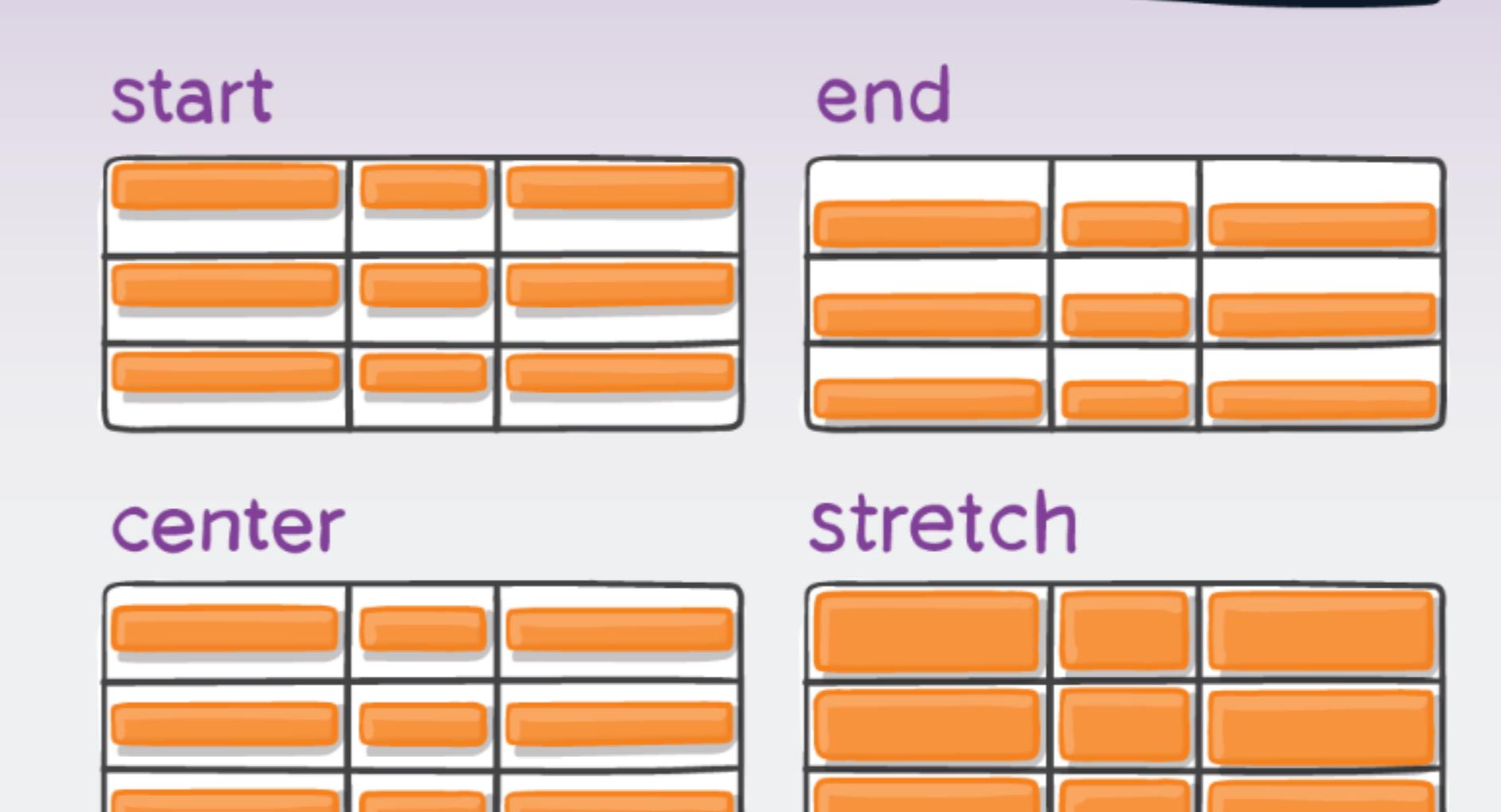
```
.container {  
  justify-items: start | end | center |  
  stretch;  
}
```

## justify-self



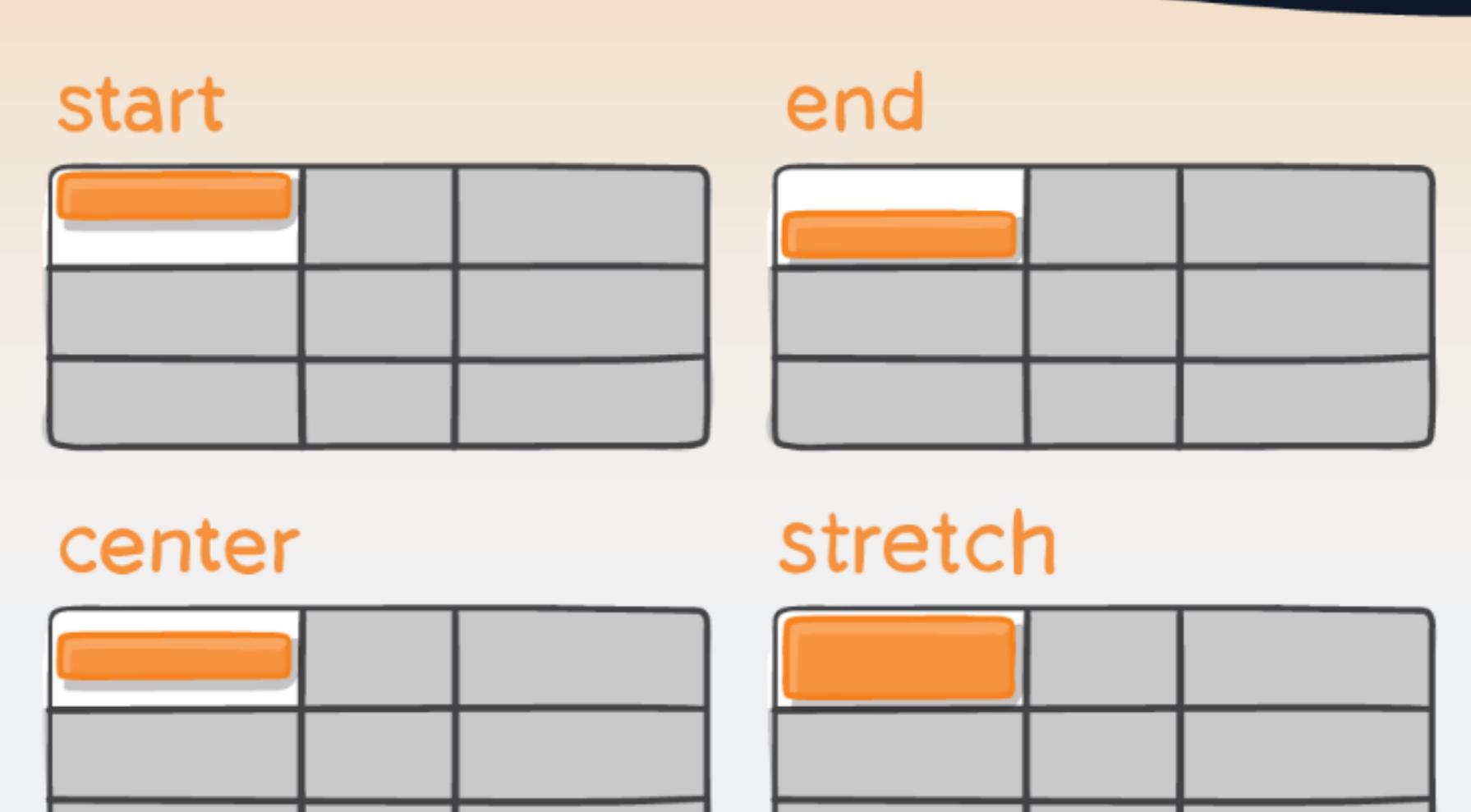
```
.item {  
  justify-self: start | end | center |  
  stretch;  
}
```

## 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**