

CSS Grid

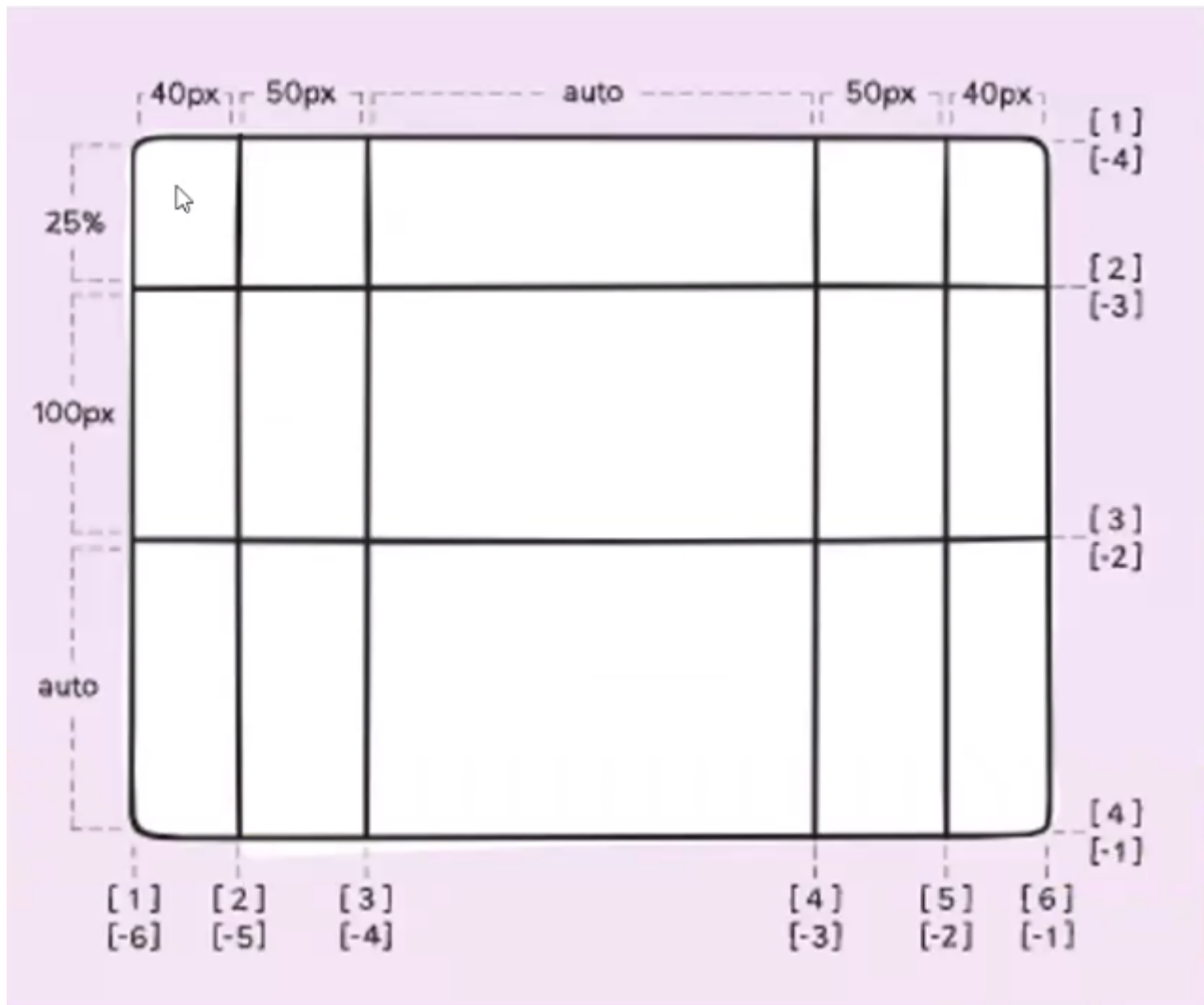


Grid CSS

Grid CSS presenta otro estándar para maquetar, en ciertos aspectos se asemeja a FlexBox, pero trabaja de otra manera.

Sobre que elegir para maquetar si *FlexBox* o *_CSS Grid*: a lo largo de este módulo iremos despejando dudas sobre cuando merece la pena usar uno u otro.

CSS Grid nos va a ayudar a maquetar elementos de nuestra aplicación que tengan forma de cuadrícula (grilla).



CSS Grid nos permite:

- Darle un alto a las filas y columnas (Sean porcentajes o pixeles o auto).
- También trabaja con fracciones para las medidas.

Introducción

CSS Grid ofrece un sistema de layout en filas y columnas. Facilita el diseño web de páginas sin tener que usar *floats* y posicionamiento, es muy potente a la hora de hacer maquetación responsiva.

El mínimo para crear un layout CSS Grid.

El HTML

```
<div class="my-grid-container">
  <div class="item">...</div>
  <div class="item">...</div>
  <div class="item">...</div>
  ...
  <div class="item">...</div>
</div>
```

```
.my-grid-container {
  display: grid | inline-grid;
}
```

Lo más importante aquí es el estilo que definimos, en el que le indicamos el *display* y tenemos dos opciones:

- *grid*: ocupa todo el espacio que tenga a la derecha.
- *inline-grid*: Sólo el espacio necesario.

Vamos a aprender a base de ejemplo, partimos de un [codesandbox](#) vacío, y elegimos la plantilla *Vanilla JS*.

Esta vez vamos a tener el fichero *index.html* y vamos a usar el fichero que trae por defecto *src/styles.css*

./index.html

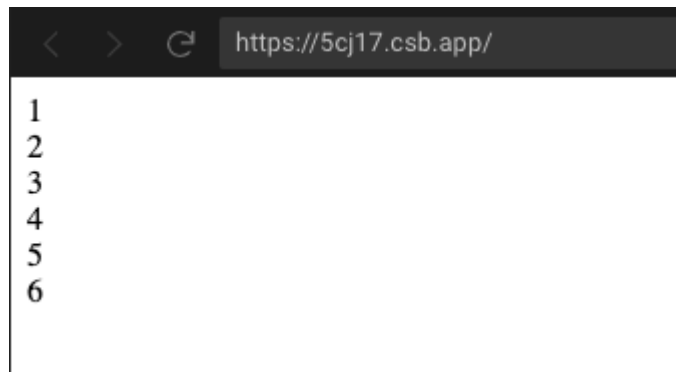
```
<head>
  <title>Parcel Sandbox</title>
  <meta charset="UTF-8" />
  + <link rel="stylesheet" type="text/css" href="./src/styles.css" />
</head>
<body>
+ <div class="grid-container">
+   <div class="item">1</div>
+   <div class="item">2</div>
+   <div class="item">3</div>
+   <div class="item">4</div>
+   <div class="item">5</div>
+   <div class="item">6</div>
+ </div>
</body>
```

Y en la parte de estilos:

./styles.css

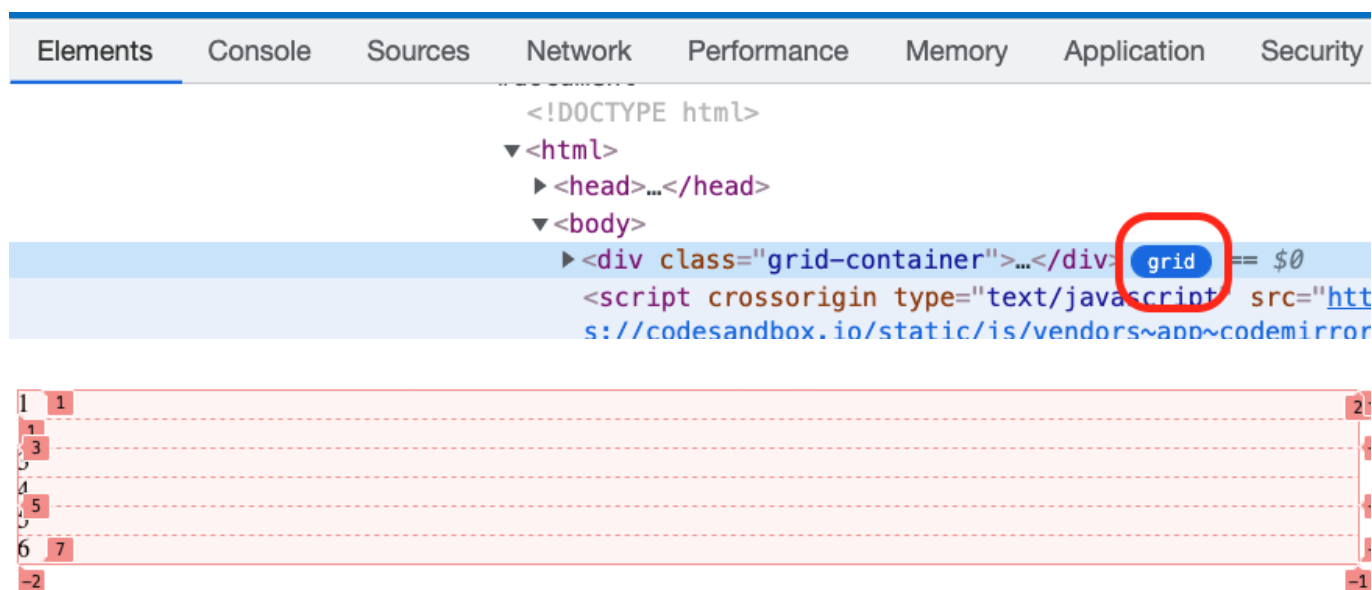
```
.grid-container {  
  display: grid;  
}
```

Vamos a ver que resultado da esto:



Bueno nada del otro jueves... ¿Qué pasa? que no estamos configurando ese Grid, no le estamos diciendo cuantas cuadrículas queremos, que disposición...

Si abrimos las dev tools de Chrome podemos ver que este navegador nos da soporte para poder configurar el Grid visualmente:



Por defecto tenemos una columna por seis filas.

Vamos a ponernos manos a la obra con la configuración:

columns & rows

De cara a configurar cuantas filas y columnas queremos tenemos dos propiedades:

- `grid-template-columns`
- `grid-template-rows`

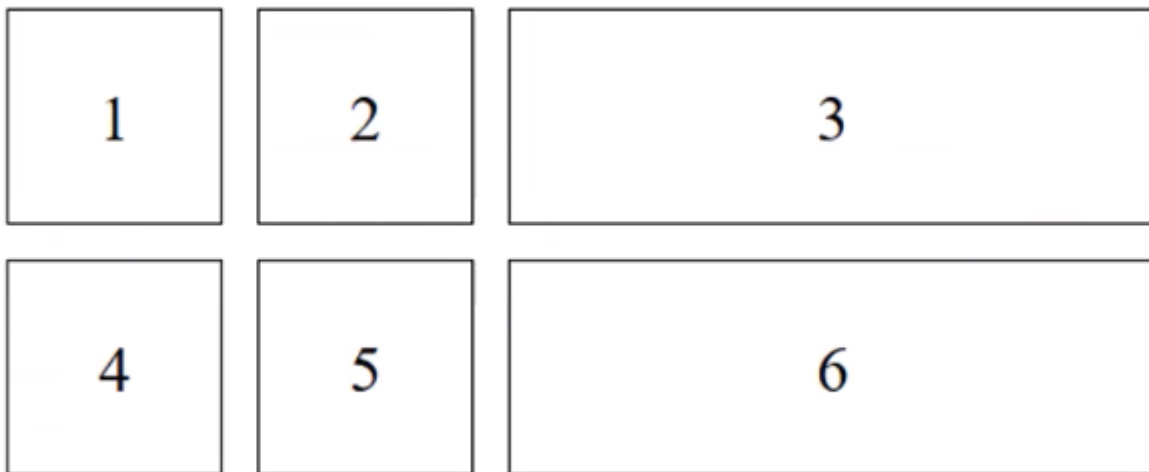
Un ejemplo de como funciona esto:

```
.my-grid-container {  
  display: grid;  
  grid-template-columns: 100px 100px 300px;  
  grid-template-rows: 100px 100px;  
}
```

Esto da como resultado:

- 3 columnas, dos de 100 pixeles y una de 300.
- 3 filas, dos de 100 pixeles

Ew decir configuramos una 3x2



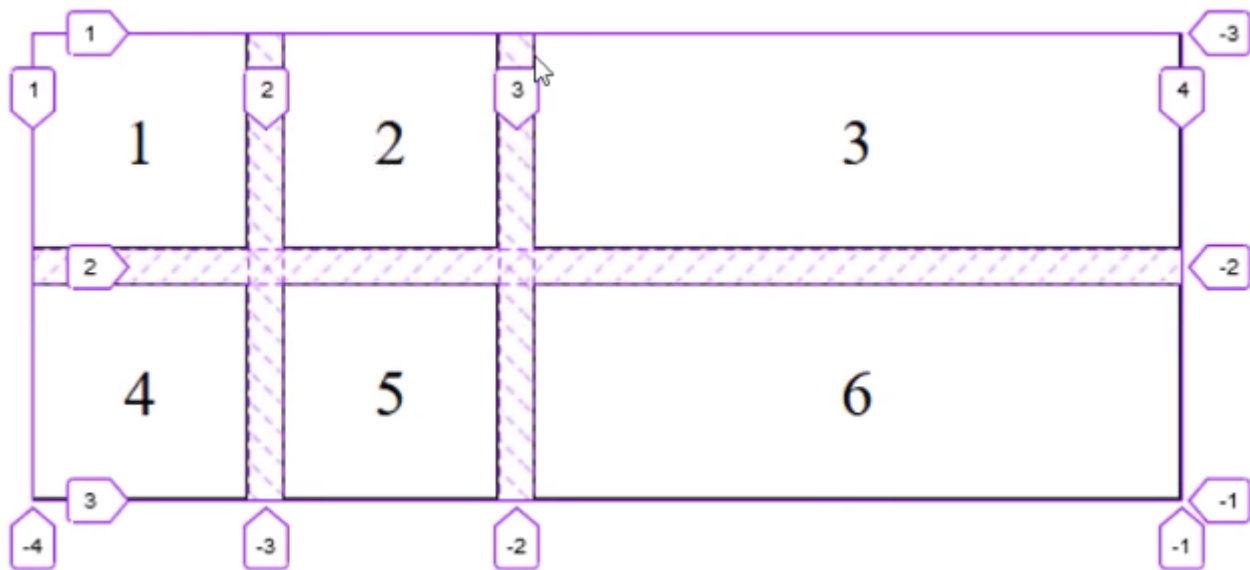
Si metemos esto en el ejemplo podemos ver que ya tenemos un resultado más elaborado



¿Y si tengo 7 elementos? Al no estar configurado, crea una nueva fila pero no le pone alto ni nada.

Para identificar donde esta cada elemento, podríamos pensar en las celdas de cada fila y columna (por ejemplo el elemento 2, esta en la fila 1, columna 2), este sistema no se usa en css-grid, lo que se usan

son las *grid-lines*



Es decir ahora decimos, que el element 2 va de la línea de columna 2 a la 3, y la línea de fila 1 a la 2 ¿ Y para qué complicarnos la vida? Porque después vamos a hacer que ciertos elementos ocupen varias casillas, etc... es mucho más fácil si trabajamos con esta aproximación.

Otro tema a tener en cuenta es que podemos enumerar, por ejemplo, las filas de la primera a la última en positivo (1,2,3), y también las podemos enumerar en negativo (la de más abajo es la -1, después la -2 y la primera la -3), lo mismo para las columnas.

De esta manera, podemos decir cosas como: quiero que un elemento empiece en la fila 1 y termine en la -1, y así nos aseguramos que ocupe todo el espacio de filas sin tener que preocuparnos de cual es el índice de la última línea de fila.

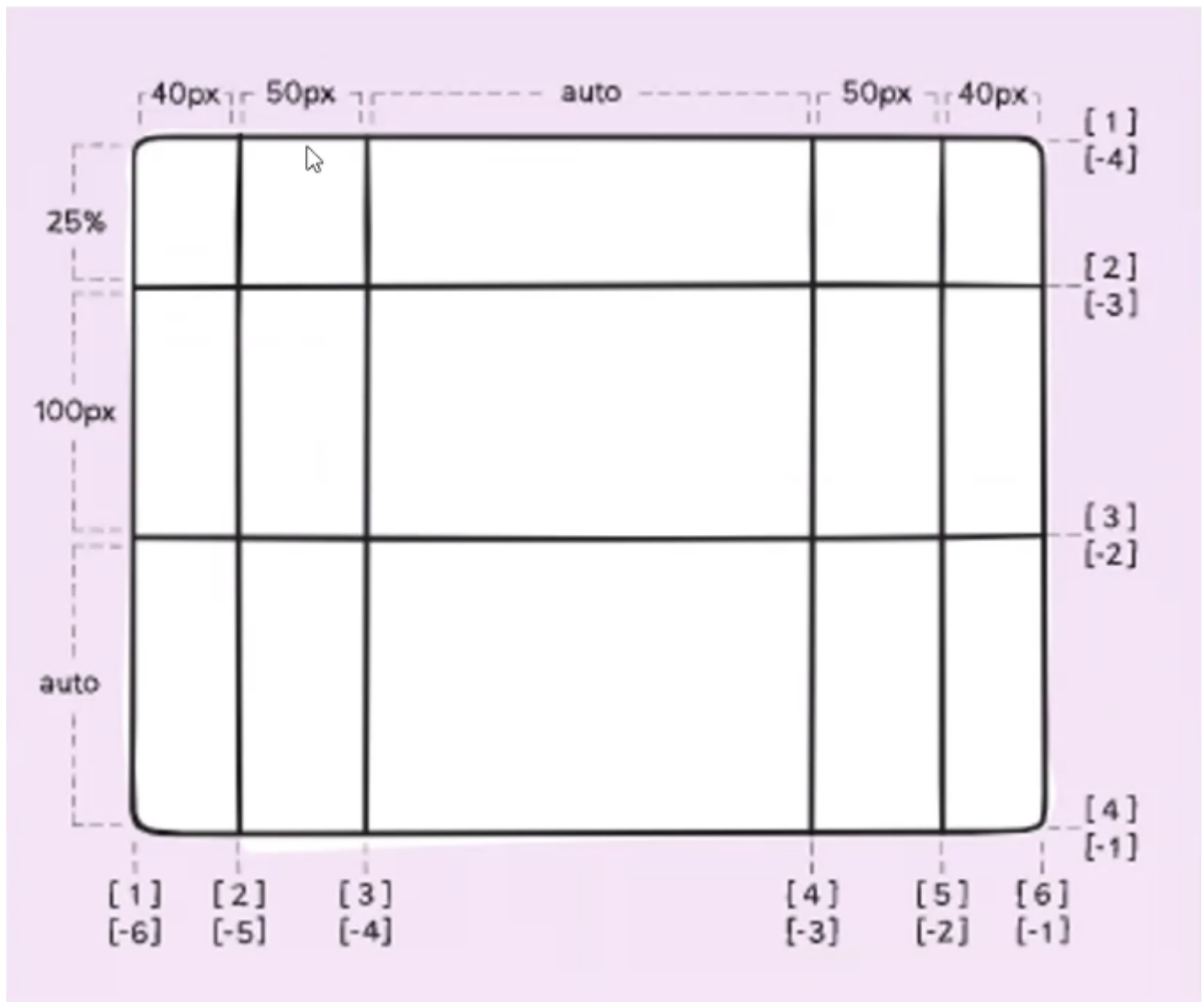
Porcentajes y Auto

Otro tema interesante es poder definir los tamaños de las filas y columnas con porcentajes, o pidiendo que tome el espacio necesario para mostrar el contenido.

Veamos el código:

- En las columnas le indicamos que queremos que la tercera columna ocupe el espacio necesario y el resto que tengan un tamaño fijo en píxeles.
- En las filas, en la primera le indicamos que queremos un 25% del espacio total disponible, en la segunda fijamos a 100 píxeles, y en la última que obtenga el espacio necesario.

```
.my-grid-container: {  
  grid-template-columns: 40px 50px auto 50px 40px;  
  grid-template-rows: 25% 100px auto;  
}
```



Otra opción interesante es usar *fr*, en este caso le estamos indicando que trabaje con una *fraccion*, vamos a montar un ejemplo:

`./index.html`

```
<body>
+   <div class="grid-container">
+       <div class="item">1</div>
+       <div class="item">2</div>
+       <div class="item">3</div>
+       <div class="item">4</div>
+       <div class="item">5</div>
+       <div class="item">6</div>
+   </div>
</body>
```

`./src/styles.css`

```
.grid-container {
    display: grid;
```

```

border: 1px solid black;
grid-template-rows: 100px 100px;
grid-template-columns: 100px 100px 100px;
}

.item {
border: 1px solid black;
background-color: khaki;
display: flex;
justify-content: center;
width: 100%;
}

```

Fr es una fracción, si solo ponemos una estamos diciendo que ocupe todo el espacio que quede libre:

```

.grid-container {
display: grid;
border: 1px solid black;
grid-template-rows: 100px 100px;
- grid-template-columns: 100px 100px 100px;
+ grid-template-columns: 100px 1fr 100px;
}

```

Si hacemos lo mismo en la tercera columna se repartiran el espacio libre entre las dos:

```

.grid-container {
display: grid;
border: 1px solid black;
grid-template-rows: 100px 100px;
- grid-template-columns: 100px 1fr 100px;
+ grid-template-columns: 100px 1fr 1fr;
}

```

También podemos decirle usar *fr* para todas y a la de en medio darle más peso usando *2fr*:

```

.grid-container {
display: grid;
border: 1px solid black;
grid-template-rows: 100px 100px;
- grid-template-columns: 100px 1fr 1fr;
+ grid-template-columns: 1fr 2fr 1fr;
}

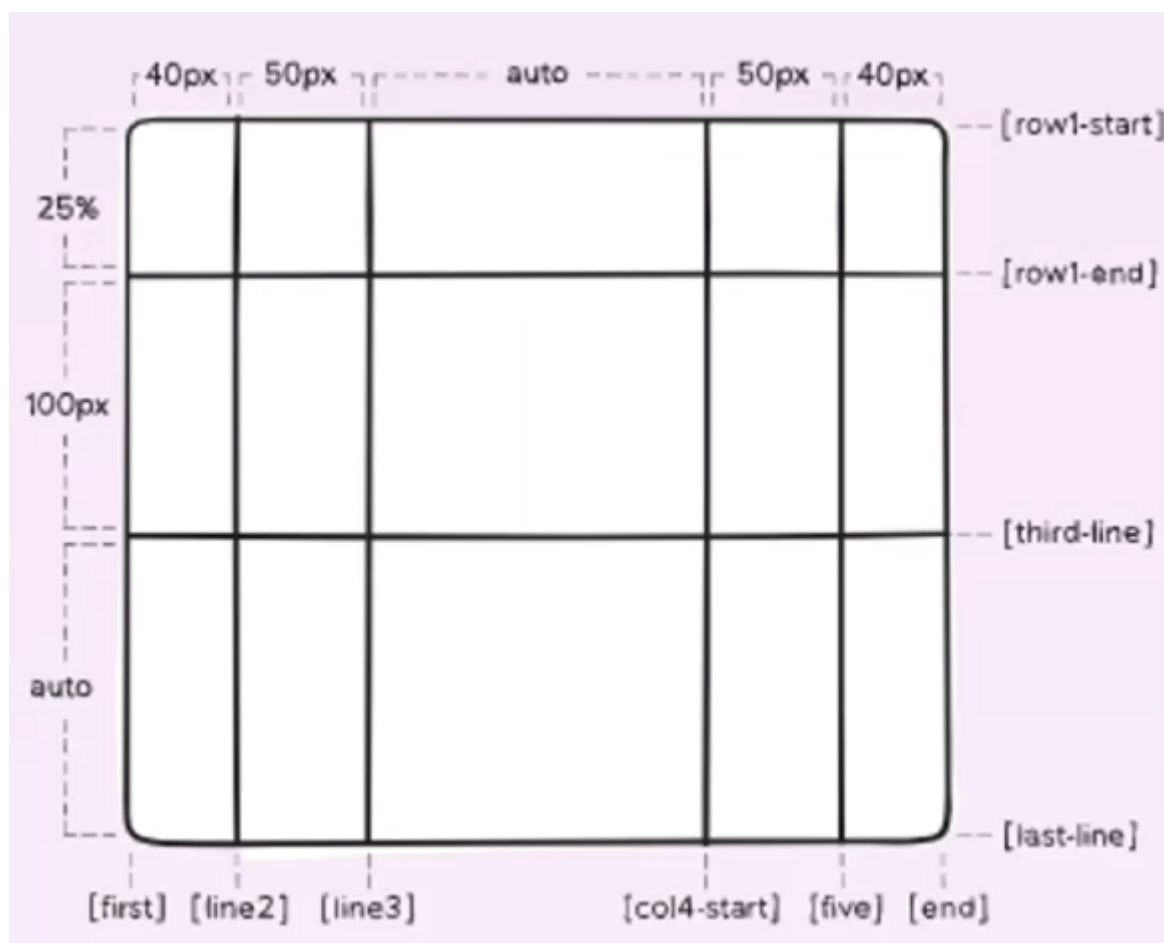
```

Para saber más: <https://www.programandoamedianoche.com/2019/05/guia-completa-para-aprender-a-utilizar-css-grid-layout/>

Nombre de líneas

A los separadores de líneas, que normalmente los nombramos como: 1, 2,3 (o -3, -2 -1), podemos también darles nombres, así nos puede ser más cómodo de manejar, esto lo podemos hacer indicando entre corchetes un nombre, antes o después de cada ancho de columna o de fila, por ejemplo:

```
.my-grid-container {  
  grid-template-columns: [first]40px[line2]50px[line3]auto[col4-start]50px[five]40px[end];  
  grid-template-rows: [row1-start]25%[row1-end]100px[third-line]auto[last-line];  
}
```



Tampoco es buena idea añadir nombres de líneas a todo lo que pillemos, porque puede costar leerlo, mejor usarlo en puntos clave.

Otra cosa que podemos hacer es darle más de un nombre de alias, es decir entre los corchetes dejamos un espacio y añadimos otro nombre más, ... esto puede ser bastante lioso, por ejemplo:

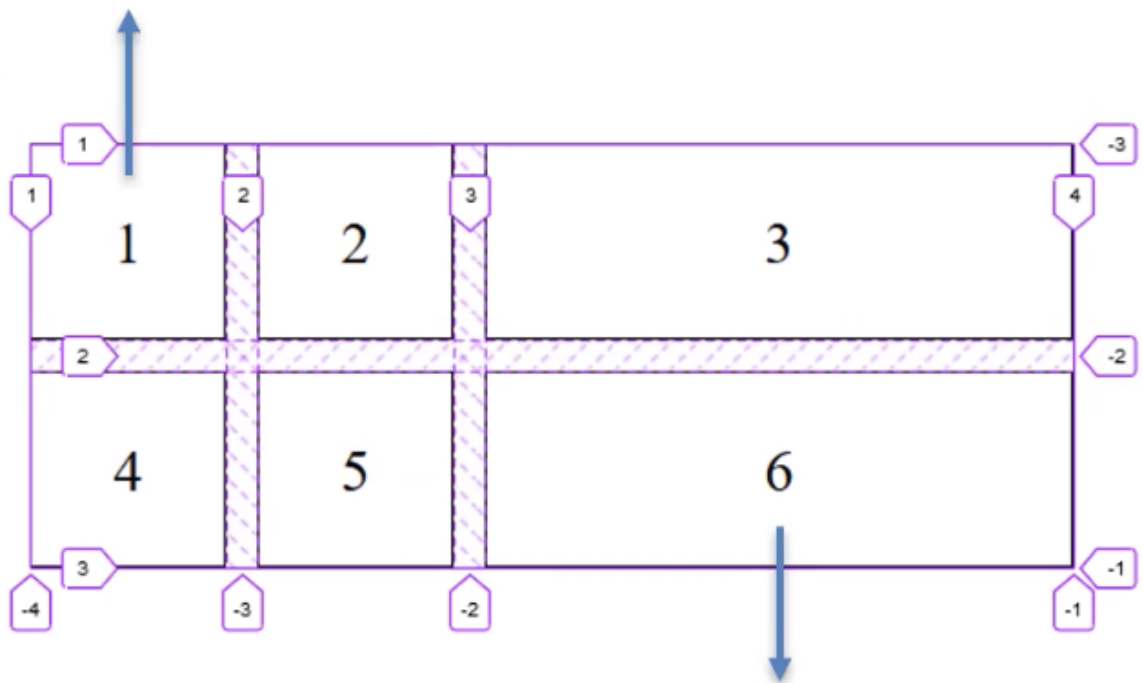
```
.container: {  
  grid-template-rows: 25%[row1-end row2-start] 25%;  
}
```

Colocando items

Vamos a indicarle a los items donde se deben de colocar, ahora en cada item podemos decirle:

- En que column start y column end cae.
- En que row start y end cae.

```
.item-1 {
  grid-column-start: 1;
  grid-column-end: 2;
  grid-row-start: 1;
  grid-row-end: 2;
}
```



```
.item-6 {
  grid-column-start: 3;
  grid-column-end: 4;
  grid-row-start: 2;
  grid-row-end: 3;
}
```

Volvemos a nuestro codesandbox:

En el HTML vamos a darle unos ids a los div:

./index.html

```
<body>
  <div class="grid-container">
-    <div class="item">1</div>
+    <div class="item" id="item1">1</div>
    <div class="item">2</div>
```

Y en el CSS vamos a indicarle un estilo para esa id:

./src/styles.css

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 300px;
  grid-template-rows: 100px 100px;
}

+ .item {
+   border: 1px solid black;
+ }

+ item1 {
+   grid-column-start: 1;
+   grid-column-end: 2;
+   grid-row-start: 1;
+   grid-row-end: 2;
+ }
```

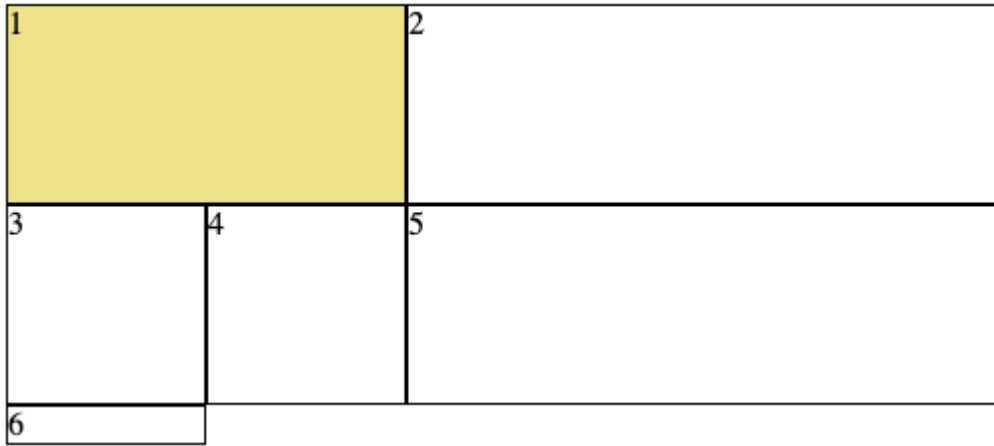
Con esto hemos obtenido exactamente el mismo resultado, a fin de cuentas le hemos dicho que se coloque en el sitio que esperaba, vamos a distinguir ahora entre:

- Elementos no posicionados: no le indicamos donde caen y es CSS grid el que decide donde se va colocando, siguiendo el orden que toque.
- Elementos posicionados: si le indicamos donde deben de caer.

¿Que pasaría si ahora le digo a *item1* que su *grid-column-end* es el 3?

```
item1 {
  grid-column-start: 1;
-   grid-column-end: 2;
+   grid-column-end: 3;
  grid-row-start: 1;
  grid-row-end: 2;
+   background: khaki;
}
```

Que el *item1* pasa a ocupar dos filas y desplaza al resto de elemento no posicionados.



Vamos buscar conflictos... ¿Y si intentamos que el item 1 ocupe la posición del 2? ¿Que va a pasar? Se nos ocurren varias alternativas:

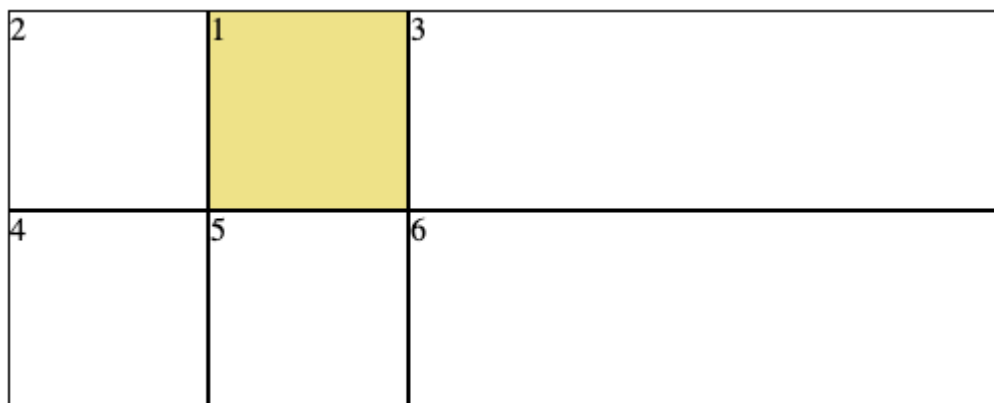
- Que el elemento 1 ocupe la posición del 2 y este uno encima del otro.
- Que el elemento 2 se desplace a la izquierda dejando el hueco al 1 al ser posicionado.
- Que el elemento 2 se desplace a la derecha y se queda un hueco donde antes estaba el 1.

Aquí *css-grid* es muy listo y nos posiciona el elemento 2 donde estaba anteriormente el 1, a fin de cuentas no es posicionado y lo coloca en el primer hueco libre.

Veámoslo en código:

```
#item1 {
-  grid-column-start: 1;
+  grid-column-start: 2;
  grid-column-end: 3;
  grid-row-start: 1;
  grid-row-end: 2;
  background: khaki;
}
```

Y con un pantallazo de como quedaría



Lo que hace CSS Grid es:

- Me defines un cuadrícula.
- Me dices que elementos hay posicionados, y los coloco en su sitio.
- El resto los voy colocando conforme encuentre huecos libres.

No es normal posicionar dos elementos en la misma posición, en este caso se pisan, se escribe uno encima del otro.

grid vs inline grid

Para que se vea todo más claro vamos añadirle un border al grid.

`./src/styles.css`

```
.grid-container {  
  display: grid;  
  grid-template-columns: 100px 100px 300px;  
  grid-template-rows: 100px 100px;  
+ border: 1px solid indianred;  
}
```

Le ponemos un borde al grid container para poder visualizar mejor el espacio que ocupa

Si te fijas el grid llega hasta el final del ancho que tenemos en la ventana, pero los items ocupan menos ¿Por qué? Porque estoy trabajando con anchos de columnas fijas en píxeles, así aunque no estemos en inline, es que no hay más hueco donde pintar.

si cambiáramos el estilo del *grid* container de *grid* a *inline-grid* vemos ya como el ancho de nuestro *grid-container* se ajusta al espacio disponible de cada item.

```
.grid-container {  
- display: grid;  
+ display: inline-grid;  
  grid-template-columns: 100px 100px 300px;  
  grid-template-rows: 100px 100px;  
  border: 1px solid indianred;  
}
```

2	1	3
4	5	6

Posicionamiento negativo

Vamos ahora a decirle al ítem 1 que se posicione de la siguiente manera:

- Que empiece en la línea de columna 2, y termine en la -1 (es decir en la última línea de columna)
- Que empiece en la línea de fila 2 y llegue a la 3 (la segunda fila)

El resultado:

- El ítem 1 se va a la segunda fila y ocupa dos casilla.
- El ítem 6 desplaza abajo porque no hay más espacio.

./src/styles.css

```
#item1 {
  grid-column-start: 2;
  - grid-column-end: 3;
  + grid-column-end: -1;
  grid-row-start: 1;
  grid-row-end: 2;
  background: khaki;
}
```

2	3	4	
5	1		
6			

grid-auto-flow

Fijate que por defecto todos los elementos no posicionados los va colocando por filas y después por columnas, ¿Y si quisieramos que fuera al revés? Esto lo podemos hacer con la propiedad *grid-auto-flow* aquí le podemos indicar si nuestra preferencia es filas o columnas, vemoas como afecta esto:

Primero vamos dejar nuestro item como estaba (en la posición 1 y sin posicionar)

Vemos que todo fluye por filas y columnas:

```
#item1 {  
-  grid-column-start: 2;  
-  grid-column-end: 3;  
-  grid-row-start: 1;  
-  grid-row-end: 2;  
  background: khaki;  
}
```

Vamos a aplicar la propiedad *grid-auto-flow* indicandole:

```
.grid-container {  
  display: inline-grid;  
  grid-template-columns: 100px 100px 300px;  
  grid-template-rows: 100px 100px;  
  border: 1px solid indianred;  
+  grid-auto-flow: column;  
}
```

Fijate como se reparten ahora los items (primero manda la columna, después la fila).

1	3	5
2	4	6

Este cambio de flow sólo afecta a los elementos no posicionados, los posicionados siguen manteniendo su espacio asignada, da igual que estemos en *grid-auto-flow row* o *column*.

grid-auto-rows

En ejemplo anteriores, vimos que definiamos, por ejemplo, 2 filas y al final añadíamos una tercera, está se creaba pero con un alto que no controlabamos.

Por ejemplo, tenemos en el html:

```
<body>
  <div class="grid-container">
    <div class="item" id="item1">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
  </div>
</body>
```

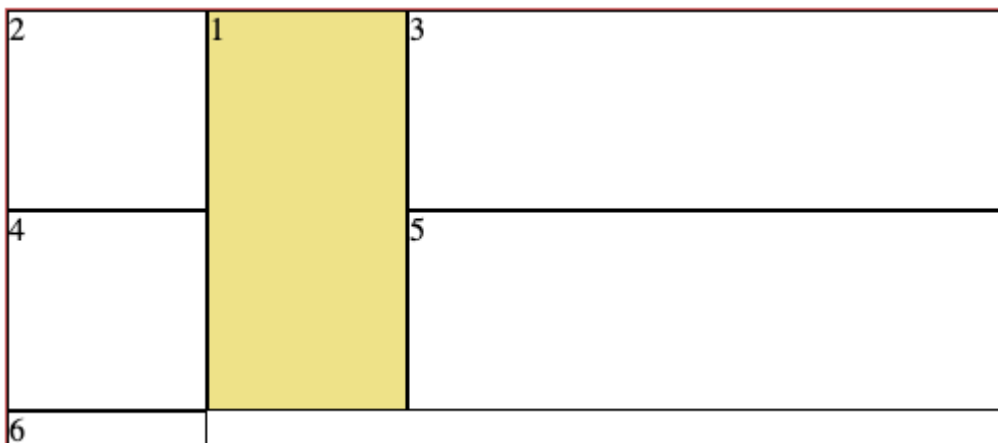
Y en el css

```
.grid-container {
  display: inline-grid;
  grid-template-columns: 100px 100px 300px;
  grid-template-rows: 100px 100px;
  border: 1px solid indianred;
}

.item {
  border: 1px solid black;
}

#item1 {
  background-color: khaki;
  grid-column: 2 / 3;
  grid-row: 1 / span 2;
}
```

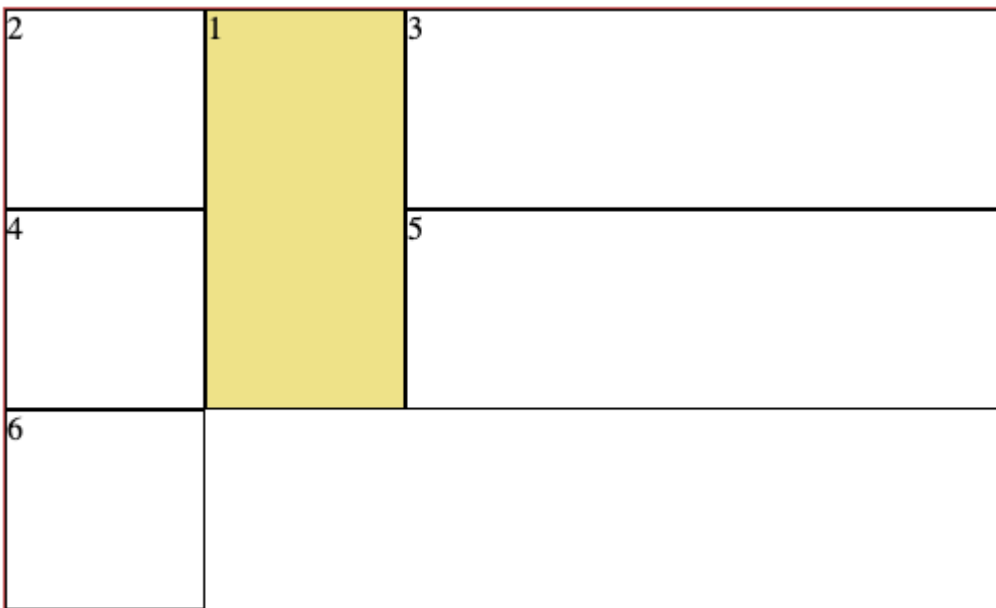
Cómo no hay espacio para el elemento 6 este se muestra en una nueva fila, pero con un alto que igual no es el que queremos:



Vamos ahora a indicarle a nuestro *grid-container* que si se crean nuevas filas que les de un alto por defecto:

```
.grid-container {  
  display: inline-grid;  
  grid-template-columns: 100px 100px 300px;  
  grid-template-rows: 100px 100px;  
  border: 1px solid indianred;  
+ grid-auto-rows: 100px;  
}
```

Ahora cuando se crea la nueva fila si le da un alto de *100px*:



También tenemos la propiedad *grid-auto-columns*, esto nos valdría si el *grid-auto-flow* estuviera en modo *columns*.

Referenciando en ítems

Hasta hora hemos usado el indicador de línea en los ítems para ver donde caen, una alternativa a esto es usar los alias que previamente hayamos definido, por ejemplo:

```
.grid-container {  
  display: inline-grid;  
- grid-template-columns: 100px 100px 300px;  
+ grid-template-columns: 100px [col-2] 100px 300px;  
  grid-template-rows: 100px 100px;  
  border: 1px solid indianred;  
  grid-auto-flow: column;  
}
```

Y en ítem 1, para colocarlo en el espacio de línea 2 en la columna, hemos usado:

```
#item1 {  
  background-color: khaki;  
  grid-column-start: 2;  
  grid-column-end: 3;  
  grid-row-start: 2;  
  grid-row-end: 3;  
}
```

Y podemos reemplazarlo por el alias que hemos creado:

```
#item1 {  
  background-color: khaki;  
-  grid-column-start: 2;  
+  grid-column-start: col-2;  
  grid-column-end: 3;  
  grid-row-start: 2;  
  grid-row-end: 3;  
}
```

Short hands

Hasta ahora hemos usado *grid-column-start*, *grid-column-end*, *grid-row-start* y *grid-row-end* esto se hace un poco pesado de usar, css *grid* nos ofrece un *shorthand grid-column* y *grid-row* que nos hacen el trabajo más llevadero.

En vez de tener que teclear:

```
#item1 {  
  background-color: khaki;  
  grid-column-start: 2;  
  grid-column-end: 3;  
  grid-row-start: 2;  
  grid-row-end: 3;  
}
```

Podemos usar esto:

```
#item1 {  
  background-color: khaki;  
  grid-column: 2 / 3;  
  grid-row: 2 / 3;  
}
```

Aquí le estamos diciendo que mi elemento, va desde el separado de columna 2 a la 3, y que mi elemento ocupa desde el separador de línea 2 al 3.

Otra opción interesante que nos ofrece este short-hand es la opción de usar *span* para decirle cuantas filas queremos que ocupen, de esta manera no tengo que ir calculando cual es el separador final de fila o columna, y si cambio el origen no tengo que ir recalculando.

Veamos un ejemplo, vamos a volver a posicionar nuestro *item1* en el row-start 1:

```
#item1 {  
  background-color: khaki;  
  grid-column: 2 / 3;  
-  grid-row: 2 / 3;  
+  grid-row: 1 / 3;  
}
```

Vamos a usar ahora el short hand *span*:

```
#item1 {  
  background-color: khaki;  
  grid-column: 2 / 3;  
  grid-row: 1 / 3;  
+  grid-row: 1 / span 2;  
}
```

Notación repeat

A veces cuando definimos muchas filas o columnas con un mismo tamaño, puede ser un poco pesado ir repitiendo valores, CSS grid tiene una ayuda para permitirnos hacer un bucle definiendo filas o columnas, veamos como.

Partimos del ejemplo original:

```
.grid-container {  
  display: inline-grid;  
  grid-template-columns: 100px 100px 300px;  
  grid-template-rows: 100px 100px;  
  border: 1px solid indianred;  
  grid-auto-flow: column;  
}
```

Si te fijas en las columnas repetimos dos veces el valor *100*, en vez de esto le podemos decir que *repita* el valor 100 dos veces:

```
.grid-container {
  display: inline-grid;
  - grid-template-columns: 100px 100px 300px;
  + grid-template-columns: repeat(2,100px) 300px;
  grid-template-rows: 100px 100px;
  border: 1px solid indianred;
  grid-auto-flow: column;
}
```

Esto nos puede ser útil si queremos crear un número grande de filas o columnas, y también al ahora de modificar un valor no tener que ir modificandola en todas las entradas.

Repeat nos permite entrar en casos más avanzados, podemos combinar varias columnas o filas en el cuerpo del repeat, por ejemplo, si tenemos

```
.grid-container {
  display: inline-grid;
  grid-template-columns: 100px 25px 100px 25px 100px 25px;
  grid-template-rows: 100px 100px;
  border: 1px solid indianred;
  grid-auto-flow: column;
}
```

Podemos definir las columnas de la siguiente manera:

```
.grid-container {
  display: inline-grid;
  - grid-template-columns: 100px 25px 100px 25px 100px 25px;
  + grid-template-columns: repeat(3, 100px 25px);
  grid-template-rows: 100px 100px;
  border: 1px solid indianred;
  grid-auto-flow: column;
}
```

Otra opción que nos permite es ponerle nombres a cada separador de línea o columna:

```
.grid-container {
  display: inline-grid;
  - grid-template-columns: repeat(3, 100px 25px);
  + grid-template-columns: repeat(3, 100px [mi-columna] 25px);
  grid-template-rows: 100px 100px;
  border: 1px solid indianred;
  grid-auto-flow: column;
}
```

¿Qué pasa con esto? Que estoy repitiendo nombres, cuando se repiten nombres podemos usar un índice para acceder al que queramos.

```
#item1 {  
  background-color: khaki;  
-  grid-column: 2 / 3;  
+  grid-column: mi-columna 2 / mi-columna 3;  
  grid-row: 1 / span 2;  
}
```

Esto es buena idea usarlo sólo si realmente aporta valor en tu escenario de desarrollo, un mal uso puede hacer que sea menos legible tu css.

Áreas

Antes de explicar lo que son las áreas, vamos a crear un layout de una aplicación con lo que ya sabemos.

Esta layout tendrá:

- Una cabecera.
- Un menu lateral (lo colocaremos a la izquierda).
- El contenido principal.
- Un footer.

Reemplazamos lo que hay en el *body* de nuestro *codesandbox* y metemos lo siguiente.

./index.html

```
<body>  
  <div class="grid-container">  
    <header>Logo App</header>  
    <nav>Menu</nav>  
    <main>Contenido principal</main>  
    <footer>Footer</footer>  
  </div>  
</body>
```

Nos vamos ahora al CSS, borramos lo que hay y arrancamos de cero.

Arrancamos por quitar el margen y padding del body, y vamos a definir nuestro grid container.

./src/styles.css

```
body {  
  margin: 0;  
  padding: 0;  
  height: 100vh;
```

```
}

.grid-container {
  display: grid;
  height: 100vh;
}
```

¿Cuántas columnas nos va a hacer falta? Vamos a plantear 3 columnas:

- Una columna para el menú lateral, que va a ocupar 150px.
- Voy a crear una columna de separación entre el menú y el contenido de 25px.
- Otra para el contenido, en este caso uso algo nuevo *1fr* le estoy diciendo que tome todo el contenido que pueda.

Y en las filas:

- El header lo voy a dejar en 100px.
- El contenido principal, que me ocupe *1fr* todo el espacio disponible.
- El footer que ocupe otros 100px.

Vamos a añadirlas:

`./src/styles.css`

```
.grid-container {
  display: grid;
+ grid-template-columns: 150px 25px 1fr
+ grid-template-rows: 100px 1fr 100px;
}
```

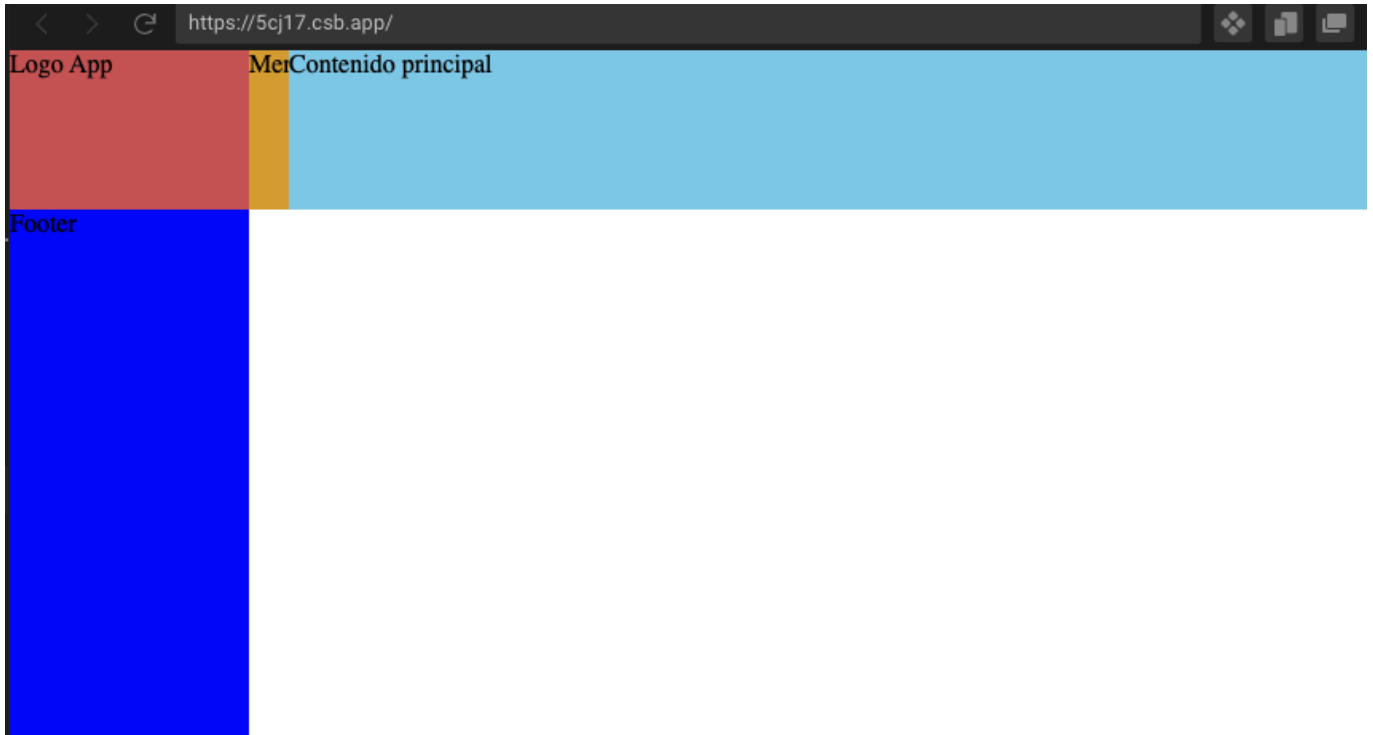
Para ver más clara cada zona, vamos a darle unos colores de fondo.

```
.grid-container {
  display: grid;
  grid-template-columns: 150px 25px 1fr
  grid-template-rows: 100px 1fr 100px;
}

+header {
+  background: indianred;
+ }
+
+ main {
+  background: skyblue;
+ }
+
+ nav {
+  background: goldenrod;
+ }
```

```
+  
+footer {  
+  background: blue;  
+}
```

Si lo dejamos todo por defecto, vemos que sale todo mal montado ¿Por qué? porque tenemos que indicarle que ciertos items ocupen varios espacios, por ejemplo la barra de navegación debería de ocupar 3 filas, el footer también...



Vamos a configurarlo:

Vamos a por el logo, este queremos que ocupe una fila entera.

./src/styles.css

```
header {  
  background: indianred;  
+  grid-column: 1 / -1;  
}
```

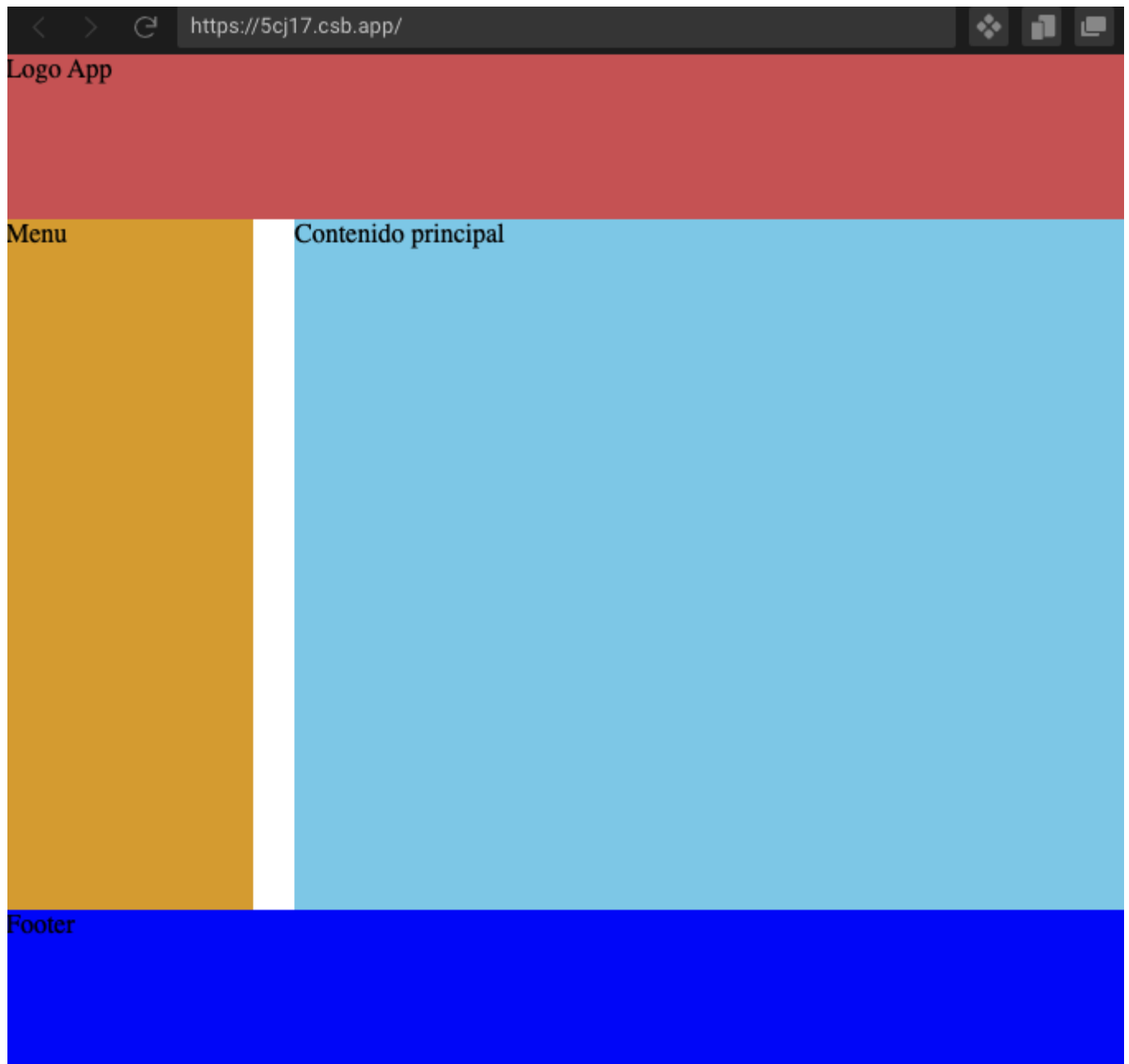
Ya lo tenemos posicionado, vamos a por el menú, que cae donde nos interesa, el problema es que el contenido principal ha caído justo donde está el separador, vamos a moverlo: queremos que empiece en la columna 3, hasta el final (de la 3 a la -1).

```
main {  
  background: skyblue;  
+  grid-column: 3 / -1;  
}
```

Y el footer, queremos que ocupe toda la fila:

```
footer {  
  background: blue;  
+  grid-column: 1 / -1;  
}
```

Ya tenemos nuestro layout montado.



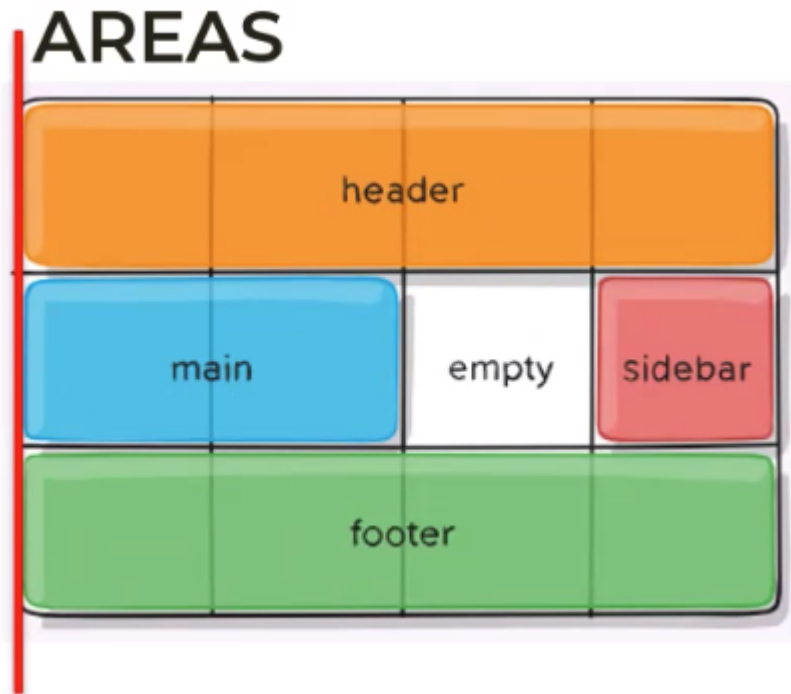
Si lo hacemos más grande o más pequeño, vemos que el *logo* y *footer* se quedan fijas y el ancho y alto del contenido se adaptan para ocupar todo el espacio restante.

Esto es interesante, pero no es muy mantenible, si a futuro añadimos nuevas filas o columnas puede ser complicado de modificar.

Para hacer esto más fácil podemos usar las áreas:

- Estas nos permiten en el mismo contenedor definir zonas de columnas.
- Estas zonas de columnas, después se las podemos añadir a cada item.

Veamos un layout parecido al que montamos en el ejemplo anterior:



Podemos configurarlo de la siguiente manera:

- Definimos una primera fila que va a ocupar cuatro columnas (tendremos para el *line name* el nombre *header* y automáticamente nos va a crear los identificadores de línea *header-start* y *header-end* para identificar la línea en la que termina cada una de estas áreas),
- En la segunda fila le indicamos que *main* ocupe dos columnas, después que se salte una columna (la de espaciado entre el sidebar y el contenido, eso lo indicamos con el caracter punto .) y que coloque en la restante el *sidebar*.
- En la última ocupamos todo el footer.

```
.container {  
  display: grid;  
  grid-template-columns: 50px 50px 50px 50px;  
  grid-template-rows: auto;  
  grid-template-areas:  
    "header header header header"  
    "main main . sidebar"  
    "footer footer footer footer";  
}
```

¿Cómo referenciaríamos esto en cada item?

```
.item-a {
  grid-area: header;
}

.item-b {
  grid-area: main;
}

.item-c {
  grid-area: sidebar;
}

.item-d {
  grid-area: footer;
}
```

Vamos ahora a modificar nuestro ejemplo para que esté configurado utilizando áreas:

Recordamos que tenemos:

- La cabecera de nuestra aplicación que ocupa una fila entera.
- Una segunda fila con un sidebar de menu, y el contenido principal (que ocupa todo lo que puede).
- Un footer que ocupa una fila entera.

Vamos primero a definir las áreas:

`./src/styles.css`

```
.grid-container {
  display: grid;
  grid-template-columns: 150px 25px 1fr;
  grid-template-rows: 100px 1fr 100px;
  height: 100vh;
+ grid-template-areas:
+   "header header header"
+   "menu . body"
+   "footer footer footer";
}
```

Y en los items:

`./src/styles.css`

```
header {
  background: indianred;
- grid-column: 1 / -1;
+ grid-area: header;
}
```

```


main {
  background: skyblue;
-  grid-column: 3 / -1;
+  grid-area: body;
}

nav {
  background: goldenrod;
+  grid-area: menu;
}

footer {
  background: blue;
-  grid-column: 1 / -1;
+  grid-area: footer;
}

```

Hemos montado el mismo layout pero usando áreas ¿Cual te parece más mantenible?

 Logo arriba en una fila, menu en una columna, espacio, contenido ocupando todo lo que queda, y footer en otra fila

A nivel de CSS Grid existe también un *short-hand* que aglutina *grid-template-rows* *grid-template-columns* y *grid-template-areas* se llama *grid-template* pero lo que genera es poco manejable.

Por ejemplo:

```

.container-1 {
  display: grid;
  grid-template:
    [row-1-start] "header header header" 1fr [row1-end]
    [row-2-start] "main . sidebar 4fr" [row2-end]
    [row-3-start] "footer footer footer" 1fr [row3-end]
    / 1fr 50px 4fr;
}

```

Aquí hacemos una mezcla de nombre de áreas, espacios de filas, y después del caracter /, definimos los ancho de columna... esto es algo un poco lioso.

Se podría convertir a:

Un tema importante a tener en cuenta es que podemos tener un contenedor padre que sea CSS Grid y anidar hijos con flexbox, en cuanto a anidar CSS Grid nos podemos encontrar con algunos problemas en navegadores antiguos.

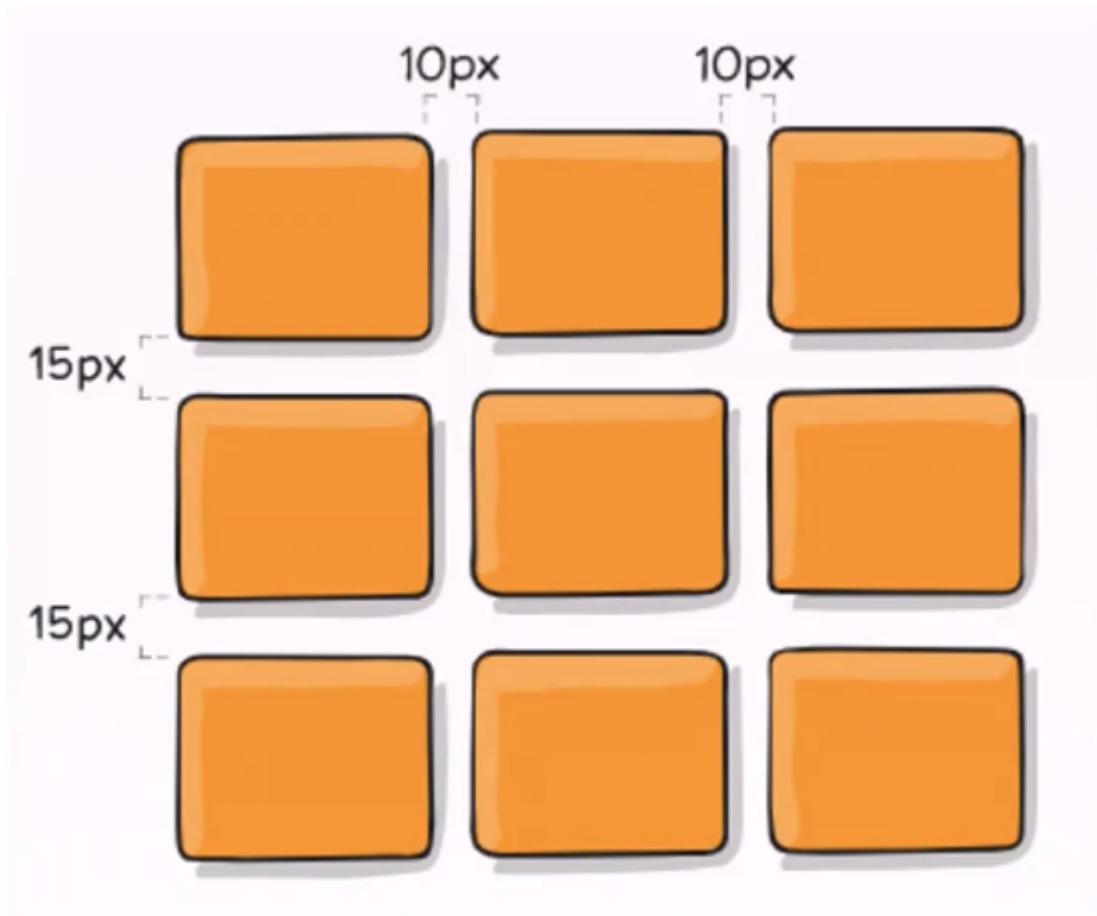
gap

Una funcionalidad muy sencilla a la vez que práctica que ofrece *css-grid* son las propiedades *row-gap* y *column-gap*, estas nos permiten añadir huecos entre filas y columnas.

Un ejemplo: imaginemos que tenemos el siguiente container:

```
.container {  
  grid-template-columns: 100px 50px 100px;  
  grid-template-rows: 80px auto 80px;  
}
```

Y queremos que haya un hueco de 10 *pixeles* entre cada columna, y un hueco de 15 *pixeles* entre cada fila.



¿Cómo podemos definir este espacio? Muy fácil, añadiendo *column-gap* y *row-gap* a este contenedor:

```
.container {  
  grid-template-columns: 100px 50px 100px;  
  grid-template-rows: 80px auto 80px;  
  + column-gap: 10px;  
  + row-gap: 15px;  
}
```

También tenemos disponible un *shorthand* en el que primero le definimos las *filas* y después las *columnas*

```
.container {
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  - column-grap: 10px;
  - row-gap: 15px;
  + gap: 15px 10px;
}
```

Si coincidieran los dos valores de gap de columnas y filas podríamos poner *gap: 15px*

Si quieres verlo en un ejemplo:

En el HTML podemos poner:

```
<body>
  <div class="grid-container">
+   <div class="item">1</div>
+   <div class="item">2</div>
+   <div class="item">3</div>
+   <div class="item">4</div>
+   <div class="item">5</div>
+   <div class="item">6</div>
  </div>
</body>
```

En el css (sin gap):

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

.grid-container {
  display: inline-grid;
  border: 1px solid blue;
  grid-template-columns: repeat(3, 100px);
  grid-auto-rows: 100px;
}

.item {
  border: 1px solid black;
}
```

Si vemos ahora el resultado tenemos:

Vamos añadirle el *gap*

```
.grid-container {
  display: inline-grid;
  border: 1px solid blue;
  grid-template-columns: repeat(3, 100px);
  grid-auto-rows: 100px;
+  row-gap: 15px;
+  column-gap: 18px;
}
```

Fijate que el gap se aplica entre *items* y no nos añade *margen* adicional.

También podríamos haber usado el *shorthand*:

```
.grid-container {
  display: inline-grid;
  border: 1px solid blue;
  grid-template-columns: repeat(3, 100px);
  grid-auto-rows: 100px;
-  row-gap: 15px;
-  column-gap: 18px;
+  gap: 15px 18px;
}
```

Si tuvieran el mismo *gap* tanto filas como columnas podríamos haber especificado directamente *gap: 15px*

justify-items

Esto nos va a recordar mucho a *flex-box*, vamos a ver como posicionar las filas y columnas dentro de nuestra cuadrícula.

Una definición más formal:

Organiza los *items* en referencia al eje *inline* (fila horizontal), el valor es aplicado a todos los items dentro del contenedor (esto es opuesto a *align-items* que organiza el eje vertical).

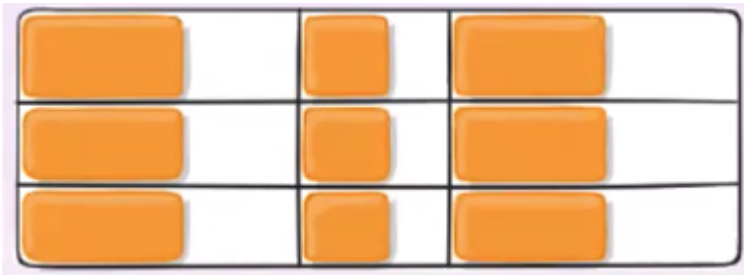
Este setting en concreto no existe en *flex-box* (más de una vez veréis que lo han colado por equivocación), es un caso un poco especial de *css-grid*, veamos su comportamiento.

Nos permite ajustar cada elemento dentro de su celda:

- Que se muestre pegado a la izquierda de la celda.
- Que se muestre pegado a la derecha de la celda.
- Que se muestre centrado en su celda.
- Que ocupe todo el espacio que tiene disponible en la celda.

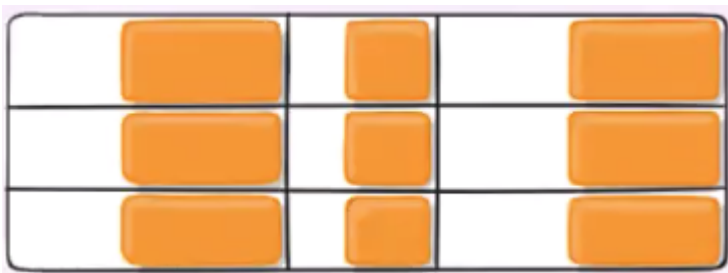
opción: *start*

```
.container {  
  justify-items: start;  
}
```



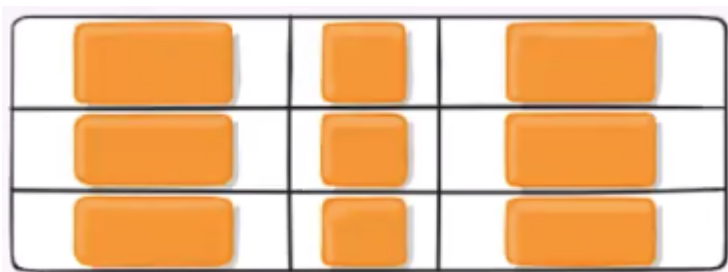
opción: *end*

```
.container {  
  justify-items: end;  
}
```



opción: *center*

```
.container {  
  justify-items: center;  
}
```



opción: *stretch*

```
.container {  
  justify-items: stretch;  
}
```



Vamos a ver esto con un ejemplo.

Sustituimos lo que hay dentro del *body* por el siguiente *html*:

```
<body>
+ <div class="center-container grid-container">
+   <div class="item">1</div>
+   <div class="item">2</div>
+   <div class="item">3</div>
+   <div class="item">4</div>
+   <div class="item">5</div>
+   <div class="item">6</div>
+ </div>
</body>
```

Y en el CSS, borramos todo y empezamos de cero, al *body* le ponemos un color oscuro, para diferenciar lo que es el *body* del contenedor.

./src/styles.css

```
body {
  background-color: darkslategrey;
  padding: 0;
  margin: 0;
  height: 100vh;
  width: 100vw;
}
```

Vamos a centrar el contenido que queremos mostrar (esto no tiene que ver con CSS-Grid es simplemente para que veamos lo que hay dentro centrado).

./src/styles.css

```
body {
  background-color: darkslategrey;
  padding: 0;
  margin: 0;
  height: 100vh;
  width: 100vw;
```



```

}

+ .center-container {
+   position: relative;
+   background-color: white;
+   width: 80vw;
+   height: 80vh;
+   left: 10vw;
+   top: 10vw;
+ }

```

Vamos al turrón, creamos un `_grid-container`, de tres columnas y dos filas.

```

.center-container {
  position: relative;
  background-color: white;
  width: 80vw;
  height: 80vh;
  left: 10vw;
  top: 10vw;
}

+ .grid-container {
+   display: grid;
+   grid-template: repeat(2, 1fr) / repeat(3, 1fr);
+   border: 1px solid black;
+ }

```

Vamos a por cada item, en este caso vamos a anidar un *flex* container para alinear los números en el centro de cada celda:

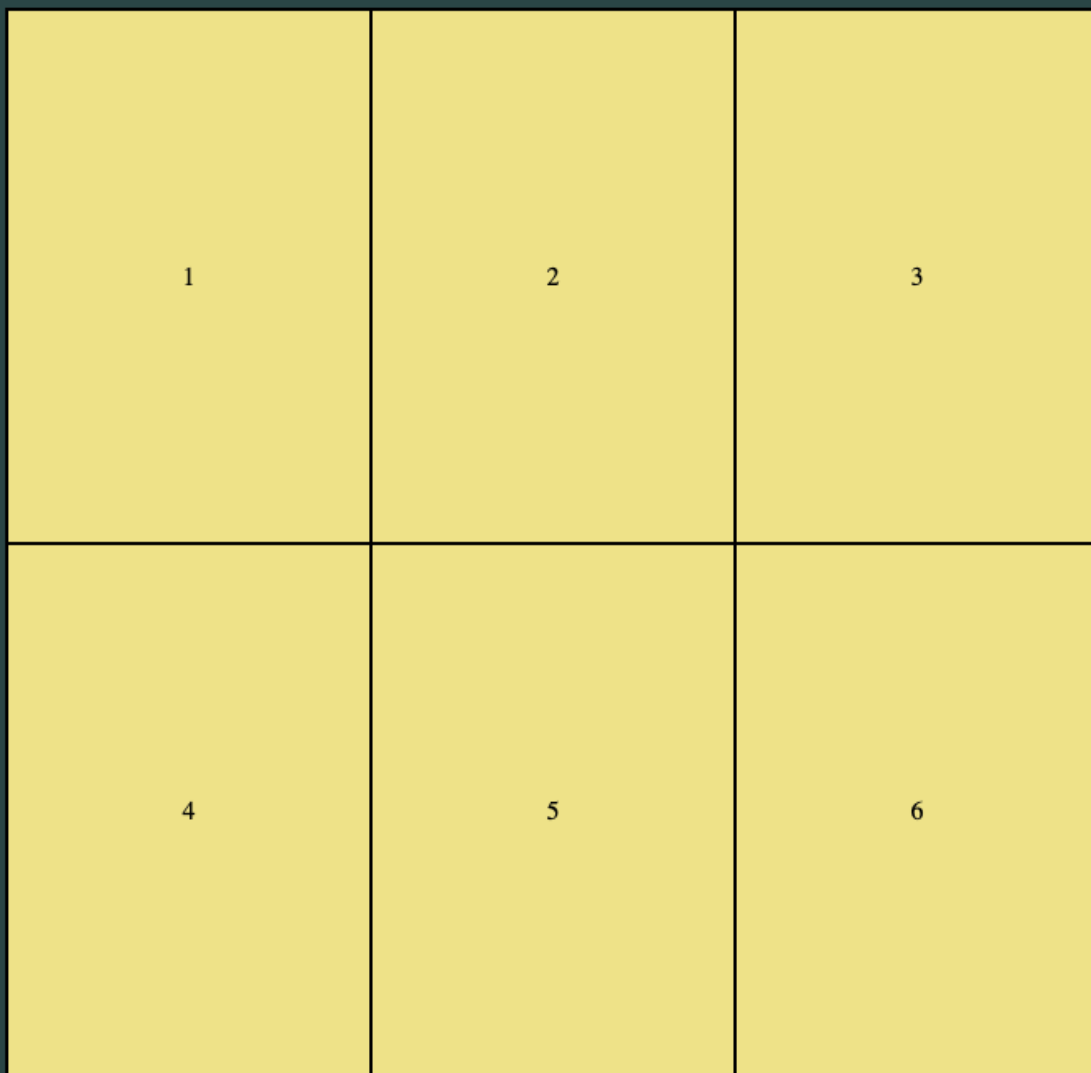
```

.grid-container {
  display: grid;
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);
  border: 1px solid black;
}

+ .item {
+   border: 1px solid black;
+   background-color: khaki;
+   display: flex;
+   justify-content: center;
+   align-items: center;
+ }

```

Ahora mismo este es el resultado que tenemos

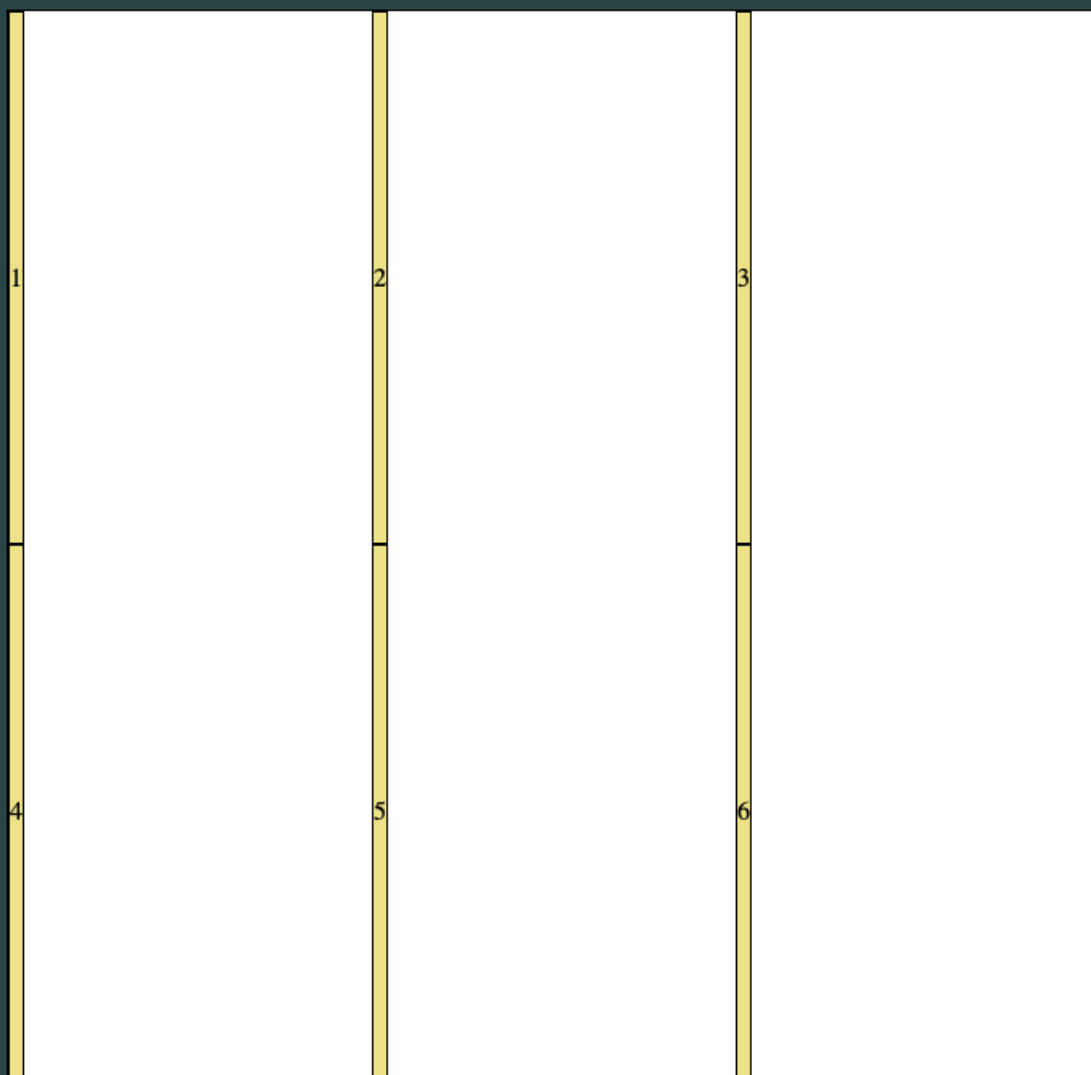


Nos ponemos a jugar con *justify-items* el valor que se asigna a esta propiedad es la de *stretch* por eso podemos ver que la del contenido de cada celda ocupa todo el espacio.

Si ahora cambiamos *justify-items* y decimos que no haga un start

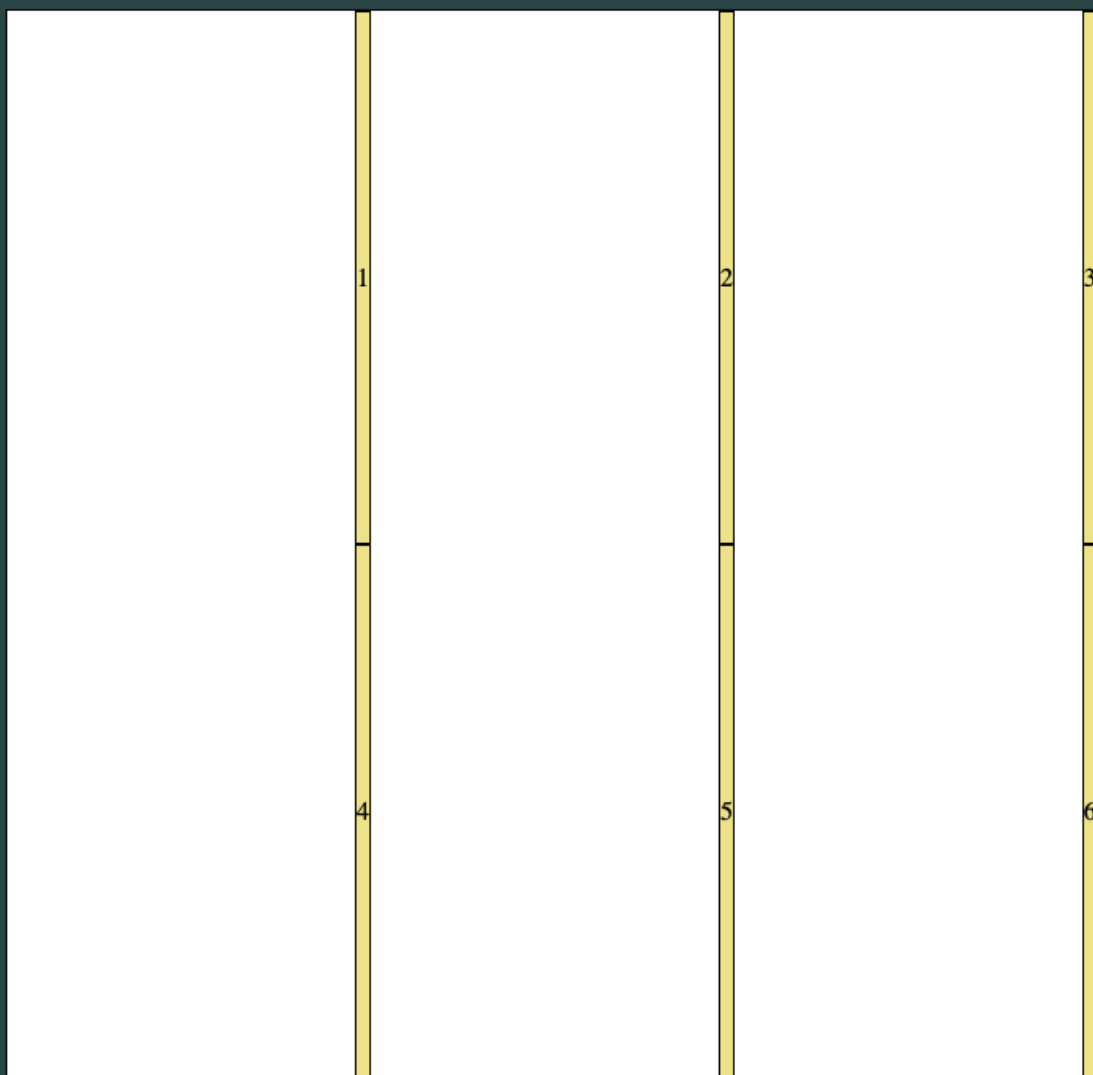
```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);  
  border: 1px solid black;  
+ justify-items: start;  
}
```

Fijate como todos los items se van a la izquierda y ocupan sólo el tamaño horizontal necesario para renderizar su contenido.



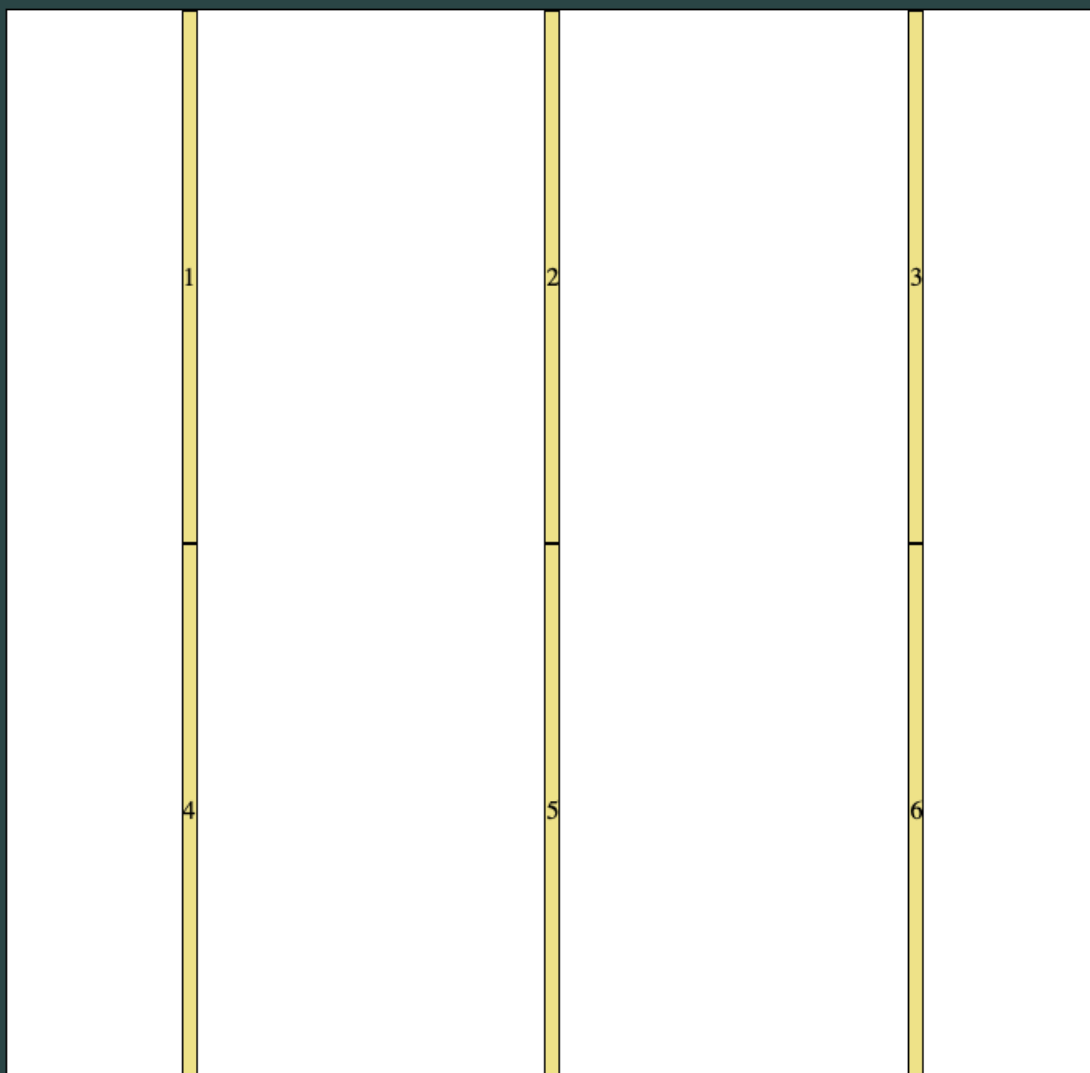
Si lo cambiamos a un *justify-items end* veremos que se alinean a la derecha:

```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);  
  border: 1px solid black;  
  - justify-items: start;  
  + justify-items: end;  
}
```



Y si lo ponemos con *center* vemos que se centran las cajas horizontalmente pero sólo ocupando el espacio necesario de cada item.

```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);  
  border: 1px solid black;  
  - justify-items: end;  
  + justify-items: center;  
}
```



A veces no queremos que se aplique el *justify* a todos los elementos, puede que haya elementos en los que queramos que tenga un valor distinto, CSS Grid nos permite aplicar esto a un sólo ítem utilizando la propiedad *justify-self*

Vamos a mostrar el elemento 2 justificado a la izquierda:

`./index.html`

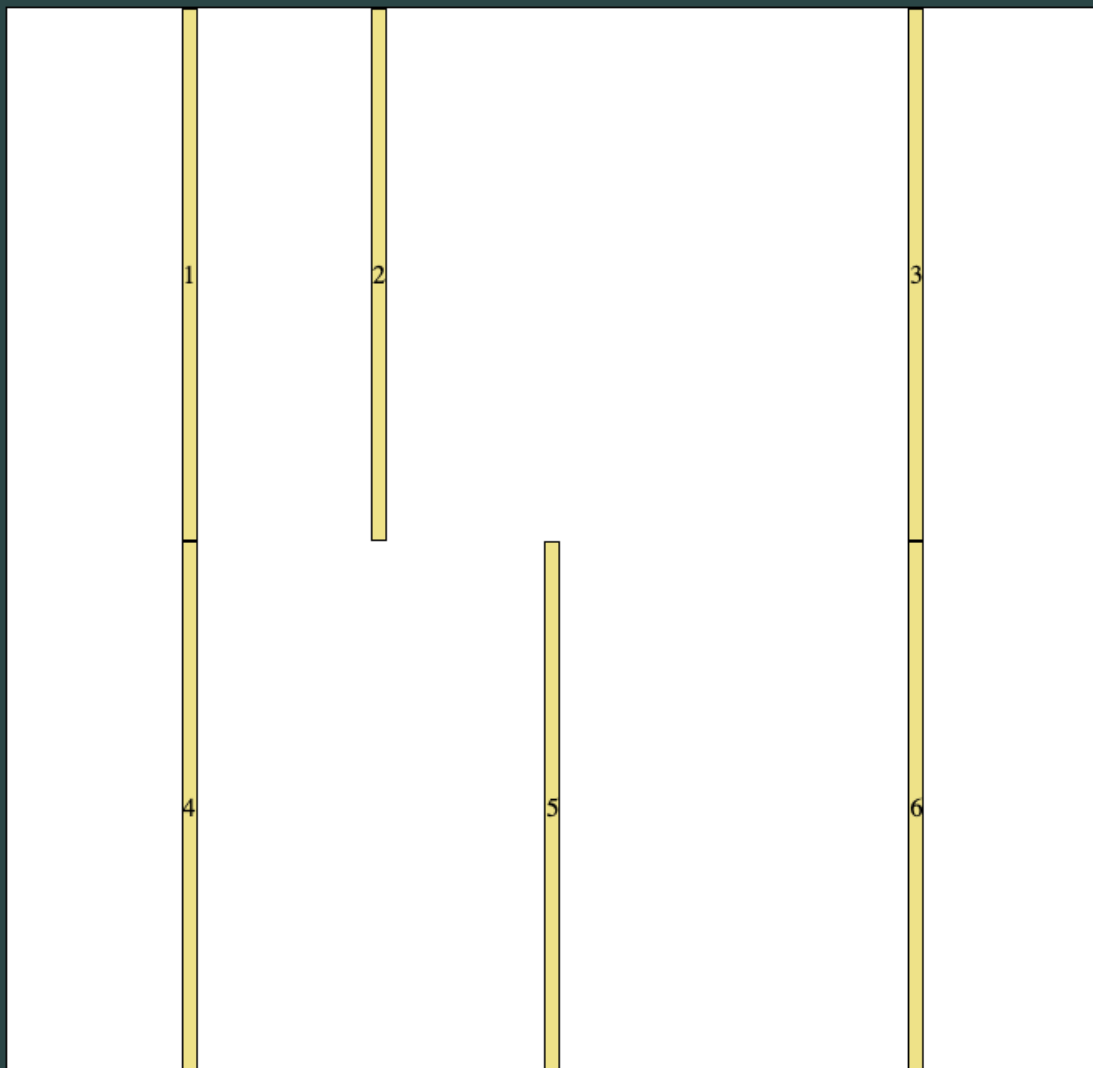
```
<div class="center-container grid-container">
  <div class="item">1</div>
  -   <div class="item">2</div>
  +   <div class="item" id="item2">2</div>
  <div class="item">3</div>
```

`./src/miestilo.css`

```
.item {  
  border: 1px solid black;  
  background-color: khaki;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}
```

```
+ #item2 {  
+   justify-self: start;  
+ }
```

Fijate como el elemento 2 se desmarca del resto



align-items

Este es el hermano de *justify-item* pero trabajando con el eje vertical, es decir.

Su valor por defecto es *stretch* por eso en el ejemplo anterior podíamos ver que cada item intentaba ocupar todo el espacio vertical.

Vamos a jugar con el resto de opciones y ver como se nos queda el ejemplo anterior.

Para evitar meter ruido vamos a eliminar el *justify-self* del item 2.

./src/mystyles.css

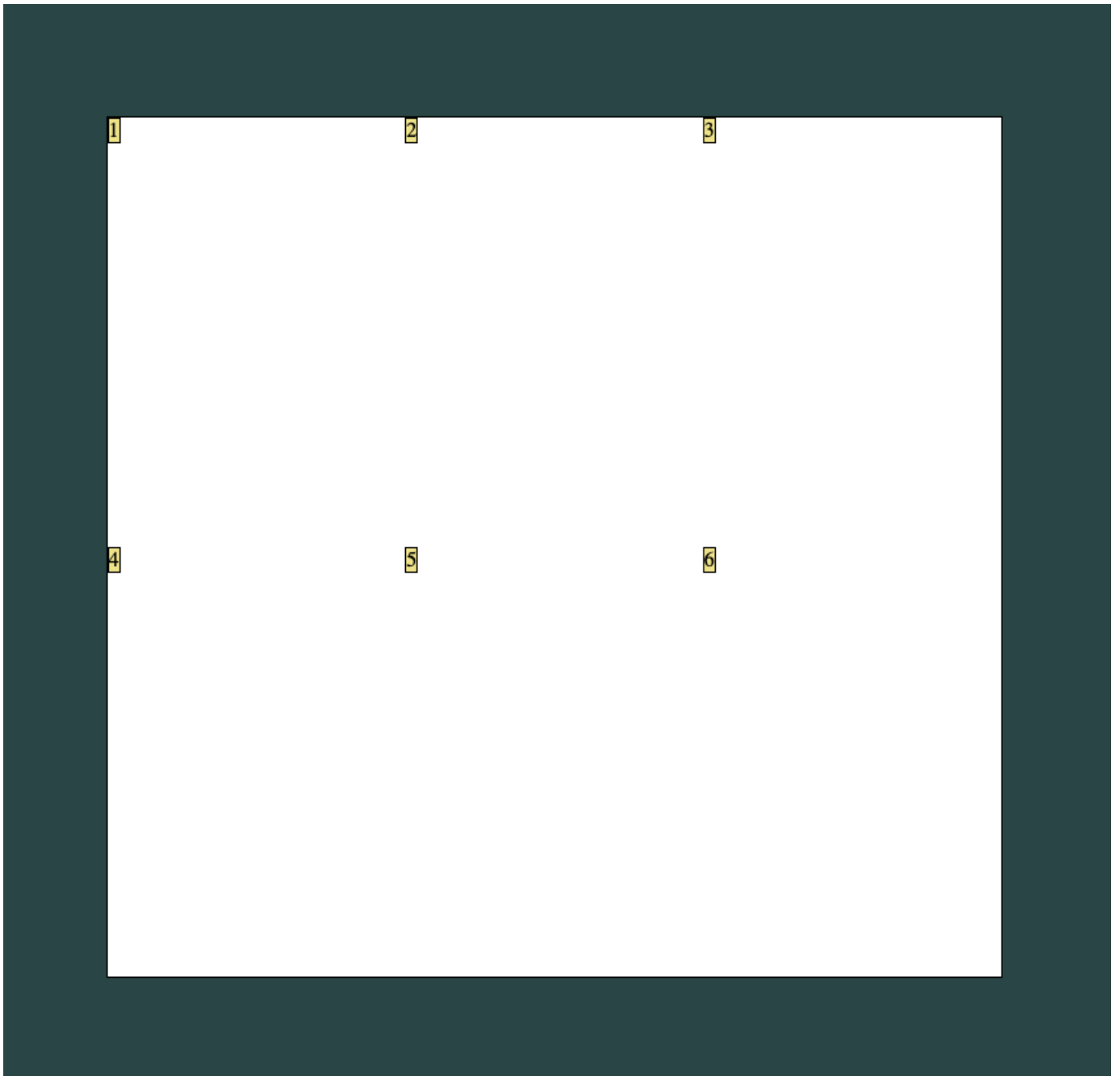
```
.item {
  border: 1px solid black;
  background-color: khaki;
  display: flex;
  justify-content: center;
  align-items: center;
}

#item2 {
-  justify-self: start;
}
```

Ahora vamos a decirle al contenedor que queremos alinear los elementos arriba, valor *start*

./src/mystyles.css

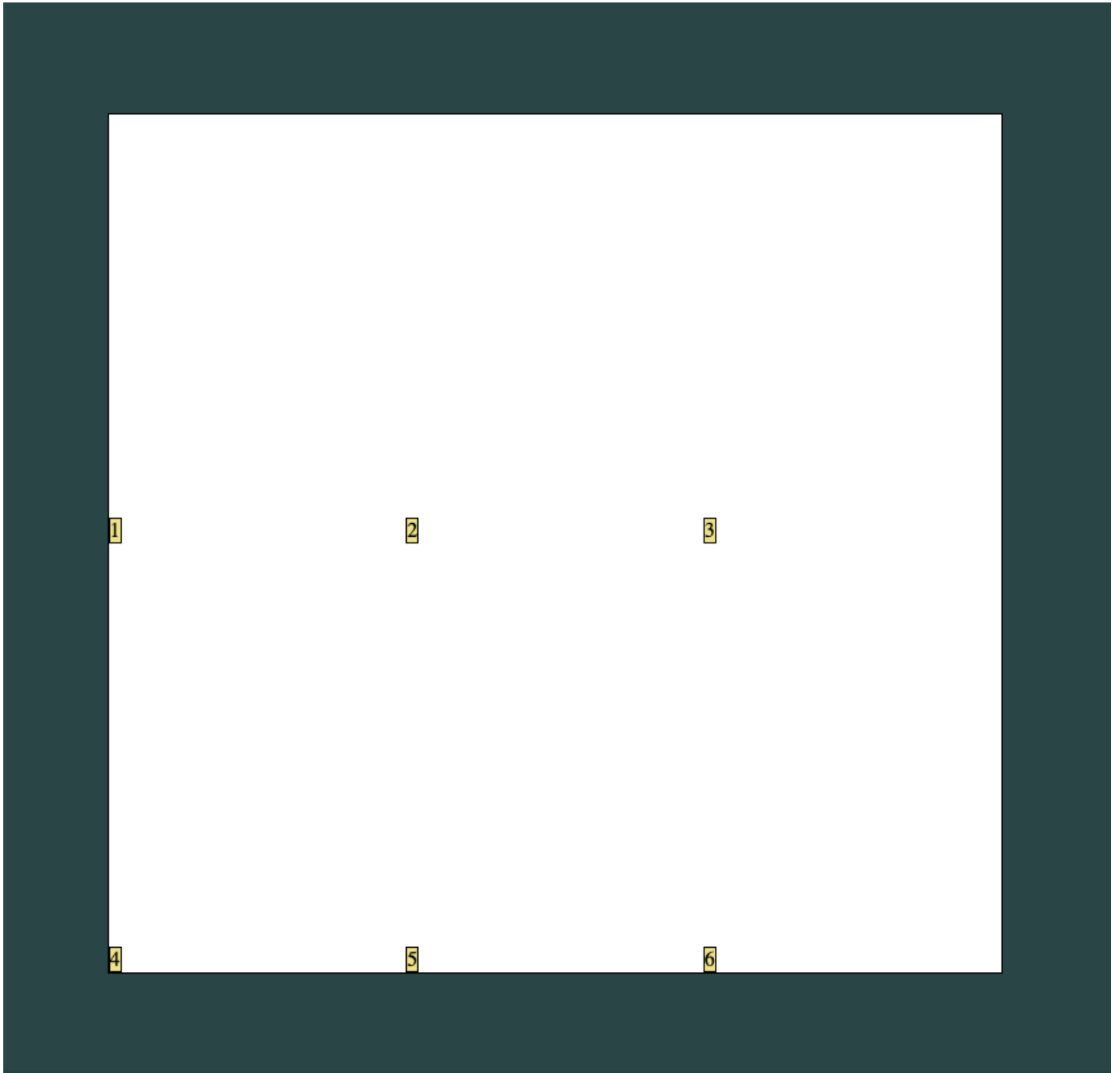
```
.grid-container {
  display: grid;
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);
  border: 1px solid black;
  justify-items: center;
+ align-items: start;
}
```



Vamos a probar con alinearlo en la parte de abajo de cada celda *end*

`./src/mystyles.css`

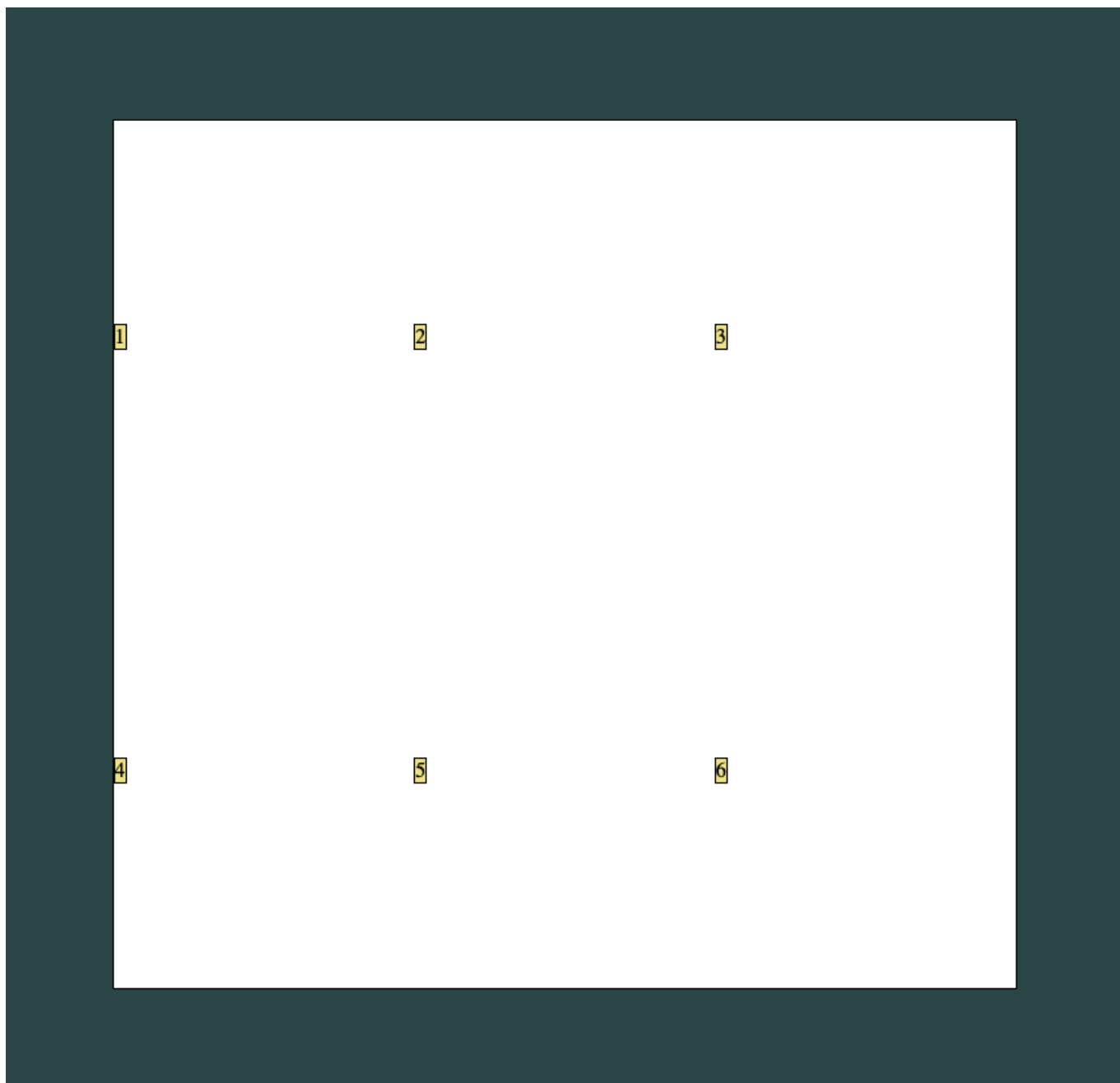
```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);  
  border: 1px solid black;  
  justify-items: center;  
  - align-items: start;  
  + align-items: end;  
}
```

Y ahora vamos a probar a centrarlos:

`./src/mystyles.css`

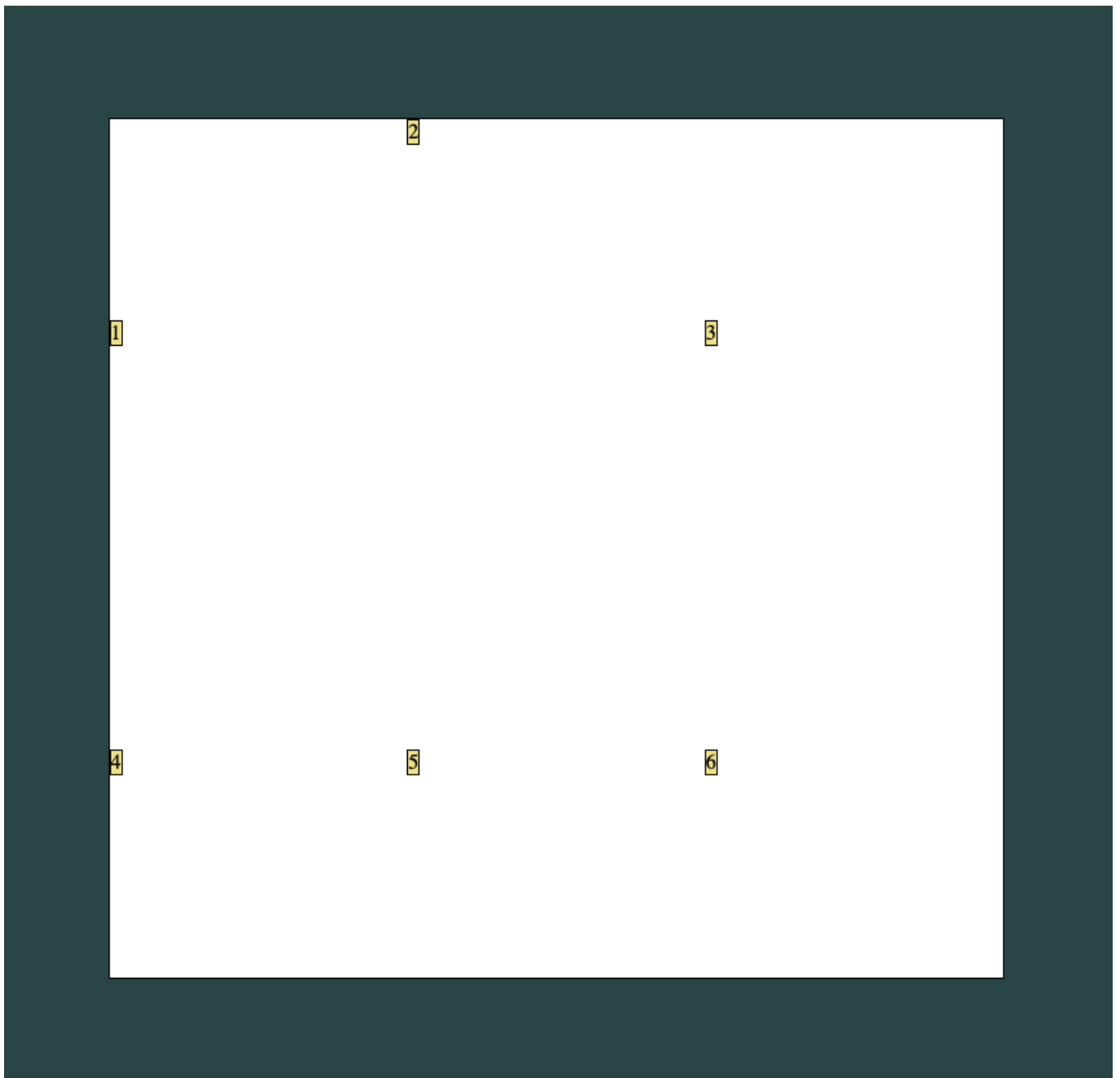
```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);  
  border: 1px solid black;  
  justify-items: center;  
  - align-items: end;  
  + align-items: center;  
}
```



Al igual que con `justify`, podemos alinear una celda en concreto, para ello vamos a decirle que el elemento con *id* `item2` lo alinee al principio:

`./src/mystyles.css`

```
#item2 {  
+ align-self: start;  
}
```



Existe un *shorthand* para unificar *justify-items* y *align-items* se llama *place-items*, tiene la siguiente sintaxis:

```
place-items: <align-items><justify-items>;
```

Si sólo se define un valor, se asigna a las dos propiedades.

En el ejemplo anterior podríamos haber hecho:

```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 1fr) / repeat(3, 1fr);  
  border: 1px solid black;  
  - justify-items: center;
```

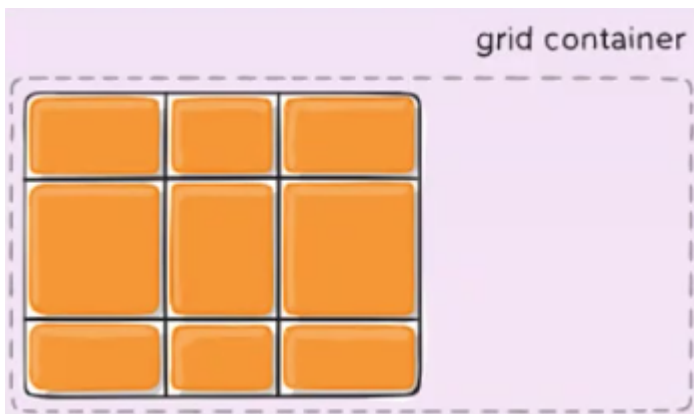
```
- align-items: center;  
+ place-items: center;  
}
```

Justify-content

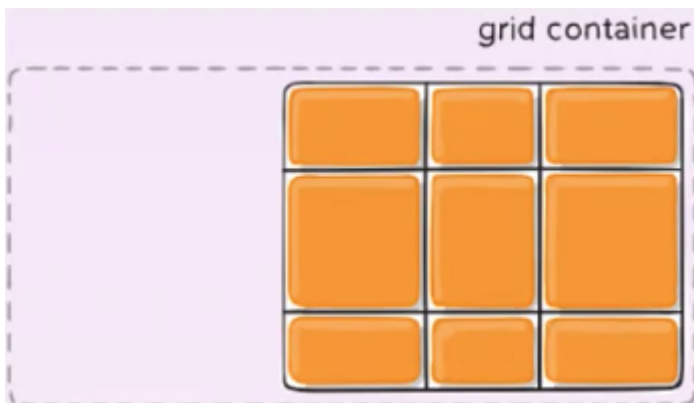
Ahora que ya sabemos como hacer un *justify* o *align* del contenido que hay dentro de una celda, nos toca aprender como hacer *justify* o *align* de las filas y columnas de nuestro grid.

Veamos los valores posibles y como se representan:

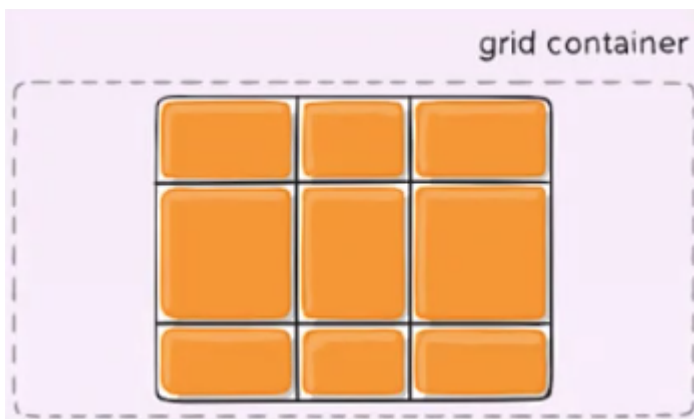
start



end



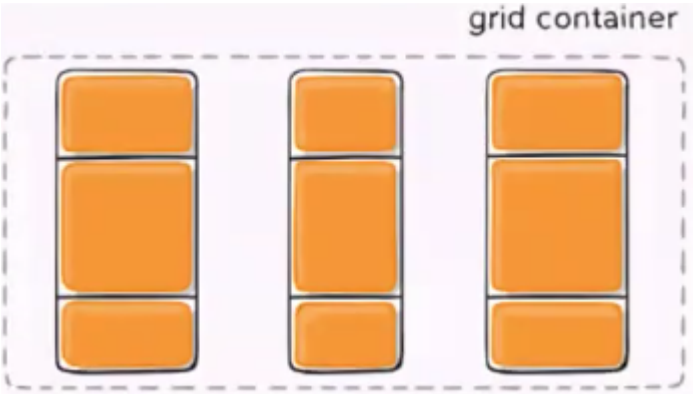
center



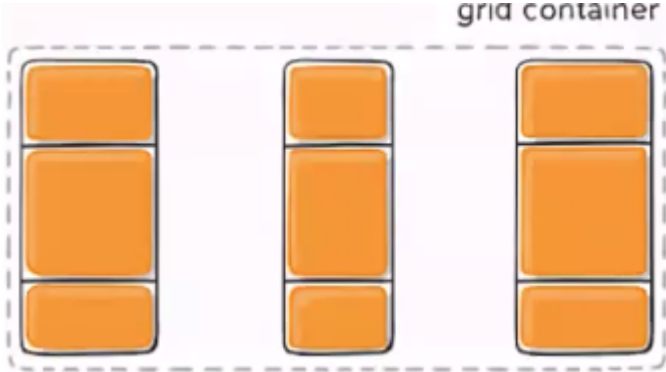
stretch



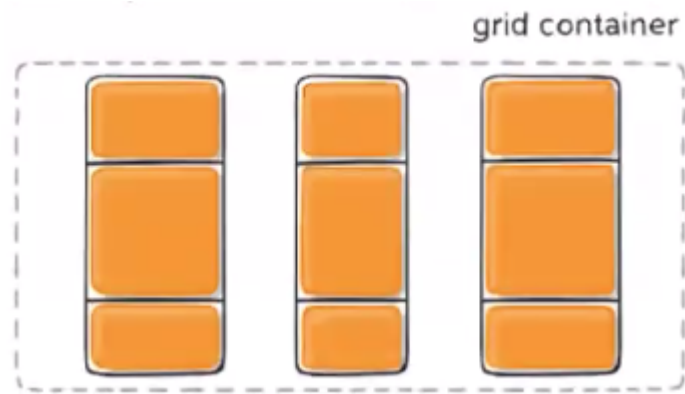
space-around



space-between



space-evenly



¡Ojo! Si intentamos probar directamente esto con el ejemplo anterior, veríamos que no tiene efecto, ¿Por qué? Porque tenemos configurado nuestro *grid* para que ocupe todo el espacio.

Vamos a cambiar el ancho de las columnas, en vez de decirle en algunas que tome todo el tamaño disponible le vamos a decir que tome cien *pixeles*

`./src/styles.css`

```
.grid-container {
  display: grid;
  - grid-template: repeat(2, 1fr) / repeat(3, 1fr);
+ grid-template: repeat(2, 100px) / repeat(3, 100px);
  border: 1px solid black;
  justify-items: start;
  align-items: center;
}
```

Para visualizar mejor los elementos, vamos a darle un *padding* a cada item de 25 *pixeles*

`./src/styles.css`

```
.item {
  border: 1px solid black;
  background-color: khaki;
  display: flex;
  justify-content: center;
+ padding: 25px;
}
```

Y vamos a eliminar el *align-self* del item 2:

`./src/styles.css`

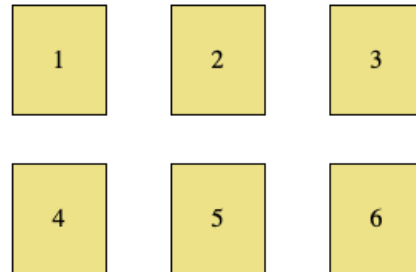
```
#item2 {
- align-self: start;
}
```

Vamos ahora a jugar con la propiedad *justify-content*, por defecto está en *start*

Vamos a colocarlo en *end*

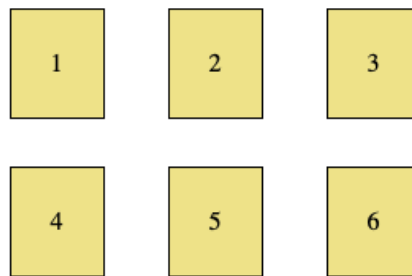
```
.grid-container {
  display: grid;
  grid-template: repeat(2, 100px) / repeat(3, 100px);
  border: 1px solid black;
  justify-items: start;
```

```
align-items: center;  
+ justify-content: end;  
}
```



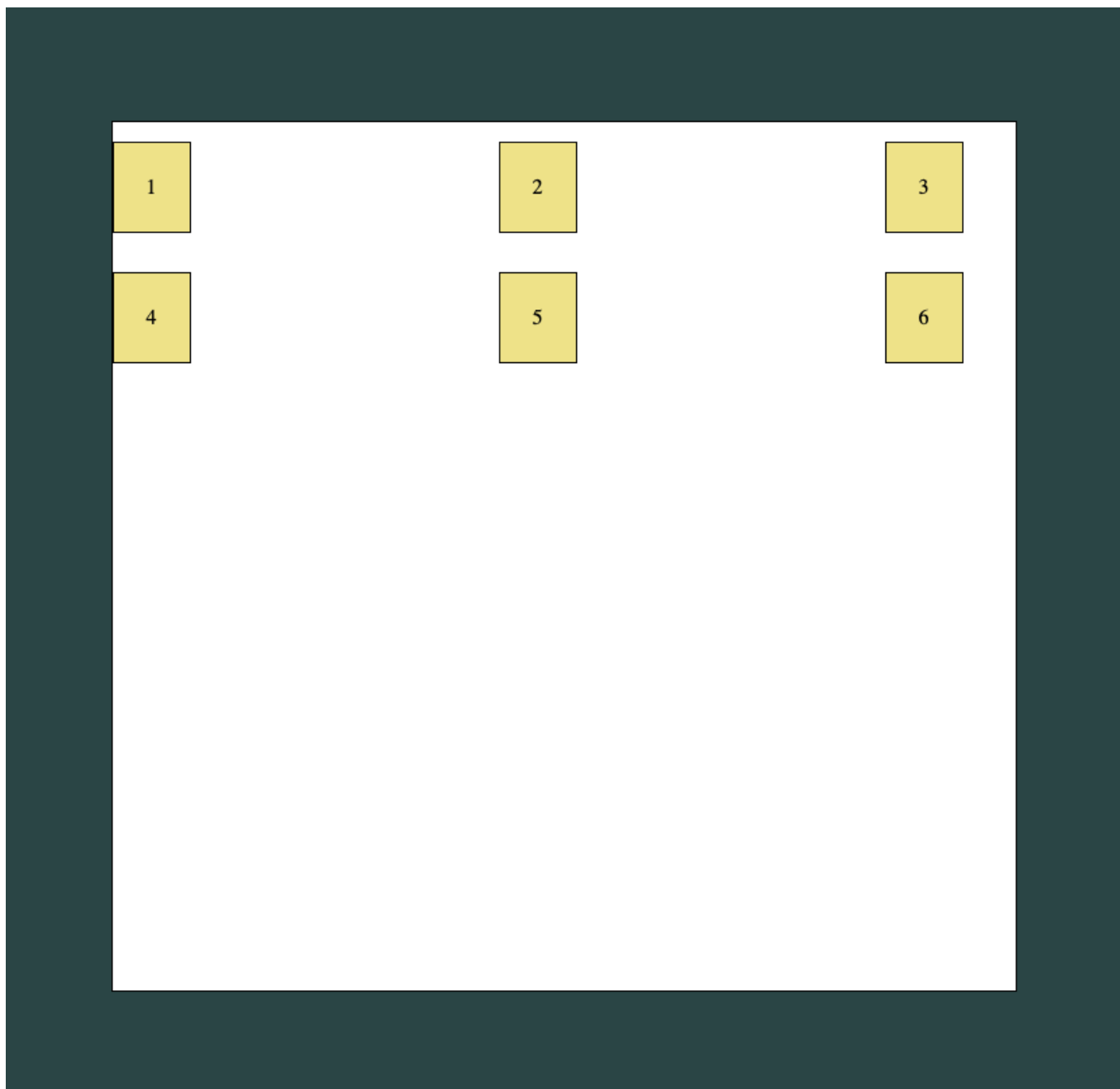
Ahora en *center*

```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 100px) / repeat(3, 100px);  
  border: 1px solid black;  
  justify-items: start;  
  align-items: center;  
- justify-content: end;  
+ justify-content: center;  
}
```



Vamos a probar *space-between*

```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 100px) / repeat(3, 100px);  
  border: 1px solid black;  
  justify-items: start;  
  align-items: center;  
  - justify-content: center;  
  + justify-content: space-between;  
}
```

¿Te animas tu a probar el resto de valores?

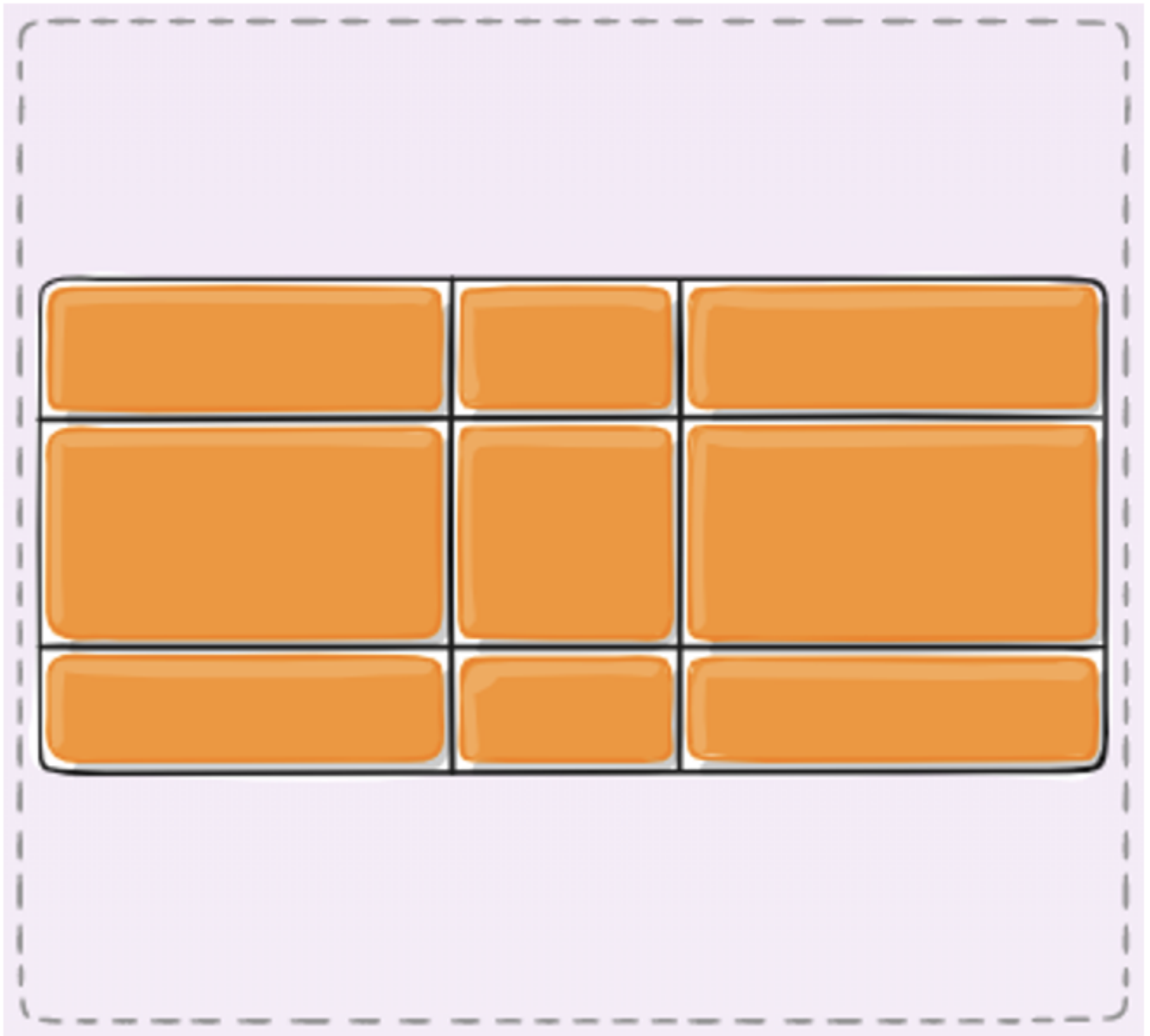
Align-content

En el apartado anterior vimos como *justify-content* nos permitía repartir horizontalmente las filas del Grid, con *align-content* podemos repartir verticalmente las filas del grid.

Veamos los valores que permite:

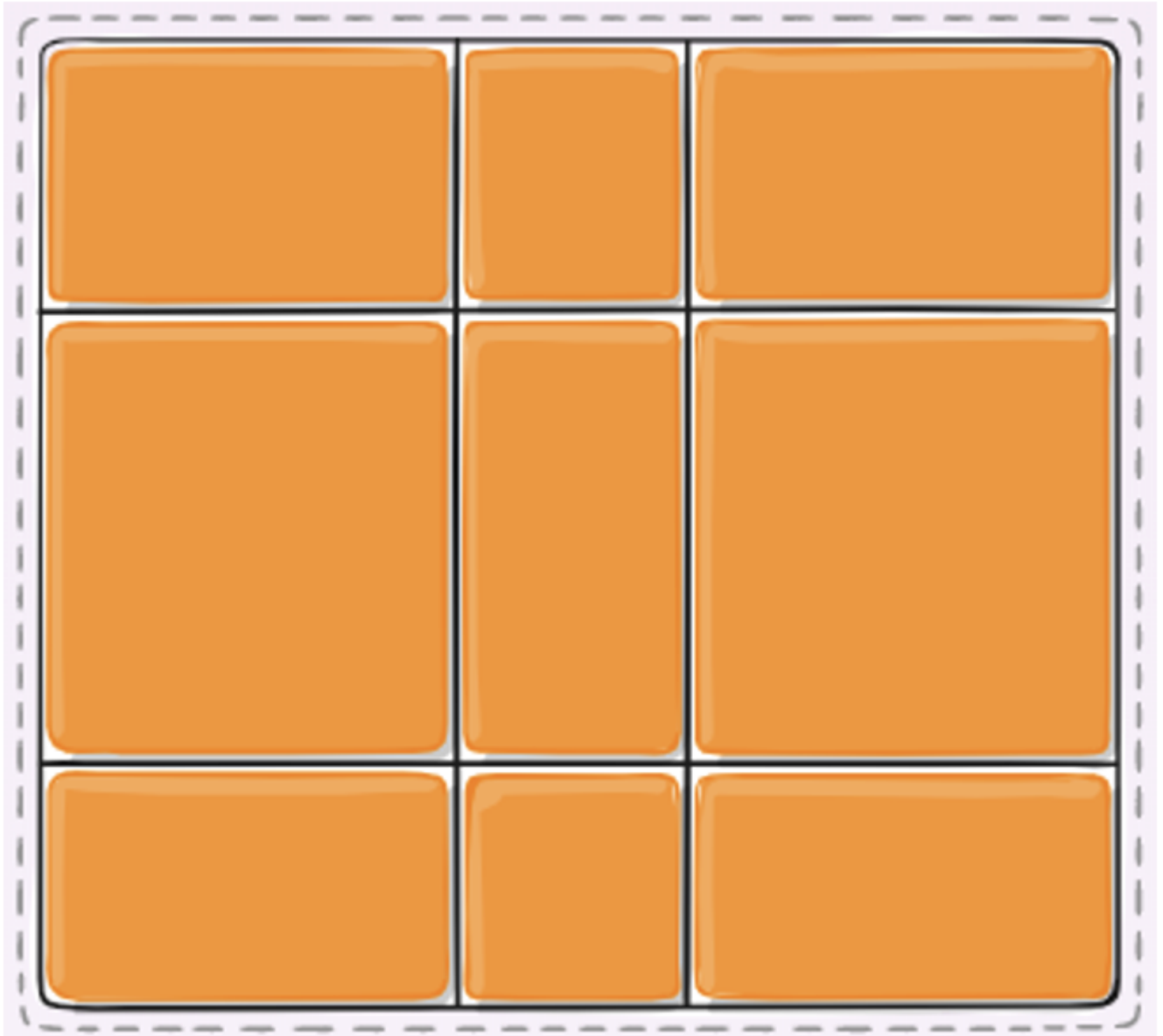
start

center



end

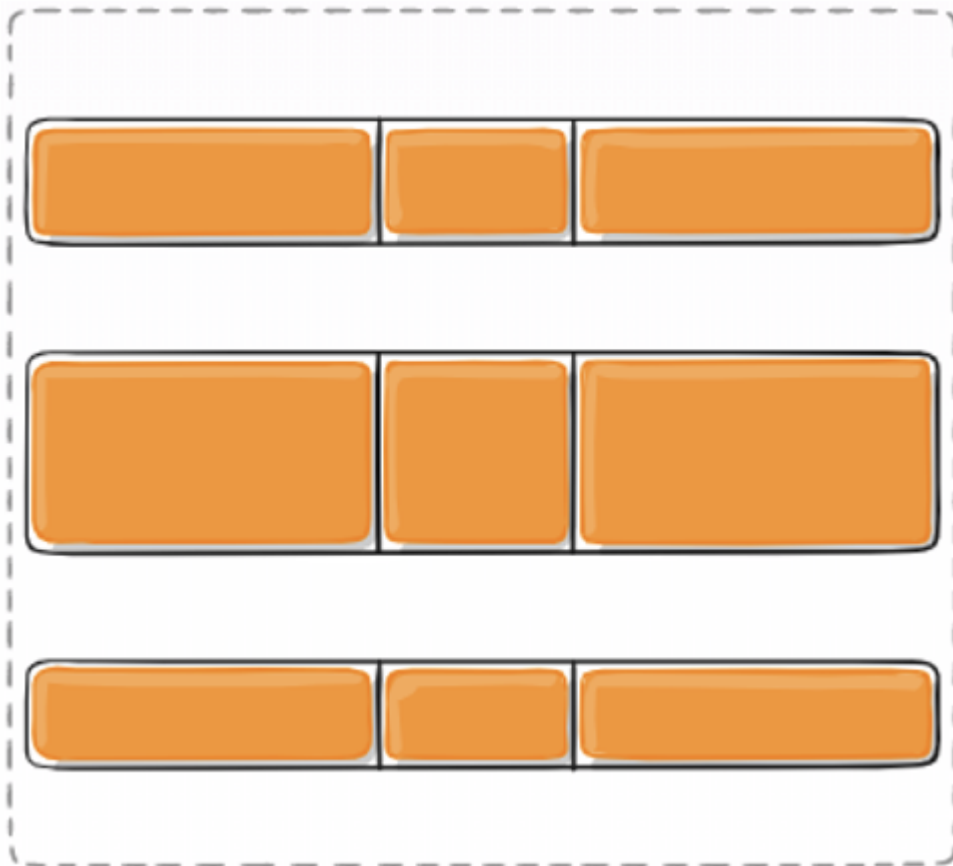
stretch



space-around

space-between

space-evenly



Si queremos probarlo en el ejemplo que tenemos sólo tendríamos que añadir la propiedad *align-content* e indicar alguno sus valores validos.

Por ejemplo para que los items se empiecen a mostrar abajo del todo:

`./src/styles.css`

```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 100px) / repeat(3, 100px);  
  border: 1px solid black;  
  justify-items: start;  
  align-items: center;  
  justify-content: space-between;  
+  align-content: end;  
}
```

También contamos con un *short-hand* para especificar de una tacada *justify-content* y *align-content* y es el *place-content*

Es decir podríamos realizar la siguiente sustitución:

```
.grid-container {  
  display: grid;  
  grid-template: repeat(2, 100px) / repeat(3, 100px);  
  border: 1px solid black;
```

```
justify-items: start;
align-items: center;
- justify-content: space-between;
- align-content: end;
+ place-content: end space-between;
}
```

El uso de los shorthands es más a gusto del consumidor, algunas veces es lioso de usar, ya que nos puede costar acordarnos de que valor representaba a que entrada.

CSS Grid vs Flexbox

Ahora nos sale la duda ¿ Cuando usar uno u otro? No hay reglas muy claras, veamos algunos motivos para decidir depende en que caso:

- Flexbox es más veterano que CSS Grid, esto es un punto a favor si tienes que dar soporte a IE11.
- Flexbox es unidimensional (un eje principal, otro secundario), mientras que CSS grid es bidimensional (cuento con una cuadrícula con filas y columnas), depende del tipo de layout que vayas a montar esto puede ser una buena pista, también mencionar que eso no quiere decir que con flexbox no puedes montar estilos de tipo cuadrícula, un ejemplo de esto lo puedes ver con Bootstrap 4, han montado diseños bidimensionales usando flexbox.
- CSS Grid se centra en la colocación precisa del contenido, cada Elemento es una celda de la cuadrícula, con flexbox es difícil predecir el comportamiento en ciertas combinaciones.
- Flexbox se centra en el flujo del contenido en lugar de su ubicación. Las anchuras o alturas de los elementos flexibles vienen determinadas por el contenido del propio elemento. Los elementos flexibles se amplían y se encogen con su contenido interno y con el espacio disponible.
- Flexbox te permite asignar espacio y alinear elementos de forma flexible. Flexbox te permite decidir cómo debe comportarse tu contenido cuando hay demasiado o no hay suficiente espacio en la pantalla.

Fuente de esta información: <https://webdesign.tutsplus.com/es/articles/flexbox-vs-css-grid-which-should-you-use--cms-30184>

Anexo - Recursos flexbox

Guía:

[A complete guide to CSS Grid](#)

Juegos

[CSS Grid Garden](#)