ÍNDICE

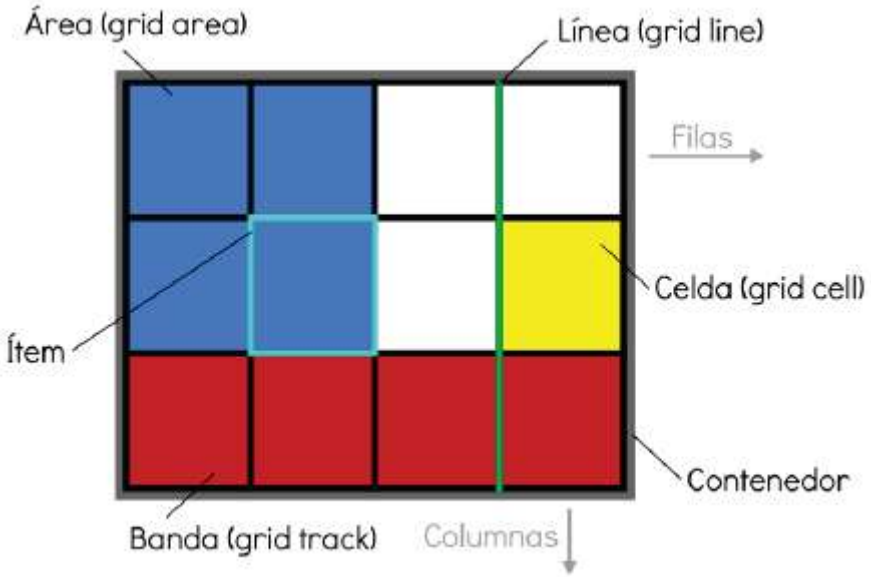
Grid CSS

El sistema **flexbox** es una gran mejora, sin embargo, está orientado a estructuras de una sola dimensión, por lo que aún necesitamos algo más potente para estructuras web. Con el paso del tiempo, muchos frameworks y librerías utilizan un **sistema grid** donde definen una cuadrícula determinada, y modificando los nombres de las clases de los elementos HTML, podemos darle tamaño, posición o colocación.

Grid CSS nace de esa necesidad, y recoge las ventajas de ese sistema, añadiéndole numerosas mejoras y características que permiten crear rápidamente cuadrículas sencillas y potentes de forma prácticamente instantánea.

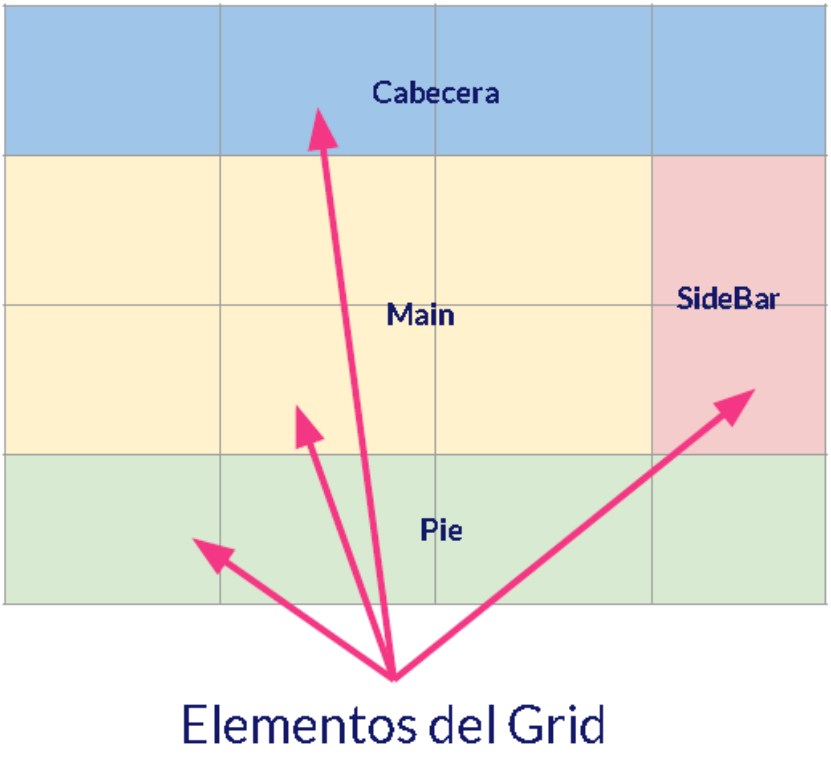
1. Conceptos

Antes de comenzar con **Grid CSS**, quizás sería conveniente dominar el sistema **Flexbox**, ya que Grid toma la filosofía y bases de él. Para utilizar **Grid CSS** necesitaremos tener en cuenta una serie de conceptos que utilizaremos a partir de ahora y que definiremos a continuación:



- **Contenedor**: Existe un elemento padre que es el contenedor que definirá la cuadrícula o rejilla.
- **Ítem**: Cada uno de los hijos que contiene la cuadrícula (elemento contenedor).
- **Celda (grid cell)**: Cada una de las celdas (unidad mínima) de la cuadrícula.
- **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.

Visto de otra manera, una página se puede maquetar de la siguiente forma:



Para utilizar cuadrículas **Grid CSS**, trabajaremos bajo el siguiente escenario:

```
<main class="grid"> <!-- Contenedor -->
  <div class="a">1</div> <!-- Cada uno de los ítems -->
  <div class="b">2</div>
  <div class="c">3</div>
  <div class="d">4</div>
</main>
```

El código HTML asociado al CSS a lo largo de este documento tendrá una estructura similar. Es decir, el contenedor con **display: grid** y los ítems, que serán los elementos del contenedor.

Para activar la cuadrícula **grid** hay que utilizar sobre el elemento contenedor la propiedad **display** y especificar el valor **grid** o **inline-grid**. Este valor incluye en cómo se comportará la cuadrícula con el contenido exterior. El primero de ellos permite que la cuadrícula aparezca encima/debajo del contenido exterior (en bloque) y el segundo de ellos permite que la cuadrícula aparezca a la izquierda/derecha (en línea) del contenido exterior.

inline-grid	Establece una cuadrícula con ítems en línea, de forma equivalente a inline-block.
grid	Establece una cuadrícula con ítems en bloque, de forma equivalente a block.

Una vez elegido uno de estos dos valores, y establecida la propiedad **display** al elemento contenedor, hay varias formas de configurar nuestra cuadrícula grid. Comencemos con las propiedades que se aplican al elemento contenedor (padre).

2. Grid con filas y columnas

Tamaño fijo

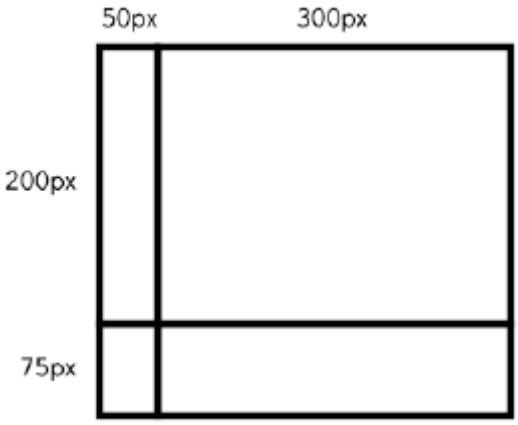
Es posible crear cuadrículas con un tamaño explícito. Para ello, sólo tenemos que usar las propiedades CSS **grid-template-columns** y **grid-template-rows**, que sirven para indicar las dimensiones de cada **celda** de la cuadrícula, diferenciando entre columnas y filas. Las propiedades son las siguientes:

grid-template-columns	[c1] [c2]...	Establece el TAMAÑO de cada columna (col 1, col 2...).
grid-template-rows	[f1] [f2]...	Establece el TAMAÑO de cada fila (fila 1, fila 2...).

Conociendo estas dos propiedades, asumamos el siguiente código CSS:

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

Esto significa que tendremos una cuadricula con **2 columnas** (la primera con 50px de ancho y la segunda con 300px de ancho) y con **2 filas** (la primera con 200px de alto y la segunda con 75px de alto). Ahora, dependiendo del número de ítems (elementos hijos) que tenga el contenedor **grid**, tendremos una cuadrícula de 2x2 elementos (4 ítems), 2x3 elementos (6 ítems), 2x4 elementos (8 ítems) y así sucesivamente. Si el número de ítems es impar, la última celda de la cuadrícula se quedará vacía.

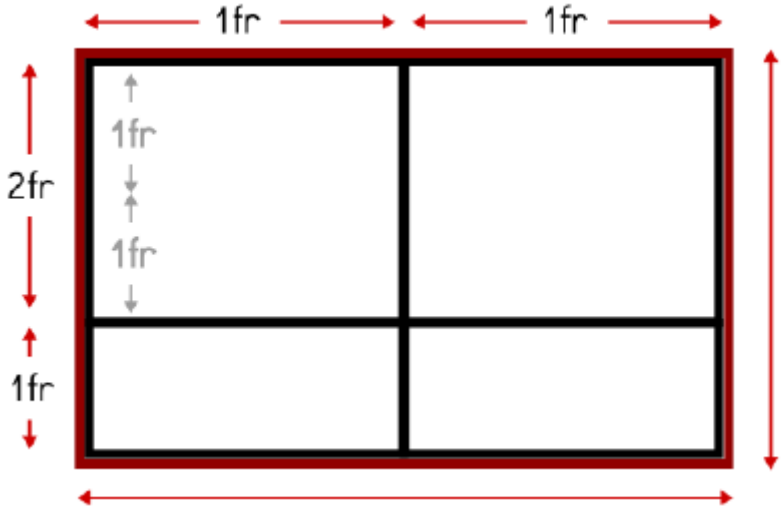


Unidad fr

En este ejemplo se han empleado **píxeles** como unidades de las celdas de la cuadrícula, sin embargo, también podemos utilizar otras unidades (e incluso combinarlas) como porcentajes, la palabra clave **auto** (que obtiene el tamaño restante) o la unidad especial **fr** (fraction), que simboliza una **fracción de espacio restante en el grid** según el tamaño definido en el contenedor. Veamos un código de ejemplo en acción:

```
#contenedor {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 2fr 1fr;
  width: 50%; /* La unidad fr calcula en función del ancho y alto */
  height: 300px;
}
```

Este nuevo ejemplo, se crea una cuadrícula de 2x2, donde el tamaño de ancho de la cuadrícula se divide en **dos columnas** (mismo tamaño de ancho para cada una), y el tamaño de alto de la cuadrícula se divide en **dos filas**, donde la primera ocupará el doble **(2fr)** que la segunda **(1fr)**. De esta forma, podemos tener un mejor control del espacio restante de la cuadrícula, y como utilizarlo.



Nota: Se pueden combinar varias unidades diferentes, pudiendo utilizar píxeles (px) y fracciones restantes (fr), porcentajes (%) y fracciones restantes (fr), o combinaciones similares con em y rem.

Filas y columnas repetitivas

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. Se puede utilizar la expresión **repeat()** para indicar repetición de valores, indicando el número de veces que se repiten y el tamaño en cuestión.

La expresión a utilizar sería la siguiente: **repeat([núm de veces], [valor o valores])**:

```
#contenedor {
  display: grid;
  grid-template-columns: 100px repeat(2, 50px) 200px;
  grid-template-rows: repeat(2, 50px 100px);
}
```

Asumiendo que tuviéramos un contenedor grid con 8 ítems hijos (o más), el ejemplo anterior crearía una cuadrícula con **4 columnas** (la primera de 100px de ancho, la segunda y tercera de 50px de ancho y la cuarta de 200px de ancho). Por otro lado, tendría **4 filas** (dos repeticiones con 50px de alto y la segunda de 100px de alto, respectivamente).

El ejemplo anterior sería equivalente al código CSS siguiente:

```
#contenedor {
  display: grid;
  grid-template-columns: 100px 50px 50px 200px;
  grid-template-rows: 50px 100px 50px 100px;
}
```

En un navegador podría quedar de la siguiente manera:

Item 1	Item 2	Item 3	Item 4
Item 5	Item 6	Item 7	Item 8
Item 9	Item 10	Item 11	Item 12
Item 13	Item 14	Item 15	Item 16

Cabe recordar, que el HTML asociado es el siguiente:

```
<main class="grid"> <!-- Contenedor -->
  <div class="item">Ítem 1</div> <!-- Cada uno de los ítems -->
  <div class="item">Ítem 2</div>
  <div class="item">Ítem 3</div>
  <div class="item">Ítem 4</div>
  <div class="item">Ítem 5</div>
  <div class="item">Ítem 6</div>
  <div class="item">Ítem 7</div>
  <div class="item">Ítem 8</div>
  <div class="item">Ítem 9</div>
  <div class="item">Ítem 10</div>
  <div class="item">Ítem 11</div>
  <div class="item">Ítem 12</div>
  <div class="item">Ítem 13</div>
  <div class="item">Ítem 14</div>
  <div class="item">Ítem 15</div>
  <div class="item">Ítem 16</div>
</main>
```

Propiedad abreviada grid-template

La definición del grid se puede resumir con una única propiedad, en lugar de especificarlo por separado con filas y columnas. Esto se lleva a cabo con la palabra reservada **grid-template**

El formato es **grid-template: grid-template-rows / grid-template-columns;**

Por ejemplo:

```
#contenedor {
  display: grid;
  grid-template: 100px 1fr / repeat(2, 50px);
```

3. Grid con huecos

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, que evita los problemas clásicos de los modelos de caja (básicamente desbordamiento): los huecos

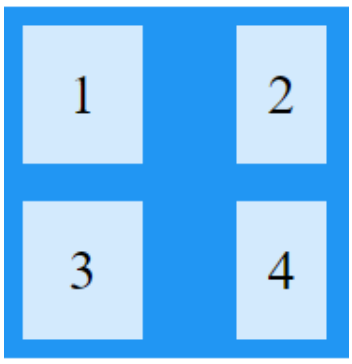
Para especificar los **huecos** (espacio entre celdas) podemos utilizar las propiedades **grid-column-gap** y/o **grid-row-gap**. En ellas indicaremos el tamaño de dichos huecos:

grid-column-gap	Establece el TAMAÑO de los huecos entre columnas (líneas verticales).
grid-row-gap	Establece el TAMAÑO de los huecos entre filas (líneas horizontales).

Un ejemplo sería el siguiente:

```
#contenedor {
  display: grid;
  grid-template-columns: 4rem 3rem;
  grid-template-rows: 4rem 3rem;
  grid-column-gap: 50px;
  grid-row-gap: 20px;
}
```

El resultado sería el siguiente:



Existe una propiedad de atajo para **grid-column-gap** y **grid-row-gap**, permitiéndonos la posibilidad de no tener que indicarlas por separado. La propiedad en cuestión sería **grid-gap** y se utilizaría de la siguiente forma:

Opción 1

```
.grid {
  /* grid-gap: <row-gap> <column-gap> */
  grid-gap: 20px 80px;
}
```

Opción 2

```
.grid {
  /* grid-gap: <row-column-gap> */
  grid-gap: 40px;
  /* Equivalente a grid-gap: 40px 40px */
}
```

4. Ajuste automático de celdas

Las propiedades grid-auto-rows y grid-auto-columns

Es posible utilizar las propiedades **grid-auto-columns** y **grid-auto-rows** para darle un tamaño automático a las celdas de la cuadrícula. Para ello, sólo hay que especificar el tamaño deseado en cada una de las propiedades.

grid-auto-rows	TAMAÑO	Indica el tamaño automático de ancho que tendrán las filas.
grid-auto-columns	TAMAÑO	Indica el tamaño automático de ancho que tendrán las columnas.

Estas propiedades son especialmente útiles si se da el caso que definimos un tamaño fijo de filas y columnas, sin embargo, en el HTML se añaden más de aquellas definidas con **grid-template-columns** y **grid-template-rows.** Por defecto, estas nuevas filas se adaptan al contenido. Sin embargo, con **grid-auto-rows** y **grid-auto-columns** aquellas celdas adicionales tomarán las dimensiones especificadas con estas propiedades.

Por ejemplo:

```
#contenedor {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 2fr 1fr;
  grid-auto-rows: 50px;
}
```

Si el HTML fuese el siguiente, solamente hemos definido 4 celdas (2 filas y 2 columnas) Las dos nuevas celdas se añadirán debajo con el tamaño especificado en **grid-auto-rows** (50 px en este caso)

```
<main class="grid"> <!-- Contenedor -->
  <div class="a">Ítem 1</div> <!-- Cada uno de los ítems -->
  <div class="b">Ítem 2</div>
  <div class="c">Ítem 3</div>
  <div class="d">Ítem 4</div>
  <div class="e">Ítem 5</div>
  <div class="f">Ítem 6</div>
</main>
```

El resultado en el navegador sería como se muestra debajo.

Item 1	Item 2
Item 3	Item 4
Item 5	Item 6

El grid por defecto crece de arriba a abajo. Es decir, en el momento que no hay más columnas definidas se pasa a la siguiente fila. Por tanto, ¿en qué situación sería útil emplear **grid-auto-columns** si nunca va a haber más columnas que las definidas? Existe una propiedad denominada **grid-auto-flow** que permite que un grid "crezca" en sentido horizontal y que veremos más adelante.

Los valores minmax, auto-fill y auto-fit

Existen varias posibilidades de que el navegador calcule los tamaños o número de celdas en con una serie de palabras clave en las propiedades **grid-template-columns** y **grid-template-rows.**

Por un lado, podemos establecer tamaños dinámicos con **minmax**. Para ello, vamos a tomar como punto de partida el siguiente código.

```
#contenedor {
  display: grid;
  width: 100%;
  grid-template-columns: repeat(4, minmax(100px, 1fr));
  grid-template-rows: repeat(2, 50px 100px);
}
```

En este ejemplo, se definen 4 columnas en un contenedor que ocupa el 100%. En función de la resolución y el ancho del navegador, tomamos que cada columna tendrá como mínimo 100px. Si quedara espacio restante, las cuatro columnas se distribuyen de forma equitativa en el espacio del contenedor (1fr).

Item 1	Item 2	Item 3	Item 4
Item 5	Item 6	Item 7	Item 8
Item 9	Item 10	Item 11	Item 12
Item 13	Item 14	Item 15	Item 16

Este valor resulta especialmente útil para un diseño adaptable a dispositivos de diferente tamaño.

En definitiva, **minmax** recibe dos parámetros, un tamaño máximo y otro mínimo. Uno de los valores debe ser absoluto y el otro relativo.

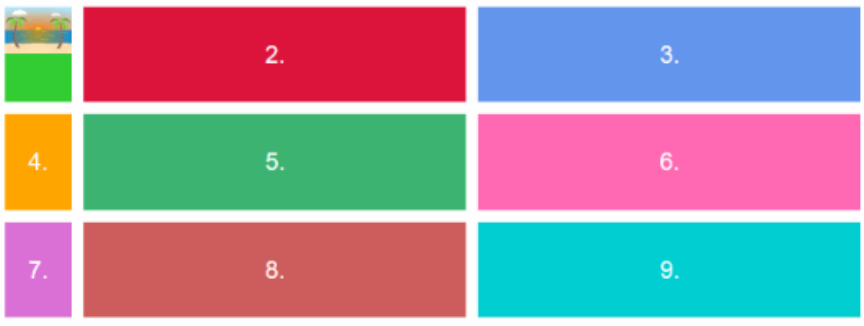
Otros ejemplos de minmax:

- **minmax**(200px, 1fr)
- **minmax**(400px, 50%)
- **minmax**(30%, 300px)
- **minmax**(100px, max-content)
- **minmax**(min-content, 400px)
- **minmax**(max-content, auto)
- **minmax**(auto, 300px)
- **minmax**(min-content, auto)

A continuación se explica el significado de algunas palabras clave empleadas como parámetro:

- **max-content:** Representa la mayor contribución max-content de los elementos de rejilla que ocupan la banda. Por ejemplo, en `grid-template-columns: max-content 1fr 1fr` la primera columna se adapta a la imagen, que es el elemento más grande.

- **min-content:** Representa la mayor contribución min-content de los elementos de rejilla que ocupan la banda. Por ejemplo, en `grid-template-columns: min-content 1fr 1fr` la primera columna se adapta a la imagen, que es el elemento más pequeño.



- **auto:** Como un máximo, idéntico a **max-content**. Como un mínimo representa el mayor tamaño mínimo, pero también tiene en cuenta **min-width** o **min-height**. Es decir, será el mínimo entre todos los elementos del grid y min-width o min-height.

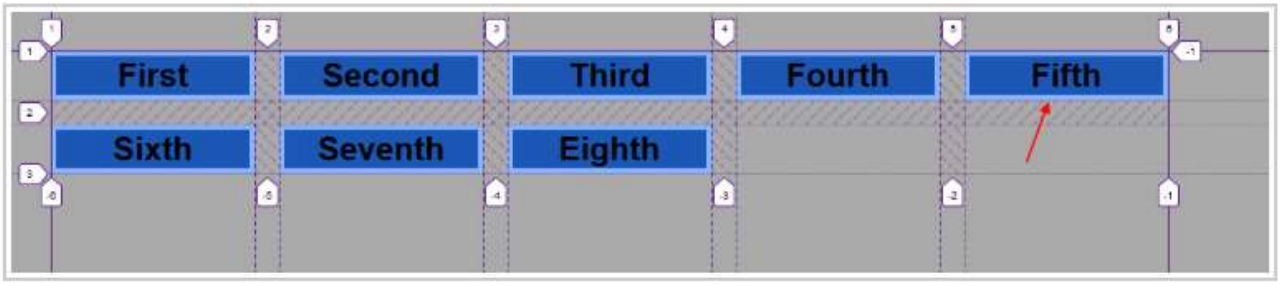
Hasta ahora todas las alternativas que se han visto parten de un número fijo de filas o columnas a la hora de definir un grid. Es decir, aunque se utilizara repeat, siempre sabíamos el número total de columnas.

Sin embargo, los valores de **auto-fill** y **auto-fit** permiten que el número de elementos del grid que recibe repeat como parámetro se calcule automáticamente.

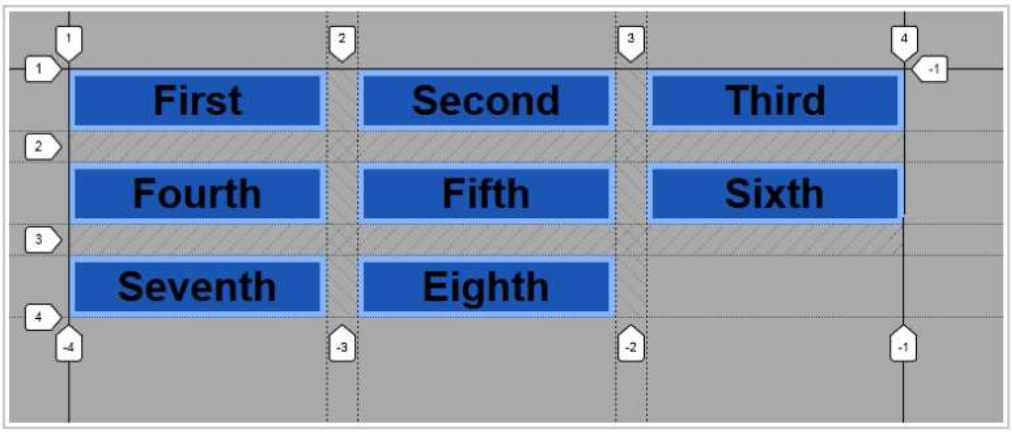
Por ejemplo, dado el siguiente código:

```
#contenedor {
  display: grid;
  width: 100%;
  grid-template-columns: repeat(auto-fill, 200px);
  grid-template-rows: repeat(2, 50px 100px);
}
```

Para un código HTML similar al que hemos visto con un contenedor e ítems, se generaría un resultado similar al de debajo en el que se crean tantas columnas de 200px como quepan.



Sin embargo, si redimensionamos las columnas se adaptan automáticamente al nuevo tamaño del contenedor.



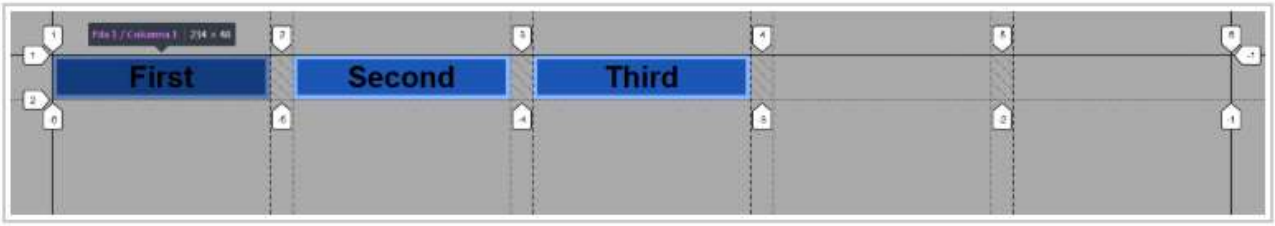
Si esto lo combinamos con minmax, se pueden obtener un código tan dinámico como el siguiente:

```
#contenedor {
  display: grid;
  width: 100%;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  grid-template-rows: repeat(2, 50px 100px);
}
```

El resultado sería similar al anterior, pero las columnas se calculan teniendo en el tamaño máximo que se pueden mostrar siempre que el mínimo sea 200px.

Hasta ahora se ha empleado solamente **auto-fill.** El valor auto-fit funciona exactamente igual en todos los casos excepto cuando falta espacio. En este caso la diferencia es la siguiente:

- **auto-fill:** Rellena la fila con el máximo de columnas. Crea columnas implícitas cuando quepan, esto se debe a que trata de llenar la fila con tantas columnas como pueda. Las nuevas columnas que son añadidas estarán vacías, pero ocuparán el espacio designado en su fila.



- **auto-fit:** Encaja las columnas disponibles en el espacio expandiéndolas tanto como sea necesario para ocupar el espacio disponible. El navegador gestiona esta situación después de llenar el espacio extra con columnas (como con auto-fill) y después colapsa las columnas vacías.

En definitiva, con auto-fit se colapsan columnas, mientras que auto-fill deja columnas vacías. Todo esto si se emplea minmax y sobra espacio. En caso contrario el comportamiento es casi siempre igual en ambos casos.

5. Grid por áreas

Mediante los **grids CSS** es posible indicar el nombre y posición concreta de cada área de una cuadrícula. Para ello utilizaremos la propiedad **grid-template-areas**, 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:

La sintaxis es la siguiente:

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.

De esta forma, es muy sencillo crear una cuadrícula altamente personalizada en apenas unas cuantas líneas de CSS, con mucha flexibilidad en la disposición y posición de cada área.

Por ejemplo, dado el siguiente código:

```
.grid {
   display: grid;
   grid-template-areas: "head head"
                        "menu main"
                        "foot foot";
}
.a{ grid-area: head; background-color: blue}
.b{ grid-area: menu; background-color: red}
.c{ grid-area: main; background-color: green}
.d{ grid-area: foot; background-color: orange}
```

Aplicando este código, conseguiríamos una cuadrícula donde:

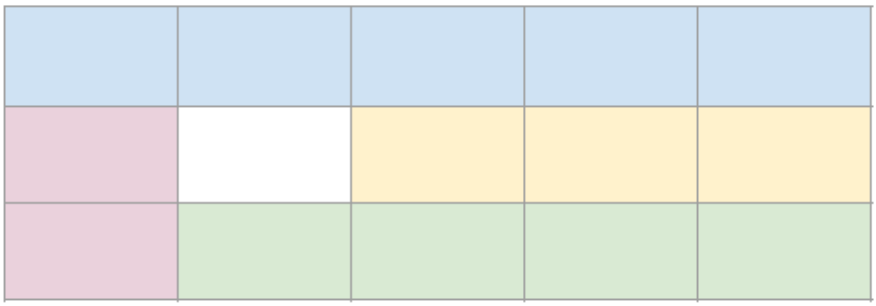
- El **Item 1**, la cabecera (head), ocuparía toda la parte superior.
- El **Item 2**, el menú (menu), ocuparía el área izquierda del grid, debajo de la cabecera.
- El **Item 3**, el contenido (main), ocuparía el área derecha del grid, debajo de la cabecera.
- El **Item 4**, el pie de cuadrícula (foot), ocuparía toda la zona inferior del grid.

IMPORTANTE: Ten en cuenta añadir contenido de texto en cada celda del grid. Si no se detectará como vacía y no se mostrará.

En la propiedad **grid-template-areas**, en lugar de indicar el nombre del área a colocar, también podemos indicar una palabra clave especial:

- La palabra clave **none**: Indica que no se colocará ninguna celda en esta posición.
- Uno o más puntos (.): Indica que se colocará una celda vacía en esta posición.

El resultado sería similar a la siguiente imagen:



```
.container {
 grid-template-columns: repeat(5,20%);
 grid-template-rows: repeat(3,100px);
 grid-template-areas:
   "cab cab cab cab cab"
   "menu . main main main"
   "menu pie pie pie pie";
}
```

```
{
  grid-area: main;
}
```

```
{
  grid-area: cab;
}
```

```
{
  grid-area: pie;
}
```

```
{
  grid-area: menu;
}
```

Hueco

6. Orden manual de elementos del grid

Las propiedades grid-column y grid-row

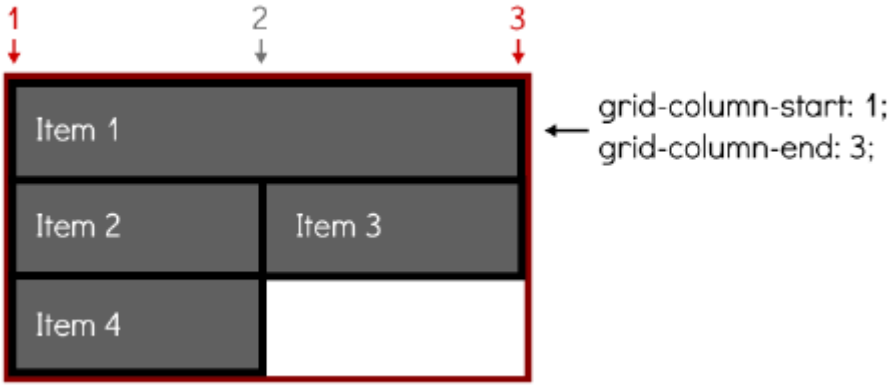
Hasta ahora hemos visto que los ítems ocupan el lugar que les corresponde según el flujo de posicionamiento de la página de acuerdo con el CSS. Sin embargo, existen una serie de propiedades que permiten alterar el orden de los elementos dentro de un grid. Estas propiedades son las siguientes:

grid-column-start	Indica en qué columna empezará el ítem de la cuadrícula.
grid-column-end	Indica en qué columna terminará el ítem de la cuadrícula.
grid-row-start	Indica en qué fila empezará el ítem de la cuadrícula.
grid-row-end	Indica en qué fila terminará el ítem de la cuadrícula.

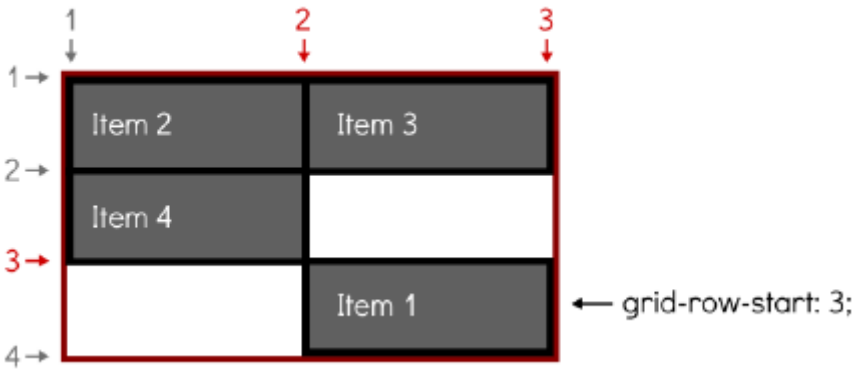
Con dichas propiedades, podemos indicar el siguiente código CSS sobre el primer ítem de una cuadrícula de 4 ítems:

```
.grid {
    display: grid;
    grid-template-columns: repeat(2, 50px);
    grid-template-rows: repeat(2, 20px);
}
.a{
    grid-column-start: 1;
    grid-column-end: 3;
}
```

De esta forma, tenemos una cuadrícula de 4 elementos, en la variamos los valores de forma que tomen posiciones diferentes, como por ejemplo, si indicamos que el ítem 1 debe comenzar en la columna 1, pero acabar en la columna 3 (que acabe en la tercera se refiere en la práctica a que ocupa la hipotética primera y segunda celda):



Además, podemos desplazar cualquier elemento desde su posición original simplemente alterando el orden con estas propiedades. Por ejemplo, vamos a mover el “Item 1” a la tercera fila.

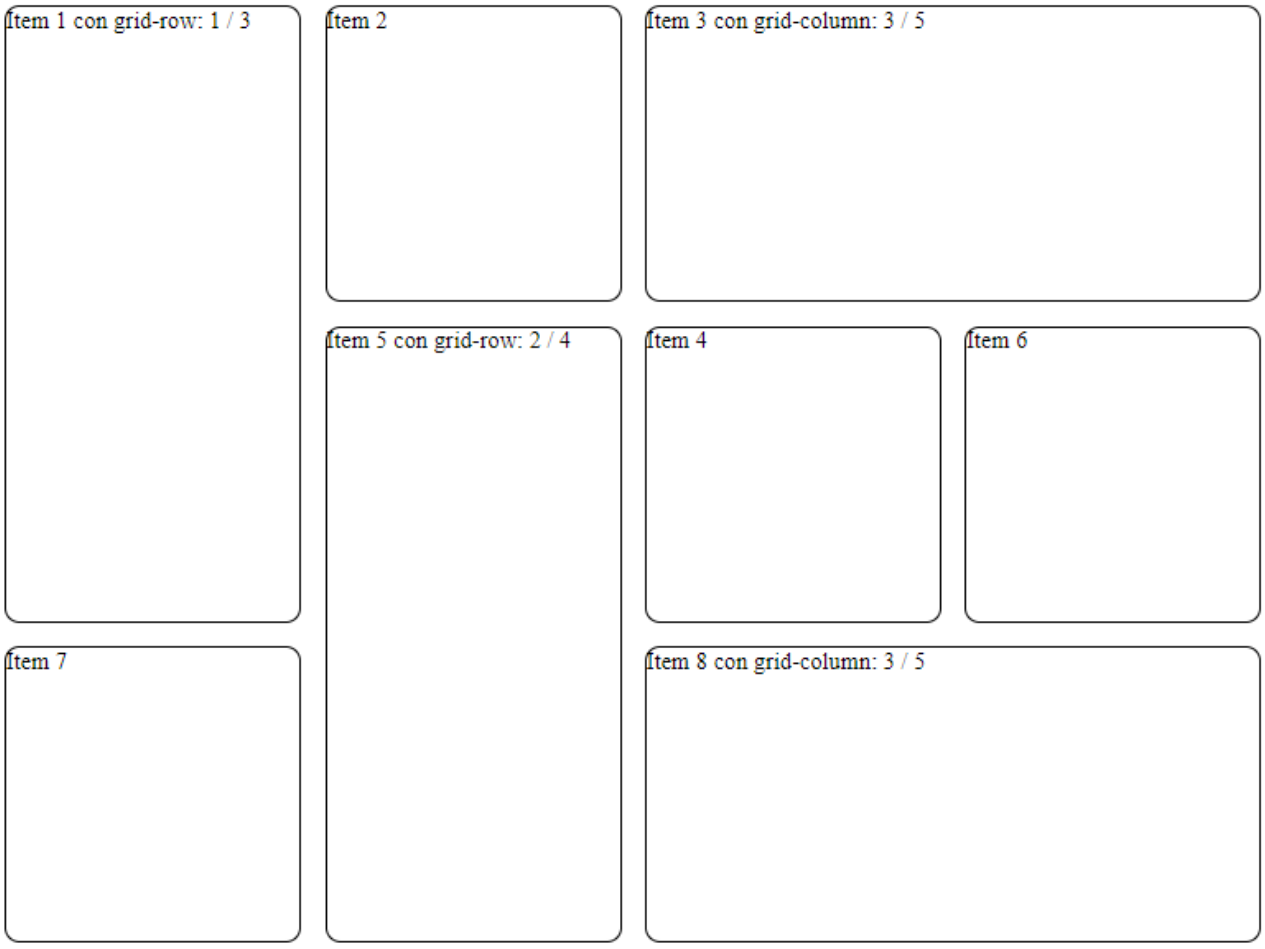


Si la posición nueva “desborda” las dimensiones definidas para las celdas del grid, entonces cogerá los tamaños de **grid-auto-rows** o **grid-auto-columns** y en el caso se no definirse estas propiedades se adaptará al contenido.

Las propiedades anteriores se pueden abreviar con **grid-column** y **grid-row** con el siguiente formato

- grid-column: grid-column-start / grid-column-end
- grid-row: grid-row-start / grid-row-end

Por ejemplo, en un grid de 4 columnas de 200px y 3 filas de 200px el resultado sería el siguiente. El HTML indica en qué columnas se están aplicando estos estilos.



Aplicando span

También es posible utilizar la palabra clave **span** seguida de un número, que indica que abarque un número determinado de columnas o filas.

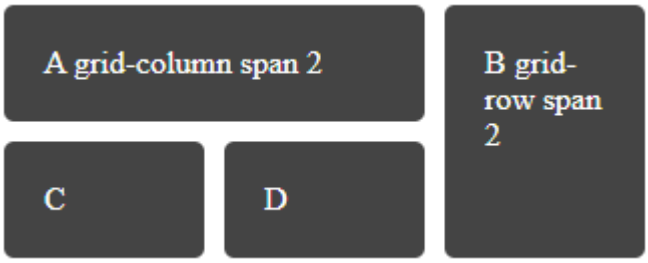
Como último parámetro se refiere a que se extiende el número de celdas que le indicamos.

```css
.grid {
    display: grid;
    grid-template-columns: repeat(3, 100px);
}

.a{
    grid-column: 1 / span 2;
}

.b{
    grid-column-start: 3;
    grid-row: 1 / span 2;
}
```

El resultado sería el siguiente con 4 ítems:



Si span fuese el primer parámetro en un grid con 6 columnas de 100px lo que hace es contar por el final. Es decir, con span 1 empieza en la penúltima columna (número 5), con span 2 en la columna 4 y así sucesivamente. El ejemplo de debajo se extiende entre las columnas 4 y 7. Es decir, **span** 2 indica la columna 4 y **grid-column-end** a 7 que abarcaría hasta la columna 6 incluida.

Orden de filas y columnas con nombre

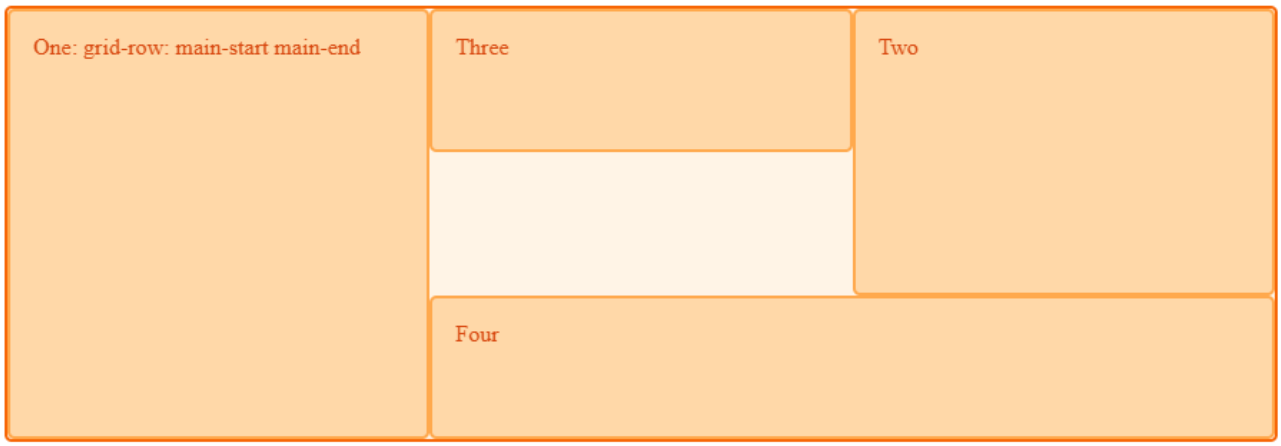
Aún no lo hemos visto, pero cuando se define un grid con **grid-template-rows** y **grid-template-columns** es posible asignar nombres en lugar de emplear el número de fila o columna como se ha hecho con las propiedades de este apartado.

La forma de definir los nombres sería la siguiente. Cada etiqueta se refiere al tamaño que se ubica a la derecha. La última etiqueta se refiere simplemente al final de las filas o columnas y no está asociada a ninguna celda real. Por ejemplo, cuando se asigna **grid-column: 1 / 4** se refiere en realidad a las tres primeras columnas. Así no tendríamos que hacer referencia a una cuarta columna que en teoría está fuera del rango y para indicar el límite. Con identificadores resulta más intuitivo.

Los nombres **siempre van entre corchetes [].**

```
.grid {
   display: grid;
   grid-template-columns:  [main-start]  1fr  [content-start]  1fr [content-end] 1fr [main-end];
   grid-template-rows:  [main-start]  100px  [content-start]  100px [content-end] 100px [main-end];
}
```

Como resultado, si tenemos un ítem denominado “One”, con **grid-row: main-start / main-end** estamos definiendo el equivalente a los valores 1 / 4. Como decíamos, [main-end] referencia solamente al final de las columnas (al igual que el valor 4, que se refiere al límite en un grid de tres columnas de 1fr).



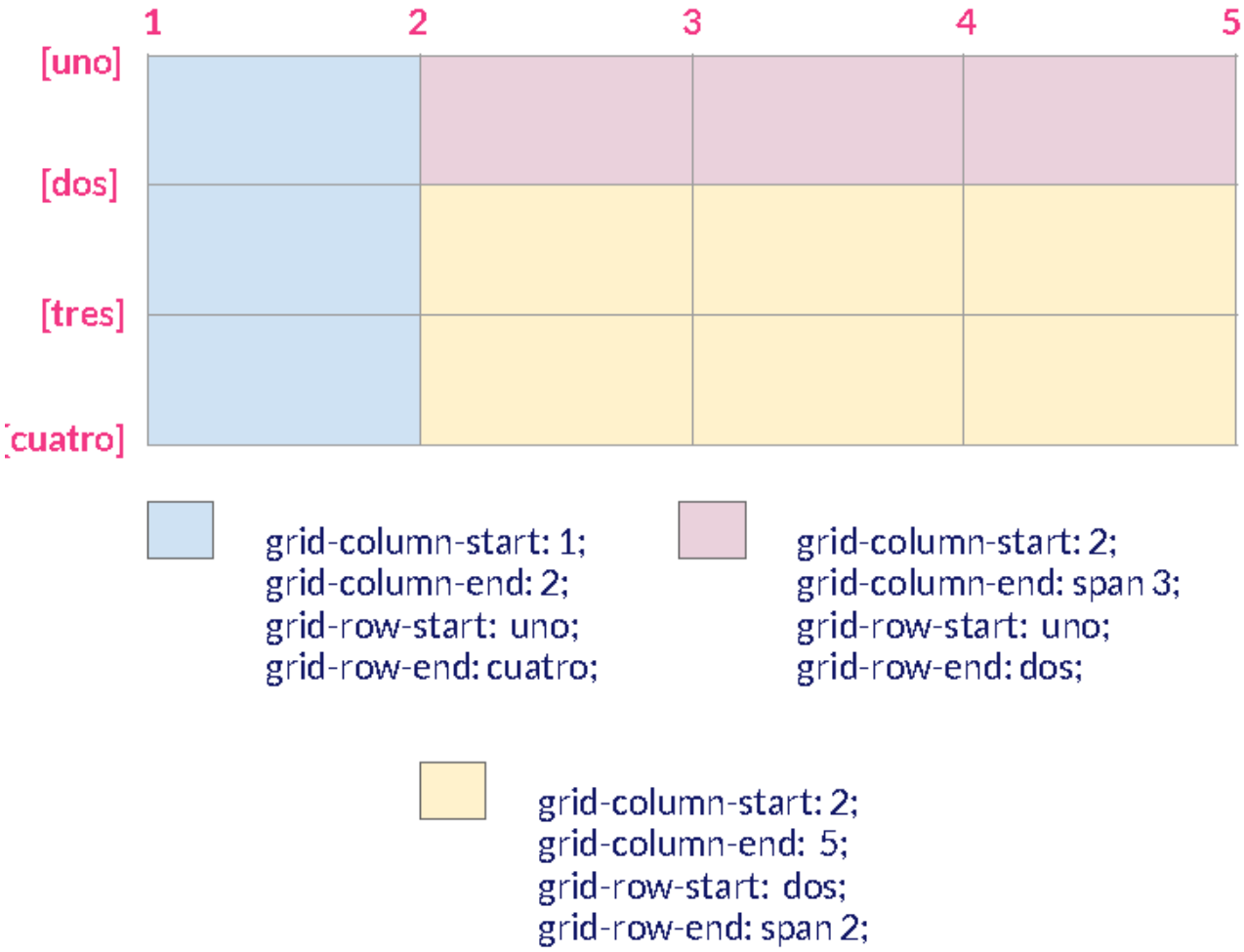
Con repeat también es posible asignar nombres duplicados

```
.grid {
   display: grid;
   grid-template-columns: repeat(12, [col-start] 1fr);
}
```

A la hora de hacer referencia a estos ítems, habría que indicar el número de celda con el mismo nombre. Por ejemplo, el primer elemento es **col-start 1** (para el primero no hace falta el orden y también valdría col-start), mientras que el quinto elemento denominado con **col-start** sería [col-start] 5 como se indica debajo.

```
.item-1 {
    grid-column: col-start / col-start 5
}
```

A continuación se muestra un último ejemplo que abarca todo lo estudiado de estas propiedades:



Propiedad order

Otra manera de cambiar el orden de los elementos de un grid es mediante la propiedad **order**, cuyo funcionamiento es muy similar a flexbox y se aplica directamente a cada ítem.

Supongamos el grid de debajo con cuatro columnas:

A cada uno de los botones se les especifica el orden de la siguiente manera siempre que a su contenedor se le aplique **display: grid** y la distribución de columnas que corresponda.

```
button:nth-of-type(1) {
    order: 3;
}
button:nth-of-type(2) {
    order: 3;
}
button:nth-of-type(3) {
    order: 3;
}
button:nth-of-type(4) {
    order: 3;
}
```

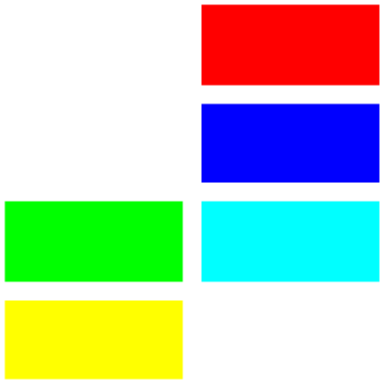
7. Dirección del grid

Hemos visto que un grid por defecto crece de arriba a abajo, siguiendo el flujo del diseño normal de una página. Sin embargo, es posible cambiar este comportamiento con **grid-auto-flow**. Los valores que admite son los siguientes:

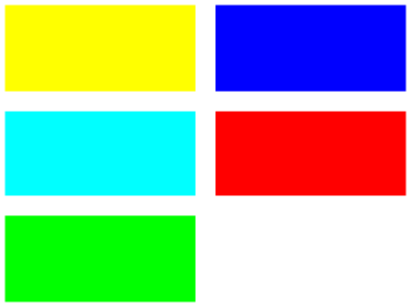
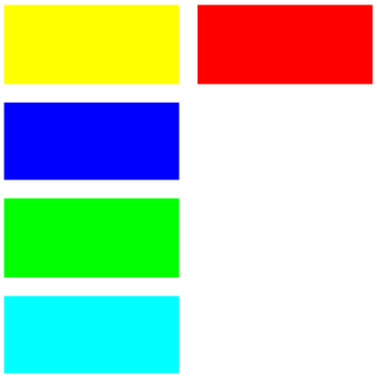
grid-auto-flow: row	El grid crece de arriba a abajo añadiendo filas cuando se alcanza aquellas definidas en **grid-template-rows**
grid-auto-flow: column	El grid crece de arriba a abajo añadiendo filas cuando se alcanza aquellas definidas en **grid-template-columns**
dense	Se puede añadir a los valores row y column de la forma **grid-auto-flow: row dense** o **grid-auto-flow: column dense**). Si se altera el orden de los elementos con **grid-column** o **grid-row** trata de rellenar los huecos con los ítems que quedan debajo, en la medida de lo posible.

El comportamiento con cada opción sería el siguiente:

El comportamiento de dense sería el siguiente. Supongamos que alteramos el orden de algunos elementos de un grid con dos columnas y cuatro filas empleando **grid-column** y **grid-row**. El primer ítem (el verde) se desplaza a la fila 3 con **grid-row-start: 3**, mientras que el cuarto ítem (el rojo) se desplaza a la columna 2 con **grid-column-start: 2**. Tenemos el siguiente resultado en función de **grid-auto-flow**

grid-auto-flow: row	grid-auto-flow: column
	

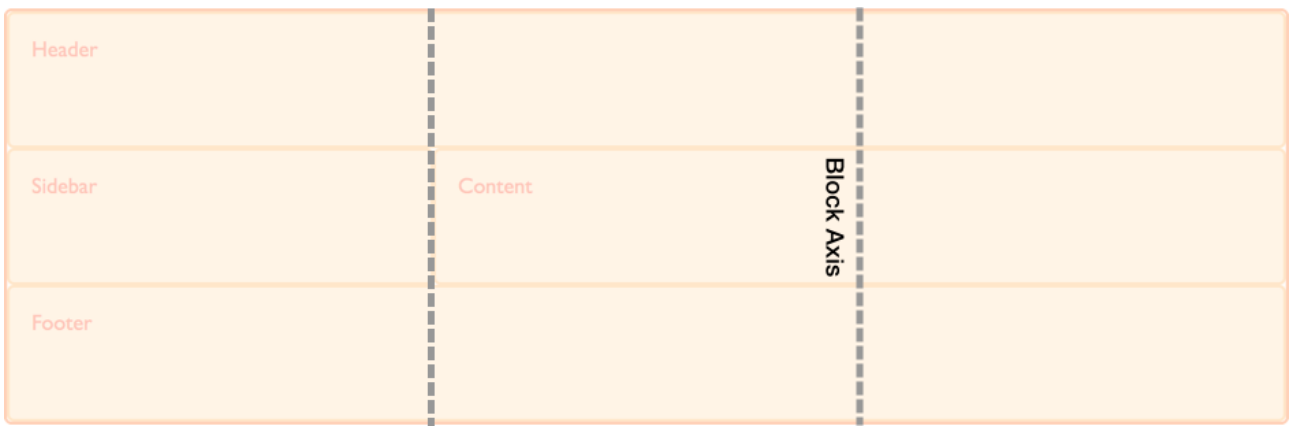
Los huecos en blanco que no rellenamos con ítems se solucionarían con dense.

grid-auto-flow: row dense	grid-auto-flow: column dense
	

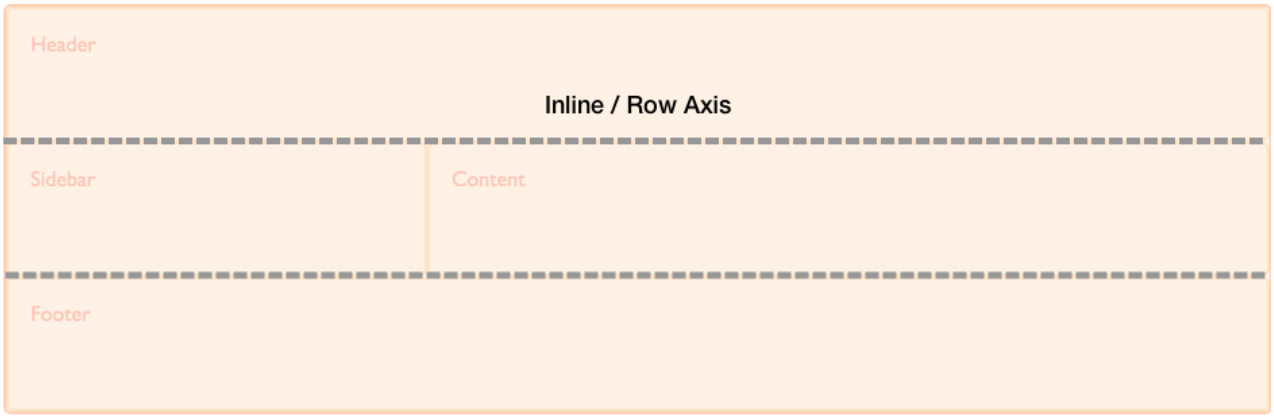
8. Alineación del grid

Al trabajar con el diseño en grid, tenemos dos ejes disponibles para alinear los elementos dentro de cada celda: el eje del bloque (block axis) y el eje en línea (inline axis).

El eje de bloque es básicamente el eje vertical.



El eje de línea se corresponde con el horizontal



Por cada celda que se crea en un grid, es posible alinear el contenido siguiendo estos ejes y con las propiedades que vamos en los próximos apartados.

Alineación horizontal y vertical de ítems

Las dos propiedades que se exponen a continuación permiten alinear cada ítem dentro de su celda correspondiente siguiendo los ejes horizontal y vertical, respectivamente.

justify-items	start | end | center | **stretch**	Distribuye los elementos en el eje horizontal.
align-items	start | end | center | **stretch**	Distribuye los elementos en el eje vertical.

Estas propiedades se aplican sobre el contenedor de la forma que se expone a continuación:

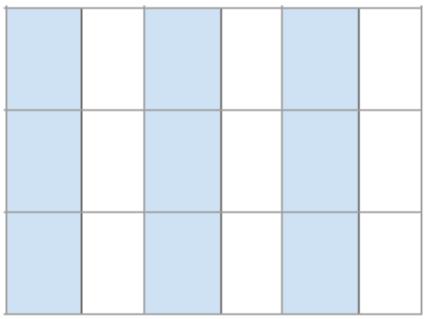
```
.grid {
   display: grid;
   grid-template: repeat(3, 1fr) / repeat (3, 1fr);
   justify-items: stretch;
   align-items: stretch;
}
```

De esta manera, la alineación se aplica uniformemente por cada elemento. El valor por defecto es **stretch** (ocupa todo el ancho de la celda), mientras que **start**, **center** y **end** significan arriba, en el centro y al final de la celda respectivamente.

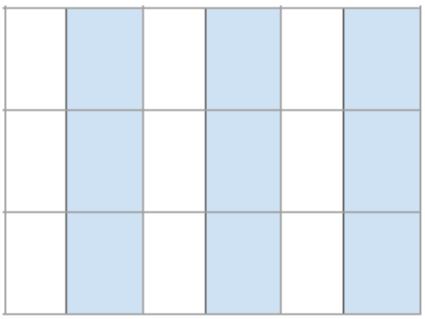
En un grid de 3x3 como en el código anterior el resultado sería el siguiente con cada tipo de alineación (el contenido es el recuadro azul y la celda contenedora creada con el grid es aquel elemento con fondo blanco).

Con justify-items

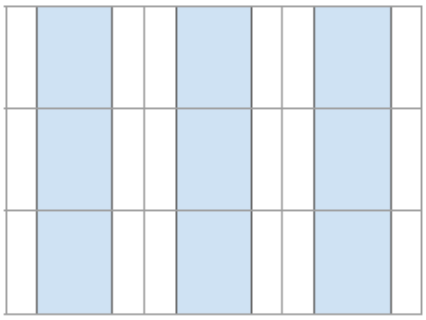
start



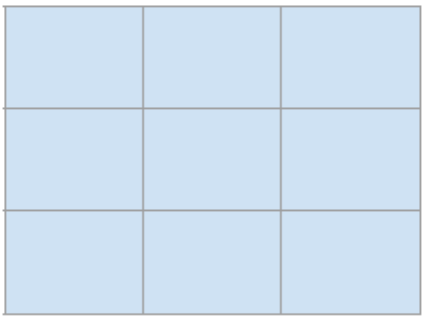
end



center

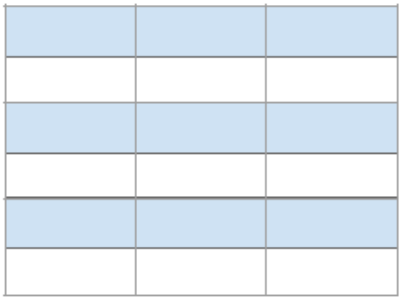


stretch

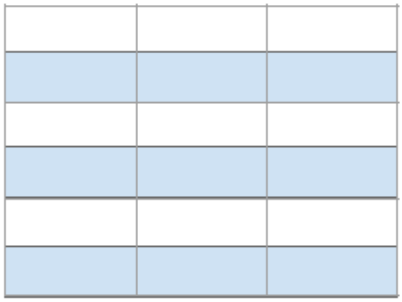


Con align-items

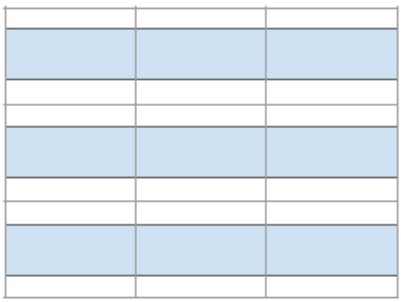
start



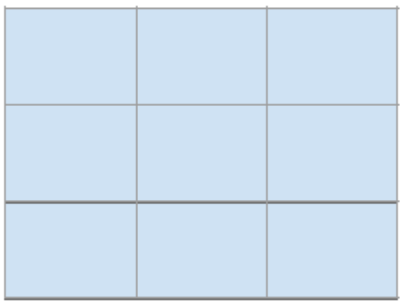
end



center



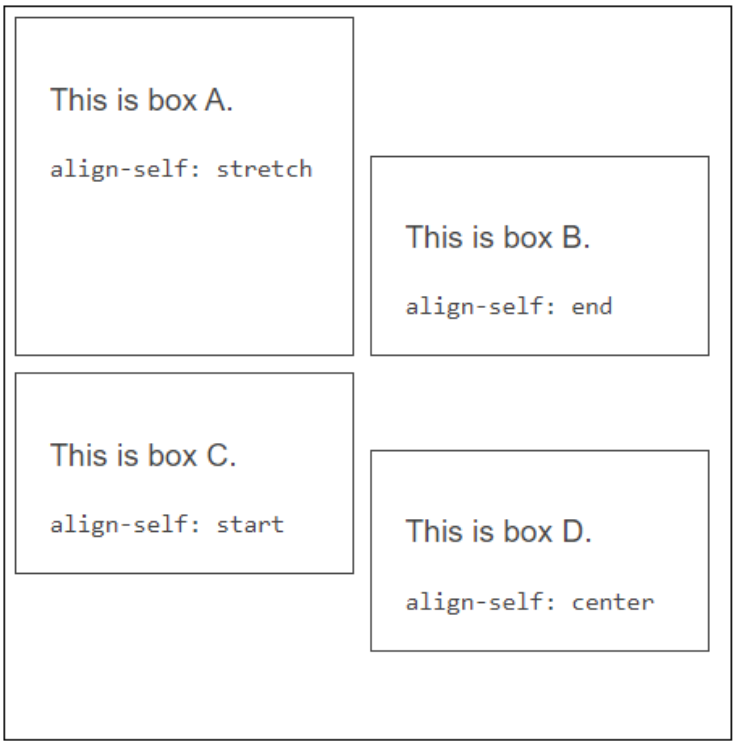
stretch



Existe la posibilidad de modificar la alineación de un ítem concreto con las propiedades respectivas **justify-self** y **align-self**. Estas propiedades se aplican en el CSS de cada ítem y no en el contenedor.

Por ejemplo, la alineación horizontal de cada ítem individual con **justify-self** en un grid de 2x2 sería la siguiente:

Mientras que el equivalente para la alineación vertical con **align-self** tendría la siguiente apariencia.



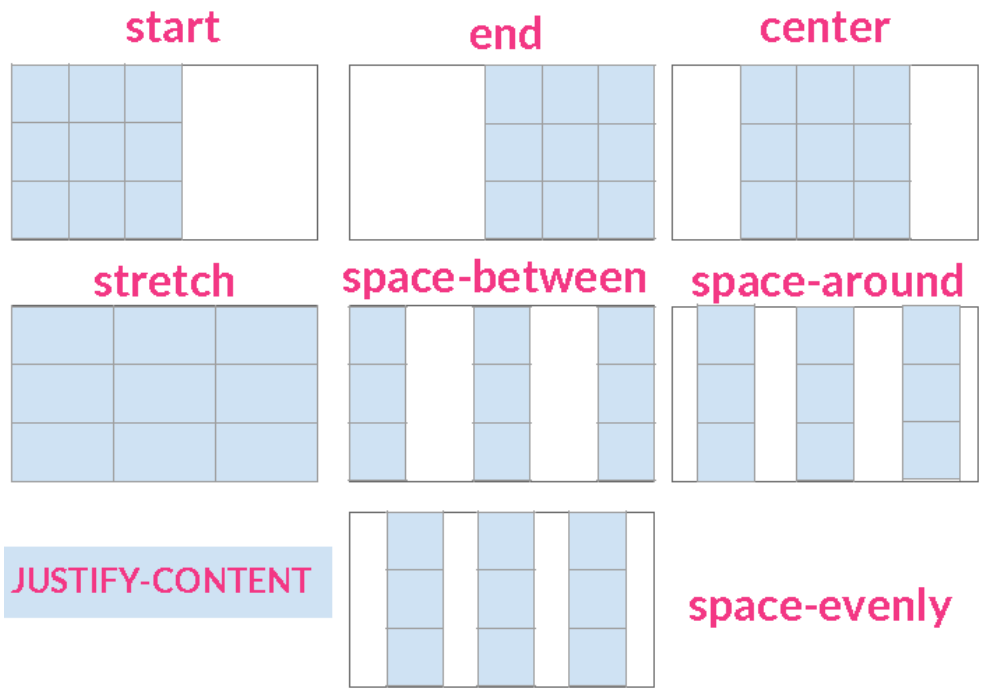
Distribución dentro del contenedor

En el caso de que el no ocupe todo el contenedor podemos distribuirlo con las propiedades CSS

justify-content (horizontal) y **align-content** (vertical).

También se aplica a nivel de contenedor. Los valores posibles son los siguientes:

Con justify-content



Con align-content