CSS Grid

Introducción a Grid

CSS Grid es un nuevo sistema que intenta **facilitar la maquetación de cajas dentro de una página web**. Los mecanismos utilizados tradicionalmente como posicionamiento relativo, absoluto, floats, elementos en bloque o en línea, o flex son insuficientes (o *muy complejos*) para generat layouts o estructuras en determinados diseños web actuales.

La idea de CSS Grid es **dividir una web en filas y columnas** (cuadrícula). Una de las ventajas de utilizar Grid es la posibilidad de **cambiar la posición de los elementos modificando simplemente el código CSS**, sin tener que realizar cambio alguno en HTML.

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 (grid tiene 2 dimensiones). 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, entre otros aspectos.

CSS Grid está activado por defecto en los navegadores mas importantes (Chrome, Mozilla, Safari, Opera) **desde Marzo 2017**, pero funciona de forma óptima desde Mayo del mismo año tras varias actualizaciones y errores que surgieron. Posteriormente se han realizado actualizaciones de mejora y actualmente todos los navegadores soportan la tecnología.

Antes de iniciarse con Grid **es necesario dominar FlexBox** (imprescindible) ya que Grid toma la base y la filosofía de este sistema. Tal y como ocurre en Flexbox, CSS Grid funciona con la idea de un contenedor-padre que alberga unos elementos-hijo. Por lo que nuestro contenedor-padre (***grid-container***), y nuestro elemento-hijo es un (***grid-item***).

- **Container**: Es el elemento padre y es el contenedor que definirá la cuadrícula o rejilla. Es donde se introducirá el atributo (***display:grid***) para especificar que es un elemento que trabaja con grid.

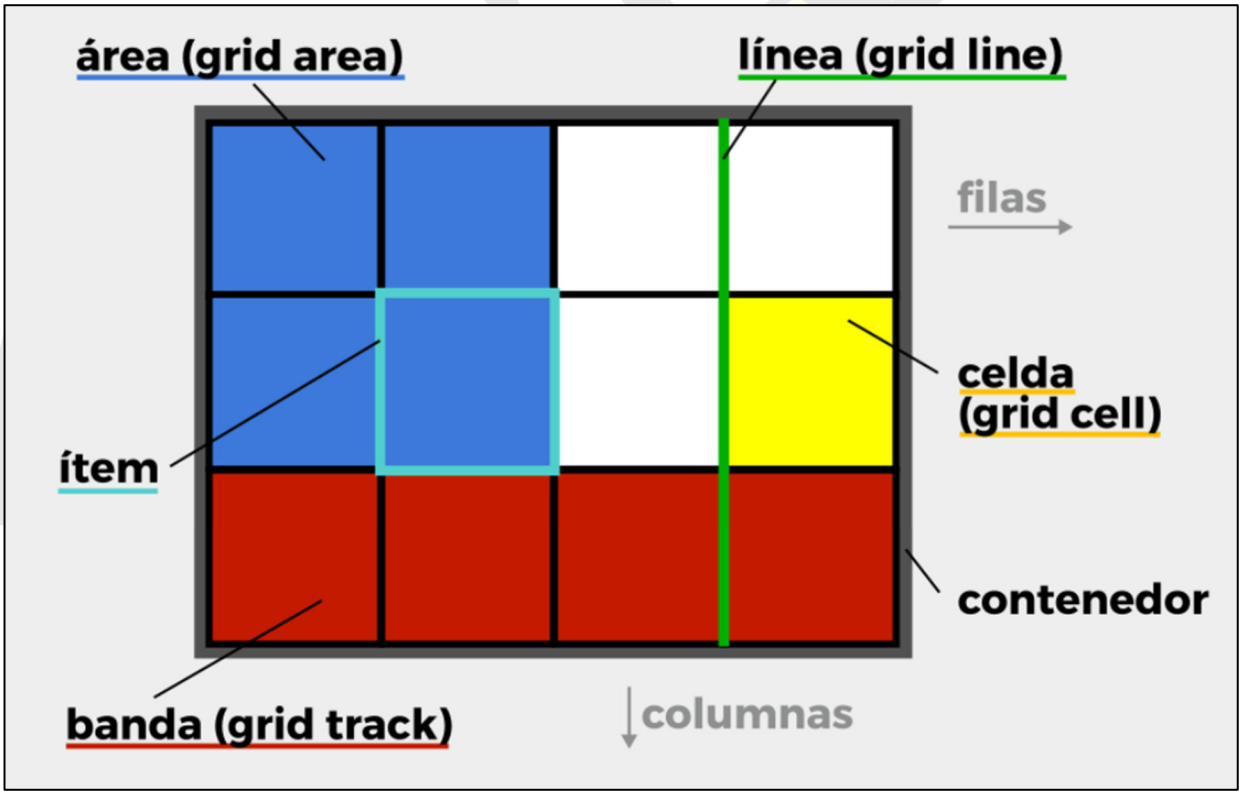
- **Ítem**: Cada uno de los elementos hijos que contiene la cuadrícula (*elemento contenedor*).

- **Línea (*grid line*)**: Separador horizontal o vertical de las celdas de la cuadrícula.

- **Celda (*grid cell*)**: Es una celda, el espacio entre dos líneas verticales adyacentes y dos líneas horizontales adyacentes. Es una «unidad» dentro de la rejilla llamada (casilla, celda, etc…) que coincide con la intersección de fila (*'grid-row'*) y columna (*'grid-column'*). Este concepto puede confundirse fácilmente con el elemento **'item'**. Los dos hacen referencia a un mismo elemento pero el elemento 'item' hace referencia a todo el contenedor mientras la celda es cada uno de los cuadrados del gris (unidad mínima).

- **Area (*grid area*)**: Región o conjunto de celdas de la cuadrícula. El espacio entre dos líneas verticales y dos líneas horizontales, no teniendo que ser forzosamente adyacentes. En la imagen, tendríamos la grid área creada entre las column grid lines 1 y 3, y las row grid lines 1 y 3.

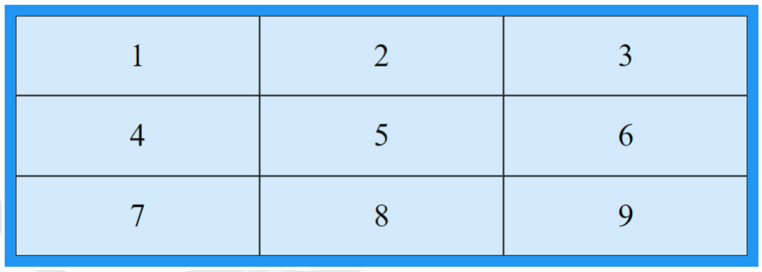
- **Banda (*grid track*)**: Es el espacio entre dos *'grid lines'* adyacentes. Pueden formar tanto columnas como filas. En la imagen inferior vemos el grid track (color marrón) entre la tercera línea de fila y la cuarta.



Para crear un elemento Grid hay que especificar sobre el elemento contenedor la propiedad *'display'* con el valor **'grid'** o **'inline-grid'**. Este valor influye en como la cuadrícula se comportará con el contenedor exterior.

- **inline-grid**: Establece una cuadrícula con ítems en línea, de forma equivalente a 'inline-block'. En este caso la celda se adapta al contenido afectando a toda la columna y tomando el espacio mínimo necesario para mostrarlo y el resto de las columnas se repartirán el restante.

- **grid**: Establece una cuadrícula con ítems en bloque, de forma equivalente al 'block', es decir la rejilla (no cada uno de los ítems) toma todo el ancho de la pantalla de navegación. En este caso todas las celdas se distribuyen con un mismo tamaño a partir del ancho del elemento contendor.



Grid Columnas y filas

Es posible especificar un tamaño fijo de las cuadrículas con las propiedades **'grid-template-columns'** y **'grid-template-rows'**, siempre desde el elemento padre. En la siguiente imagen se especifica el tamaño en píxeles de cada columna y de cada fila.

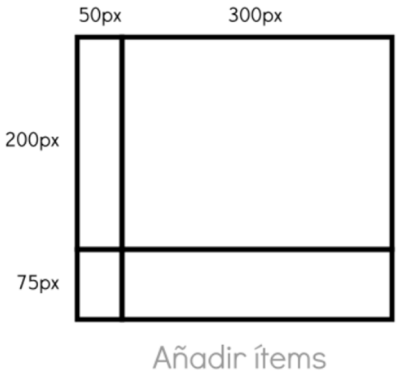
- **grid-template-columns**: Establece el tamaño de cada columna separadas por comas.
- **grid-template-rows**: Establece el tamaño de cada una de las filas separadas por comas.

Ejemplo 1: Creación de un grid de dos formas especificando el número de ítems y sus proporciones, en el primer caso en píxeles y en el segundo caso fracciones (fr).

```
<div class="grid"> <!-- contenedor -->
  <div class="a">Item 1</div> <!-- cada uno de los ítems del grid -->
  <div class="b">Item 2</div>
  <div class="c">Item 3</div>
  <div class="d">Item 4</div>
</div>
```

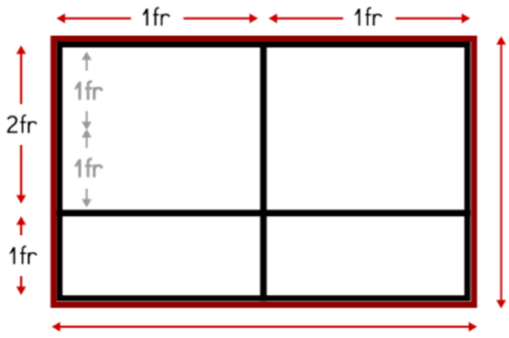
1

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



2

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



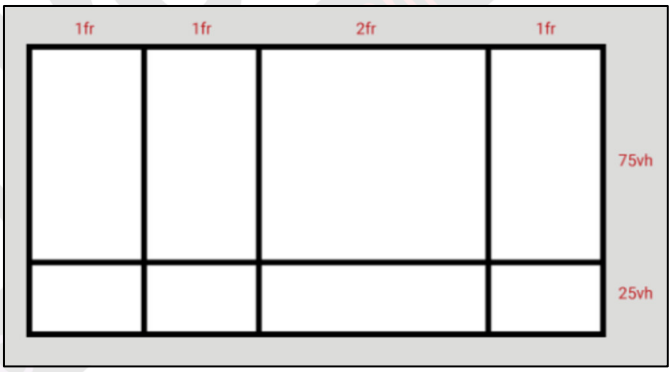
Es posible utilizar otros tipos de unidades además de los píxeles que se ajustan seguramente de forma más adecuada al espacio del contenedor como **porcentajes (%)** o **fracciones (fr)**. Incluso se puede realizar la combinación de diferentes unidades dentro de la estructura Grid.

Ejemplo 2: Creación de un grid de dos formas diferentes con 8 ítems dentro de un contenedor y especificando con diferentes unidades el tamaño de la misma.

1

```
 </div class="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 class="item-7"> </div>
   </div class="item-8"> </div>
</div>
```

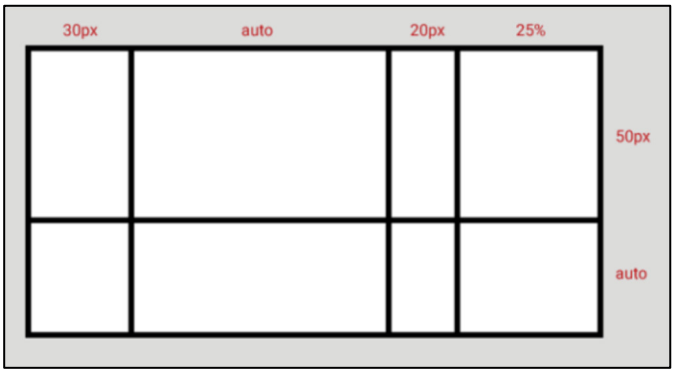
```
.container{
   display: grid;
   grid-template-columns: 1fr 1fr 2fr 1fr;
   grid-template-rows: 75vh 25vh;
}
```



2

```
 </div class="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 class="item-7"> </div>
   </div class="item-8"> </div>
</div>
```

```
.container{
   display: grid;
   grid-template-columns: 30px auto 20px 25%;
   grid-template-rows: 50px auto;
}
```



Si se añaden más ítems que columnas o filas declaradas con **'grid-template-columns'** o **'grid-template-rows'** estas se configuran como si diesen la vuelta y volvieran a comenzar. Es importante asegurarse que el número de ítems coinciden con el total que deseamos mostrar.

En algunas situaciones las propiedades **'grid-template-columns'** y **'grid-template-rows**' se pueden configurar de tal forma que no se tenga que repetir código innecesariamente una y otra vez. En este caso se utiliza el valor **'repeat'** y entre paréntesis se especifica el número de repeticiones y la configuración de las columnas o filas.

Las dos imágenes siguientes realizan la misma acción pero la sintaxis es diferente, una tabla de 4 X 4. En el siguiente ejemplo la configuración de las columnas será para 4 ítems. El primero de 100px, los dos siguientes por repetición de 50px, y la última de 200px.

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

Método simplificado

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

Método normal

Cuando configuramos un grid especifico de tantas filas ('grid-template-rows') y columnas ('grid-template-column') podemos tener el inconveniente que si generamos una web dinámica se generen mas casillas que las configuradas.

En este caso para asegurarse que todas las casillas nuevas que pudieran aparecer tendrán el mismo tamaño que el resto del grid se deberá utilizar la propiedad **'grid-auto-rows'** o **'grid-auto-columns'** especificando el tamaño de las nuevas casillas de la rejilla en caso de desbordamiento respecto a su configuración inicial.

En el siguiente código se puede visualizar la configuración de la rejilla con espacio entre las casillas y la configuración de filas adicionales o sobrantes por combinaciones (**'grid-auto-rows'**).

```
main{
    height: 100vh; /* Altura de la caja 100% viewport*/

    display:grid; /* Especifica que sus elementos hijos son grid */
    grid-template-columns: 1fr 1fr 1fr 1fr; /* Tamaño columnas */
    grid-template-rows: 1fr 2fr repeat(2, 1fr); /*Tamaño filas */

    grid-gap: 10px; /*Shorthand espacio filas y columnas */
    grid-auto-rows: 1fr; /*Cualquier fila adicional sera como el resto*/
}
```

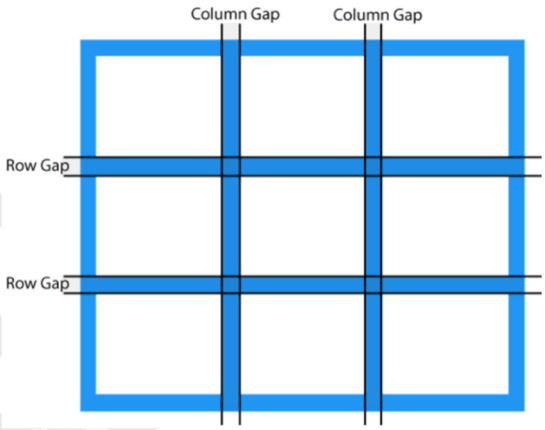
Espaciado entre items

De inicio, el espacio entre los **'grid items'** es nulo, no existe un espacio vacío entre ellos. A veces es necesario ese espaciado entre nuestros ítems de la rejilla. Este problema lo solucionaremos con dos propiedades y una que engloba a las dos:

- **'row-gap'**, Hace referencia al espaciado existente entre dos filas contiguas. (anteriormente denominado *'grid-row-gap'*).

- **'column-gap'**: Hace referencia al espaciado existente entre dos columnas contiguas. (anteriormente denominado *'grid-column-gap'*).

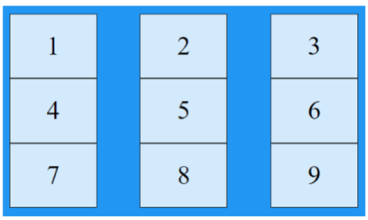
- **'grid-gap'**: Este es el método abreviado *'shorthand'* conjunto de filas y columnas. Cuando lo usemos, hay que tener en cuenta una cosa muy importante. El primer grid gap se corresponde con la segunda grid line, que es la que realmente está entre dos elementos de nuestra rejilla.

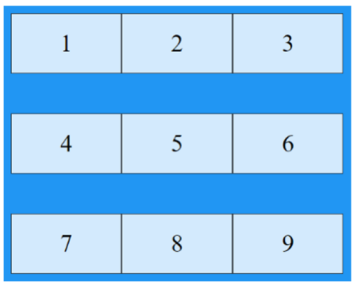


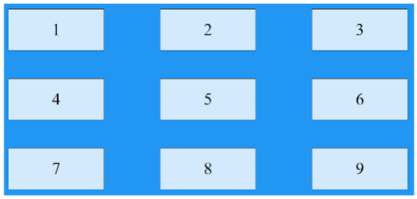

```
.grid-container {
  display: grid;
  grid-column-gap: 50px;
}
```

```
.grid-container {
  display: grid;
  grid-row-gap: 50px;
}
```

```
.grid-container {
  display: grid;
  grid-gap: 50px 100px;
}
```







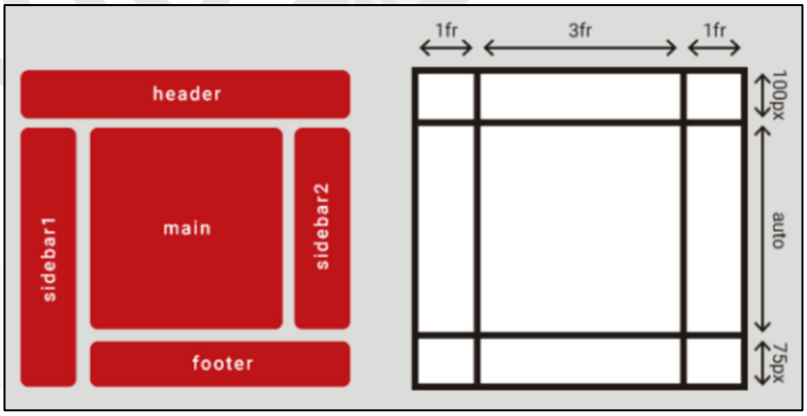
Posicionando elementos: Grid Items

Tras conocer los elementos esenciales de creación de grid es necesario **aprender como posicionar y mover los diferentes elementos que componen nuestra web** dentro de la rejilla, es decir, es necesario entrar de lleno en la manipulación de los Grid Items.

Para conocer el funcionamiento utilizaremos un ejemplo. A continuación se muestra el código HTML y CSS para generar una estructura grid.

```
<div class="container">
        <div class="header"> </div>
        <div class="sidebar1"> </div>
        <div class="main"> </div>
        <div class="sidebar2"> </div>
        <div class="footer"> </div>
</div>
```

```
.container{
        display: grid;
        grid-template-rows: 100px auto 75px;
        grid-template-columns: 1fr 3fr 1fr;
}
```



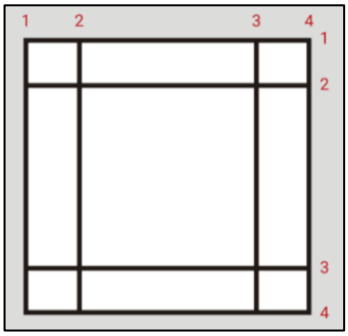
Después de la creación de la rejilla (grid) nos planteamos la siguiente pregunta:

¿Cómo posicionamos cada uno de los *divs* existentes dentro de nuestro *container* en la rejilla que hemos creado ocupando un número de celdas específicas?

Existen diferentes métodos para realizar esta acción:

1. Posicionado mediante grid lines (*por número*)

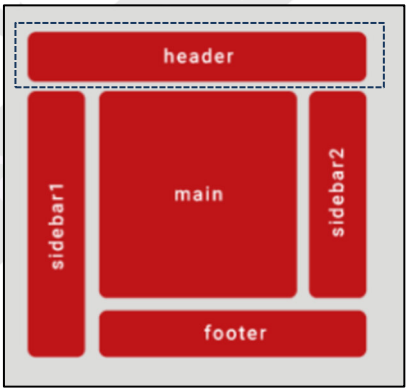
Por defecto CSS grid ordena las rejillas de forma numérica, de izquierda a derecha y de arriba abajo. Esta es la distribución estándar de un grid.

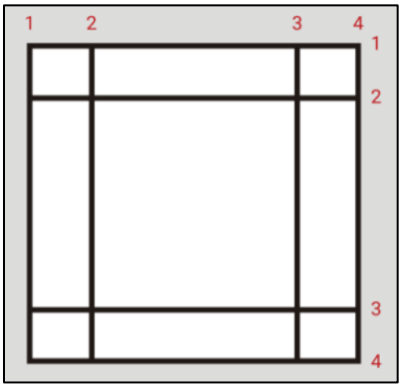


Para empezar es necesario especificar donde empiezan y donde acaban el conjunto de filas y columnas que vamos a utilizar. Las propiedades que se deben utilizar son **'grid-row-start'**, **'grid-row-end'**, **'grid-column-start'**, y **'grid-column-end'**.

A continuación, se configura la caja con el estilo *'header'* para que ocupe toda la parte superior de la rejilla a modo de cabecera. Se determina la **fila de inicio y de fin**, y la **columna de inicio y de fin**, de tal forma que todo lo que queda dentro de los limites formará parte de la caja.

```
.header {
        grid-row-start: 1;
        grid-row-end: 2;
        grid-column-start: 1;
        grid-column-end: 4;
}
```



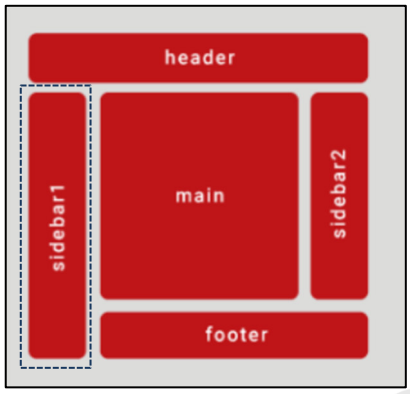


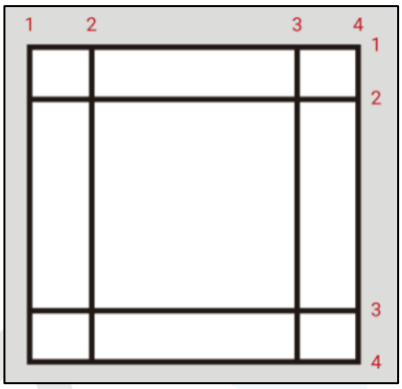
El mismo código CSS se puede realizar utilizando el **método abreviado** para las propiedades **'grid-column'** y **'grid-row'**. La diferencia respecto al sistema anterior es la eficiencia de código al escribir menos líneas. En este caso si solo se especifica un número toma como referencia la fila o la columna. Cuando se combinan varias filas o columnas es necesario la utilización de la barra (/).

```
.header {
        grid-row: 1;
        grid-column: 1 / 4;
}
```

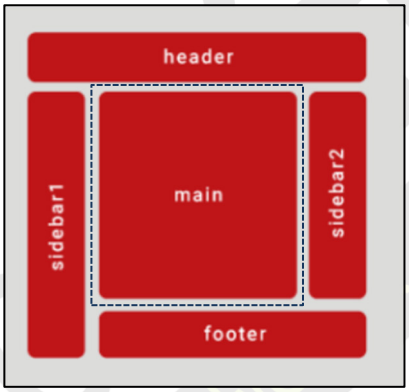
Después es necesario configurar el resto de las cajas y sus correspondientes estilos para darle el formato que deseamos.

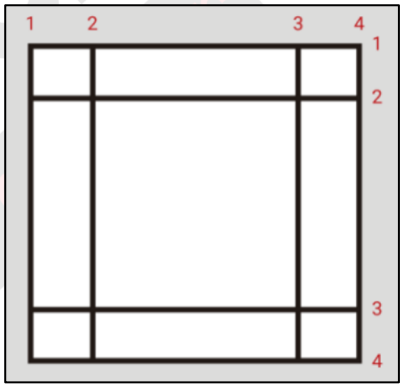
```
.sidebar1 {
        grid-row: 2 / 4;
        grid-column: 1;
}
```



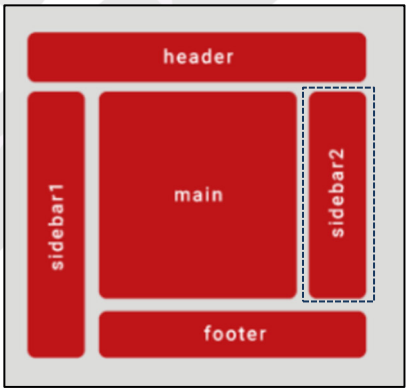


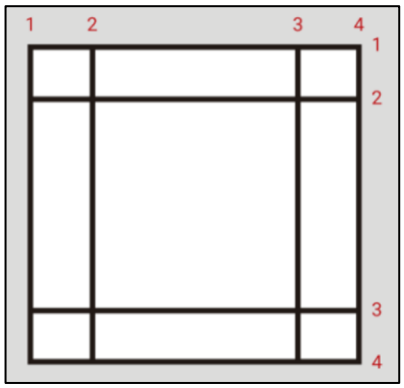

```
.main {
        grid-row: 2;
        grid-column: 2;
}
```



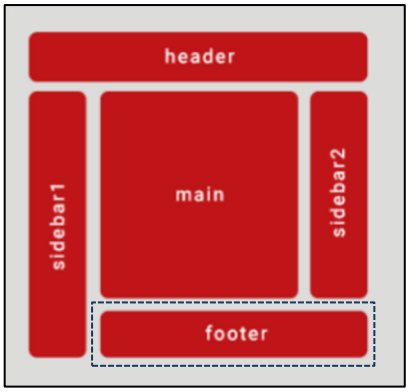


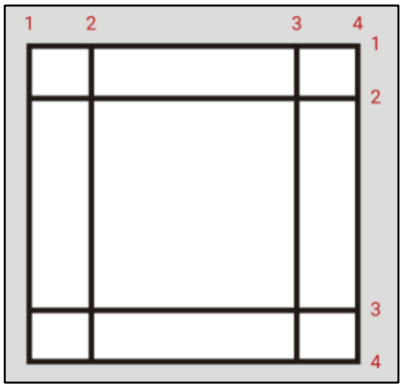

```
.sidebar2 {
        grid-row: 2;
        grid-column: 3;
}
```






```
.footer {
        grid-row: 3;
        grid-column: 2 / 4;
}
```





2. Posicionado mediante grid areas (***por nombre***)

Mediante la propiedad '**grid-area**' aplicada a nuestros Grid Items, asignaremos los límites tanto verticales como horizontales de nuestros elementos. Es como si cogiésemos a la vez las propiedades '**grid-row**' y '**grid-column**', y las comprimiésemos en una sola.

Este sistema de creación se puede realizar de dos formas aunque el resultado final es el mismo.

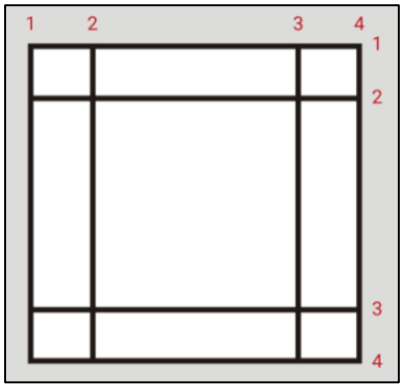
En el caso de las '**grid-area**', el orden de configuración por orden que seguiremos será el que se muestra a continuación. Curiosamente siguen un sentido antihorario, justamente todo el contrario al resto de propiedades CSS cuando se deben construir.

```
Grid-area: límit superior / limit izquierdo / límit inferior / límit derecho
```

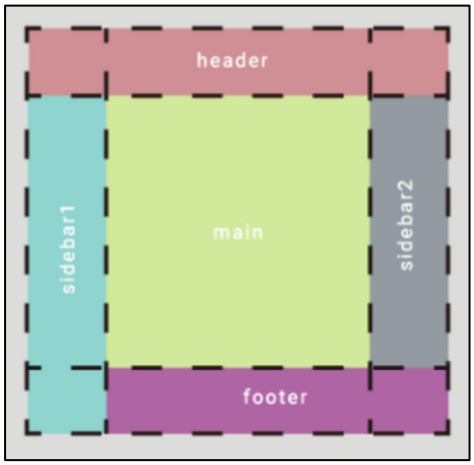
Primera forma de creación: (**por números**) RECOMENDADO

Esta forma de configurar las áreas utiliza los números para definir cada uno de los limites de las áreas que hacen referencia a las filas de la rejilla. Sigue el sentido antihorario.

```
.container{
        display: grid;
        grid-template-rows: 100px auto 75px;
        grid-template-columns: 1fr 3fr 1fr;
}
```




```
.header {
        grid-area: 1 / 1 / 2 / 4;
}

.sidebar1 {
        grid-area: 2 / 1 / 4 / 2;
}

.main {
        grid-area: 2 / 2 / 3 / 3;
}

.sidebar2 {
        grid-area: 2 / 3 / 3 / 4;
}

.footer {
        grid-area: 3 / 2 / 4 / 4;
}
```



Segunda forma de creación de áreas: (**por palabras**)

Esta forma utiliza palabras para definir cada una de las 'casillas' que conforman nuestro grid. Aquí es necesario utilizar la propiedad **'grid-template-areas'** y configurarla en la caja madre. Las cajas hijas solo harán referencia al nombre en la configuración que se haya realizado en la caja madre.

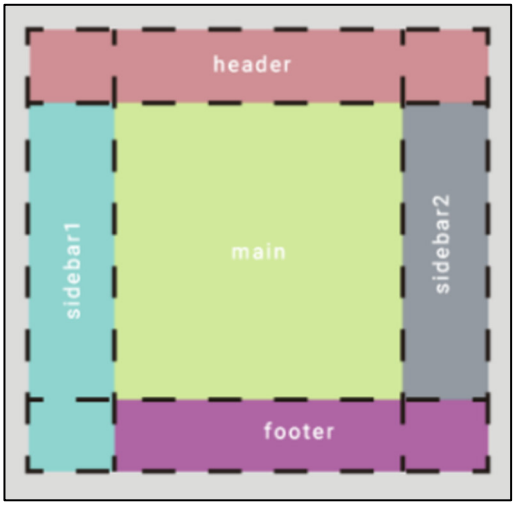
Se define cada fila de la rejilla en la caja madre escribiendo el nombre de las cajas que lo formarán. Si algun área ocupa más de una fila se escribirá la configuración en dos filas independientes (ejemplo 'side1').

- Entre las diferentes filas no hay que escribir ningún carácter de separación como la coma (error habitual).
- Todas las filas tendrán el mismo número de palabras para que la cuadrícula esté perfectamente equilibrada en el número de filas y columnas.

Ejemplo 1

```
.header {
        grid-area: head;
}

.sidebar1 {
        grid-area: side1;
}

.main {
        grid-area: main;
}

.sidebar2 {
        grid-area: side2;
}

.footer {
        grid-area: foot;
}
```

```
.container{
        display: grid;
        grid-template-rows: 100px auto 75px;
        grid-template-columns: 1fr 3fr 1fr;
        grid-template-areas: "head head head"
                             "side1 main side2"
                             "side1 foot foot"
}
```

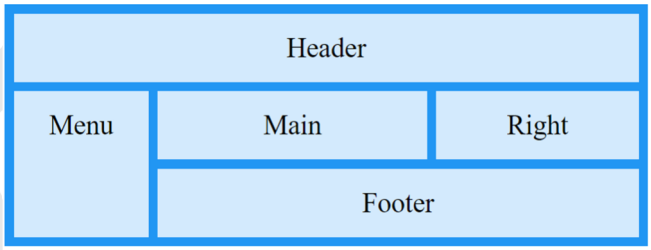


Ejemplo 2

```
.grid-container {
  background-color: #2196F3;
  padding: 10px;

  display: grid;
  grid-gap: 10px;
  grid-template-areas:
  'header header header header header header'
  'menu main main main right right'
  'menu footer footer footer footer footer';
}
```

```
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }
```

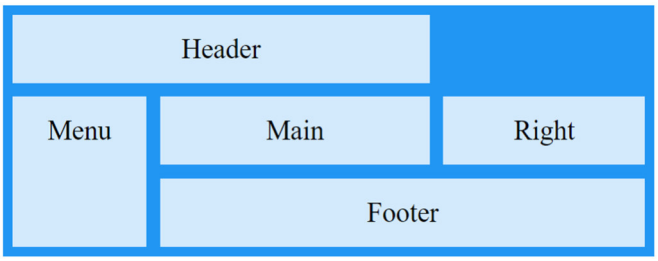
```
<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">Main</div>
  <div class="item4">Right</div>
  <div class="item5">Footer</div>
</div>
```



Si la configuración de nuestra rejilla tuviese espacios en blanco entonces se substituye el nombre por un punto. Esta es la forma de decirle a grid que hay uno, o mas, espacios en blanco.

En el siguiente ejemplo se puede ver como las dos últimas posiciones que contiene el 'header', que en el apartado anterior estaban ocupadas, ahora se han dejado vacías debido a los dos puntos que se han sustituido es la definición de **'grid-template-areas'**.

```
.grid-container {
  display: grid;
  grid-template-areas:
  'header header header header . .'
  'menu main main main right right'
  'menu footer footer footer footer footer';
  grid-gap: 15px;
  background-color: #2196F3;
  padding: 10px;
}
```

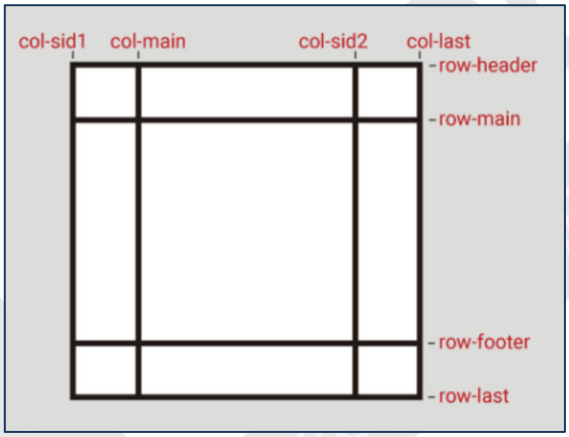


3. Posicionado mediante grid lines (***por nombre columna***) [POCO UTILIZADO]

El resultado final es exactamente igual que en el apartado anterior pero la sintaxis es diferente. En lugar de utilizar números para definir filas y columnas se utilizan nombres que previamente han sido definidos en la creación del grid.

La declaración de los nombres se asigna entre corchetes y siempre van precediendo a la unidad que especifica el tamaño de la fila o columna. La siguiente imagen muestra la declaración del grid asignando nombres a las filas y columnas.

```
.container{
        display: grid;
        grid-template-rows: [row-header] 100px [row-main] auto [row-
footer] 75px [row-last];
        grid-template-columns: [col-sid1] 1fr [col-main] 3fr [col-sid2]
1fr [col-last];
}
```




```
.header {
        grid-row: row-header;
        grid-column: col-sid1 / col-last;
}

.sidebar1 {
        grid-row: row-main / row-last;
        grid-column: col-sid1;
}

.main {
        grid-row: row-main;
        grid-column: col-main;
}

.sidebar2 {
        grid-row: row-main;
        grid-column: col-sid2;
}

.footer {
        grid-row: row-footer;
        grid-column: col-main / col-last;
}
```

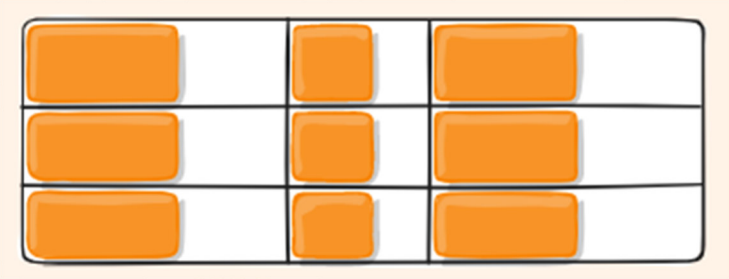
Posicionamiento CSS para Grid

Existen una serie de propiedades que se pueden utilizar para colocar los ítems dentro de la cuadrícula. Con ellas podemos distribuir los elementos de una forma muy sencilla y cómoda. Dichas propiedades son **justify-items** y **align-items**. Funcionan exactamente igual que las propiedades en Flex.

Estas propiedades se aplican sobre el elemento contenedor padre, pero afectan a los ítems hijos, por lo que actúan sobre la distribución de cada uno de los hijos. En el caso de que queramos que uno de los ítems hijos tengan una distribución diferente al resto, aplicamos la propiedad **justify-self** o **align-self** sobre el ítem hijo en cuestión. Las propiedades funcionan igual que **justify-items** y **align-items.**

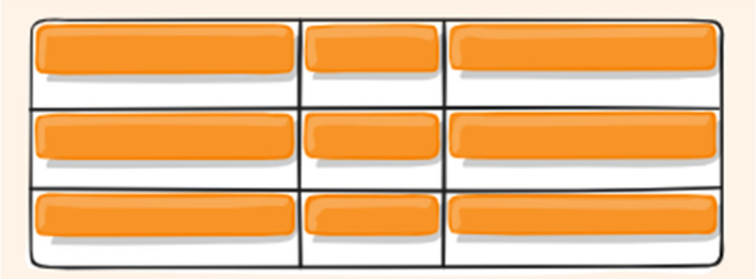
También podemos utilizar las propiedades **justify-content** o **align-content** para modificar la distribución de todo el contenido en su conjunto, y no sólo de los ítems por separado:

- **justify-items**: Propiedad que al igual que flexbox permite alinear el contenido de los elementos de cada celda del grid en el eje principal que por normal general es **el horizontal** (start, center, end, stretch que es el valor por defecto). Se aplica sobre el elemento padre.



Por defecto el valor del atributo es **strech** y aprovecha el máximo espacio de la caja.

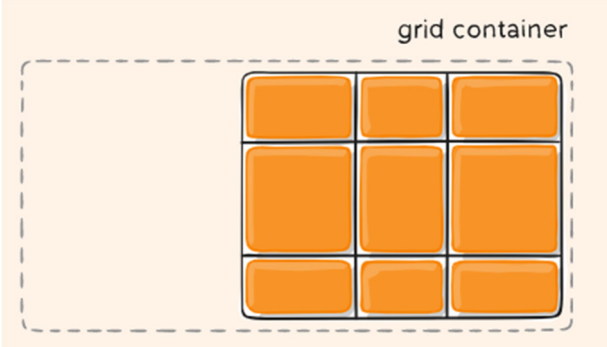
- **align-items**: Propiedad que al igual que flexbox permite alinear el contenido de los elementos de cada celda de forma **vertical** (start, end, center, stretch). Se aplica sobre el elemento padre.



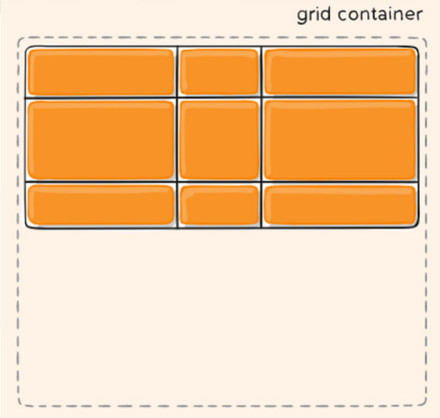
Por defecto el valor del atributo es **strech** y aprovecha el máximo espacio de la caja.

En el caso de que queramos que uno de los ítems hijos tengan una distribución diferente al resto, aplicamos la propiedad **justify-self** o **align-self** sobre el ítem hijo en cuestión, sobre escribiendo su distribución. Los valores de las propiedades son exactamente iguales que las anteriores pero afectando solo a un elemento hijo.

- **justify-content**: Propiedad que se utiliza para modificar la distribución de todo el contenido en su conjunto de forma **horizontal** (*start, end, centr, strech, space-around, space-between, space-evenly*). Se aplica sobre el elemento padre.



- **align-content**: Propiedad que se utiliza para modificar la distribución de todo el contenido en su conjunto de forma **vertical** (*start, end, centr, strech, space-around, space-between, space-evenly*). Se aplica sobre el elemento padre.



Si vamos a crear estructuras grid donde necesitamos utilizar las cuatro propiedades anteriores, es mejor utilizar un atajo donde simplificamos el código resultante. Es el caso de las propiedades **'place-items'** -> [*align-items*] [*justify-items*] i **'place-content'** -> [*align-content*] [*justify-content*].

Métodos y propiedades CSS para Grid

Grid dispone de una serie de métodos que permite establecer unas configuraciones muy interesantes a la cuadrícula para que se adapten de la forma más correcta a los diferentes tamaños del viewport.

- **auto-fill**: Es una propiedad que permite distribuir el numero de columnas en función del ancho que se haya especificado a la caja madre.

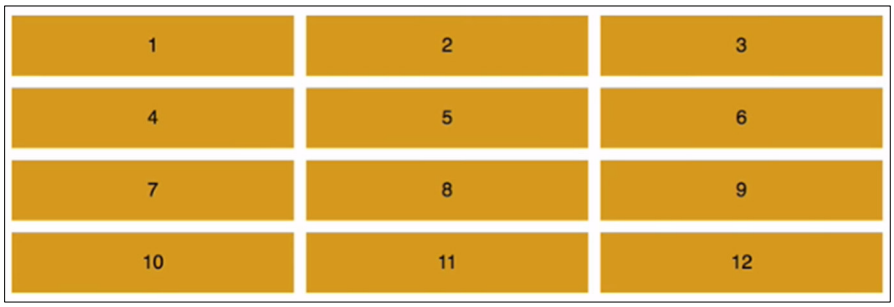
- **minmax()**: Este método permite establecer el mínimo y máximo de las columnas o las filas para evitar que las celdas del grid adopten un tamaño inapropiado para el dispositivo.

```
/*Crea 8 columnas de minimo 100px y maximo 1 fraccion*/
grid-template-columns: repeat(8, minmax(100px, 300px));

/*Crea 8 columnas de minimo 100px y maximo 1 fraccion*/
grid-template-columns: repeat(8, minmax(100px, 1fr));

/*Crea 8 columnas de ancho minimo al contenido de la celda y maximo 1 fraccion*/
grid-template-columns: repeat(8, minmax(min-content, 1fr));

/*Crea 8 columnas de ancho min y max respecto al contenido de la celda*/
grid-template-columns: repeat(8, minmax(min-content, max-content));
```

- **grid-auto-row**: Indica el tamaño automático del ancho que tendrán las columnas.

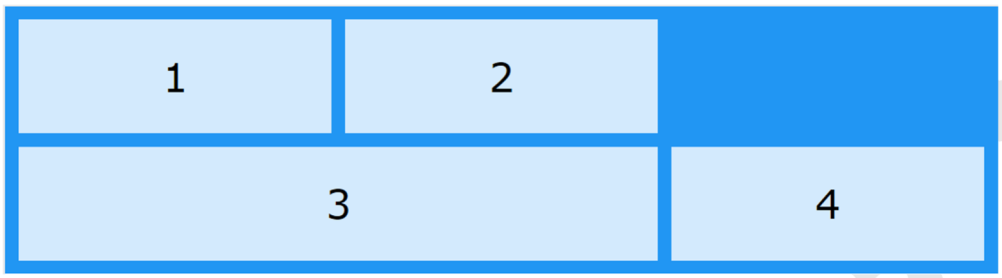
- **grid-auto-columns**: Indica el tamaño automático del alto que tendrán las filas.

- **grid-auto-flow**: Es una propiedad de grid que determina cual será el flujo de creación de la cuadrícula. De la misma forma que sucedía con flexbox disponemos de un flujo que por defecto tiene el valor de **'row'** pero también puede ser **'column'**. Otros valores posibles son **'dense'**, **'row-dense'** y **'column-dense'**. El valor dense rellena de forma automática intentando cubrir todos los huecos de la cuadrícula si aparecen elementos más pequeños más adelante.

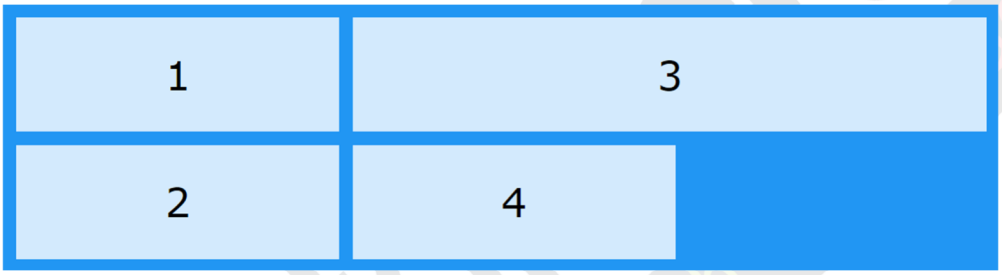


A continuación se muestran diferentes distribuciones de cajas en formato grid que están directamente asociadas con los posibles valores de la propiedad.

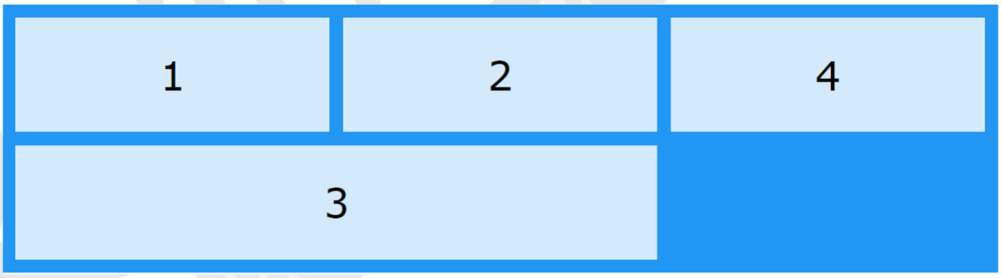
- **grid-auto-flow: row;**



- **grid-auto-flow: column;**



- **grid-auto-flow: row-dense;**



- **grid-auto-flow: column-dense;**