CSS GRID

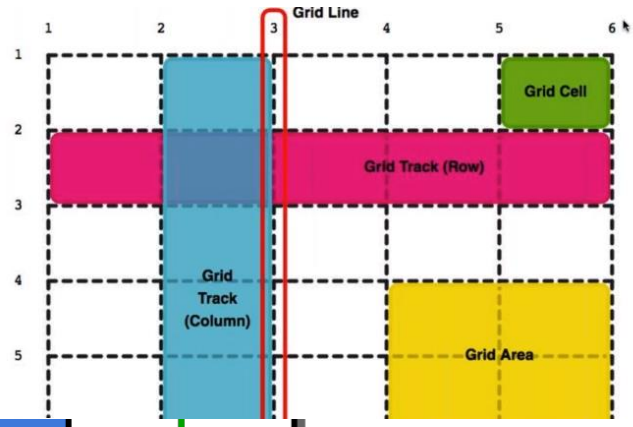
Uno de los procesos más problematicos y frustrantes de CSS es el proceso de colocar y distribuir los elementos a lo largo de una página. Mecanismos como posicionamiento, floats o elementos en bloque o en línea, suelen ser insuficientes o complejos para crear un layout o estructuras para páginas web actuales.

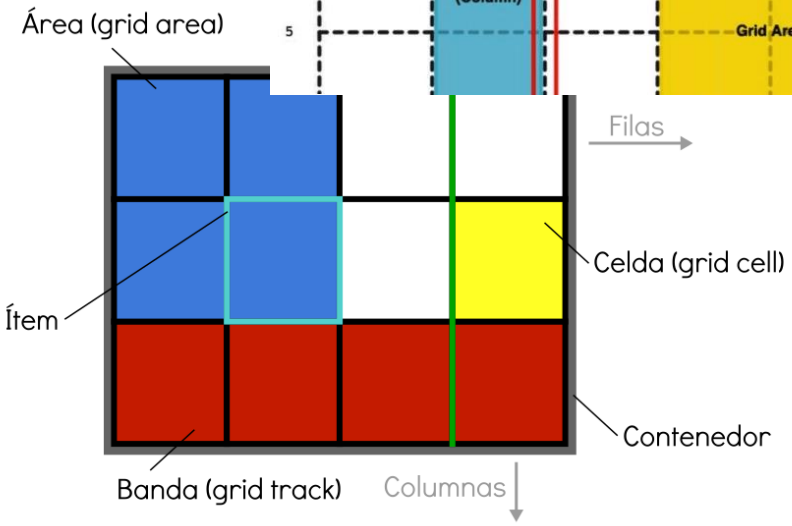
El sistema *flexbox* es una gran mejora, sin embargo, está orientado a estructuras de una sola dimensión, por lo que aún necesitamos algo más potente para estructuras web. Con el paso del tiempo, temenos muchos frameworks y librerías que utilizan un *sistema grid* donde definen una cuadrícula determinada, 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, añadiendole numerosas mejoras y características que permiten crear rápidamente cuadrículas sencillas y potentes.

Conceptos

Grid toma la filosofía y bases de FlexBox. Para utilizar **Grid CSS** necesitaremos tener en cuenta una serie de conceptos 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 uno de los cuadritos (unidad mínima) de la cuadrícula.

Area (grid area): 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.

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

```HTML
<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>
```

Para activar la cuadrícula **grid** hay que utilizar sobre el elemento contenedor la propiedad `display` y especificar el valor **grid** o **inline-grid**.

Display: grid → permite que la cuadrícula aparezca encima/debajo del contenido exterior (en bloque)
Display: inline-grid → **permite que la cuadrícula aparezca a la izquierda/derecha (en línea) del contenido exterior.**

Lo mínimo
/* crear un contenedor grid*/
https://codepen.io/paqui-molina/pen/yLLwmoe

Grid con filas o columnas explícitas
https://codepen.io/paqui-molina/pen/xxxBvXX

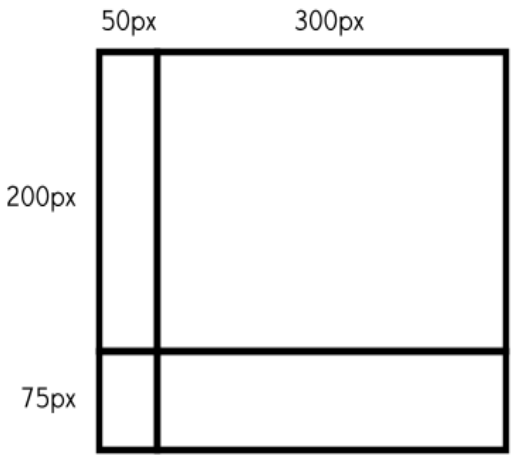
Para crear cuadrículas con un tamaño explícito.

`grid-template-columns`
`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 → establece el tamaño de las columnas (eje horizontal)

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

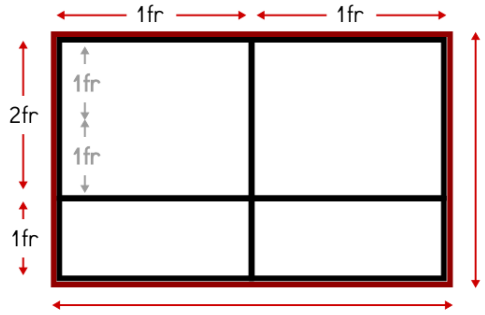


grid-template-rows → establece el tamaño de las filas (eje vertical)

En este ejemplo hemos utilizado **píxels** 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**.

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



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: los huecos (gutters).

Para especificar los huecos (espacio entre celdas) utilizamos

`grid-column-gap` y/o `grid-row-gap`.

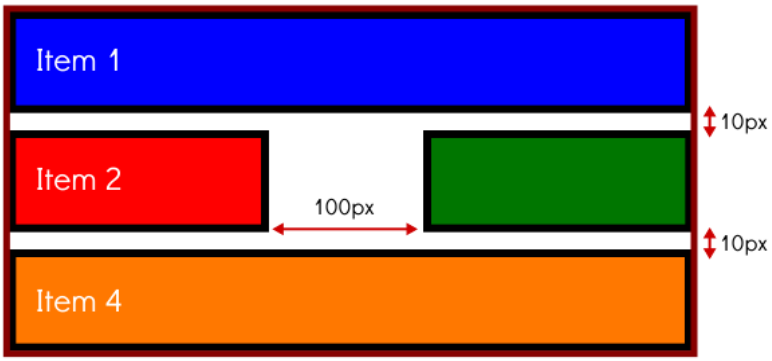
En ellas indicaremos el tamaño de dichos huecos:

`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*).

`Grid-gap` →`<row-gap> <column-gap>` ***`/ *versión corta* /`***

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



Unidades de medida

fr → Espacio libre // unidad flexible

1fr 1fr 1fr → 3 columnas del mismo tamaño

1fr 2fr 1fr → 3 columnas, la segunda el doble de las otras

Ejemplos:
grid-template-columns: 200px 1fr 1fr;

grid-template-columns: 20% 1fr;

grid-template-rows: 1fr 1fr 1fr; /* esto funciona si el contenedor tiene una dimension de alto; mheight:200vh;*/

Ajuste automático de celdas

https://codepen.io/paqui-molina/pen/GRjqYvR

Determinarn el comportamiento de los tracks que se generan implícitamente o que su comportamiento no viene descrito en la definición explícita del grid.

`grid-auto-columns` y `grid-auto-rows` se usa para darle un tamaño automático a las celdas.
`Grid-auto-flow` para indicar el flujo de elementos en la cuadrícula, y especificar por donde se irán añadiendo,

https://codepen.io/paqui-molina/pen/WNGxaZo

`grid-auto-columns` → `tamaño`

`grid-auto-rows` → `tamaño`

`grid-auto-flow` → `row | column | dense`

Función repeat()
Filas y columnas repetitivas
En las propiedades `grid-template-columns` y `grid-template-rows` podemos indicar expresiones de repetición, indicando celdas que repiten un mismo patrón de celdas varias veces. La expresión a utilizar sería la siguiente: `repeat([núm de veces], [valor o valores])`.

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

Esto es igual a

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

Función minmax()
Establece un mínimo y un máximo en columnas o filas.

Grid-template-columns: minmax(150px, 250px) 1fr 1fr → 3 columnas, la primera como mínimo 150px, las otras dos se reparten el espacio

Grid-auto-rows: minmax(100px, auto); -> Una fila con un alto mínimo de 100px que se ajustará si el contenido es más alto.

Keywords: autofill, autofit.
https://css-tricks.com/auto-sizing-columns-css-grid-auto-fill-vs-auto-fit/
Autofill → relleno automático. Crea tantos tracks de la medida indicada como quepan en el contenedor SIN DESBORDAR

autofit → ajuste automático. Elimina los tracks vacíos

auto-fill vs. auto-fit (gridbyexample.com)

Ya sabemos dibujar un grid, ahora toca colocar

Propiedades referentes a la posición de los hijos de la cuadrícula

`grid-column-start` → Indica en que columna empezará el ítem de la cuadrícula.
`Grid-column-end` → Indica en que columna terminará el ítem de la cuadrícula.
`Grid-row-start` → Indica en que fila empezará el ítem de la cuadrícula.
`Grid-row-end` → Indica en que fila terminará el ítem de la cuadrícula.

Versión corta

```css
.a {
  /* grid-column: <grid-column-start> <grid-column-end> */
  /* grid-row: <grid-row-start> <grid-row-end> */
  grid-column: auto;
  grid-column: 4 / 6;
  grid-column: span 3;
  grid-column: span 3 / 6;
}
```

Ejemplos:

https://codepen.io/paqui-molina/pen/xQpdqa

Ejemplo: colocar elementos en el grid

https://codepen.io/cesalberca/pen/aEdGwj/

Ejemplo:

https://css-tricks.com/snippets/css/complete-guide-grid/
https://www.w3schools.com/css/css_grid.asp
Grid by Example - Usage examples of CSS Grid Layout

Grid por áreas. Otra forma de colocar los ítems

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:

Grid-template-areas → **Indica la disposición de las áreas en el grid. Cada texto entre comillas simboliza una fila.**

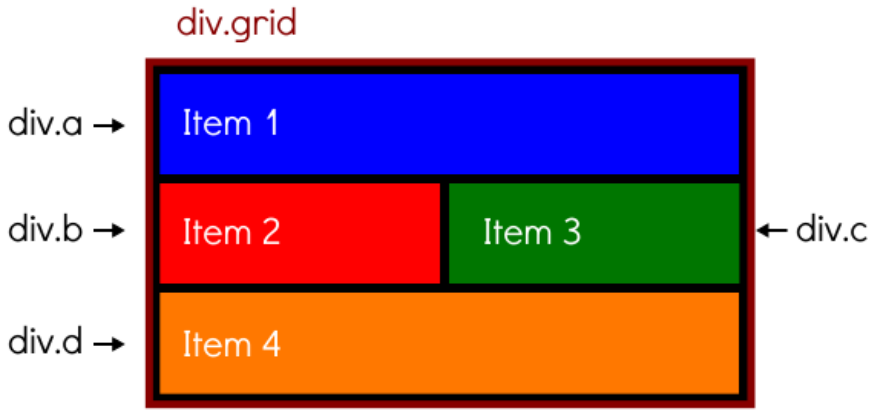
Grid-area → Indica el nombre del área. Se usa sobre ítems hijos del grid.

Así, es más fácil crear una cuadrícula altamente personalizada en apenas unas cuantas líneas de CSS, con mucha flexibilidad en la disposición y posición de cada área:

```
.grid {
  display:grid;
  grid-template-areas: "head head"
                       "menu main"
                       "foot foot";
}

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

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



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.

Ejemplo: Grid-template-areas

Grid Areas with Media Queries

Propiedades para el grid

GRID. Alineación, justificación items (codepen.io)

GRID. Justificación, alineación tracks (codepen.io)

Para colocar los ítems dentro de la cuadrícula podemos usar: justify-items y align-items que ya conocemos de flexbox,

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 elemento contenedor padre, pero afectan a los ítems hijos, por lo que actuan 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, sobreescribiendo su distribución.

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-content →start | end | center | stretch | space-around | space-between | space-evenly

Align-content →start | end | center | stretch | space-around | space-between | space-evenly

Propiedades para ítems grid hijos

Hasta ahora, salvo algunas excepciones como justify-self, align-self o grid-area, hemos visto propiedades CSS que se aplican solamente al contenedor. A continuación, vamos a ver ciertas propiedades que en su lugar, se aplican a cada ítem hijo de la cuadrícula, para alterar o cambiar el comportamiento específico de dicho elemento, que no se comporta como la mayoría.

Propiedades que hemos visto:

justify-self → altera la justificación del item hijo en el eje horizontal

align-self → altera la alineación del item hijo en el eje vertical

`grid-area` → Indica un nombre al área especificada, para su utilización con `grid-template-areas`.

Propiedad order

GRID. Order property (codepen.io)

Ejemplo grid

GRID. Z-index property, overlap (codepen.io)