

CSS Grid

INTRODUCCIÓN

Presenta un sistema de cuadrícula bidimensional para CSS.

Una cuadrícula es un conjunto de líneas horizontales y verticales que se intersectan - un grupo define columnas y el otro filas. Los elementos se pueden colocar en la cuadrícula respetando estas columnas y filas.

Tamaños fijos y flexibles

- a) Fijo: Definir con tamaños fijos: cm, px,...
- b) Flexibles: porcentajes o fr (fracción).

Posicionamiento de elementos

Puede colocar elementos en una ubicación precisa en la cuadrícula utilizando números de línea, nombres o seleccionando un área de la cuadrícula.

Grid también contiene un algoritmo para controlar la ubicación de elementos que no tienen una posición explícita en la cuadrícula.

Creación de líneas adicionales para alojar contenido

Usted puede definir una cuadrícula explícita con grid layout. La especificación Grid es lo suficientemente flexible como para permitir agregar filas y columnas adicionales cuando sea necesario.

Control de alineación

Grid contiene características de alineación para poder controlar la forma cómo se alinean los elementos una vez colocados en un área de cuadrícula y cómo está alineada toda la cuadrícula.

Control de contenido superpuesto

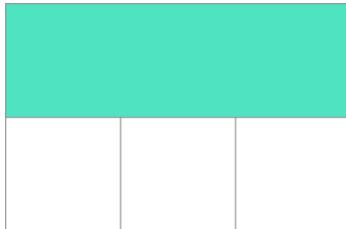
Se puede colocar más de un elemento en una celda de la cuadrícula o área, las cuales pueden solaparse o superponerse total o parcialmente entre sí. Controlar con [z-index](#).

Declaración de contenedor GRID

```
display: grid  
display: inline-grid
```

Tan pronto como hagamos esto todos los *hijos directos* de ese elemento se convertirán en *elementos de la cuadrícula*.

VÍAS, FILAS Y COLUMNAS DEL GRID



En la imagen se puede ver una vía resaltada. Es la vía de la primera fila en nuestra cuadrícula.

Definir filas y columnas- Propiedades:

- [grid-template-columns](#): Define el tamaño de las vías de cada columna.
- [grid-template-rows](#). : Define el tamaño de las vías de cada fila.

Éstas definen las vías de la cuadrícula. Una vía de cuadrícula es el área entre las dos líneas - horizontales o verticales- dentro de la cuadrícula.

La Unidad fr

Representa una fracción del espacio disponible en el contenedor de la cuadrícula.

EJEMPLO:

[grid-template-columns](#): 2fr 1fr 1fr; → El espacio disponible se divide en cuatro. Dos partes corresponden a la primera vía y una parte a cada una de las dos vías restantes.



[grid-template-columns](#): 500px 1fr 2fr; → La primera vía tiene 500 píxeles, por lo que este ancho fijo se sustraerá del espacio disponible. El espacio restante se divide en tres y se asigna en proporción a las dos vías flexibles.



Listando vías con la notación `repeat()`

Las cuadrículas grandes con muchas vías o celdas pueden utilizar la notación `repeat()` con el fin de repetir todas o una sección de la lista de vías.

```
repeat([número de veces], [valor o valores]);
```

EJEMPLO:

```
grid-template-columns: 1fr 1fr 1fr;  
grid-template-columns: repeat(3, 1fr);
```

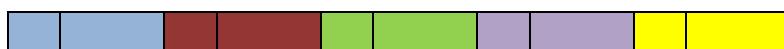
1fr	1fr	1fr
-----	-----	-----

```
grid-template-columns: 20px repeat(6, 1fr) 20px;
```

20px	1fr	1fr	1fr	1fr	1fr	1fr	20px
------	-----	-----	-----	-----	-----	-----	------

La notación de repetición toma una lista de vías específicas:

grid-template-columns: repeat(5, 1fr 2fr); → la cuadrícula consistirá de 10 vías, una vía 1fr seguida por una vía 2fr, repetida cinco veces.



Cuadrícula implícita y explícita

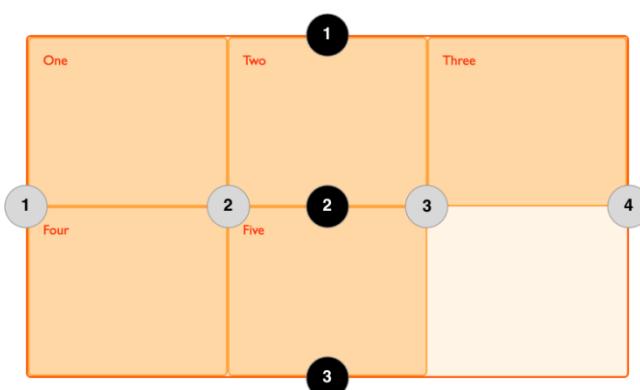
- a) Si especifico la cantidad de filas → cuadrícula explícita
- b) Si especifico sólo grid-template-columns → la cuadrícula irá poniendo filas según lo necesite el contenido. Eso es cuadrícula implícita.
- Si coloca algo fuera de la cuadrícula ya definida, o si debido a la cantidad de contenido, se necesitarán más vías o celdas, entonces grid crea filas y columnas en la cuadrícula implícita.

Estas vías varían su tamaño automáticamente de forma predeterminada, así que *ajustarán su tamaño basadas en el contenido dentro de ellas*.

- También se puede definir un tamaño para el conjunto de vías creadas en la cuadrícula *implícita* con las propiedades `grid-auto-rows` y `grid-auto-columns`.

LÍNEAS DE LA CUADRÍCULA

Debe tenerse en cuenta que cuando definimos una cuadrícula definimos las vías de la cuadrícula, no las líneas. Grid luego nos da las líneas numeradas a utilizar al posicionar elementos.



Las líneas están numeradas según el modo de escritura del documento.

En un idioma de izquierda a derecha, la línea 1 está al lado izquierdo de la cuadrícula.

En un idioma de derecha a izquierda, está en el lado derecho de la cuadrícula.

Posicionando elementos de acuerdo a las líneas en los hijos de un padre grid.

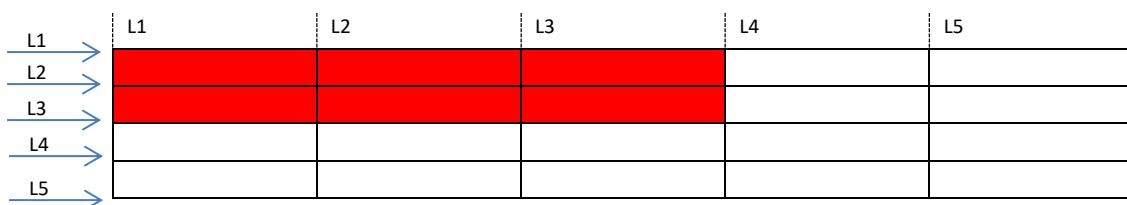
Las propiedades `grid-column-start`, `grid-column-end`, `grid-row-start` y `grid-row-end`

EJEMPLO:

```
.box1 {  
    grid-column-start: 1;  
    grid-column-end: 4;  
    grid-row-start: 1;  
    grid-row-end: 3;  
}
```

El elemento se coloca partiendo de la *línea* de la columna 1, y se extiende a la *línea* de la columna 4 → Se extiende 3 columnas

Y comienza en la *línea* de la fila 1 y termina en la *línea* de fila la 3 → Se extiende 2 filas.



[EJEMPLO: GRID- CREACION-MANIPULACION INICIAL](#)

[EJEMPLO: GRID- DISEÑARESTRUCTURA PAGINA CON GRID](#)

PROPIEDADES

- `grid-area`
 - Las áreas del grid son cuadradas (definidas con `grid-column/row-start/end`).
 - `grid-template-areas`: Consiste en la definición de áreas (según la cuadrícula dibujada) y asignarles nombres.
Posteriormente con `grid-area` se asociarán a elementos html (que ocuparán estos espacios definidos). En el elemento definido como `display:grid`

PASOS PARA DEFINIR GRID AREAS:

1. *Contenedor*: Definir las filas y columnas del grid con `grid-template-rows` y `grid-template-columns`

2. *Contenedor*: Las cuadrículas del grid se pueden nombrar de izquierda a derecha y de arriba a abajo, por filas con `grid-template-areas`
3. *Hijos*: Asocio ese nombre a una etiqueta HTML con `grid-area`

EJEMPLO:

1. Definir el grid

```
main{
  display:grid;
  grid-template-columns: 150px 1fr;
  grid-template-rows: 50px 1fr 30px;}
```

Esto me da un grid de 2 columnas (cada una de un tamaño: 150px 1fr) x 3 filas (cada una de un tamaño: 50px 1fr 30px)

1	2
3	4
5	6

2. Ponerles nombre con `grid-template-areas`

```
grid-template-areas: "head head"
                     "nav   main"
                     "nav   foot";
```

Head	Head
Nav	Main
Nav	Foot

3. Asociarlas a un/os elemento/s html con `grid-area`

```
#page > header {
  grid-area: head;
  background-color: #8ca0ff;
}

#page > nav {
  grid-area: nav;
  background-color: #ffa08c;
}

#page > main {
  grid-area: main;
  background-color: #fffff64;
}
```

Resultado:

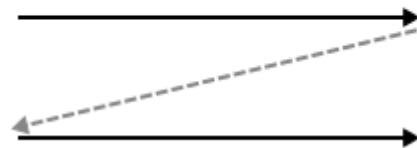


4. **grid-auto-flow:** controla cómo funciona el algoritmo de auto-llenado de celdas especificando cómo se autocolocarán los elementos.

Valores:

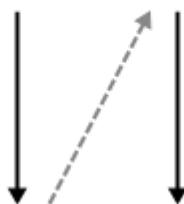
- **row:** Por defecto. Los items se colocan llenando las filas. Si es necesario añade más filas.

grid-auto-flow: row



- **column:** los items se colocan llenando las columnas. Si es necesario añade más columnas

grid-auto-flow: column



- **dense:** El algoritmo dense intenta llenar los huecos antes en la cuadrícula, si más tarde aparecen elementos más pequeños. Esto puede hacer que los elementos parezcan desordenados, cuando al hacerlo se llenarían los huecos dejados por elementos más grandes.

Si se omite, se utiliza un algoritmo donde el algoritmo de ubicación solo se mueve "hacia adelante" en la cuadrícula al colocar elementos, nunca retrocede para llenar los huecos. Esto asegura que todos los elementos colocados automáticamente aparezcan "en orden", incluso si esto deja huecos que podrían haberse llenado con elementos posteriores.

5. `grid-auto-rows`/ `grid-auto-columns`: especifica el tamaño (fila/columna) de una cuadrícula creada *implícitamente* (las explícitas van con `grid-template-columns/rows`)

VALORES:

`<length>`

Es una medida no negativa.

EJEMPLO:

```
/* Valores <longitud> */
grid-auto-rows: 100px;
```

`<porcentaje>`

Es un `<porcentaje>` no negativo relativo al tamaño del bloque del contenedor de la grilla. Si el tamaño del bloque del contenedor de la grilla es indefinido, el valor del porcentaje es tratado como `auto`.

EJEMPLO:

```
/* Valores <porcentaje> */
grid-auto-rows: 10%;
```

`<flex>`

Es una dimensión *no negativa* con la unidad `fr` especificando el factor flex de la vía. Cada vía con valor `<flex>` toma una parte del espacio restante en proporción con su factor flex.

Cuando aparece fuera de una notación `minmax()`, implica un mínimo automático (Ejemplo: `minmax(auto, <flex>)`).

EJEMPLO:

```
/* Ejemplo de valores <flex> */
grid-auto-rows: 0.5fr;
grid-auto-rows: 3fr;
```

`max-content`

`grid-auto-rows: max-content;`

Representa la longitud máximo del contenido de cada uno de los ítems de la vía en la cuadrícula (grid)

`min-content`

```
grid-auto-rows: min-content;
```

Representa la longitud mínima del contenido de cada uno de los ítems de la vía en la cuadricula (grid)

```
minmax(mínimo, máximo)
```

Define el rango del tamaño, dicho tamaño debe ser mayor o igual al (mínimo) y menor o igual al (máximo).

- a) Si el parámetro (máximo) es más pequeño que el parámetro (mínimo), entonces el parámetro (máximo) es ignorado y la función se tratará con el parámetro (mínimo).
- b) Si el valor <flex> pone como mínimo el factor flex de la pista, será tratado como cero (o el contenido mínimo, si el contenedor grid es inicializado con el tamaño mínimo permitido).

EJEMPLO:

```
/* Ejemplo de valores minmax() */  
grid-auto-rows: minmax(100px, auto);  
grid-auto-rows: minmax(max-content, 2fr);  
grid-auto-rows: minmax(20%, 80vmax);
```

```
auto
```

```
grid-auto-rows: auto;
```

Es una palabra clave que es idéntica a contenido máximo si es un máximo. Como mínimo representa el valor mínimo más grande (como esté especificado por min-width/min-height) de los elementos de la grilla que ocupan la pista de la grilla.

EJEMPLOS MEZCLADOS:

```
/* Múltiples valores de tamaño de pista(fila) */  
grid-auto-rows: min-content max-content auto;  
grid-auto-rows: 100px 150px 390px;  
grid-auto-rows: 10% 33.3%;  
grid-auto-rows: 0.5fr 3fr 1fr;  
grid-auto-rows: minmax(100px, auto) minmax(max-content, 2fr)  
minmax(20%, 80vmax);  
grid-auto-rows: 100px minmax(100px, auto) 10% 0.5fr fit-  
content(400px);
```

EJEMPLO: GRID-AUTO-ROWS

6. **grid-column / grid-row:** Especifica dónde empiezan y acaban las vías (mediante números de líneas).

```
grid-column: <grid-column-start> / <grid-column-end>
```

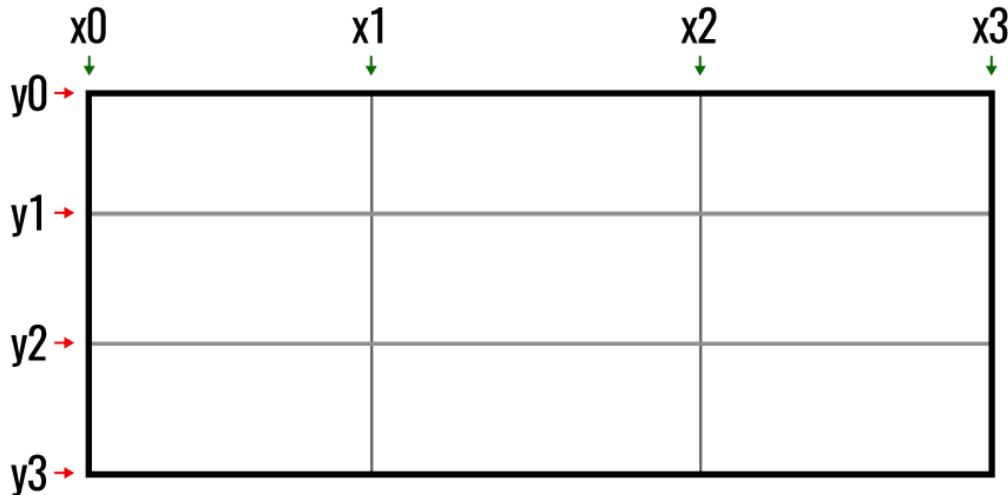
```
grid-row: <grid-row-start> / <grid-row-end>
```

Valores posibles :

auto

Indica auto-colocación, un span automático, o por defecto un span de 1.
<integer> && <custom-ident>?

- a) Si se especifica un valor negativo se empezará a contar desde el final del eje.
- b) Si se indica un nombre, <custom-ident>, sólo las líneas con ese nombre serán contadas (las defino al definir el grid)



```
grid-template-columns: [x0] 1fr [x1] 1fr [x2] 1fr [x3];
/* [x3]-> no necesita especificarle anchura porque es la última
línea*/
grid-template-rows: [y0] 1fr [y1] 1fr [y2] 1fr [y3];
/* [y3]-> no necesita especificarle anchura porque es la última
línea*/
```

- c) El valor 0 es inválido.

7. Huecos entre filas y columnas:

- **grid-gap**: Abreviatura para el espacio entre filas y columnas.
 - 1. Un único valor: ancho para ambas
 - 2. Dos valores: filas columnas
- 8. **grid-template**: Abreviatura para definir el grid <filas> / <columnas>

9. Alineación:

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

Estas 2 últimas propiedades se *definen* 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, sobre escribiendo su distribución.

10. Alineación: 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

11. Atajo de alineaciones:

- **place-items**: [align-items] [justify-items]
- **place-content**: [align-content] [justify-content]

12. **grid**: Permite definir todas las propiedades *grid* explícitas ([grid-template-rows](#), [grid-template-columns](#), y [grid-template-areas](#)), implícitas ([grid-auto-rows](#), [grid-auto-columns](#), y [grid-auto-flow](#)), y relativas al espacio entre las celdas ([grid-column-gap](#) y [grid-row-gap](#)) en una sola declaración.

```
grid: <grid-template> <grid-auto-flow> <grid-auto-rows> / <grid-auto-columns>
```

EJEMPLOS:

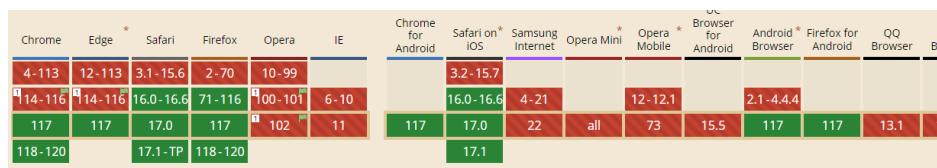
```
.grid {  
  grid: 100px 20px;  
  grid: 200px repeat(2, 100px) 300px;  
  grid: row;  
  grid: column dense;  
  grid: row 200px;  
  grid: row 400px / 150px;  
}
```

RESUMEN:

Propiedad	Valor	Explicación	Se define en
<code>display</code>	<code>grid inline-grid</code>	Define la cuadrícula (Contenedor)	Contenedor
<code>grid-template-columns</code>	AnchoCol1 AnchCol2 ...	Ancho /Alto columnas/filas—Explicita (=la que defino)	Contenedor
<code>grid-template-rows</code>	Altofila1 altofila2 ...		
<code>grid-auto-rows</code>	longitud porcentaje valor_flex max-content min_content max_content minmax	Tamaño (fila/columna) – IMPLÍCITA (=las que se añadirán automáticamente si no llega con la explícita)	Contenedor
<code>grid-column-start</code> <code>grid-column-end</code>	Número	Comienzo-Fin de la vía en base a las líneas	Hijo
<code>grid-column: X / Y</code> <code>grid-row: X/Y</code>	X: Línea de inicio Y: Línea de final		
<code>grid-column: X / span Y</code> <code>grid-row: X/span Y</code>	X: Línea de inicio Y: Cuántas líneas se expande. Línea final X+Y		
<code>grid-template-areas</code>	“nombre1” “nombre2” ...	Poner nombre a las vías	Contenedor
<code>grid-area</code>	“nombre”	Asociar a la etiqueta un área	Hijo
<code>grid-auto-flow</code>	<code>row column dense</code>	Algoritmo de auto llenado	Contenedor
<code>grid-gap</code>	- Un valor: igual separación filas y columnas - Dos valores: filas columnas	Hueco de separación entre las filas y columnas	Contenedor
<code>column-gap</code> <code>row-gap</code>	Hueco entre columnas Hueco entre filas		

SUBGRID

Se puede declarar en un elemento las columnas o filas como subgrid cuando el elemento padre está declarado como grid. De esta manera, el elemento hijo herederá las propiedades del elemento contenedor.



¿CUÁNDO USAR DISPLAY:FLEX O DISPLAY:GRID?

1. Grid está basado en el contenedor, Flex está basado en el contenido.

- En una flexbox, el tamaño de la celda (flex-item) está definido dentro del propio item.
 - En grid, el tamaño de la celda (grid-item) está definido en el contenedor grid.
-
- Un caso de uso ideal para flexbox es cuando tienes un conjunto de ítems y quieres espaciarlos uniformemente en un contenedor. Dejas que el tamaño del contenido decida cuánto espacio ocupa cada ítem. Si los ítems se envuelven (wrap) en una nueva línea, calcularán su espacio basándose en su tamaño y el espacio disponible *en esa línea*.
 - Cuando usas CSS Grid Layout creas un diseño y luego colocas elementos en él, o permites que las reglas de auto-placement coloquen los elementos en las celdas de la cuadrícula de acuerdo con esa cuadrícula estricta. Existe la posibilidad de crear pistas que respondan al tamaño del contenido, sin embargo, también cambiarán toda la pista.

Si estás usando flexbox y estás deshabilitando parte de la flexibilidad, probablemente necesites usar CSS Grid Layout

2. Grid tiene la propiedad gap, flexbox no.

- Grid: **grid-column-gap**
- Flex: **justify-content, flex-items**

3. Flexbox is Unidimensional, Grid es Bidimensional

- Flexbox es mejor para trabajar con elementos en una única fila/columna.
- Grid es mejor para trabajar con elementos en más de 1 fila/columna.

4. Flexbox Wraps vs Grid Wraps

Cuando los items dentro del contenedor no caben en la anchura del contenedor, ambos modelos tienen la opción de desplazar items a una nueva fila.

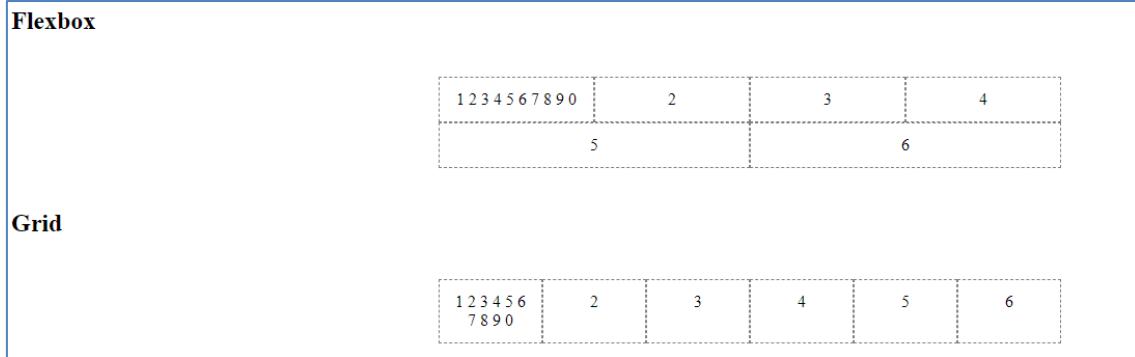
- Flex:

`flex: 1 1 100px` → para darles a los flex-items un tamaño inicial de 100px y permitirles crecer/decrecer (**grow/shrink**).

`flex-wrap: wrap` → Para permitir a los elementos que no caben fluir a otra línea.

- Grid: Utiliza la combinación de las funciones `minmax` y `auto-fill` (llenado)/`auto-fit` (acomodación) dentro de la propiedad `grid-template-columns` property.

`grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));` → Crea tantas columnas (mínimo:100px-máximo 1fr) como quepan en el contenedor.
`grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));`
→ Acomoda los items hijos en el contenedor e ignora su anchura. El segundo valor tiene que estar especificado en la unidad fr.



EJEMPLO: AUTO-FILL Y AUTO-FILL

5. Grid es para maquetación, Flex para alineación

- Puedes considerar utilizar Flexbox cuando:
 - Tienes un diseño pequeño que implementar: Es ideal cuando tienes una maquetación con pocas filas y columnas.
 - Necesitas alinear elementos: Lo único que hay que tener en cuenta es que el contenedor debe ser flex (display: flex) y después definir la flex-direction que quieras
 - Necesitas un diseño content-first: Es ideal para crear páginas web si conoces exactamente cómo se va a ver el contenido que se mostrará.
- CSS grid es mejor cuando:
 - Tienes un diseño complejo que implementar.
 - Necesitar tener hueco entre los elementos de bloque
 - Necesitas sobreponer elementos: Es sencillo utilizando las propiedades grid-column y grid-row. En flexbox necesitas para hacerlo márgenes, transformaciones o posiciones absolutas.
 - Necesitas un diseño basado en la maquetación.

Fuentes:

https://developer.mozilla.org/es/docs/Web/CSS/CSS_Grid_Layout/Conceptos_B%C3%A1sicos_del_Posicionamiento_con_Rejillas

<https://lenguajecss.com/css/maquetacion-y-colocacion/grid-css/>

<https://www.adictosaltrabajo.com/2018/01/30/maquetacion-con-css-grid/#responsividad-avanzada>