# ¿Qué es CSS Grid Layout?

1/13

Bienvenido y bienvenida a este curso de CSS Grid Básico. Pondremos en práctica algunos conceptos que ya hemos aprendido y conoceremos otros nuevos para posicionar elementos dentro de contenedores de la manera en que deseemos. Primero, entendamos lo que es CSS Grid.

**Qué es CSS Grid**

Css grid es una especificación de CSS para realizar layouts más dinámicos. Anteriormente sólo se podía usar tablas para realizar lo que ahora podemos hacer con Grid de manera mucho más sencilla. Nos permite dividir una página en áreas o regiones principales, por definir la relación en términos de tamaño, posición y capas construidas a partir de HTML

Interfaz de usuario gráfica

Descripción generada automáticamente

CSS Grid introdujo un sistema de grilla que es una cuadrícula con columnas (columns) y filas (rows), con ellas podemos ubicar elementos de manera más fácil.

Interfaz de usuario gráfica

Descripción generada automáticamente

Podemos crear, por ejemplo, un layout simple con el header, main y footer ya que cada bloque conforma un elemento de la grilla que se puede ubicar. Como los layouts pueden cambiar mucho, Grid nos ayuda a posicionar y reposicionar los elementos cuantas veces necesitemos.

Te recomiendo echar un vistazo a la [guía completa de CSS Grid](https://css-tricks.com/snippets/css/complete-guide-grid/).

# Conceptos para comenzar

2/13

Antes de empezar a trabajar con CSS Grid, es importante que conozcamos algunos conceptos básicos y así tener un mejor entendimiento de nuestro código. Vamos a ello.

**Qué es contenedor**

Es el elemento que se va a convertir en una grilla. Algo como una caja en la que vamos agregando elementos.

Texto

Descripción generada automáticamente

**Qué son items**

Son los elementos que están dentro del contenedor. Estos elementos pueden ser de cualquier tipo como links, botones, imágenes, etc. Todos ellos se convertirán en grid items. Esto quieres decir que por defecto van a tener propiedades que los ayudan a trabajar muy bien con nuestro sistema de grilla.

Texto

Descripción generada automáticamente

**Qué son líneas**

Son los elementos que limitan o dividen las filas o columnas de una grilla. Como vemos en el ejemplo, cuatro líneas dividen las tres columnas (columns), y se lee de izquierda a derecha. Tres líneas dividen las 2 filas (rows) y se enumeran de arriba hacia abajo.

Escala de tiempo

Descripción generada automáticamente

**Qué es celda**

Es la unidad mínima que nosotros podemos tener en una grilla. Está delimitada por 4 líneas y normalmente solo ocupa 1 fila y 1 columna.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

**Qué es track**

Es un grupo de celdas, que solo puede estar en una fila o en una misma columna.

Gráfico, Histograma

Descripción generada automáticamente

**Qué es área**

Es un grupo de celdas, estas pueden usar varias filas o varias columnas a la vez.

Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

Bastante simple, ¿verdad? Pues ya estamos listos para empezar a codificar nuestra primera grilla.

# Propiedades del contenedor

3/13

Ahora vamos a conocer algunas propiedades el contenedor que nos permiten construir grillas especificando la cantidad de filas, columnas y los tamaños que manejan entre ellos.

## Qué es el display grid

Define el elemento como contenedor de cuadrícula y establece un nuevo contexto de formato de cuadrícula para su contenido.

<div class="contenedor">

<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>

El div principal o “contenedor” va a ser el padre de nuestra grilla. Los seis div internos serán los elementos o grid items.

.contenedor {

border: 5px solid #e1bee7;

background-color: #fff1ff;

}

.item {

border: 5px solid #00bcd4;

font-size: 4rem;

}

En nuestra hoja de estilos agregamos un color de fondo y un borde al contenedor principal, mientras que un borde y un tamaño de fuente a los elementos del contenedor.

Al renderizar el código en el navegador obtendríamos este resultado:



Los elementos se posicionan uno debajo de otro y funciona como un solo bloque, ya que por defecto el navegador asigna un contenedor como block. Sólo con el hecho de agregar la propiedad display: grid al código de nuestro contenedor principal, el navegador ya lo interpreta como tal, a pesar de no cambiar visualmente. Vamos a generar algunas columnas.

.contenedor {

border: 5px solid #e1bee7;

background-color: #fff1ff;

display: grid;

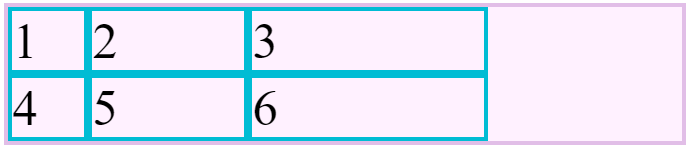
grid-template-columns: 100px 200px 300px ;

}

### Grid-template-columns y grid-template-rows

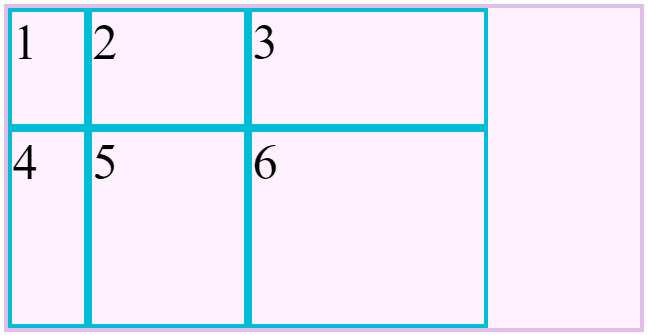
Define las columnas y filas de la cuadrícula con una lista de valores separados por espacios. Los valores representan el tamaño de la pista y el espacio entre ellos representa la línea de la cuadrícula.

**grid-template-columns** nos permite especificar el número de columnas y su ancho. En este caso usamos tres tamaños, cada uno **100px** mayor al anterior. Vemos en el navegador que al llegar a la tercera columna, los elementos se mueven a la siguiente fila.



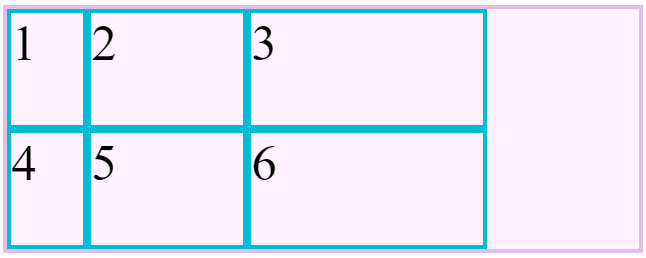
**grid-template-rows** nos permite especificar el número de filas y su ancho. En este caso usamos dos tamaños, el primero de 150px y el segundo de 250px.

**grid-template-rows**: 150**px** 250**px**;



grid-auto- Nos permite definir una altura a todas las columnas y filas. Es decir, si agregáramos otra columna o fila también tendría esta dimensión.

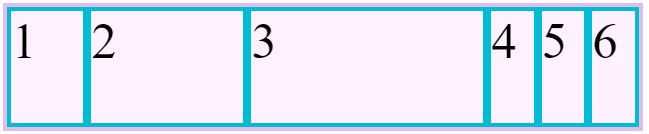
**grid-auto-rows**: 150**px**;



### **Grid-auto-flow**

Nos permite cambiar el orden en el que están los grid items, este valor por defecto es row (fila), pero lo podemos cambiar.

**grid-auto-flow** nos permite modificar cómo se está llenando la grilla. Cuando los elementos no entran en la cantidad de filas definidas, crea nuevas. **grid-auto-flow: column** crea nuevas columnas.



Como observamos, todos los elementos se han alineado en una sola fila, esto es porque no hemos definido en el código la cantidad de filas que queremos. Vamos a establecer que las filas solo tengan dos dimensiones, mientras que todas las columnas tendrán **100px**. De esta manera:

**grid-template-rows**: 100**px** 200**px**;

**grid-auto-columns**: 100**px**;

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

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Vemos en el navegador que incluso el orden en que se organizan los elementos cambia. Todas las columnas tienen un mismo ancho. Puedes usar estas propiedades para jugar según lo que busques.

## ****Qué son gaps****

Es un espacio entre filas y columnas.

**row-gap** nos permite definir el tamaño de la brecha entre filas. Por supuesto, también lo puedes hacer en las columnas con **column-gap**. En este caso usamos\*\*\*😗\*\*

**row-gap**: 15**px**;

**column-gap**: 30**px**;

**Ojo**: los espacios solo se crean entre filas o columnas, no con los extremos del contenedor.

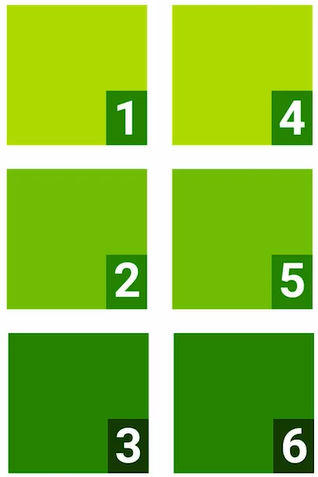
Imagen que contiene Forma

Descripción generada automáticamente

**gap**, por otro lado nos permite crear un espaciado tanto para filas y columnas. Por ejemplo: **gap: 40px** crea el mismo espaciado entre sí, tanto en filas como en columnas.

## Completa este reto

¡Aprendimos muchoooo! Ahora te toca participar. Como reto tendrás que construir un grid de tres filas y dos columnas con un gap de 20px entre ellas. Muy sencillo, ¿verdad? Como referencia, los números de los elementos deben coincidir con los de la siguiente imagen:



Como dato adicional, te dejo una página en la que podrás jugar y practicar tu conocimiento sobre CSS Grid y el manejo de filas y columnas. Para ello ve a [Grid Garden](https://cssgridgarden.com/" \l "es" \t "_blank).

# Propiedades de alineación

4/13

Ahora vamos a revisar algo que consideramos un super poder en CSS Grid. Nos permitirá alinear los elementos en nuestro contenedor de la manera en que deseemos. También puede ver el concepto de [alineación desde el diseño gráfico.](https://platzi.com/clases/1228-fundamentos-diseno/10122-que-es-la-alineacion/)

## Cómo funcionan las propiedades de alineación

Vamos al editor de texto. Creamos un contenedor con lo que hemos aprendido. Usaremos este contenedor para ejemplificar cómo funcionan las propiedades de alineación.

.contenedor {

border: 5px solid #e1bee7;

background-color: #fff1ff;

display: grid;

grid-gap: 15px;

grid-template-columns: 150px 150px 150px;

grid-auto-rows: 150px;

grid-auto-flow: column;

height: 600px;

}

Imagen que contiene Diagrama

Descripción generada automáticamente

### **Propiedades de alineación de los ítems**

#### **Justify-items**:

Todos los que empiezan con justify nos ayudan a ordenar los elementos de manera horizontal en el espacio. **justify-items: center** nos da:

Imagen que contiene Gráfico en cascada

Descripción generada automáticamente

#### **Align-items**:

Todos los que empiezan con align nos ayudan a alinear los elementos de manera vertical en el espacio. **align-items: center** nos da:

Imagen que contiene Aplicación

Descripción generada automáticamente

#### **Place-items**:

Es la mezcla de justify-items y align-items. **place-items: center** nos da:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

### **Propiedades de alineación del contenedor:**

Estas propiedades son las que ajustan la grilla completa al espacio en la que ella vive. No tiene que ver nada con los elementos. La grilla funciona como un bloque.

#### Justify-content: \*justify-content:

center nos da este resultado:

Imagen que contiene Diagrama

Descripción generada automáticamente

#### Align-content: \*align-content:

center nos da este resultado:

Icono

Descripción generada automáticamente con confianza media

#### Place-content: place\*-content:

center nos da este resultado:

Icono

Descripción generada automáticamente

Además de estas propiedades de alineación, tenemos otras que dan directamente propiedades a los hijo de la grilla o grid items. Es decir, podemos alinear los elementos dentro de los contenedores de manera independiente. Estas son:

* **Justify-self**
* **Align-self**
* **Place-self**

## Completa este reto

¿Listo para poner a prueba tus conocimientos?

Como reto, tendrás que crear un grid de 2x2 en que los elementos se ubiquen en la esquina inferior derecha del contenedor. El contenedor debe estar centrado.

¡Muchos éxitos!

Gráfico

Descripción generada automáticamente

# Propiedades de ubicación

5/13

Ya conoces las [propiedaddes del contenedor](https://platzi.com/clases/2474-css-grid/42184-propiedades-del-contenedor/" \t "_blank) y las [propiedades de alineación](https://platzi.com/clases/2474-css-grid/42185-propiedades-de-alineacion/). Vamos ahora a conocer las de ubicación.

## Cuáles son las propiedades de ubicación

Imaginemos que tenemos una grilla de 4x4 con sólo dos elementos. Podemos moverlos para que ocupen más espacio que una celda, o estén en una celda distinta a la que deberían estar.

Para las columnas, las propiedades que nos van a ayudar son:

* **grid-column-start**: con esta propiedad nosotros vamos a decirle al elemento en que línea de columna debe comenzar.
* **grid-column-end**: con esta propiedad nosotros vamos a decirle al elemento hasta que línea de la columna va a llegar.
* **grid-column**: es una mezcla de grid-column-start y grid-column-end. Los valores que necesita son el valor inicial y el valor final, estos van separados de un “/” (diagonal).

Para las filas las propiedades son iguales, solo que vamos a trabajar en las filas:

* **grid-row-start**
* **grid-row-end**
* **grid-row**

Además de estas propiedades, tenemos:

* **grid-area**: con esta propiedad declaramos solo una vez donde va a comenzar tanto en columna como en fila y donde va a terminar tanto en columna como en fila.

Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

## Cómo agregar las propiedades de ubicación

Vamos al editor de texto para comprender mejor estos conceptos. Primero creamos un contenedor con tres elementos a los que les agregamos clases para poder aplicar estilos individuales a cada uno.

<**div** class="contenedor">

<**div** class="item item-1">1</**div**>

<**div** class="item item-2">2</**div**>

<**div** class="item item-3">3</**div**>

</**div**>

En nuestro archivo CSS llamamos al contenedor con su clase y le aplicamos propiedades similares a las que hemos trabajado anteriormente.

Ajustamos el ancho de fuente en 4rem para todas las clases.  
Le damos un borde y color de fondo distinto a cada elemento del contenedor para poder diferenciarlo.

.contenedor {

border: 5px solid #e1bee7;

background-color: white;

display: grid;

grid-template-columns: 150px 150px 150px;

grid-template-rows: 75px 75px 75px;

place-content: center;

}

.item {

font-size: 4rem;

}

.item-1 {

border: 5px solid #f8bbd0;

background-color: #ffeeff;

}

.item-2 {

border: 5px solid #e1bee7;

background-color: #fff1ff;

}

.item-3 {

border: 5px solid #b2ebf2;

background-color: #e5ffff;

}

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### Item 1

Ahora, editemos el primer elemento para que ocupe todo el ancho de la primera fila. Debe empezar a numerar los elementos de izquierda a derecha.

Mucho ojo, el número no es la columna en sí, sino la línea en que termina la columna. Entonces si queremos que ocupe hasta la tercera celda, tendremos que colocar grid-column-end: 4.

.item-1 {

border: 5px solid #f8bbd0;

background-color: #ffeeff;

grid-column-start: 1;

grid-column-end: 4;

}

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteCalendario

Descripción generada automáticamente

### Item 2

En este caso queremos que ocupe la primera celda de las filas 2 y 3 dentro de la primera columna. Como la primera fila ya está ocupada, debemos empezar de la fila 2 y terminar en la 4.

grid-row: 2/4

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### Item 3

El tercer item ocupa tanto filas como columnas, por lo que conviene usar grid-area. El orden en que debemos colocar el valor es: fila y columna en que empieza, fila y columna en que termina.

.item-3 {

border: 5px solid #b2ebf2;

background-color: #e5ffff;

grid-area: 2/2/4/4;

}

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## ¿Existen atajos para usar las propiedades?

Podemos usar grid-template-areas para acomodar todos los elementos con mucha más rapidez. grid-template-areas declaramos areas dentro de la grilla.

Para ello, agregamos la información a la tabla como si fuera dentro de una grilla, de esta manera:

.contenedor {

border: 5px solid #e1bee7;

background-color: white;

display: grid;

grid-template-columns: 150px 150px 150px;

grid-template-rows: 75px 75px 75px;

grid-template-areas:

"header header header"

"side main main"

"side main main";

place-content: center;

}

Luego, colocamos el nombre de la celda creada anteriormente a cada item.

}

.item-1 {

border: 5px solid #f8bbd0;

background-color: #ffeeff;

grid-area: header;

}

.item-2 {

border: 5px solid #e1bee7;

background-color: #fff1ff;

grid-area: side;

}

## Completa este reto

Aprendiste varias formas de llegar a un mismo resultado. Ya sólo te queda jugar para seguir practicando lo aprendido. Pero adivina, te tengo un reto para ello.

Debes recrear la grilla que ves en al siguiente imagen. Estás en la total libertad de usar el método que consideres el mejor o simplemente el que quieras usar. Recuerda que en programación no hay una sola manera de lograr las cosas.  
Gráfico, Gráfico de rectángulos

Descripción generada automáticamente con confianza media

# Funciones especiales

6/13

¿Qué te ha parecido CSS Grid hasta ahora? Apuesto a que no sabías que tiene funciones que lo hacen aún más genial. Veamos algunas de ellas usando un código como ejemplo para que sea más práctico.

## Funciones especiales en CSS grid

Primero creamos un contenedor principal con otros 9 contenedores dentro:

<**div** class="contenedor">

<**div** class="item item-1">1</**div**>

<**div** class="item item-2">2</**div**>

<**div** class="item item-3">3</**div**>

<**div** class="item item-4">4</**div**>

<**div** class="item item-5">5</**div**>

<**div** class="item item-6">6</**div**>

<**div** class="item item-7">7</**div**>

<**div** class="item item-8">8</**div**>

<**div** class="item item-9">9</**div**>

</**div**>

Luego, añadimos los estilos generales que hemos trabajado anteriormente.

.contenedor {

border: 5px solid #e1bee7;

background-color: #fff1ff;

display: grid;

grid-template-columns: 30px 200px 60px;

grid-template-rows: 100px 100px 100px;

}

.item {

border: 5px solid #00bcd4;

font-size: 2rem;

}

Imagen que contiene Icono

Descripción generada automáticamente

## Funciones minmax y repeat

Uno de los detalles a solucionar es que al reducir el ancho de la pantalla, el tamaño de las celdas no cambia, porque es absoluto. Si agregáramos más texto a la celda, esta se rompería de todas maneras porque no hemos cambiado el tamaño.

### **Minmax**:

Nos ayuda a declarar un tamaño mínimo y máximo para el ancho o alto de una celda. Podemos modificar el tamaño del contenedor sin importar el tamaño del contenido.

Entonces cambiamos los valores establecidos en las columnas; la primera en un ancho mínimo de 30px y un máximo de 300px, y la tercera con un mínimo de 60px y un máximo de 250px.

grid-template-columns: minmax(30px, 300px) 200px minmax(60px, 250px);

Calendario

Descripción generada automáticamenteCalendario

Descripción generada automáticamente

Como notamos, las celdas se adaptan al ancho del contenido, pero al redimensionarlas mucho más pequeñas, se desborda por el alto.

### **Repeat**:

Imaginemos que queremos una grilla con 12 columnas del mismo ancho. No vamos a escribir 12 veces el valor del ancho, sino una vez usando **repeat**.

Entonces, para que la celda se adapte al tamaño del contenido sin importar la dimensión, definimos la cantidad de celdas en que queremos aplicar la acción (tres) y damos la propiedad de auto. Si colocáramos un valor en pixeles realmente no solucionaríamos el problema.

grid-**template**-rows: repeat(3, **auto**);

Imagen que contiene Tabla

Descripción generada automáticamenteImagen que contiene Cuadrado

Descripción generada automáticamente

Notamos que ahora las celdas inicialmente son menos altas debido que sólo ocupan el espacio que ocupa el contenido. Y al dimensionarlas los contenedores no se rompen.

Genial, ¿verdad? Con estas funciones puedes jugar mucho con el contenido y redimensionarlo según tus intereses. Y lo mejor de todo, **evitar que tus grillas se rompan**.

# Keywords especiales

7/13

CSS tiene varias palabras claves que suelen resultar muy útiles al momento de crear código. Veamos algunas de ellas:

## Palabras clave especiales para creación de código

* **fr** : es una unidad relativamente nueva de medida especial de CSS Grid para darle ancho o alto a filas y columnas. 1fr representa una fracción del total de columnas o filas.
* **min-content** : ajusta el ancho de la celda lo mínimo posible sin romper su contenido.
* **max-content** : ajusta el ancho de la celda lo máximo posible para mostrar su contenido.
* **auto-fill** : logra que la grilla ocupe el 100% del espacio que tiene. Agrega columnas “fantasma” que rellenan el espacio sobrante del contenedor.
* **auto-fit** : ensancha las columnas para que ocupen todo el espacio del contenedor.
* **auto-fill** y **auto-fit:** ayudan a la grilla a ocupar el 100% del espacio disponible.

## Cómo funcionan las palabras clave especiales

Veamos cómo funcionan estas **keywords** directamente en el código:  
Primero, creamos un index.html con cuatro contenedores. Llevando el 3 y 4 distintas mezclas y espaciados.

<**div** class="contenedor">

<**div** class="item item-1">1</**div**>

<**div** class="item item-2">2</**div**>

<**div** class="item item-3">3 3 33 3 3 3 3 3 3 3 3 3 3 33333 3333 333</**div**>

<**div** class="item item-4">444 4 4 4 4 4 4 4 4 4444 4444 44 4 4 4 4</**div**>

</**div**>

Luego aplicamos en los estilos:  
Cuatro columnas de **100px** sin especificar su alto.

.contenedor {

border: 5px solid #e1bee7;

background-color: #fff1ff;

display: grid;

grid-template-columns: repeat(4, 100px);

}

.item {

border: 5px solid #00bcd4;

font-size: 2rem;

}

En el navegador observamos que todas las columnas tienen el mismo ancho.  
Texto

Descripción generada automáticamente con confianza media

Piensa en las columnas como fracciones: si la grilla tiene 4 columnas, todas del mismo ancho, cada columna es igual a una fracción de la grilla.  
Entonces, si en vez de colocar **100px** de ancho, colocamos **1fr**, logramos que cada columna siempre tenga el mismo ancho, ocupando siempre el 100% del ancho de la pantalla.

grid-template-columns: **repeat**(4, 1fr);

Imagen que contiene Forma

Descripción generada automáticamente

También podemos agregar distintos valores proporcionales al ancho de la columa. Por ejemplo, podemos hacer que una columna sea del doble de ancho de la anterior, y así con las demás.

grid-template-columns: 1fr 3fr 1fr 2fr;

Imagen que contiene Tabla

Descripción generada automáticamente

Podemos cambiar una sola columna para que use el ancho mínimo que ocupa su contenido con min-content. De esta manera, al redimensionar la pantalla, el ancho de las demás columnas cambiará, pero el de la modificada no, porque ya ocupa el ancho mínimo desde el principio.

grid-template-columns: 1fr 3fr min-content 2fr;

Imagen que contiene Tabla

Descripción generada automáticamente

Lo mismo pasa si queremos que la celda use el ancho máximo. Tomará el espacio necesario para mostrar su contenido en una sola línea.

grid-template-columns: 1fr 3fr max-content 2fr;

Forma, Rectángulo

Descripción generada automáticamente

Para explicar el **auto-fill** y **auto-fit**, creemos otro contenedor.

<**div** class="contenedor-1">

<**div** class="item item-1">1</**div**>

<**div** class="item item-2">2</**div**>

<**div** class="item item-3">3</**div**>

<**div** class="item item-4">4</**div**>

</**div**>

<**div** class="contenedor-2">

<**div** class="item item-1">1</**div**>

<**div** class="item item-2">2</**div**>

<**div** class="item item-3">3</**div**>

<**div** class="item item-4">4</**div**>

</**div**>

Aplicamos estilos:

.contenedor-1 {

border: 5px solid #e1bee7;

background-color: #fff1ff;

display: grid;

grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));

}

.contenedor-2 {

border: 5px solid #e1bee7;

background-color: #fff1ff;

display: grid;

margin-top: 100px;

grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));

}

Para que funcione, es necesario especificar el ancho mínimo y máximo del contenedor. En este caso queremos que comience desde 100px y crezca hasta llegar a una fracción de la grilla. No colocamos valores absolutos para que el contenedor siempre se ajuste al ancho de la pantalla.

Diagrama

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Como vemos en la grilla con **auto-fit** las columnas se expanden hasta ocupar todo el ancho. Mientras que en **auto-fill** se agregan columnas fantasma para rellenar el espacio faltante. Sin embargo, al reducir el tamaño vuelven a ocupar el mismo espacio mínimo.

Has aprendido bastante sobre CSS Grid, ¡ahora vamos a implementarlo a un [proyecto](https://platzi.com/clases/2474-css-grid/42189-proyecto/)!