Bootcamp Full Stack

CSS avanzado

Uso básico de Grid

Grid

Grillas en CSS

Grid es otro valor de **`display`** posible.

```
div { display: grid;}
```

La intención de este apartado es comprender las potencialidades que nos brinda esta propiedad cuando es **utilizada de forma complementaria**, en nuestra interfaz, para lograr maquetar el diseño prototipado. **No es para nada un reemplazo de flex**.

Cada una de esas dos propiedades tiene su contexto. Parte de la experiencia que ganamos con desarrolladores o maquetadores es entender cuándo conviene más una forma u otra.

El hecho de conocer el funcionamiento de todas las formas de trabajo es suficiente para poder, con el tiempo, hacerlo correctamente.

`display: grid`

Alimplementar **`grid` en el contenedor**, por ejemplo en la siguiente estructura, no veríamos grandes cambios.

```
<div>


   <p>1 parrafo</p>
   <p>2 parrafo</p>
   <p>3 parrafo</p>
   <p>4 parrafo</p>

                    </div>
```

Necesitamos implementar más propiedades para comenzar a trabajar.

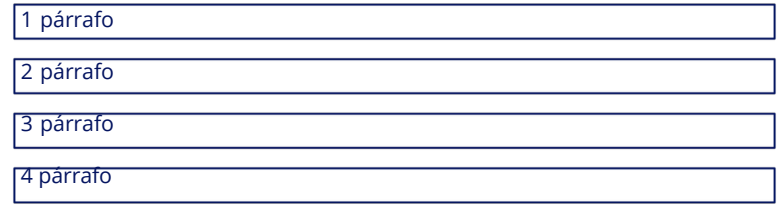
Veamos un ejemplo de utilización simple y los resultados que obtenemos.

En un **sistema de grillas** es fundamental tener presente que **tenemos filas y columnas**.
Para **determinar cuántas columnas** tendremos, `trabajamos con` **`grid-template-columns.`**
Esta propiedad toma **valores de longitud,** así como también la palabra **auto, para adaptarse al espacio disponible**.

También es importante saber cuántos elementos tenemos en total para comprender el resultado final. En la estructura anterior, un formato de **4 elementos,** con los siguientes estilos como vemos en la imagen:

```
div { display: grid; }

p { border: 1px solid black;}
```

Resultado

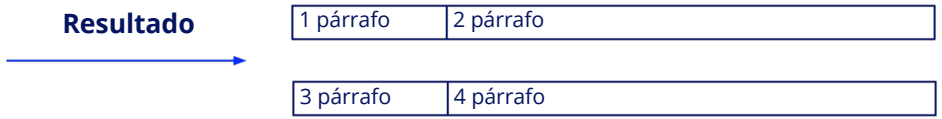


No es del todo prometedor y no implica demasiados cambios, veamos una imagen:

Sin embargo, si al código anterior le sumamos:

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;   }

p { border: 1px solid black;}
```

Resultado



Dependiendo de cuántas medidas indiquemos será la cantidad de columnas que se trabajarán.

Al trabajar con **grid-template-columns** los valores posibles son diversos según el objetivo planteado acorde al diseño a seguir.

El **porcentaje**, como siempre, toma en cuenta el valor del contenedor. Veamos un ejemplo:

index.html > html

```
12
13      <div id="contenedor">
14
15          <div>1</div>
16
17          <div>2</div>
18
19          <div>3</div>
20
21          <div>4</div>
22
23                  </div>
```

A partir del html del ejemplo en la izquierda y un **css** como el que se muestra debajo:

s > # estilos.css > #contenedor

```
1   #contenedor { width: 100px;
2   grid-template-columns: 50% 20%;
3   display: grid;}
4
5   #contenedor div {
6
7   border:1px solid red;
8
9   }
```

Resultado:

1	2
3	4

Si a partir del ejemplo anterior, nuestros estilos dijesen que ambas columnas ocupan un 50% (en este caso, dado que el *parent* tiene un width de 100px, sería 50px cada una), nos encontraríamos repitiendo dos veces el mismo valor.

```
css > # estilos.css > #contenedor div
1   #contenedor { width: 100px;
2   grid-template-columns: 50% 50%;
3   display: grid;}
4
5   #contenedor div {
6
7   border:1px solid red;
8
9
10  }
11
```

Resultado:

1	2
3	4

Para simplificarlo, podemos trabajar de la siguiente forma:

```
css > # estilos.css > ...
1   #contenedor { width: 100px;
2   grid-template-columns: repeat(2, 50%);
3   display: grid;}
4
5   #contenedor div {
6
7   border:1px solid red;
8
9
10  }
11
```

Resultado:

1	2
3	4

Trabajo con fr

Esta medida indica un proceso en **fracciones.** Donde 1fr es una parte del todo. Para entenderlo mejor sigamos el mismo ejemplo anterior, si queremos lograr el mismo resultado pero utilizando fr, nuestro css debería decir lo siguiente:

```
css > # estilos.css > ...
1   #contenedor { width: 100px;
2   grid-template-columns: repeat(2, 1fr);
3   display: grid;}
4
5   #contenedor div {
6
7   border:1px solid red;
8
9   }
```

Resultado:

1	2
3	4

Utilizando la medida anterior, podemos fijar un width a la primera columna y uno distinto a las restantes, por ejemplo con el siguiente html:

index.html > html > body > div#contenedor > section

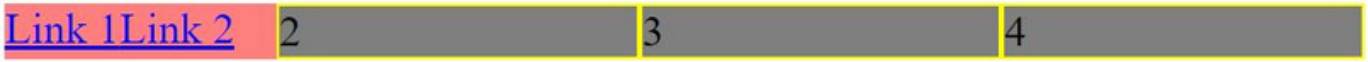
```
12
13      <div id="contenedor">
14
15          <nav><a href="#">Link 1</a><a href="#">Link 2</a></nav>
16
17          <section>2</section>
18
19          <section>3</section>
20
21          <section>4</section>
22
23              </div>
24
```

Y utilizando, junto al html del *slide* anterior, el css que vemos a continuación:

css > # estilos.css > ...

```css
#contenedor  { width: 500px;
grid-template-columns: 100px repeat(3, 1fr);
display: grid;}

#contenedor section  {

background-color: rgba(0,0,0,0.5);
border: 1px solid yellow;



}


#contenedor nav { background-color: rgba(255,0,0,0.5);}

```

El resultado de este último ejemplo será:

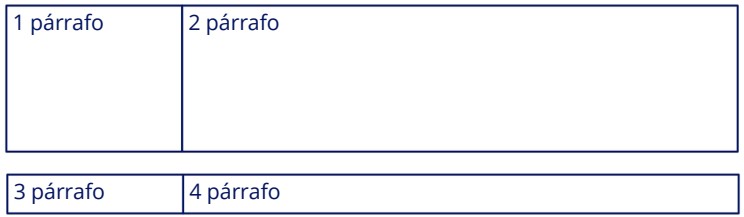


Trabajar con filas

El trabajo con **filas,** es similar. Lo realizamos a través de la propiedad **grid-template-rows**. **Esta propiedad indica el alto de las filas,** luego de que ya estén determinadas las columnas con `la propiedad` **`grid-template-column.`**

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;    }

p { border: 1px solid black;}
```

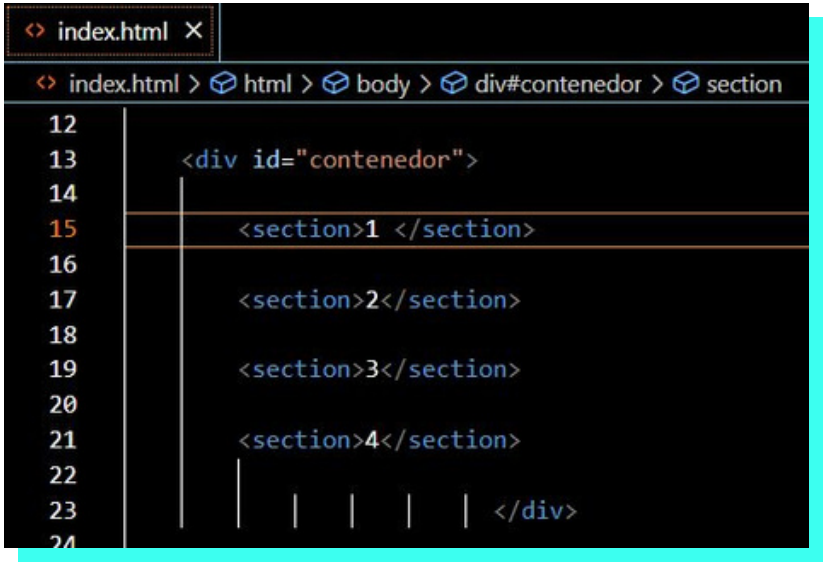
Resultado



fr y filas

Al igual que con grid-template-columns, en **grid-template-rows** se puede trabajar con `fr` u otras medidas de longitud.

Por ejemplo, con el siguiente html:



Utilizando el **css** debajo junto al html de la pantalla anterior:

estilos.css

```
css > # estilos.css > ...
1   #contenedor  { width: 500px;
2   grid-template-columns: 2fr 1fr;
3   grid-template-rows: 1fr 2fr;
4   display: grid;}
5
6   #contenedor section  {
7
8   background-color: rgba(0,0,0,0.5);
9   border: 1px solid yellow;
10
11  }
12
```

Resultado:

1	2
3	4

auto

El valor **auto** es utilizado muchas veces para que el espacio restante se tome automáticamente, por ejemplo:

```
<div id="contenedor">
    <section>1 </section>
    <section>2</section>
    <section>3</section>
    <section>4</section>
    <section>5 </section>
    <section>6</section>
    <section>7</section>
    <section>8</section>
</div>
```

Y utilizando el **css** que vemos a continuación:

css > # estilos.css > ...

```
1   #contenedor  { width: 500px;
2   grid-template-columns:  40px 50px  auto 50px  40px;
3   grid-template-rows:  25%  100px  auto ;
4   display: grid;}
5
6   #contenedor section  {
7
8   background-color: rgba(0,0,0,0.5);
9   border: 1px solid yellow;
10
11  }
```

El resultado será:

Generar espacios en la grilla

Column-gap y row-gap

Para generar espacios trabajamos con:

- **La propiedad column-gap:** representa el espacio entre las columnas.
- **La propiedad row-gap:** representa el espacio entre las filas.

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;
    column-gap:10px;
    row-gap:5px;
    }
```

Grid-gap y gap

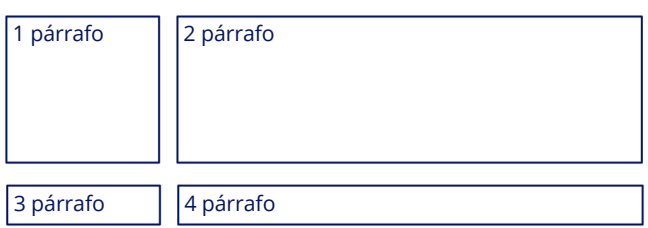
Como vimos, para generar espacios trabajamos con **column-gap** o **row-gap**. De todas maneras, siempre es interesante saber cómo trabajar de forma más ágil y sencilla.

La propiedad **grid-gap** (actualmente reemplazada por la propiedad **gap**) nos permite en una línea indicar el **column-gap** y el **row-gap**.

Siempre el primer valor responde a las filas y el segundo a las columnas, por tanto el ejemplo anterior sería:

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;
    gap: 5px 10px;
    }

p { border: 1px solid black;
    margin: 0; }
```

Resultado



Alinear el contenido

Para **alinear el contenido dentro de la grilla,** debemos trabajar con **justify-content**.

El valor **start** de esta propiedad permite orientar el contenido hacia el comienzo del contenedor:

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;
    grid-column-gap:10px;
    justify-content: start; }

p { border: 1px solid ■black;}
```

También, es posible alinear hacia el final, como podemos ver en la imagen, con el valor **end**:

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;
    grid-column-gap:10px;
    justify-content: end; }
```

Para **alinear o distribuir los elementos en forma horizontal o vertical,** podemos trabajar con las mismas propiedades que usamos en el trabajo con **`displayflex:`** **`justify-content`** o **`align-items:`**

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;
    gap: 15px 10px;
    justify-content: space-evenly;
     }

p { border: 1px solid ■black;
    margin: 0; }
```

Otra similitud con **flex**, es que también encontramos valores como **space-around** y **space-between:**

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;
    gap: 15px 10px;
    justify-content: space-between;
    }

p { border: 1px solid black;
    margin: 0; }
```

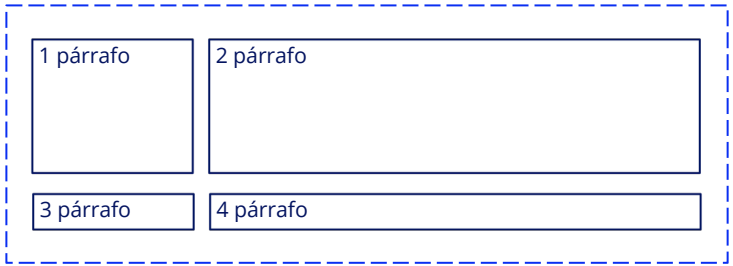
Resultado

Para **alinear en forma vertical** se utiliza **`align-content.`**

Podemos centrar el contenido con **`center`**:

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:120px;
    gap: 15px 10px;
    justify-content: space-between;
    align-content: center;
    border: 1px dashed blue;
    padding: 20px;

    }

p { border: 1px solid black;
    margin: 0; }
```

Resultado



Como ocurre en la alineación horizontal, en la vertical contamos con palabras como **end, start, y center.**

```
div { display: grid; width: 400px;
    grid-template-columns: 100px 300px;
    grid-template-rows:100px;
   padding: 10px;
   gap: 10px 15px;
    justify-content: space-between;
    align-content:end;
    border: 1px dashed blue;
    height: 400px;
    padding: 20px;


    }

p { border: 1px solid black;
    margin: 0; }
```

Revisión

- Repasar los conceptos vistos de **grid**.
- Implementar estas nuevas propiedades.
- Ver todos los videos y materiales necesarios antes de continuar

¡Sigamos trabajando!

IT College