Bootcamp Full Stack

CSS avanzado

Uso avanzado de Grid

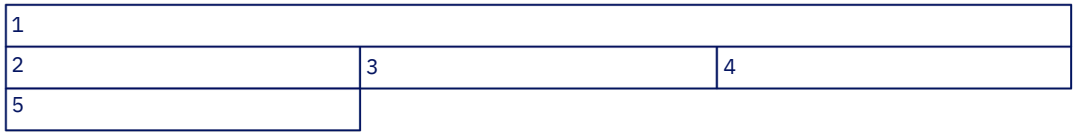
Expandir celdas

Para **expandir celdas** debemos trabajar con **`grid-column-start`** y **`grid-column-end`**. Estas propiedades permiten indicar dónde comienza y termina la celda.

En un sistema de tres columnas, si queremos que nuestra celda ocupe todo el espacio de la primera fila, el valor será *1* en start y *4* en end.

```
main { display: grid; grid-template-columns: auto auto auto ;}
div { border: 1px solid ■black; }

#expandir { grid-column-start: 1; grid-column-end: 4; }
```

Resultado



Para expandir celdas avanzando entre las filas, debemos trabajar con **grid-row-start** y **grid-row-end**. El concepto es el mismo.

En este caso, si queremos que nuestro elemento tome la segunda fila también lo haremos como se muestra en el ejemplo debajo.

Nótese que hemos hecho algunos cambios en las propiedades anteriores para lograr que esta nueva propiedad se efectivice:

```
main { display: grid; grid-template-columns: auto auto auto ;
       grid-template-rows: auto auto auto ;}
div { border: 1px solid black; }

#expandir { grid-column-start: 1; grid-column-end: 3;
            grid-row-start: 1; grid-row-end:3 ;  }
```

¿Sabías que...?

Existen propiedades muy interesantes como **grid-area**, que nos permite **nombrar áreas** para construir una grilla.

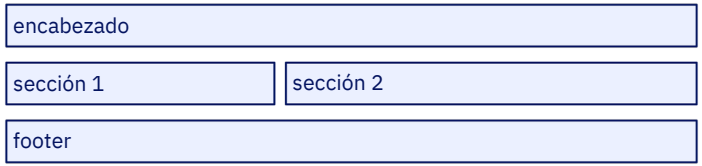
En nuestro siguiente ejemplo, nombramos múltiples áreas para construir una interfaz gráfica. Para lograr este proceso nuestros estilos se verán como se muestra en la diapositiva a continuación.

```
main { display: grid;
    grid-template-areas: 'encabezado encabezado encabezado' 'seccion1 seccion2 seccion2' 'footer footer footer';
    grid-gap: 4px;}
div { border: 3px solid black; background-color: lightblue; }

#encabezado { grid-area: encabezado;}
.seccion1{ grid-area:seccion1;}
.seccion2 { grid-area: seccion2;}
#footer { grid-area:footer;}
```

Con la siguiente estructura **generada en nuestro .html:**

```
<main>
    <div id="encabezado">encabezado</div>
    <div class="seccion1">seccion 1</div>
    <div class="seccion2">seccion 2</div>
    <div id="footer">footer</div>

</main>
```

Resultado



Grid-column

Con `grid-template-columns y` grid-template-rows generamos la grilla en sí. La propiedad grid-column permite ubicar específicamente un ítem en una grilla.

Hasta ahora, lo que hacíamos era dejar que el *flow* natural de los elementos de html decidieran dónde ubicarse.

Veamos, por ejemplo, el siguiente **html**:

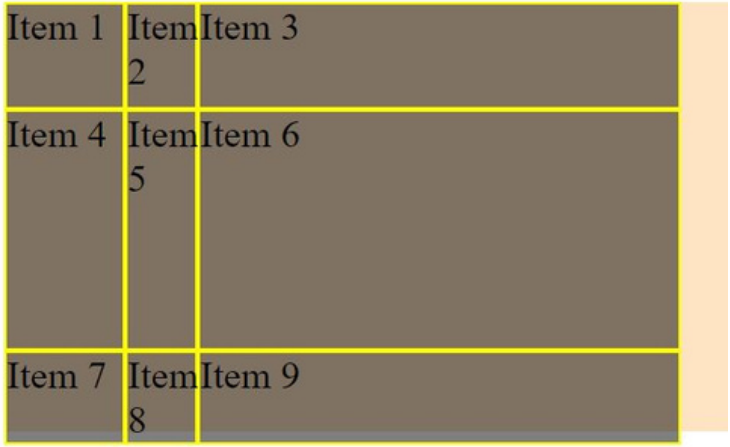
```
<div id="contenedor">

    <section id="item1">Item 1 </section>

    <section>Item 2</section>

    <section>Item 3</section>

    <section>Item 4</section>

    <section>Item 5 </section>
    <section>Item 6 </section>
    <section>Item 7 </section>
    <section>Item 8 </section>
    <section>Item 9 </section>


    </div>
```

A partir del html del *slide* anterior, trabajado junto con el **css** debajo:

css > # estilos.css > ...

```
1   #contenedor  { width: 600px;
2   grid-template-columns:  50px 30px 200px;
3   grid-template-rows:  25%  100px  auto ;
4   display: grid;
5   background-color:  bisque;
6   }
7
8   #contenedor section  {
9
10  background-color:  rgba(0,0,0,0.5);
11  border: 1px solid  yellow;
12
13  }
14
```

Resultado:



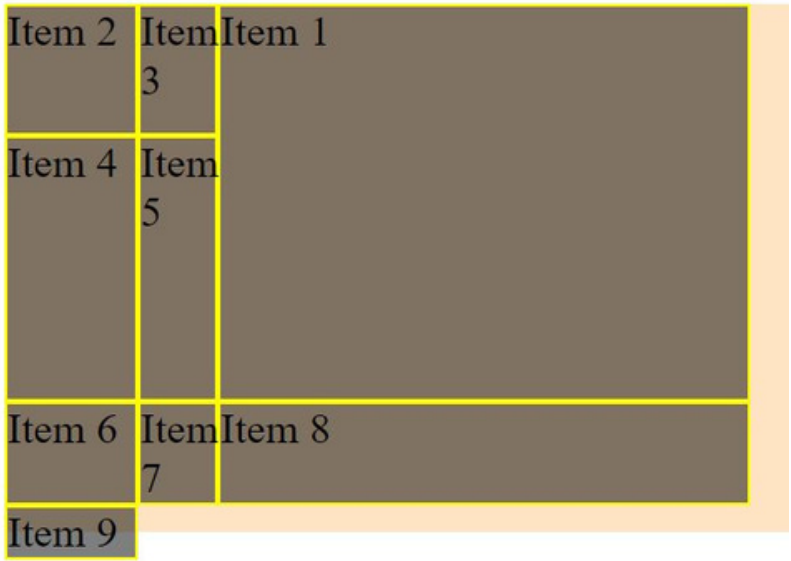
Si en el css, agregamos al **item1** lo siguiente, el resultado ubicará este ítem en el lugar específico. En este caso, columna 3 fila 1:

```
css > # estilos.css > ...
1  #contenedor  { width: 600px;
2    grid-template-columns:  50px 30px 200px;
3    grid-template-rows:  25%  100px  auto ;
4    display: grid;
5    background-color:  bisque;
6    }
7
8  #contenedor section  {
9
10   background-color:  rgba(0,0,0,0.5);
11   border: 1px solid  yellow;
12
13   }
14   #item1 { grid-column:3; grid-row: 1;}
```

Podemos ir más allá, e incluso utilizar esta propiedad para trabajar con el **span**, pero de forma más simple:

```
#item1 { grid-column:3; grid-row: 1/3;}
```

El resultado será que, aparte de ubicar el ítem en la columna 1, tomará 2 celdas de filas generando el siguiente resultado:



grid-auto -flow

En el resultado anterior, la celda 9 cae por una falta de previsibilidad de tamaños y espacios.

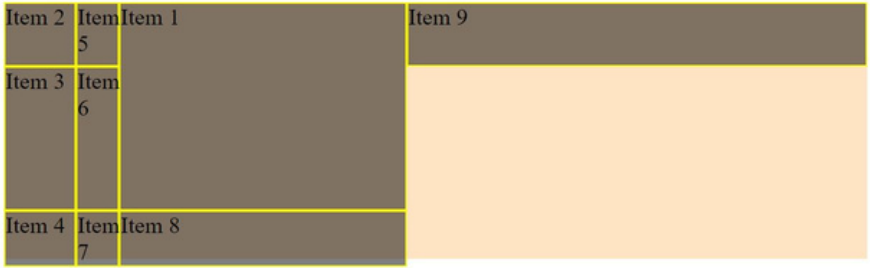
Pero podemos resolver cualquier tipo de inconveniente de este tipo sumando la propiedad **grid-auto-flow**, que permite indicar dónde queremos se posicionen aquellas celdas que no tengan espacio indicado explícitamente.

Por ejemplo:

```
css > # estilos.css > #contenedor
1   #contenedor {
2       width: 600px;
3       grid-template-columns: 50px 30px 200px;
4       grid-template-rows: 25% 100px auto;
5       display: grid;
6       background-color: bisque;
7       grid-auto-flow: column;
8   }
9
10  #contenedor section {
11
12      background-color: rgba(0, 0, 0, 0.5);
13      border: 1px solid yellow;
14
15  }
```

```
19  #item1 {
20      grid-column: 3;
21      grid-row: 1/3;
22  }
```

Resultado:



Seguramente, en este punto te preguntarás cuándo trabajar con grid y cuándo hacerlo con flex. **En general, grid se utiliza donde el diseño muestra claras filas y columnas,** no simplemente elementos encolumnados. Usualmente se utiliza en calendarios y galerías de imágenes.

Pero dejemos que ChatGPT nos ayude a conocer más ejemplos. Te sugerimos **trabajar en ChatGPT con el siguiente *prompt:***

Usos comunes de display: grid

Ejemplo de respuesta obtenida en ChatGPT para implementar este valor de display:

1. Diseño de formularios: Puedes utilizar `display: grid` para crear formularios con diseño personalizado, organizando los campos y etiquetas en una cuadrícula ordenada.

css — Copy code

```css
.form {
  display: grid;
  grid-template-columns: 1fr 2fr;
  gap: 10px;
}

.label {
  grid-column: 1 / 2;
}

.input {
  grid-column: 2 / 3;
}
```

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