LEMONCODE

HTML

Introducción a CSS Grid

A diferencia de Flexbox, que es un sistema unidimensional, CSS Grid Layout es un sistema de diseño bidimensional que permite crear diseños de páginas complejos y responsivos con facilidad. Ahora exploraremos las características básicas de Grid Layout con algunos ejemplos.

Para comenzar, necesitamos configurar un contenedor y añadir elementos dentro de él. Hemos creado un `div` con la clase `grid-container` y hemos agregado algunos ítems. Primero, vamos a aplicar estilos a nuestros ítems para visualizarlos mejor.

```html
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>CSS Grid Layout</title>
        <link rel="stylesheet" href="style.css" />
    </head>
    <body>
        <div class="grid-container">
            <div class="grid-item">1</div>
            <div class="grid-item">2</div>
            <div class="grid-item">3</div>
            <div class="grid-item">4</div>
            <div class="grid-item">5</div>
            <div class="grid-item">6</div>
        </div>
    </body>
</html>
```

```css
/* style.css */

.grid-item {
    background-color: #f1f1f1;
    padding: 20px;
    font-size: 30px;
    text-align: center;
}
```

Al igual que en Flexbox, para utilizar CSS Grid, debemos establecer la propiedad `display` del contenedor con el valor `grid`. En este caso, utilizaremos la clase `grid-container` con `display: grid`. Para definir las columnas y filas, utilizaremos `grid-template-columns` y `grid-template-rows`.

Notarás que utilizo una unidad de medida que quizás no hayas visto antes. La unidad `fr` en CSS Grid Layout es una medida flexible que representa una fracción del espacio disponible en el contenedor de la Grid. La palabra "fr" es una abreviatura de "fracción". Esta unidad nos permite dividir el espacio libre del contenedor de manera proporcional entre las filas y columnas que indiquemos, simplificando la distribución del espacio en todo el diseño de la Grid.

Por ejemplo, si tienes un contenedor con un ancho de 900 píxeles y deseas dividirlo en 3 columnas, puedes usar `1fr 1fr 1fr`, y cada una de esas partes ocupará 300 píxeles. Si quisieras dividir el espacio disponible en 4 partes, podrías utilizar `1fr 2fr 1fr`, y la columna del medio ocuparía el doble de espacio. Además, si añadimos un `gap` de 20 píxeles, podrás ver claramente que la columna del medio ocupa el doble de espacio.

```
/* style.css */

.grid-container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    gap: 20px;
}

.grid-item {
    background-color: #f1f1f1;
    padding: 20px;
    font-size: 30px;
    text-align: center;
}
```

Hay una forma de simplificar este código para no tener que escribir cada columna una y otra vez. Podemos utilizar la función `repeat` en `grid-template-columns` para indicar que queremos repetir 3 veces la unidad `fr`. Así que en lugar de escribir `1fr 1fr 1fr`, podemos escribir `repeat(3, 1fr)`. Obtendremos el mismo efecto, pero escribiremos menos código. Esto es especialmente útil cuando se trabaja con un gran número de columnas.

```
/* style.css */

.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 20px;
}

.grid-item {
    background-color: #f1f1f1;
    padding: 20px;
    font-size: 30px;
```

```
    text-align: center;
}
```

Esta técnica también se puede aplicar a la propiedad `grid-template-rows`, que permite definir filas en lugar de columnas.

Los elementos de la Grid se colocan automáticamente en las celdas, y son flexibles, adaptándose al espacio disponible. Sin embargo, también podemos especificar la posición exacta de un elemento utilizando `grid-column-start` y `grid-column-end`. Por ejemplo, si seleccionamos el primer hijo de nuestros ítems y le asignamos `grid-column-end: 3`, verás que se ha modificado su posición.

```
/* style.css */

.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 20px;
}

.grid-item {
    background-color: #f1f1f1;
    padding: 20px;
    font-size: 30px;
    text-align: center;
}

.grid-item:nth-child(1) {
    grid-column-end: 3;
}
```

Además, podemos definir áreas en la Grid y colocar elementos dentro de esas áreas. Primero, asignamos nombres a las áreas utilizando `grid-template-areas` en el contenedor y definimos los nombres de las áreas. Por ejemplo, podríamos establecer "header header header" seguido de "sidebar main main" y finalmente "footer footer footer". Esto suele ser un diseño de página muy común.

```
/* style.css */

.grid-container {
    display: grid;
    grid-template-areas:
        "header header header"
        "sidebar main main"
        "footer footer footer";
    gap: 20px;
}

.grid-item {
    background-color: #f1f1f1;
```

```
    padding: 20px;
    font-size: 30px;
    text-align: center;
}

.grid-item:nth-child(1) {
    grid-column-end: 3;
}
```

Luego, asignamos un nombre de área a cada uno de nuestros hijos, utilizando `grid-area`. Por ejemplo, podemos decir que el primer elemento será `grid-area: header`, y verás que ocupa las tres columnas especificadas. Si asignamos el nombre de área "sidebar" al segundo hijo, "mail" al tercero y "footer" al cuarto, verás cómo se posicionan los elementos.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>CSS Grid Layout</title>
        <link rel="stylesheet" href="style.css" />
    </head>
    <body>
        <div class="grid-container">
            <div class="grid-item">1</div>
            <div class="grid-item">2</div>
            <div class="grid-item">3</div>
            <div class="grid-item">4</div>
        </div>
    </body>
</html>
```

```
/* style.css */

.grid-container {
    display: grid;
    grid-template-areas:
        "header header header"
        "sidebar main main"
        "footer footer footer";
    gap: 20px;
}

.grid-item {
    background-color: #f1f1f1;
    padding: 20px;
    font-size: 30px;
    text-align: center;
```

```
}

.grid-item:nth-child(1) {
    grid-area: header;
}

.grid-item:nth-child(2) {
    grid-area: sidebar;
}

.grid-item:nth-child(3) {
    grid-area: main;
}

.grid-item:nth-child(4) {
    grid-area: footer;
}
```

Es importante destacar que los hijos de un contenedor Grid pueden ser también Grid o incluso Flexbox. Además, es posible anidar un Grid dentro de otro. Simplemente debemos aplicar la propiedad `display: grid` al elemento hijo de la Grid.

A todos los elementos con la clase `grid-item` les hemos añadido `text-align: center` para centrar los números. Para demostrar que un hijo de una Grid puede ser Flexbox, hemos agregado `display: flex` y `justify-content: center` y como puedes ver los números se posicionan en el centro.

```
/* style.css */

.grid-container {
    display: grid;
    grid-template-areas:
        "header header header"
        "sidebar main main"
        "footer footer footer";
    gap: 20px;
}

.grid-item {
    background-color: #f1f1f1;
    padding: 20px;
    font-size: 30px;
    display: flex;
    justify-content: center;
}

.grid-item:nth-child(1) {
    grid-area: header;
}

.grid-item:nth-child(2) {
    grid-area: sidebar;
```

```
}

.grid-item:nth-child(3) {
    grid-area: main;
}

.grid-item:nth-child(4) {
    grid-area: footer;
}
```

Es importante aclarar que utilizar Grid Layout no invalida la posibilidad de utilizar Flexbox, y viceversa. Ambos sistemas se pueden combinar para lograr resultados más complejos y completos.

Para practicar y experimentar más con Grid Layout, te recomiendo utilizar herramientas online como "Layoutit" o "Grid Garden". Estas herramientas te permiten crear grids más complejas y probar su funcionamiento. Además, puedes copiar el código generado para usarlo en tus propios proyectos.

Con estos recursos, podrás seguir practicando y mejorando tus habilidades. Pronto nos enfocaremos en el proyecto final.