**CSS Grid Layout**

**Objetivos de Aprendizaje**

Al final de esta clase, los estudiantes serán capaces de:

1. Comprender la estructura básica de CSS Grid.
2. Crear y organizar elementos en una cuadrícula usando propiedades de CSS Grid.
3. Definir áreas de grid y posiciones de elementos.
4. Construir layouts responsivos usando fracciones (fr), unidades relativas y minmax.
5. Aplicar estos conceptos en ejemplos prácticos y casos de uso.

**1. Introducción a CSS Grid Layout**

* **Explicación**: CSS Grid es un sistema de diseño bidimensional que permite organizar elementos en filas y columnas. Con CSS Grid, se pueden crear layouts complejos y adaptables de forma intuitiva.
* **Ejemplo Visual**: Muestra un diseño de cuadrícula básica en una página y destaca que CSS Grid divide el área de trabajo en filas y columnas.

**2. Primera Estructura de Grid Básica**

**Paso 1: Crear el Contenedor y los Elementos de Grid**

* **HTML**:

html

Copiar código

<div class="grid-container">

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

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

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

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

</div>

* **CSS**:

css

Copiar código

.grid-container {

display: grid;

}

.item {

background-color: lightblue;

padding: 20px;

border: 1px solid #333;

text-align: center;

font-size: 1.5rem;

}

* **Explicación**:
  + La propiedad display: grid; transforma el contenedor en una cuadrícula.
  + Los elementos .item son las celdas del grid.
* **Discusión**: Sin más propiedades de grid, todos los elementos se apilan en una columna. Pregunta a los estudiantes: ¿cómo podemos organizarlos en filas y columnas?

**3. Definir Columnas con grid-template-columns**

* **CSS**:

css

Copiar código

.grid-container {

display: grid;

grid-template-columns: 1fr 1fr; /\* Dos columnas de igual tamaño \*/

}

* **Explicación**:
  + grid-template-columns define el número y tamaño de las columnas en el grid.
  + 1fr significa "una fracción" del espacio disponible. Aquí, dos columnas de igual ancho.
* **Caso Práctico**: Cambia el código para tener tres columnas (grid-template-columns: 1fr 1fr 1fr;) y observa el cambio.

**Ejercicio Rápido:**

* Pide a los estudiantes que creen un grid con cuatro columnas de igual tamaño. Pregunta: ¿qué pasa si añadimos más elementos?

**4. Controlando Filas con grid-template-rows**

* **CSS**:

css

Copiar código

.grid-container {

display: grid;

grid-template-columns: 1fr 1fr;

grid-template-rows: 100px 100px; /\* Dos filas de 100px de altura \*/

}

* **Explicación**:
  + grid-template-rows define la altura de cada fila en el grid.
  + Puedes usar valores en píxeles, porcentajes, o fr para hacer filas de tamaño variable.
* **Caso Práctico**: Cambia una de las filas a 2fr y observa el cambio.

**5. Espacio Entre Elementos: gap**

* **CSS**:

css

Copiar código

.grid-container {

display: grid;

grid-template-columns: 1fr 1fr;

grid-template-rows: 100px 100px;

gap: 10px; /\* Espacio entre las celdas \*/

}

* **Explicación**:
  + gap agrega espacio entre las filas y columnas. Es útil para crear separación visual sin usar margin.
* **Ejercicio**: Pide a los estudiantes que cambien gap a 20px y observen el cambio. ¿Cómo afecta el diseño?

**6. Posicionamiento de Elementos en la Cuadrícula**

* **HTML con Identificadores**:

html

Copiar código

<div class="grid-container">

<div class="item item1">1</div>

<div class="item item2">2</div>

<div class="item item3">3</div>

<div class="item item4">4</div>

</div>

* **CSS con Áreas de Grid**:

css

Copiar código

.grid-container {

display: grid;

grid-template-columns: 1fr 1fr;

grid-template-rows: 100px 100px;

gap: 10px;

}

/\* Posicionamiento personalizado \*/

.item1 {

grid-column: 1 / 3; /\* Ocupa ambas columnas \*/

}

.item2 {

grid-row: 2 / 3; /\* Se posiciona en la segunda fila \*/

}

* **Explicación**:
  + grid-column y grid-row permiten definir la posición y tamaño de los elementos en la cuadrícula.
* **Caso Práctico**: Cambia la posición de los elementos y observa cómo afecta al diseño.

**7. Áreas de Grid con grid-template-areas**

* **CSS**:

css

Copiar código

.grid-container {

display: grid;

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

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

grid-template-areas:

"header header header"

"sidebar content content"

"footer footer footer";

gap: 10px;

}

.header { grid-area: header; background-color: #f39c12; }

.sidebar { grid-area: sidebar; background-color: #2ecc71; }

.content { grid-area: content; background-color: #3498db; }

.footer { grid-area: footer; background-color: #e74c3c; }

* **HTML**:

html

Copiar código

<div class="grid-container">

<div class="header">Encabezado</div>

<div class="sidebar">Barra Lateral</div>

<div class="content">Contenido Principal</div>

<div class="footer">Pie de Página</div>

</div>

* **Explicación**:
  + grid-template-areas permite nombrar áreas y asignarlas a elementos específicos. Facilita el diseño de layouts complejos.
* **Discusión**: ¿Por qué es útil para el diseño de páginas web?

**8. Unidades Responsivas: minmax() y auto-fit**

* **CSS**:

css

Copiar código

.grid-container {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));

gap: 10px;

}

* **Explicación**:
  + minmax() permite definir tamaños mínimos y máximos para columnas, mientras que auto-fit ajusta automáticamente el número de columnas según el ancho disponible.
* **Caso Práctico**: Cambia el tamaño mínimo de las columnas y observa cómo se ajusta el diseño al tamaño de la ventana.

**9. Ejercicio Final: Crear un Layout Completo**

* **Objetivo**: Usando todo lo aprendido, pide a los estudiantes que creen un layout similar a una página web con encabezado, barra lateral, contenido y pie de página.
* **Directrices**:
  + Usa grid-template-areas para definir el layout.
  + Agrega espacio entre los elementos con gap.
  + Haz el diseño responsivo con minmax() y auto-fit.

**10. Creación de un Mapa para Diseñar el Grid y Planificar el Layout**

CSS Grid, es útil para planificar visualmente antes de codificarlo. Esto puede hacerse dibujando un mapa del grid en papel o utilizando herramientas digitales para entender cómo se colocarán los elementos en el espacio. Este punto cubrirá cómo planificar el layout, usando un mapa visual con grid, padding, y margin.

**Paso 1: Definir el Propósito y el Contenido de Cada Área**

Antes de diseñar un grid, define las áreas que vas a necesitar. Piensa en los elementos principales de una página web, como:

* **Encabezado**: El título o barra de navegación de la página.
* **Barra lateral**: Opcionalmente, un área para enlaces o menús adicionales.
* **Contenido principal**: Donde se ubicará la mayor parte del contenido.
* **Pie de página**: Información de contacto o derechos de autor.

**Paso 2: Boceto de un Mapa Visual del Grid**

Dibuja un boceto en papel o usa herramientas digitales (como Figma o herramientas de diagramación como Lucidchart) para planificar la estructura. Aquí te doy un ejemplo:

markdown

Copiar código

----------------------------------------

| Encabezado |

----------------------------------------

| Barra Lateral | Contenido |

| | Principal |

----------------------------------------

| | |

| Pie de Página |

----------------------------------------

Este layout tiene un encabezado en la parte superior, un contenido principal al lado derecho y una barra lateral a la izquierda. El pie de página ocupa todo el ancho en la parte inferior.

**Paso 3: Convertir el Mapa en CSS Grid**

Basándonos en el boceto, vamos a diseñar el layout usando CSS Grid. Primero, creamos las áreas del grid en CSS:

css

Copiar código

.grid-container {

display: grid;

grid-template-columns: 1fr 3fr; /\* Barra lateral y contenido \*/

grid-template-rows: 100px 1fr 100px; /\* Encabezado, contenido principal, pie de página \*/

gap: 20px; /\* Espacio entre los elementos \*/

grid-template-areas:

"header header"

"sidebar main-content"

"footer footer";

padding: 20px; /\* Espacio interno dentro del grid \*/

}

.header {

grid-area: header;

background-color: #f39c12;

padding: 15px; /\* Espacio dentro del encabezado \*/

}

.sidebar {

grid-area: sidebar;

background-color: #2ecc71;

padding: 15px; /\* Espacio dentro de la barra lateral \*/

}

.main-content {

grid-area: main-content;

background-color: #3498db;

padding: 15px; /\* Espacio dentro del contenido \*/

}

.footer {

grid-area: footer;

background-color: #e74c3c;

padding: 10px; /\* Espacio dentro del pie de página \*/

}

**Paso 4: Ajuste de Padding y Margin**

* **Padding**: Añade espacio interno dentro de cada área, entre el borde del elemento y su contenido. Esto ayuda a que los elementos tengan "respiración" y se vean menos apretados.
  + Por ejemplo, padding: 15px; en el encabezado, la barra lateral y el contenido principal.
* **Margin**: Espacio externo, que separa el grid del resto de la página o contenedor principal.
  + Si quieres que todo el layout esté centrado en la página, puedes usar margin: 0 auto; en el .grid-container para centrarlo horizontalmente.
* **Gap**: Espacio entre las celdas del grid. Esto es diferente del padding o margin porque se aplica solo a la separación entre elementos de la cuadrícula y no afecta el borde externo del contenedor.
  + Aquí estamos usando gap: 20px; para separar cada área del grid entre sí.

**Paso 5: Ajustar el Tamaño y la Responsividad del Grid**

Para hacer que el diseño se vea bien en dispositivos de diferentes tamaños, podemos ajustar el número de columnas y el tamaño de los elementos usando minmax() y auto-fit:

css

Copiar código

.grid-container {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); /\* Se ajusta automáticamente \*/

gap: 20px;

padding: 20px;

}

Con esta configuración:

* Cada columna tendrá un ancho mínimo de 200px.
* Si hay suficiente espacio, se añadirán más columnas automáticamente, ajustándose al tamaño del contenedor.

**Ejemplo Completo con Todo Aplicado**

Combina todos los conceptos para el layout:

html

Copiar código

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Layout Planificado con CSS Grid</title>

<style>

/\* Contenedor principal de grid \*/

.grid-container {

display: grid;

grid-template-columns: 1fr 3fr;

grid-template-rows: 100px 1fr 100px;

gap: 20px;

grid-template-areas:

"header header"

"sidebar main-content"

"footer footer";

padding: 20px;

margin: 0 auto;

max-width: 1200px;

}

/\* Estilos para cada área \*/

.header {

grid-area: header;

background-color: #f39c12;

padding: 15px;

color: white;

text-align: center;

font-size: 1.5rem;

}

.sidebar {

grid-area: sidebar;

background-color: #2ecc71;

padding: 15px;

color: white;

font-size: 1.2rem;

text-align: center;

}

.main-content {

grid-area: main-content;

background-color: #3498db;

padding: 15px;

color: white;

font-size: 1.2rem;

}

.footer {

grid-area: footer;

background-color: #e74c3c;

padding: 10px;

color: white;

text-align: center;

font-size: 1rem;

}

</style>

</head>

<body>

<div class="grid-container">

<div class="header">Encabezado</div>

<div class="sidebar">Barra Lateral</div>

<div class="main-content">Contenido Principal</div>

<div class="footer">Pie de Página</div>

</div>

</body>

</html>

**Explicación de los Resultados:**

1. **Encabezado y Pie de Página**: Ocupan el ancho completo gracias a grid-template-areas.
2. **Barra Lateral y Contenido Principal**: Están organizados en dos columnas con un gap que los separa.
3. **Padding**: Da espacio interno a cada área.
4. **Margin en Contenedor**: Centra el contenedor en la página y limita el ancho máximo.
5. **Responsividad**: Puedes hacer que las columnas se ajusten según el espacio disponible usando auto-fit y minmax().