Diplomatura en

Programación FullStack

CSS
Layouts con CSS: Flexbox y Grid

Flexbox y Grid

Flexbox y Grid: Dos herramientas poderosas para el diseño web modernoFlexbox y Grid son dos módulos de CSS que revolucionaron la forma de crear diseños web. Ambos permiten crear layouts flexibles y responsivos con gran facilidad, pero cada uno tiene sus propias fortalezas y debilidades.

Flexbox se enfoca en la organización unidimensional de elementos, ya sea en filas o columnas. Es ideal para:

- Alinear elementos: Distribuye el espacio de manera uniforme entre los elementos o los coloca en los extremos del contenedor.
- Crear diseños responsivos: Permite que los elementos cambien de tamaño y posición según el tamaño de la pantalla.
- Mostrar elementos en una sola línea: Perfecto para barras de navegación, menús y pies de página.

Grid, por otro lado, se centra en la creación de layouts bidimensionales, como cuadrículas. Es ideal para:

- Diseñar estructuras complejas: Permite crear layouts con múltiples filas y columnas, con diferentes tamaños y espacios entre ellas.
- Posicionar elementos con precisión: Ofrece un control granular sobre la ubicación de cada elemento dentro de la cuadrícula.
- Crear diseños adaptables: Se adapta perfectamente a diferentes tamaños de pantalla, manteniendo la estructura del diseño.

¿Cuándo usar Flexbox o Grid?

En general, Flexbox es una buena opción para diseños simples con una sola dirección, mientras que Grid es más adecuado para layouts más complejos y multidimensionales.

Aquí hay una tabla que resume las principales diferencias:

Característica	Flexbox	Grid
Dimensionalidad	Unidimensional (filas o columnas)	Bidimensional (filas y columnas)
Alineación	Fácil y flexible	Gran control y precisión
Responsividad	Buena	Excelente
Complejidad	Simple	Complejo
Casos de uso	Barras de navegación, menús, pies de página	Layouts complejos, cuadrículas, dashboards

En resumen:

Flexbox: Ideal para diseños simples, alineación de elementos y layouts responsivos unidimensionales.

Grid: Perfecto para layouts complejos, estructuras multidimensionales y diseños adaptables a cualquier tamaño de pantalla.

Flexbox

Flexbox (o Flexible Box Layout) es un módulo de diseño de CSS diseñado para ayudar a los desarrolladores web a crear y organizar elementos de una manera más flexible y eficiente dentro de un contenedor.

1. Ejes principales y secundarios: Flexbox permite organizar elementos en un contenedor flex en dos ejes:

Eje principal: Es el eje definido por la propiedad flex-direction, que puede ser row (horizontal de izquierda a derecha), row-reverse (horizontal de derecha a izquierda), column (vertical de arriba a abajo) o column-reverse (vertical de abajo a arriba).

- row: Organiza los elementos en una fila de izquierda a derecha.
- row-reverse: Organiza los elementos en una fila de derecha a izquierda.
- column: Organiza los elementos en una columna de arriba a abajo.
- column-reverse: Organiza los elementos en una columna de abajo a arriba.

Eje secundario: Es el eje perpendicular al principal, también conocido como eje cruzado. La alineación a lo largo de este eje se controla mediante la propiedad 3 align-items en el contenedor flex.

2. Propiedades de contenedor: El contenedor flex tiene varias propiedades que controlan la disposición de sus hijos:

justify-content: Controla la alineación de los elementos a lo largo del eje principal. Las opciones incluyen:

- flex-start: Los elementos se alinean al principio del eje principal.
- flex-end: Los elementos se alinean al final del eje principal.
- center: Los elementos se centran a lo largo del eje principal.
- space-between: Los elementos se distribuyen uniformemente a lo largo del eje principal, dejando espacios iguales entre ellos.
- space-around: Los elementos se distribuyen con un espacio igual alrededor de cada uno.
- space-evenly: Los elementos se distribuyen de manera equitativa, dejando espacios iguales entre cada uno y a los extremos.

align-items: Controla la alineación de los elementos a lo largo del eje secundario. Las opciones incluyen:

- flex-start: Los elementos se alinean al principio del eje secundario.
- flex-end: Los elementos se alinean al final del eje secundario.
- center: Los elementos se centran a lo largo del eje secundario.
- baseline: Los elementos se alinean de modo que sus líneas base estén alineadas.
- stretch: Los elementos se estiran para llenar el espacio a lo largo del eje secundario.

flex-wrap: Define si los elementos deben ocupar más de una línea o no:

- nowrap: Todos los elementos permanecen en una sola línea, sin importar cuántos haya.
- wrap: Los elementos se distribuyen en varias líneas si es necesario.
- wrap-reverse: Los elementos se distribuyen en varias líneas, pero en orden inverso.

align-content: Controla cómo se distribuye el espacio entre las líneas de elementos, en caso de que haya más de una:

- flex-start: Las líneas se alinean al principio del eje secundario.
- flex-end: Las líneas se alinean al final del eje secundario.
- center: Las líneas se centran a lo largo del eje secundario.
- space-between: Las líneas se distribuyen uniformemente, dejando espacios iguales entre ellas.
- space-around: Las líneas se distribuyen con un espacio igual alrededor de cada una.
- space-evenly: Las líneas se distribuyen equitativamente, dejando espacios iguales entre cada una y en los extremos.

3. Propiedades de los elementos: Los elementos hijos del contenedor flex también tienen propiedades específicas:

- flex-grow: Define cuánto puede crecer un elemento en relación con sus hermanos. Por ejemplo, un valor de 2 significa que el elemento puede crecer el doble de lo que crecería un elemento con flex-grow: 1.

- flex-shrink: Define cuánto puede encogerse un elemento en relación con sus hermanos. Por ejemplo, un valor de 1 significa que el elemento se encogerá a la misma velocidad que sus hermanos cuando no haya suficiente espacio.

- flex-basis: Define el tamaño inicial del elemento antes de que se aplique flex-grow o flex-shrink.

- align-self: Permite anular la alineación establecida por align-items en el contenedor, solo para ese elemento. Las opciones son las mismas que align-items: flex-start, flex-end, center, baseline, stretch.

¿Qué es Grid CSS?

Grid CSS es un sistema de layout bidimensional que divide el espacio de renderizado en filas y columnas, creando una cuadrícula. Permite a los desarrolladores controlar el posicionamiento y el tamaño de los elementos hijos de un contenedor de manera muy precisa.

Terminología clave en Grid CSS:

- Grid Container: Es el elemento padre que utiliza display: grid o display: inline-grid. Dentro de este contenedor se definen las filas y columnas.
- Grid Item: Son los hijos directos del grid container.Grid Line: Son las líneas divisorias horizontales y verticales que definen las filas y columnas.
- Grid Cell: Es el espacio entre dos líneas adyacentes.
- Grid Track: Es la dimensión de una fila o columna entre dos líneas adyacentes.
- Grid Area: Es un espacio rectangular en la cuadrícula, formado por varias celdas contiguas.

En resumen, Grid CSS es una poderosa herramienta de layout que facilita el diseño de interfaces complejas de manera más intuitiva y con mayor control que otros métodos anteriores. Es altamente recomendable aprovechar su flexibilidad y capacidades para crear diseños responsivos y adaptables.

Propiedades del Grid Container:

display: grid | inline-grid
- grid crea un bloque de nivel de cuadrícula.
- inline-grid crea un elemento en línea de nivel de cuadrícula.

grid-template-columns / grid-template-rows
- Definen el número de columnas/filas y sus tamaños.
- Valores posibles: auto, un tamaño fijo (px, em, etc.), fr (fracción del espacio disponible), repeat(), minmax(), etc.

gap / column-gap / row-gap
- Establecen el espacio entre filas y columnas.

grid-template-areas

- Define un patrón de áreas en la cuadrícula utilizando nombres.
- Los nombres se referencian en grid-area de los elementos hijos.

grid-auto-columns / grid-auto-rows

- Especifican el tamaño de las filas/columnas implícitas (no definidas explícitamente).

grid-auto-flow

- Controla cómo se colocan automáticamente los elementos hijos en el grid.

justify-content / align-content

- Alinean las filas/columnas dentro del contenedorGrid.

justify-items / align-ítems

- Alinean los elementos hijos dentro de sus celdas.

Además, hay propiedades abreviadas como grid-template y grid que permiten establecer varias propiedades a la vez.

Es importante mencionar que Grid CSS también ofrece funciones útiles como repeat(), minmax(), fit-content() y palabras clave como auto-fill y auto-fit que brindan un control aún más preciso sobre el diseño de la cuadrícula.

Ejemplo de Grid Container

```css
.container {
    display: grid;
    grid-template-columns: 1fr 1fr 2fr;
    grid-template-rows: 50px auto 100px 100px;
    grid-template-areas:
        "nav nav nav"
        "header header header"
        "aside footer footer"
        "aside footer footer";
    gap: 10px; /* Espacio entre las celdas */
}
```

Propiedades de los Grid Items:

grid-column-start / grid-column-end / grid-row-start / grid-row-end

- Especifican las líneas de inicio y fin para el elemento en la cuadrícula.
- Se pueden usar números de línea o la palabra clave span.

grid-column / grid-row

- Shortcuts para grid-column-start/end y grid-row-start/end.

grid-área

- Asigna un nombre al elemento, para referenciarlo en grid-template-areas.

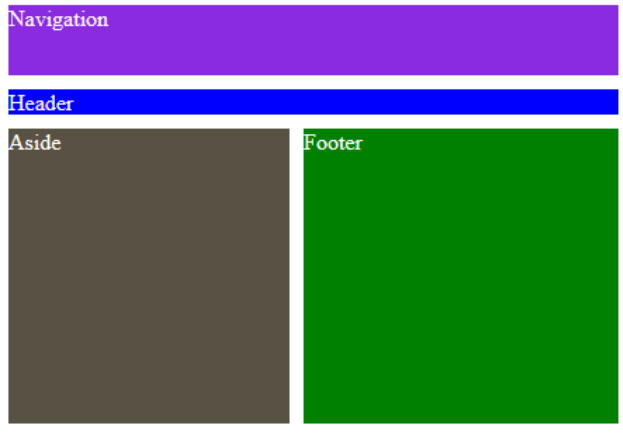
justify-self / align-self

- Alinean el elemento dentro de su celda.

Además, hay propiedades abreviadas como grid-template y grid que permiten establecer varias propiedades a la vez.

Es importante mencionar que Grid CSS también ofrece funciones útiles como repeat(), minmax(), fit-content() y palabras clave como auto-fill y auto-fit que brindan un control aún más preciso sobre el diseño de la cuadrícula.

Ejemplo de Grid Item




```html
<div class="container">
   <header class="item1">Header</header>
   <nav class="item2">Navigation</nav>
   <aside class="item3">Aside</aside>
   <footer class="item4">Footer</footer>
</div>
```

```css
.item1 {
    grid-area: header;
    background-color: blue;
}

.item2 {
    grid-area: nav;
    background-color: blueviolet;
}

.item3 {
    grid-area: aside;
    background-color: rgb(88, 82, 69);
}

.item4 {
    grid-area: footer;
    background-color: green;
}
```

Funciones en CSS Grid

Una visión general de cómo utilizar repeat(), minmax(), fit-content(), auto-fill y auto-fit en CSS Grid para crear layouts flexibles y responsivos.

repeat()

La función repeat() te permite repetir un patrón de definición de columnas o filas varias veces, lo cual es útil para crear layouts dinámicos.

```
.container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    /* Crea tres columnas de igual tamaño */
}
```

minmax()

La función minmax() es útil para definir un tamaño mínimo y máximo para tus columnas o filas, permitiendo que se ajusten dinámicamente dentro de esos límites.

```
.container {
    display: grid;
    grid-template-columns: minmax(100px, 1fr);
    /* Columnas con un mínimo de 100px y un máximo de 1fr */
}
```

fit-content()

La función fit-content() se utiliza para ajustar el tamaño de una columna o fila al contenido de sus celdas, hasta un máximo especificado.

```
.container {
    display: grid;
    grid-template-columns: fit-content(300px);
    /* Columnas que se ajustan al contenido hasta un máximo de 300px */
}
```

auto-fill vs auto-fit

Las palabras clave auto-fill y auto-fit se utilizan con repeat() para controlar cómo se llenan las columnas o filas en un contenedor.

- auto-fill llena la fila o columna con tantas pistas como sea posible, incluso si quedan vacías.
- auto-fit colapsa las pistas vacías y expande las que tienen contenido para ocupar el espacio disponible.

```
.container-fill {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
/* Llena el contenedor con columnas de al menos 100px */
}

.container-fit {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
/* Ajusta las columnas al contenido y expande para llenar el espacio */
}
```

El HTML

```html
<div class="container-fill">
    <!-- Elementos que llenarán el contenedor usando auto-fill -->
    <div>Item 1</div>
    <div>Item 2</div>
    <div>Item 3</div>
</div>

<div class="container-fit">
    <!-- Elementos que ajustarán el tamaño del contenedor usando auto-fit -->
    <div>Item 1</div>
    <div>Item 2</div>
    <div>Item 3</div>
</div>
```

Ejercicio práctico:

Barra de navegación superior (Flexbox):

1. Tendrá tres enlaces: Inicio, Acerca de, y Contacto.
2. Los enlaces estarán distribuidos uniformemente y centrados verticalmente.

Cuerpo principal (CSS Grid):

1. Tendrá tres columnas de igual tamaño.
2. Contendrá tres tarjetas o bloques de contenido, uno en cada columna.

Código HTML para el ejercicio:

```html
<body>
    <nav class="navbar">
        <a href="#">Inicio</a>
        <a href="#">Acerca de</a>
        <a href="#">Contacto</a>
    </nav>

    <section class="content">
        <div class="card1">Contenido 1</div>
        <div class="card2">Contenido 2</div>
        <div class="card3">Contenido 3</div>
    </section>
</body>
```