25 CSS: Propiedad `display`

La propiedad `display` es una de las propiedades CSS más fundamentales e importantes. Determina cómo se genera la caja (o cajas) de un elemento y cómo participa en el flujo del layout de la página. Con `display`, podemos cambiar el comportamiento intrínseco de un elemento (por ejemplo, hacer que un `<span>` actúe como un bloque, o un `<div>` como parte de una línea) y, crucialmente, podemos activar modos de layout avanzados como Flexbox y Grid.

Valores Fundamentales (Recap):

Estos valores controlan el tipo de caja *externa* (cómo interactúa con sus hermanos) y el formato *interno* básico (cómo se comporta su contenido).

1. `block`:
   - Hace que el elemento genere una caja de nivel de bloque.
   - Empieza en una nueva línea y ocupa el ancho disponible si `width` es `auto`.
   - Respeta `width`, `height`, `margin` vertical/horizontal, y `padding` vertical/horizontal.
   - *Uso:* Convertir un elemento en línea (como `<a>` o `<span>`) en un bloque estructural.
2. `inline`:
   - Hace que el elemento genere una(s) caja(s) de nivel de línea.
   - Fluye horizontalmente con el texto y otros elementos en línea.
   - **Ignora** `width` y `height`.
   - **Ignora** `margin-top`, `margin-bottom`. Afecta mínimamente el layout vertical con `padding-top` y `padding-bottom`.
   - Respeta `margin-left`, `margin-right`, `padding-left`, `padding-right`.
   - *Uso:* Es el valor por defecto de elementos como `<span>`, `<a>`, `<em>`. Raramente se usa para *cambiar* un elemento a `inline` si no lo era, pero es importante entenderlo.
3. `inline-block`:
   - Genera una caja de nivel de bloque, pero el elemento en sí fluye horizontalmente como un elemento en línea (no empieza en una nueva línea).
   - **Respeta** `width`, `height`, y **todos** los `margin` y `padding` (verticales y horizontales).
   - El ancho por defecto se ajusta al contenido (no ocupa todo el ancho disponible).
   - *Uso:* Ideal para elementos que deben estar en línea pero necesitan dimensiones y espaciado vertical controlables (botones, iconos, avatares pequeños, etc.).

Valor para Ocultar Elementos:

1. `none`:
   - **Elimina completamente** el elemento y sus descendientes del flujo del documento y del árbol de renderizado.
   - El elemento **no ocupa ningún espacio** en la página. Es como si no existiera en el HTML para propósitos de layout.

- Sus descendientes también desaparecen.
- **Diferencia clave con `visibility: hidden;`:** `visibility: hidden;` oculta el elemento visualmente, pero **sigue ocupando su espacio** en el layout. `display: none;` lo elimina por completo.
- *Uso:* Ocultar/mostrar elementos dinámicamente (con JavaScript o interacciones CSS), eliminar contenido no relevante para ciertos dispositivos (aunque hay mejores técnicas para responsividad), etc. ¡Cuidado con la accesibilidad al usar `display: none;`! El contenido desaparece también para lectores de pantalla.

Valores para Modos de Layout Avanzados:

Estos valores transforman el elemento en un **contenedor** de un tipo de layout específico, afectando principalmente a cómo se disponen sus **elementos hijos directos**.

1. `flex`:
   - Convierte el elemento en un **contenedor Flexbox de nivel de bloque**.
   - Sus hijos directos se convierten en **ítems flex** y se disponen según las reglas de Flexbox (por defecto, en una fila horizontal).
   - El contenedor en sí se comporta como un elemento de bloque (empieza en nueva línea, ocupa ancho disponible).
   - *Uso:* Principalmente para layouts unidimensionales (filas o columnas). Excelente para alinear ítems, distribuirlos espacialmente y manejar contenido de tamaño flexible. (Flexbox se tratará en detalle en una unidad posterior).
2. `inline-flex`:
   - Convierte el elemento en un **contenedor Flexbox de nivel de línea**.
   - Sus hijos directos se convierten en ítems flex, igual que con `display: flex`.
   - El contenedor en sí fluye horizontalmente como un elemento en línea (no empieza en nueva línea).
   - *Uso:* Cuando necesitas las capacidades de Flexbox para los hijos, pero quieres que el contenedor entero se integre en una línea de texto o fluya con otros elementos inline/inline-block.
3. `grid`:
   - Convierte el elemento en un **contenedor Grid de nivel de bloque**.
   - Sus hijos directos se convierten en **ítems grid** y pueden ser colocados en una cuadrícula bidimensional (filas y columnas) definida en el contenedor.
   - El contenedor en sí se comporta como un elemento de bloque.
   - *Uso:* Ideal para layouts bidimensionales complejos (estructuras de página, galerías complejas, etc.). (Grid Layout se tratará en detalle en una unidad posterior).
4. `inline-grid`:
   - Convierte el elemento en un **contenedor Grid de nivel de línea**.
   - Sus hijos directos se convierten en ítems grid, igual que con `display: grid`.
   - El contenedor en sí fluye horizontalmente como un elemento en línea.

- *Uso:* Similar a `inline-flex`, para usar Grid Layout en un contenedor que se integra en el flujo en línea.

Otros Valores Menos Comunes:

1. `list-item`:
   - Hace que el elemento se comporte como un `<li>` de una lista. Genera una caja de bloque principal y una caja de marcador (viñeta/número) de nivel de línea.
   - *Uso:* Para crear listas con elementos que no son `<li>` semánticamente, o para experimentar.
2. `table`, `table-row`, `table-cell`, `table-caption`, `table-header-group`, `table-footer-group`, `table-row-group`, `table-column`, `table-column-group`:
   - Permiten a elementos arbitrarios comportarse como las diferentes partes de una tabla HTML.
   - *Uso:* Antes de Flexbox y Grid, se usaban a veces para ciertos tipos de layouts (especialmente alineación vertical). Hoy en día, su uso principal debería ser para datos tabulares o en casos muy específicos donde su lógica de layout sea necesaria. Evita usarlo para layouts generales de página.
3. `contents`:
   - Un valor especial y potencialmente complicado. Hace que el elemento **no genere su propia caja**, pero sus **elementos hijos sí participan normalmente en el layout** como si fueran hijos directos del padre del elemento con `display: contents`.
   - *Uso:* Puede ser útil en escenarios muy específicos con Flexbox o Grid donde quieres que los hijos de un contenedor intermedio actúen como si estuvieran directamente en el contenedor principal, sin que la caja intermedia interfiera.
   - **¡Advertencia de Accesibilidad!** Históricamente (y aún en algunos casos), usar `display: contents` puede eliminar la semántica del elemento del árbol de accesibilidad, haciéndolo invisible para tecnologías asistivas. Úsalo con extrema precaución y prueba la accesibilidad a fondo.

Display Exterior e Interior:

Conceptualmente, `display` controla dos aspectos:

- **Display Exterior (Outer display type):** Cómo la caja del elemento interactúa con sus elementos hermanos (`block`, `inline`).
- **Display Interior (Inner display type):** Cómo se disponen los hijos directos del elemento (`flow` (normal), `flex`, `grid`, `table`, etc.).

Muchos valores `display` establecen ambos. Por ejemplo:

- `display: block;` → Exterior: `block`, Interior: `flow` (layout normal de bloque/línea)
- `display: inline;` → Exterior: `inline`, Interior: `flow`
- `display: flex;` → Exterior: `block`, Interior: `flex`
- `display: inline-flex;` → Exterior: `inline`, Interior: `flex`
- `display: grid;` → Exterior: `block`, Interior: `grid`
- `display: inline-block;` → Exterior: `inline`, Interior: `flow`
