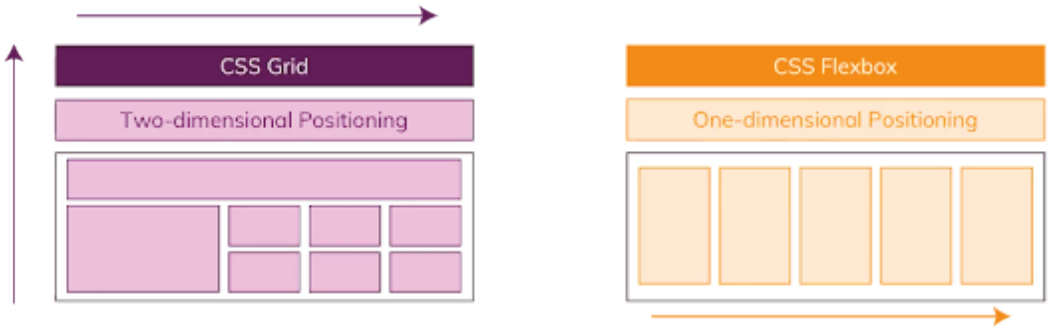
Grids

Qué es?

- Es una herramienta que nos permite maquetar/ estructurar el sitio. Funciona a dos dimensiones (filas **y** columnas - largo y ancho), a diferencia de flexbox que funciona en una sola dimensión (filas **o** columnas).



- Grids y flexbox pueden usarse juntos, conviven bien. Lo que no se puede obviamente es aplicar ambos display a una misma clase/etiqueta/id, es uno o el otro en ese sentido. Pero supongamos que tenemos un **div padre** con varios **divs hijos**, podemos aplicar grids al padre y flex a los hijos para organizar los elementos que estos últimos tengan dentro.

Algunos usos prácticos:

- Crear una galería de imágenes
- Reemplazar las tablas
- Construir el cuerpo donde estará la información en una sección
- Construir toda una sección de un sitio (header, nav, sections y footer) - **Avanzado**

Como se usa:

Vamos a dividir está sección en dos partes: el uso de grids normal (por llamarlo de alguna manera) y el uso de “grids-template-areas”.

El uso de grids más simple: vamos a ver como usar las filas y columnas de grids para armar una estructura y al final, a modo de ejemplo, vamos a ver como crear una galería de fotos simple que tenga también un banner, con grids.

Para esto vamos a usar las siguientes propiedades CSS relacionadas a grids:
- Display: grid;
- Grid-template-columns
- Grid-template-rows

La forma práctica de aplicar grids es primero trabajar la base del html y luego pasar al css. Primero creamos la **caja contenedora** (puede ser un div o una etiqueta semántica según corresponda) y dentro de ella creamos los hijos. **Los hijos** pueden ser elementos sueltos o más divs que tengan dentro elementos. Una vez que tengamos esto, definimos una clase para nuestra caja container (en el ejemplo será "container_grids") y le ponemos una clase a los hijos (en el ejemplo será "grid_item").

Pasamos al Css, donde lo primero que hacemos es definir el alto y ancho (width y height) de la caja contenedora. A esta caja le vamos a aplicar display: grid. Y vamos a trabajar las columnas y filas de este contenedor.

La idea es definir, dentro del espacio del caja container, cuantas filas y columnas va a haber y que espacio de esta caja va a ocupar cada una. Entonces, la idea es hacer dos filas de imagenes (dos imagenes en este caso) y un banner a la derecha (se recomienda mucho dibujar a mano primero la idea, pensar la distribución de los elementos dentro del container). En el código, vamos a ir a la clase "container_grids" y se refleja de la siguiente manera:

- **grid-template-columns:** 500px 500px auto; —> Esto significa que vamos a tener 3 columnas, la primera va a ocupar 500px, la segunda ocupara 500px y la última va ocupar el espacio restante de la caja. (En este caso no se exactamente el ancho en px de la caja porque use % y no chequee cuánto sería, pero si trabajamos en px sería más fácil precisar el ancho de cada uno). La última fila representa el ancho que va a ocupar el banner.

- **grid-template-rows:** 300px 300px; —> Esto significa que cada fila va a ocupar 300px del alto total de la caja. Esto nos interesa para el espacio de las fotos pero no para el banner, ya que este va a ocupar todo el alto del contenedor (600px que es el alto que le dimos al contenedor).

- Las otras propiedades de esta clase (container_grids) ya las conocemos asique no voy a entrar en detalles en eso. Cualquier duda siempre se puede comentar la propiedad y ver que cambia 😅.

Ahora vamos a la siguiente clase que nos importa: **grid_item**. Esta clase la aplicamos a los hijos del container. En esta clase podríamos definir el tamaño de cada div pero no es necesario ya que definimos el espacio que tienen que ocupar en la clase anterior. Asique lo único diferente que hicimos fue usar justify y align-**self**. Lo que hace, como hacíamos en flexbox, es centrar el contenido en el medio del div (hay diferentes variantes como en flex, los invito a probarlas si quieren en el mismo archivo a ver como va cambiando).

Si fuese solo una galería de imágenes donde todos los divs son iguales y ocupan el mismo espacio, podríamos hacer esto desde el padre con justify y align-**items**, pero al tener el banner van a ver que si aplicamos esta propiedad al padre, el banner queda corrido. Entonces usamos justify y align-self en los hijos, a excepción del banner que va a tener una clase a parte.

Con justify y align-**content** aplicado al padre lo que se hace es mover TODO el contenido de la caja (el conjunto de elemento) y acomodarlo de una manera u otra en la caja.

Por último, el banner va a tener su propia clase ("banner") con las siguiente propiedades: una altura determinada (mismo alta de la caja para ocupe todo el espacio), fondo de color y display: flex con sus propiedades correspondientes para acomodar la info dentro del banner. Después van a ver en el css que di tamaño a las imágenes los h, esto es importantísimo para que el contenido entre bien.

Grids por areas

Se puede también usar grid-template-area, una forma sencilla de crear un espacio más personalizado, usado generalmente para estructurar todo una sección de la forma que querramos. El problema es que, sin comprender lo anterior, hacer esto va a ser mil veces más complejo. Suelen cometerse errores en los tamaños de las filas y columnas, y la distribución. Por eso mi recomendación es primero practicar armar cosas más bien chicas con grids y una vez que le agarran la mano, pasar a esto. Igualmente creo hay formas más sencillas de armar cada sección.

La idea sería la siguiente: tener un grid contenedor que contenga dentro todos los elementos que formaran parte de la sección (no se usa solo para esto, pero es la forma más clara de explicarlo). Hasta ahora es prácticamente igual a lo anterior. En el div padre vamos a usar la propiedad **grid-template-areas**, ahí especificamos el orden de cada área. En el padre vamos a especificar también grid-template-rows y columns.

En los hijos vamos a usar la propiedad grid-area para nombrar cada área y que se corresponda con las áreas que usamos en grid-template-area. Recomiendo ver el ejemplo de la presentación de la clase de grids (diapositivas 34, 35 y 36) para terminar de comprender la idea.

Otras propiedades:

- **grid-column-gap:** espacio entre columnas.
- **grid-row-gap:** espacio entre filas.