**CSS GRID**

NOTA:

<https://platzi.com/blog/flexbox-vs-css-grid-cual-es-la-diferencia/>

Mi resumen:  
Flexbox para componentes  
CSS Grid Layout para estructura

3- Css Grid Layout. (Mis apuntes de la clase).

En css podemos interpretar cualquier cosa como una caja y así darle estilos.

-Grid Container: Nos permite poner las especificaciones ahí, facilitándonoslo de esa manera el trabajo.

-Grid Item: Es donde se guarda el contenido dinámico que se haga dentro de la pagina. Este es hijo directo del Grid Container.

-Grid line: son las lineas divisoras, tanto horizontales como verticales.

-Grid Track: Son las filas y columnas.

-Grid Cell: Es el espacio donde dos lineas horizontales y dos lineas verticales se juntan.

-Grid Area: Es la suma de n grid cells.

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

## **4- CLASE 1: Definiendo columnas**

Se debe poner el display de grid al contenedor

.container{

display: grid;

}

Para definir columnas debemos ponerle la propiedad de grid-template-columns al contenedor, en esta debemos definir el tamaño de cada una de las columnas, por ejemplo: si en la propiedad ponemos los siguientes valores

.container{

grid-template-columns: 200px200px200px;

}

estamos estableciendo tres columnas, cada una de 200px.

También podemos poner medidas dinámicas, tales como porcentajes, se definen exactamente igual que como en el ejemplo anterior:

.container{

grid-template-columns: 25%50%25%;

}

También podemos combinar las medidas relativas con las medidas absolutas:

.container{

grid-template-columns: 25%200px25%;

}

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

5- **CLASE 2: Definiendo filas**

Al igual que como definimos las columnas definimos las filas, lo único que cambia es el final de la propiedad, es decir que esta vez ya no será grid-tempate-columns sino que será grid-tempate-rows.

.container{

grid-template-columns: 25%200px25%;

grid-template-rows: 300px300px;

}

\_ Si tenemos el mismo código de la clase pasada aparecen 2 filas más debido al numero de items, la altura de estas es implícita, debido a que no definimos el valor. Nosotros antes definimos explicitamente el tamaño de 3 columnas y 2 filas\_

Podemos definir los tamaños de las filas y las columnas con la misma propiedad, esta propiedad es grid-template, debemos definir primero los tamaños de las filas, luego separandolos con un slash debeos definir los tamaños de las columnas

**grid**-template: 300px 150px / 25%200px 25%;

**Apuntes:**

De la misma forma que al definir columnas, cuando vamos a definir una fila, le añadimos al elemento padre la propiedad grid-template-rows y le agregamos el numero de valores por medida de las filas que vayamos a maquetar explícitamente (grid explicito).

Para realizar un trabajo automático a la hora de designar filas y columnas, usamos la propiedad grid-template, y separamos la configuración de las filas y columnas explicitas con un /.

**Grid explícito:** Es todo aquello a lo que le designamos una regla al tener el display como grid.

**Grid implícito:** Es aquello que se encuentra designado por defecto y no hemos tocado con ninguna regla cuando se tiene el display de grid.

<https://codepen.io/JuanDiegoS/pen/wbLRaw>

IMPORTANTE: Ahora que conocemos grid, ¿cuando es necesario que usemos flex?

Si se necesita un diseño que sus elementos se alineen en una sola dimensión (fila o columna) utiliza **flexbox**. Si por el contrario se necesita un diseño que tus elementos se alineen en 2 dimensiones (filas y columnas) utiliza **Grid** Es decir:

Si tenemos 5 elementos cards y utilizamos **flex** vemos que en la siguiente fila los dos elementos restantes 4 y 5 que no se acomodaron en el viewport de la pantalla pasan a ocupar todo el ancho de la segunda fila:

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

6- **Grids identados y tipos de displays**

Para centrarnos en un elemento especifico, aun cuando todos tengan la misma clase, debemos usar la pseudoclase :nth-of-type(#delElemento).

Para asignarle propiedades de Grid a un subelemento usamos la misma declaración de display: grid y luego copiamos y pegamos todas las reglas que teníamos asignadas del padre (en caso de que queremos que el subelemento herede las reglas).

\*\*Notas: \*\*

* Para que un subelemento herede todas las configuraciones Grid del elemento padre usaríamos para resumir la declaración de display: subgrid, pero aun no existe en los navegadores al día de hoy.
* Para crear un grid en línea usamos la propiedad display: y le asignamos el valor de inline-grid. Esta propiedad tampoco se encuentra disponible, pero lo estará.

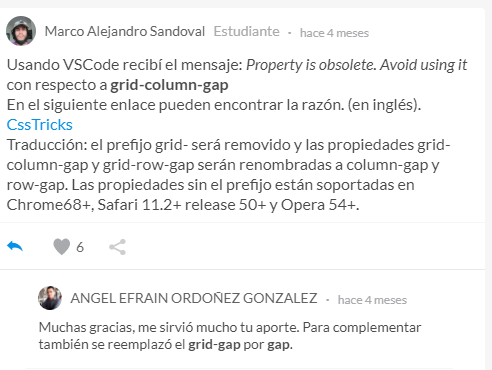
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

7- Espaciado entre filas y columnas

**Anotaciones:**

Para lograr un espaciado vertical de la manera tradicional, usamos la propiedad margin-right y le añadimos como valor la medida deseada. De esta misma forma para un espaciado horizontal usamos la propiedad margin-bottom y añadimos el valor deseado.

La mejor forma de hacerlo para un elemento Grid es con la propiedad grid-gap, con lo cual consecuentemente le añadimos el valor deseado de la medida de espaciado, esta propiedad funciona para el espaciado vertical y horizontal. Individualmente usamos grid-column-gap: (espaciado en columnas) y grid-row-gap: (espaciado en filas).



::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

**8- Repetidores, unidades de medida y funciones**

Resumen

Existen nuevas propiedades que se puede aplicar a los selectores que disponen de un display tipo grid y estos son:

\*\*repeat(veces-a-repetir,valor-de-media): \*\*  
Es una función de css que permite repetir un valor propieda según una cantidad de veces. El primer valor es la veces que se repetirá y el segundo parámetro es el valor medida.

**minmax(valor-mínimo, valor-máximo):**  
Función que permite recibir dos parámetros el primero es el valor mínimo que se puede tomar y el segundo hace referencias al valor máximo que puede tomar.

\*\*1fr: \*\*  
Es la unidad que nos permite distribuir el contenido grid de una manera más homogénea posible.1fr es equivalente a un 33.3333%.

Esta tres propiedades se pueden mezclar para manipular de una forma correcta la distribución del grid.

NOTA: La diferencia entre auto y fr. Auto organiza el contenido externo según el espacio que necesite el contenido internamente de cada componente html es decir: Sí un componente html internamente tiene o necesita mayor espacio para representarse auto lo que hace es que externamente le da mayor espacio en cambio fr realiza una distribución más o homogénea

código html

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

9-

Respuesta a:

# Definiendo áreas de contenido

**¿Cómo organizar los elementos?**  
A cada sección del html le asignamos un Grid-area  
Y **la magia la veremos** en el contenedor (container) **congrid-template-areas**  
"header header"  
“left contenido”  
"footer footer"  
Una enorme ventaja es que el orden de los elementos en el HTML es independiente.  
Hubiera estado genial si, hubiera comenzado el diseño con mobile-first y luego desktop.

Más notas:

Un sitio web está compuesto por filas columnas sí utilizamos las especificaciones grid en nuestras reglas css en tal sentido el footer y el header ocupan tantas columnas tengamos en nuestra sección de contenido.  
**vh o vw** : Estas medidas tienen múltiples usos, una de ella es si queremos que un contenido ocupe ocupe todo el ancho y alto de nuestro pantalla o sirve sí queremos que nuestro texto varíe de acuerdo a las medidas del dispositivo. Las medidas vh y vw son medidas relativas de acuerdo al viewport.  
es decir:  
a) 1vh = 1% de la altura del viewport.  
b) 100vh = altura del viewport.  
c) 1vw = 1% del ancho del viewport.  
d) 100vw = ancho del viewport.  
**grid-template-areas: "nombre-area-1"“nombre-area-2” "nombre-area-n"**  
La propiedad CSS grid-template-areas especifica nombres para cada una de las grid areas.  
**grid-area**:  
indica la posición en el grid que contendré el elemento hijo. Esta porpiedad va en el selector hijo

**código html**

**Apuntes.**

Pueden definirse didácticamente las áreas de contenido con la propiedad grid-template-areas: y como valor podemos poner entre comillas las áreas que van a ocupar los ítems, si una fila ocupa mas de un elemento, se pueden separar los nombres de los elementos con espacios. Posteriormente se le debe añadir a los elementos hijos una segunda clase, que tendrá el nombre del área designada. Finalmente se deben configurar las respectivas clases en css.