**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.

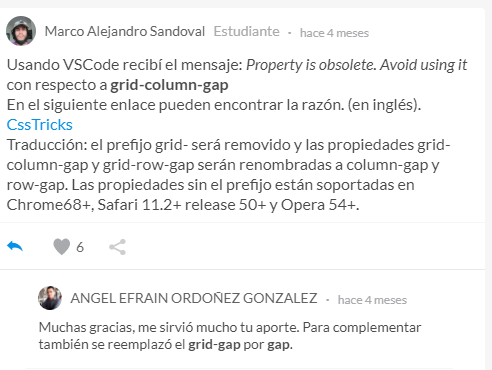
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

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) **con grid-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.

 grid-template-areas: "header header" "izquierda contenido" "footer footer";

.header {

            grid-area: header;

        }

        .left {

            grid-area: izquierda;

        }

        .contenido {

            grid-area: contenido;

        }

        .footer {

            grid-area: footer;

        }

<body>

    <section class="container">

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

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

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

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

    </section>

</body>

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

**grid-column-start:**  
Propiedad CSS especifica la posición inicial (grid-line) de un elemento hijo dentro del grid. Es decir esta propiedad no toma como referencia los espacios definido (grid-cell) para saber dónde empezar.  
Ejemplo:  
grid-column-start: 1

**grid-column-end:**  
Propiedad CSS especifica la posición final (grid-line) de un elemento hijo dentro del [grid.Es](http://grid.es/" \t "_blank) decir esta propiedad no toma como referencia los espacios definido (grid-cell) para saber dónde empezar.  
Ejemplo:  
grid-column-end: 4

**span:**  
Valor-propiedad que indica cuánto espacio va tomar nuestro columna. Es decir esta propiedad no cuenta los grid-line. Esta propiedad cuenta los espacios es decir los grid-cell.  
Ejemplo:  
grid-column: 1/ span 2

**grid-column: grid-line-incio / grid-line-final**

Es igual a usar grid-column-start y grid-column-end. Este elemento indica la posición inicial (grid-line) y la posición final (grid-line) que ocupará el elemento hijo(grid-cell).  
Ejemplo:

1. grid-column:1/3
2. grid-column: 1/-1 La declaración anterior permite a una columna adaptarse dinámicamente a todo el ancho del grid-layout

# Definiendo el tamaño de las columnas dentro de un grid

Respuesta a:

Definiendo el tamaño de las columnas dentro de un grid

Tamaño de columnas

Para definir el número de Grid lines que ocupará un elemento, usamos la propiedad grid-column-start y grid-column-end, esto para definir desde que línea comenzará determinado elemento y en qué línea terminará. Para resumir estos parámetros podemos usar la propiedad grid-column: y de esta forma ponerle los parámetros en los que empieza y termina el elemento (Separados por /).

Para indicar que queremos usar espacios y no líneas, usamos de antemano “span”.

Cuando queremos ahorrarnos los parámetros de inicio y final de un elemento, podemos usar -1 en ambos casos.

Para definir el espacio que va a ocupar una columna se le indica de la siguiente manera con las siguientes propiedades:  
Ejemplo.  
`.item:nth-of-type(1){  
background: blue;  
grid-column-start: 1;  
grid-column-end: 4;

}`

Donde la funcion “grid-column-start: ;” indica el inicio desde el cual comenzara a ocupar espacio, donde el valor “1” indica que será a partir de la 1era línea.

A su vez, la funcion “grid-column-end: ;” indica el final del espacio que debe tomar, donde el valor “4” indica que será hasta la 4ta linea.

También se puede usar la siguiente propiedad :  
`.item:nth-of-type(8){  
grid-column: 2 / 4;

}`

Donde “2” es la línea de inicio y “4” la línea final, de manera que:

`.item:nth-of-type(8){  
grid-column: linea de inicio / linea final;

}

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

11- **Definiendo el tamaño de filas dentro de un grid**

Para definir el tamaño, en filas, de nuestros items, tenemos las siguientes propiedades para lograrlo:  
**grid-row-start:** define el comienzo del tamaño en # de filas de un item  
**grid-row-end:** define el final del tamaño en # de filas de un item

La forma corta de hacer esto mismo es:  
**grid-row:** inicio / final;

También podemos definir el tamaño de nuestro item por número de filas que deseamos que ocupe. Para lograrlo podemos utilizar la propiedad span a la cual le asignamos el número de filas a utilizar. Por ejemplo:  
.item:nth-of-type(1) { grid-row: 1 / span 3; }  
El ejemplo anterior le diría a nuestro navegador que el item 1 tiene un alto de tres filas, empezando a partir de la linea 1.

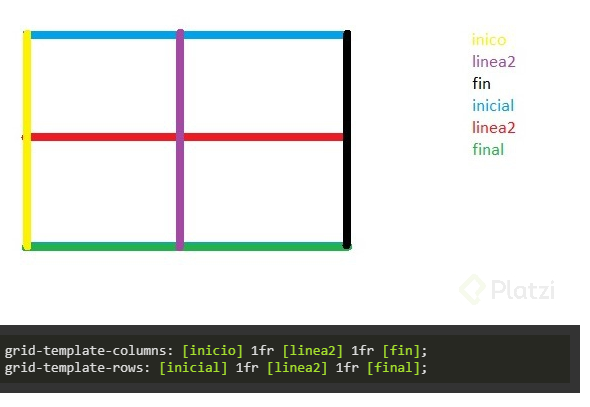
Podemos hacer lo anterior un poco más dinámico si no especificamos la línea de inicio de nuestro item:  
.item:nth-of-type(1) { grid-row: span 3; }  
Esto lo que hará es darle un alto de tres filas a nuestro item, no importando la posición de éste.

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

12- **Definiendo el nombre de lineas**

Algo muy útili para crear layouts es nombrar nuestras líneas. Esto lo podemos hacer al momento de definir nuestras filas y nuestras columnas, poniendo el nombre dentro de corchetes [ nombre ]. Por ejemplo:  
grid-template-column: [inicio ] 1fr [linea-2] 1fr [final];  
grid-tempate-row: [inicio] 200px [linea-2] 200px [final];

Para definir el tamaño de nuestros items por medio del nombre de las líneas solo basta con indicar el nombre de la línea de inicio y el nombre de la línea que marca el final del tamaño. Algo así:  
grid-column: inicio / linea-2;  
grid-row: inicio / final;



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

13- **Manejando el grid implícito**

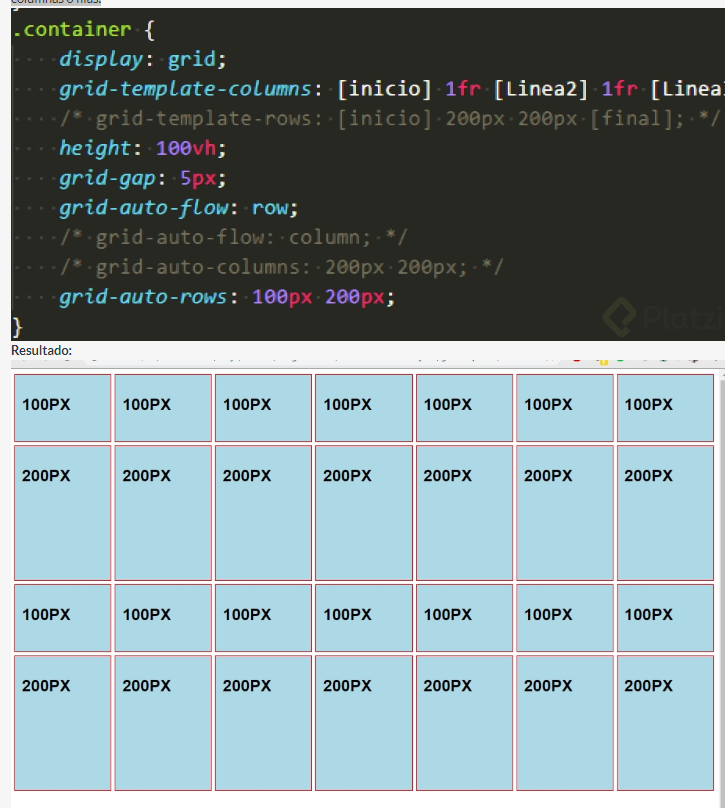
**Manejando grid implícito:** Cuando columnas o filas no tienen un valor explicito, es decir, cuando no asignamos valores específicos para controlarlas, se ordenan de manera implícita, hay propiedades que existen para manipular nuestro gird implícito:

**grid-auto-flow:** Con esta propiedad manipulamos el orden del grid implícito. Si ingresamos el valor **column** ,todo lo que sea grid implícito se apilara como columna y si ingresamos **row** ,como fila.

**grid-auto-column:** Con esta propiedad podemos darle valores a cada columna.

**grid-auto-row:** Con esta propiedad podemos darle valores a cada fila.

**NO ESPECIFICANDO:** Si yo no especifico filas o columnas, con **grid-auto-row** o **grid-auto-column** , podemos manipular todas las columnas o filas.



Más notas:

* si no hemos definido específicamente la ubicación de un elemento, el algoritmo colocación automática de Grid lo colocará en el próximo espacio (y suficientemente grande) disponible.
* Dónde no haya espacio disponible en la fila actual, comenzará a buscar en la siguiente fila, aún si ello deja huecos.  
  Podemos cambiar éste orden de búsqueda alterando grid-auto-flow de row a column.
* grid-auto-flowacepta una palabra reservada para describir el planteamiento de “cargado”. Por defecto éste valor es sparse, pero podemos alterar ésto a dense que intenta cubrir todas los huecos disponibles.

**Anotaciones:**

Para manejar el grid implícito, podemos usar la propiedad grid-auto-flow, de esta forma podemos cambiar la manera en la que se van a ordenar los elementos implícitos, es decir, si se ordenaran en filas o columnas, con valor por defecto de row.

Otra manera en la que podemos ordenar los elementos implícitos es con la propiedad de grid-auto-row, para asignar una medida a los elementos y ordenarlos a la vez a modo de filas, y respectivamente con la propiedad grid-auto-column.

En caso dado de que no se especifique el template para las filas o para columnas, la declaración faltante se asignara arbitrariamente una configuración para que ocupe el resto de tamaño de la pantalla.

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

**14- Alineación de contenido**

En el grid existe la forma de hacer que los grid-item se alineen según ciertas propiedades.

**justify-items:**  
Permite alinear el contenido (grid-item) de forma horizontal. Sus valores pueden ser: start end center stretch. Esta propiedad la usa el padre.

* Justify-item:start, Alinea los elementos(grid-item) hacia el principio (izquierda) y los elementos (grid-item) tendrán como ancho de dimensión el ancho de su contenido.
* justify-items: end, Alinea los elemento (grid-item), hacia el final (derecha) y los elementos (grid-item) al igual que start tendrán como ancho de dimensión el ancho de su contenido
* justify-items: center, Alinea los elemento (grid-item), hacia el centro (medio) y los elementos (grid-item) al igual que start y que end tendrán como ancho de dimensión el ancho de su contenido.
* justify-items: stretch, va a estirar el contenido según halla definido anteriormente en: grid-template, grid-template-columns, grid-template-rows.

**align-items:**  
Permite alinear el contenido (grid-item) de forma vertical. Sus valores pueden ser: start end center stretch. Esta propiedad la usa el padre.

* align-items: toma los mismo valores que justify-content pero su alineación lo hace de forma vertical.

El valor por defecto de align-item y justify-item es el strectch. strectch va a estirar el contenido según haya definido anteriormente en: grid-template, grid-template-columns, grid-template-rows.

**Se puede tomar un elemento (grid-item) alinearlo solamente a el**

* align-self:  
  alinea un elemento (grid item) de forma vertical y independiente, a los demás elemento que pertenecen al grid, puede tomar valores cómo : start, center, end, stretch. Propieda que usa el hijo.
* justify-selt:  
  alinea un elemento (grid-item) de forma horizontal y independiente a los demás elemento que pertenece al grid, puede tomar valores cómo: start, center, end, strectch
* **Anotaciones:**
* Para cambiar la configuración de alineación de nuestro contenido, podemos usar las propiedades align-items y justify-items para su respectiva alineación vertical y horizontal (respectivamente).
* Para alinear un solo elemento en específico, podemos repetir la propiedad que ya habíamos estudiado en clases pasadas, la cual es .item:nth-of-type(#){} y le asignamos las propiedades justify-self y align-self para su alineación propia.

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

15- **Alineación de filas y columnas**

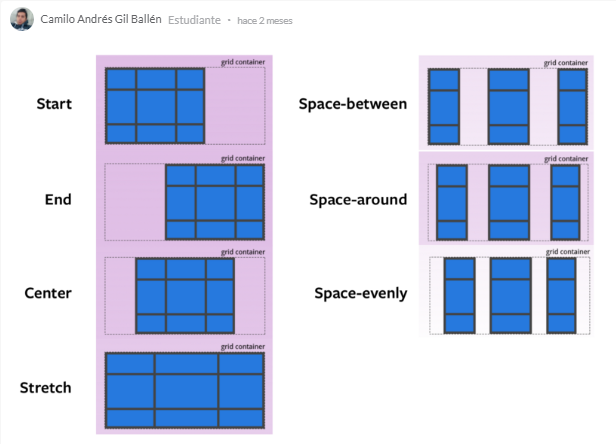
**justify-content:**  
Es una propiedad que permite alinear de formar horizontal el grid completamente. Esta puede tomar el valor propiedad de: start, end, center, stretch, space-around, space-between, space-evenly. El valor por defecto que recibe esta propiedad es start. Esta propiedad la utiliza el padre.

* justify-content:start  
  Es el valor por defecto.
* justify-content:end  
  Posiciona el grid horizontalmente hacia el final (Derecha).
* justify-content:center  
  Posiciona el grid en el centro, lo hace horizontalmente.
* justify-content: space-around  
  Asigna espacios al rededor de cada columna, pero este espacio entre columna se suma. Es decir sí tenemos 3 columnas A,B,C la propiedad space-around asigna un espacio de 100px al principio y final pero entre A , B , C esos 100px se duplica ya que A del lado derecho tiene 100px y B del lado izquierdo tiene 100px y por eso debe haber un espacio mayor entre ellos. Lo mismo ocurre entre B y C.
* justify-content: space-evenly  
  Asigna espacio más homogéneos o equivalentes alrededor de las columnas que espace-around. Lo que se diferencia del anterior propiedad es que ese espacio se puede ver reducido a medida que las pantalla son más pequeñas.
* justify-content: space-between  
  Asigna espacio entre las columanas del grid pero al final y al principio no asigna espacios.

**aling-content:**  
Es una propiedad que permite alinearde formar horizontal el grid completamente. Esta puede tomar el valor propiedad de: start, end, center, stretch, space-around, space-between, space-evenly. Esta propiedad la utiliza el padre.

**Anotaciones:**

Para alinear las filas y columnas usamos las propiedades justify-content y align-content. Los valores que tienen incluidos son start, end, center, stretch, space-between, space-around y space-evenly



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

16- Layout de InstagramUn juego para reforzar <https://cssgridgarden.com/#es>

Hola a todos, soy un viejo alumno del curso y deseo compartirles una **Guia de CSS Grid Layout** basada en los conocimientos adquiridos a lo largo de este curso con definiciones y excelentes ejemplos.

La guía cuenta con un **Pagina Web Demostrativa**, en la cual, están integrados algunos proyectos incluidos en esta sección.

NOTA: El proyecto cuenta con su repositorio de GitHub para que puedan clonarlo.

Guia: [https://github.com/luisvalladaresc/css-grid](https://github.com/LuisValladaresC/css-grid)  
Pagina Web: <https://luisvalladaresc.github.io/css-grid/>

Hola, veo que muchos tienen problemas haciendo funcionar el hover.

1. **Tengan cuidado con errores de typo**. Pueden copiar y pegar el nombre que le pusieron a la clase en el mero html.
2. Chequen sus **displays** y dónde colocan los **position**.

**Pro tip:** vayan depurando su código mientras lo van escribiendo, no hasta el final o cuando Leonidas lo haga,(él es pro, sabe cuando algo anda mal y cómo corregirlo). Esto les dará la capacidad de predecir el comportamiento y les dará un entendimiento más profundo del tema, además de qué les irá funcionando todo de maravilla.

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

17- terminando el layout de Instagram

Anotaciones:

Por acá dejo mi versión, la cual hice luego de ver esta clase.  
Use solo GRID para lo que creo que es el layout como tal y FLEX para manejar los elementos dentro de dicho layout. <https://github.com/WuilfredoAz/GRID-Instagram>

Les comparto mi ejercicio y el repositorio de mi proyecto.

<https://github.com/behagoras/instagramdesign>

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

18- Notas del ejercicio de Layout Mansory

* Si se desea configurar los elementos grid para que continuen (o empiece) inmediatamente despues  
  del otro elemento se declara solo el grid-row-end seguido de la cantidad de espacios a ocupar.  
  EX: grid-row-end: span 3;
* Para que las columnas se distribuyan dependiendo de la cantidad de espacio disponible en el viewport  
  se usa el atributo auto-fill de la propiedad repeat (que seria mas o menos igual a lo que hace flex con el wrap)  
  EX: .padre {grid-template-column: repeat(auto-fill, 250px);}  
  NOTA: Esta forma de distribucion llenara la pantalla de tantas columnas quepan en el viewport (asi esten vacias)
* Layout deseado, se puede usar grid-auto-flow, el cual es una propiedad que modifica la colocación automatica  
  de cada elemento en el grid. Su valor por defecto es el “sparse” el cual indica que si un elemento no  
  cabe en el “espacio” que dejo otro elemento mas grande para colocarlo este saltara hasta encontrar un espacio  
  donde quepa el elemento siguiente.  
  Sin embargo con la propiedad “dense” esto se puede alterar, ya que con esta propiedad se indica que  
  en el espacio disponible coloque cualquier elemento que quepa en dicho espacio sin importar su orden.  
  NOTA: En el Dense se necesita declarar medidas de alto y ancho a los contenedores .hijos (al menos 100% para cada una)
* En caso de que quedaran espacios vacios o no se ajustara los elementos correctamente al