



Facultad Regional Rosario

Universidad Tecnológica Nacional

Clase 07. DESARROLLO WEB

GRIDS



OBJETIVOS DE LA CLASE

- Conocer las Grids.
- Aplicar las nuevas formas de modelar con Grids.

GLOSARIO:

Clase 6

Flexbox: es un modo de diseño que nos permite crear estructuras para sitios web de una forma más fácil. No se trata de una propiedad de CSS, sino de un conjunto de ellas. Se basa sobre un contenedor (padre) para ordenar a sus ítems (hijos).

- **Row (flex-direction):** esta propiedad nos va a permitir especificar si queremos que los flex items se dispongan en filas o columnas.
- **Row-reverse (flex-direction):** con el valor row-reverse (fila inversa) los flex items se apilan en una fila de derecha a izquierda.
- **Column (flex-direction):** con este valor, los flex items se apilan en una columna de arriba hacia abajo.

- **Column-reverse:** con este valor, los flex items se apilan en una columna de abajo hacia arriba.
- **Flex-wrap:** permite especificar si queremos que los ítems puedan saltar a una nueva línea, cuando el contenedor flexible se quede sin espacio.
- **Wrap (flex-wrap):** los flex items (hijos) pueden romper la línea del eje horizontal, si les es necesario para conservar las características de sus dimensiones. Esto es de izquierda a derecha, y de arriba a abajo.
- **Wrap-reverse (flex-wrap):** esta vez el orden es de izquierda a derecha, y de abajo a arriba.

GLOSARIO:

Clase 6

- **Flex-flow:** Es la forma abreviada (shorthand) o rápida para las propiedades: flex-direction y flex-wrap. Se pone primero la propiedad de flex-direction, y luego la de flex-wrap.
- **Justify-content:** nos va a permitir alinear los elementos. Esto puede ser de forma vertical u horizontal, según lo especifiquemos con flex-direction. Nos va a ayudar a distribuir los flex items (hijos) en el contenedor (padre), cuando los ítems no utilicen todo el espacio disponible en su eje principal actual.
- **Flex-start (justify-content):** consiste en alinear los flex items (hijos) al lado izquierdo.
- **Flex-end (justify-content):** consiste en alinear los flex items (hijos) al lado derecho.
- **Center (justify-content):** consiste en alinear los flex items (hijos) al centro.
- **Space-between (justify-content):** es hacer que los flex items (hijos) tomen la misma distancia o espaciado entre ellos dentro del contenedor flexible, quedando el primer y último elemento pegados con los bordes del contenedor en el eje principal.
- **Space-around (justify-content):** muestra los flex items (hijos) con el mismo espacio de separación entre sí. El espaciado entre los bordes lo toman del contenedor padre.
- **Space-evenly:** hace que el espacio entre los flex items (hijos) sea igual. No es lo mismo que space-around.

GLOSARIO:

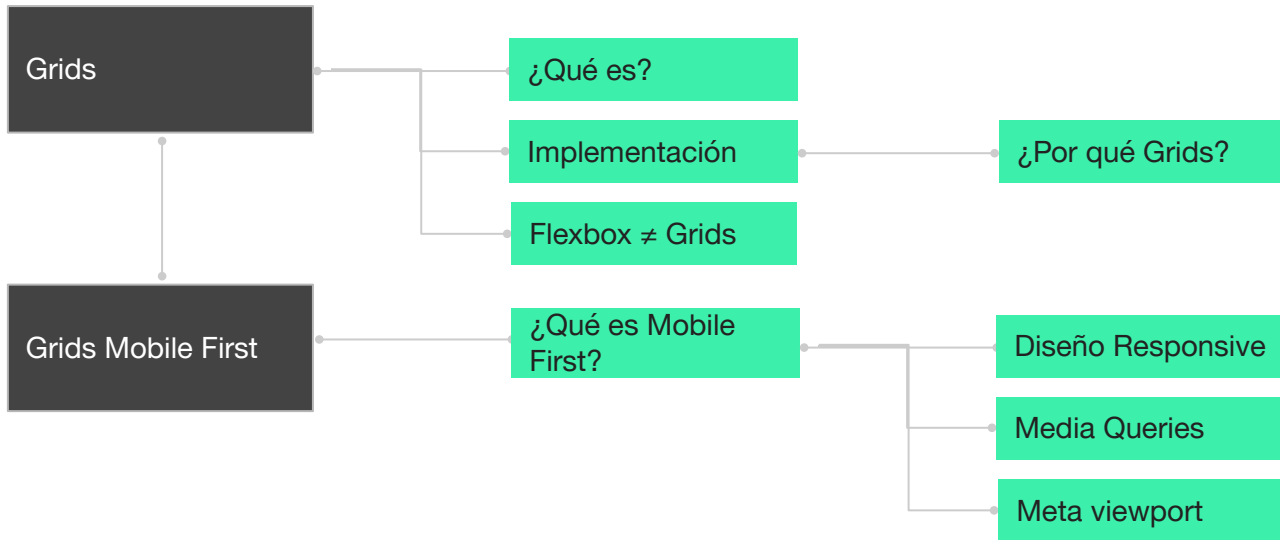
Clase 6

- **Stretch (align-items):** tratará de llenar toda la altura (o anchura) del contenedor, siempre y cuando los hijos no tengan propiedades de dimensión definidas.
- **Align-content:** esta propiedad sólo tiene efecto cuando el contenedor flexible tiene varias líneas de flex items (hijos). Si se colocan en una sola línea, esta propiedad no tiene ningún efecto sobre el diseño.
- **Order:** esta propiedad permite modificar el orden de aparición de un elemento. Recibe como valor numeros enteros.
- **Flex-basis:** define el ancho de un elemento inicial. Este valor por defecto viene configurado en “auto”. Actúa como “máximo” o “mínimo”, al usar flex-grow o flex-shrink.
- **Flex-grow:** esta propiedad define la capacidad de un elemento de crecer, cuando en el contenedor todavía hay espacio sobrante.
- **Flex-shrink:** básicamente es lo mismo que flex-grow, pero con el espacio faltante.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 7

¡Para
recordar!



GRIDS



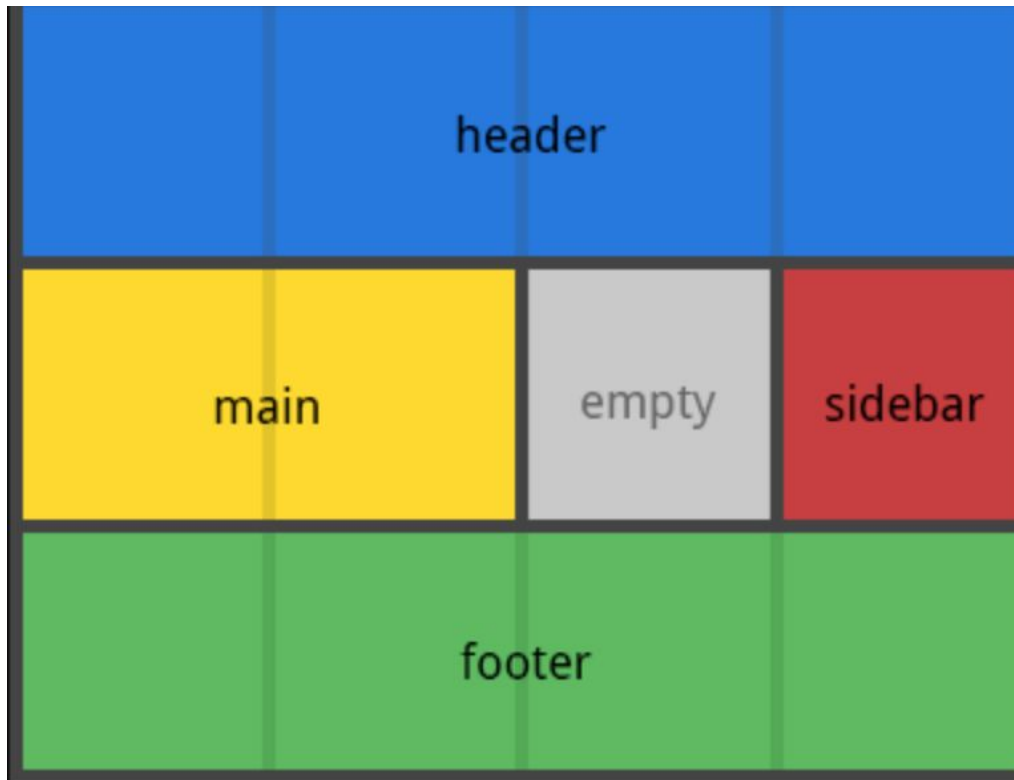
¿QUÉ ES?

CSS Grid es el sistema de maquetación más potente que hay disponible. Se trata de un sistema en 2D que permite definir filas y columnas (a diferencia de, por ejemplo, Flexbox, el cual funciona en una única dimensión).

El grid layout permite alinear elementos en columnas y filas. Sin embargo, son posibles más diseños con CSS grid que como lo eran con las tablas.

También nos permite tener más control sobre qué posición ocupan los hijos tanto en filas como en columnas.

CSS Grid layout contiene funciones de diseño dirigidas a los desarrolladores de aplicaciones web. El CSS grid se puede utilizar para lograr muchos diseños diferentes. Se destaca por dividir una página en regiones principales, o definir la relación en términos de tamaño, posición y capas, entre partes de un layout.





IMPLEMENTAR GRIDS

¿Position? ¿Float? ¿Elementos de Bloque?

¿Elementos de líneas?

¿Es suficiente crear un layout/estructuras para páginas web actuales?

Y... ¿Flexbox?

¿Necesitaremos algo más potente para estructuras web?

IMPLEMENTAR GRIDS



Los complementos vistos anteriormente suelen ser insuficientes, o a veces un poco complejos para crear un layout/estructuras para páginas web actuales.

Flexbox fue una gran mejora, pero está orientado a estructuras de una sola dimensión.

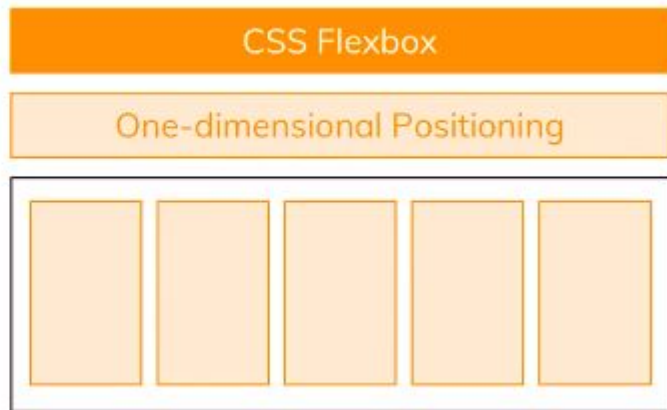
Muchos frameworks y librerías utilizan un sistema grid, donde definen una cuadrícula determinada, y cambiando los nombres de las clases de los elementos HTML es posible trabajar muchos atributos.

¿POR QUÉ GRIDS?

- Grid CSS surge de la necesidad de algo más potente, y toma las ventajas del sistema Flexbox, sumándole muchas mejoras y características que permiten crear muy rápido cuadrículas sencillas y potentes.
- Grid toma la filosofía y la base del sistema Flexbox. Esto no significa que lo reemplaza, sino que pueden convivir.
- Está pensado para estructuras grandes y complejas.

DIFERENCIA ENTRE FLEXBOX Y GRIDS

Flexbox

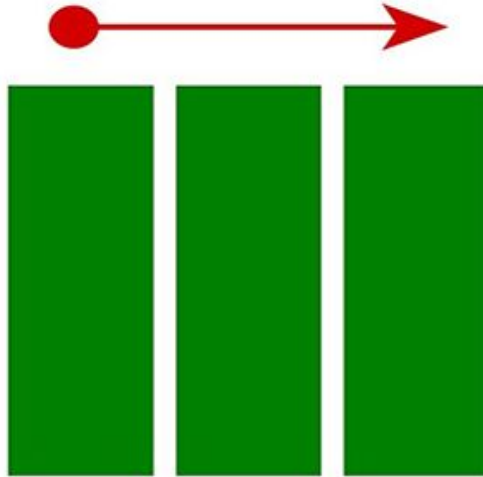


Grids



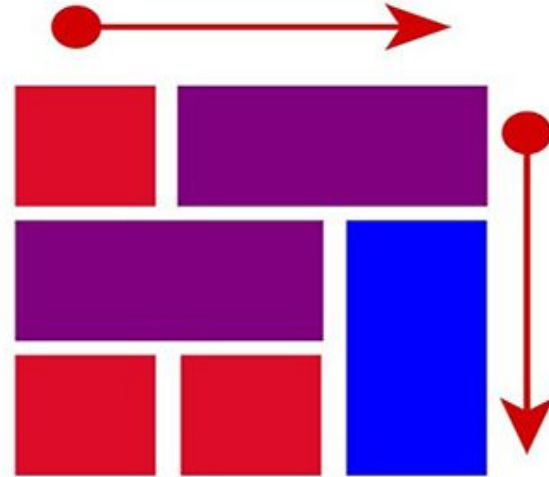
Ambos son mucho más potentes que cualquier técnica que haya existido antes.

DIFERENCIA ENTRE FLEXBOX Y GRIDS



Flexbox

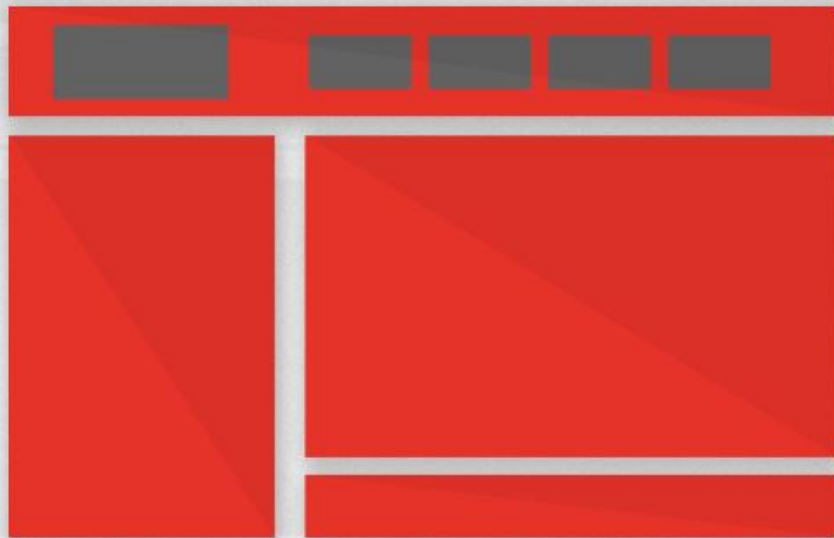
Unidimensional



CSS Grids

Bidimensional

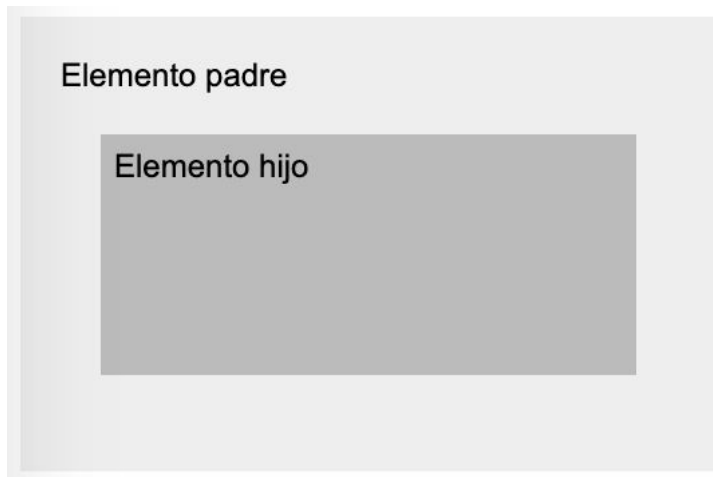
PUEDEN CONVIVIR



CSS GRID vs. FLEXBOX

PROPIEDADES DEL PADRE

Activamos la cuadrícula Grid utilizando, sobre el elemento contenedor, la propiedad display con el valor grid. Esto influye en cómo se comportará la cuadrícula con sus elementos hijos aunque inicialmente no se aprecia un cambio.



```
<section>  
  Elemento padre  
  <div>  
    Elemento hijo  
  </div>  
</section>
```

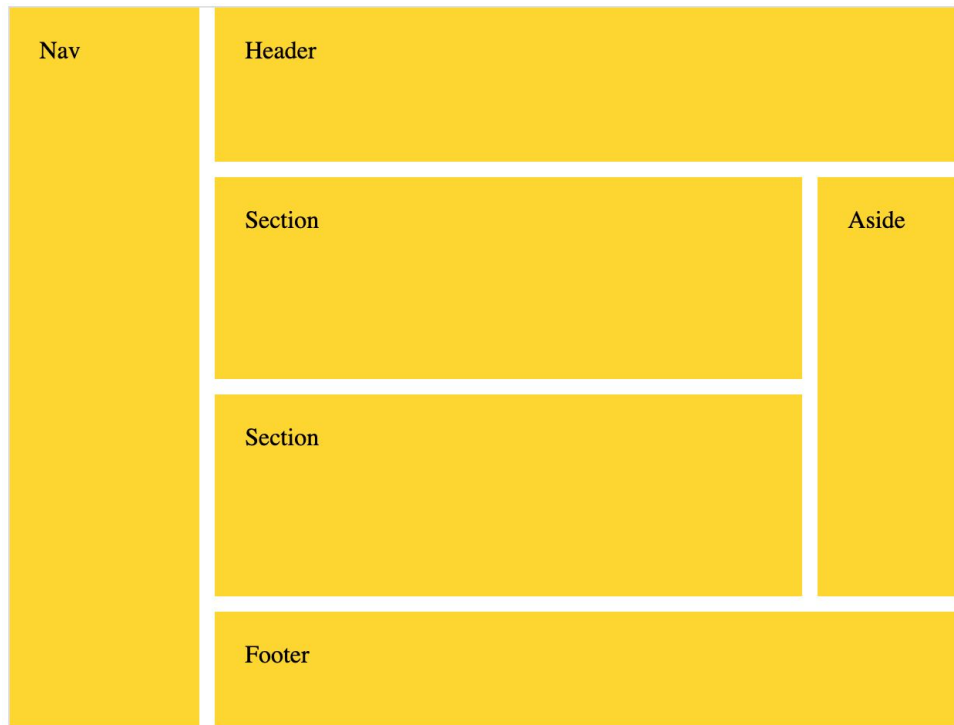
PROPIEDADES DEL PADRE

En primer lugar, aplicas la propiedad “**display: grid**” al elemento padre.
Luego puedes usar lo siguiente para crear la estructura principal:

grid-template-columns	Establece el TAMAÑO de cada columna (<u>col 1, col 2...</u>).
grid-template-rows	Establece el TAMAÑO de cada fila (<u>fila 1, fila 2...</u>).
grid-template-areas	Indica la disposición de las áreas en el grid. Cada texto entre comillas simboliza una fila.
grid-column-gap	Establece el TAMAÑO de los huecos entre columnas (<u>líneas verticales</u>).
grid-row-gap	Establece el TAMAÑO de los huecos entre filas (<u>líneas horizontales</u>).

EL OBJETIVO

Este tipo de estructura no es posible con Flexbox, por eso podemos pensar en usar **Grid**:



FILAS Y COLUMNAS EXPLÍCITAS

Es posible crear cuadrículas con un tamaño definido. Para ello, sólo tienes que usar las propiedades CSS *grid-template-columns* y *grid-template-rows*, las cuales sirven para indicar las dimensiones de cada celda de la cuadrícula, diferenciando entre columnas y filas.

Propiedad	Descripción
<i>grid-template-columns</i>	Establece el tamaño de las columnas (eje horizontal).
<i>grid-template-rows</i>	Establece el tamaño de las filas (eje vertical).

FILAS Y COLUMNAS EXPLÍCITAS

Veamos la forma más simple de crear una grilla, especificando cuántas columnas y filas queremos.

```
.grid {  
  display: grid;  
          /* 2 columnas */  
  grid-template-columns: 300px 100px;  
          /* 2 filas */  
  grid-template-rows: 40px 100px;  
}
```

```
<section class="grid">  
  <div>Item 1</div>  
  <div>Item 2</div>  
  <div>Item 3</div>  
  <div>Item 4</div>  
</section>
```

FILAS Y COLUMNAS EXPLÍCITAS

	300px	100px
40px	Item 1	Item 2
100px	Item 3	Item 4

FILAS Y COLUMNAS EXPLÍCITAS

Unidad creada para ser usada en grid (**fr (fraction)**)

Nota: también es posible utilizar otras unidades y combinarlas, como porcentajes o la palabra clave **auto** (que obtiene el tamaño restante).

```
.grid {  
  display: grid;  
  grid-template-columns: 2fr 1fr;  
  grid-template-rows: 3fr 1fr;  
}
```

FILAS Y COLUMNAS EXPLÍCITAS

Cuadrícula de 2x2, donde el tamaño de ancho de la cuadrícula se divide en **dos columnas** (una el doble de tamaño que la siguiente), y el tamaño de alto de la cuadrícula se divide en dos filas, **donde la primera ocupará el triple (3 fr) que la segunda (1 fr):**

	2fr	1fr
3fr	Item 1	Item 2
1fr	Item 3	Item 4

FILAS Y COLUMNAS REPETITIVAS

Si necesitas hacer muchas columnas y filas iguales, puedes usar lo siguiente:

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(12, 1fr);  
  grid-template-rows: repeat(12, 1fr)  
}
```

repeat([número de veces], [valor o valores])

FILAS Y COLUMNAS REPETITIVAS

Deberíamos hacer los divs necesarios, pero la grilla está lista para acomodar a sus ítems.

[illegible]

section.grid 626 × 240

Así “se ve” la pantalla dividida en grillas.

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

GRID POR ÁREAS

Ahora veremos otra forma de crear grillas, de una forma más flexible.

Es posible indicar el nombre y la posición concreta de cada área de la cuadrícula. Utiliza la propiedad `grid-template-areas`, donde debes especificar el orden de las áreas. Luego, en cada *ítem hijo*, usas la propiedad `grid-area` para indicar el nombre del área en cuestión.

De esta forma, es muy sencillo crear una cuadrícula altamente personalizada en apenas unas cuantas líneas de CSS.

GRID POR ÁREAS

Entonces, siguiendo la línea del objetivo que tenemos podemos hacer lo siguiente:

GRID POR ÁREAS

HTML base para todos los ejemplos. Es muy importante marcar la estructura HTML.

```
<div id="grilla">
  <header class="border">Header</header>
  <section id="productos" class="border">Section</section>
  <section id="servicios" class="border">Section</section>
  <nav class="border">Navegacion</nav>
  <aside class="border">Aside</aside>
  <footer class="border">Pie de pagina</footer>
</div>
```

GRID POR ÁREAS

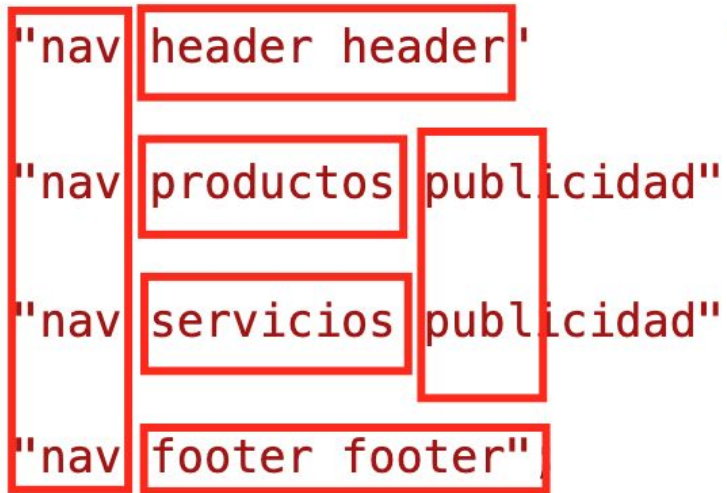
CSS a usar:

```
#grilla {  
  display: grid;  
  grid-template-areas:  
    "nav header header"  
    "nav productos publicidad"  
    "nav servicios publicidad"  
    "nav footer footer";  
  grid-template-rows: 100px 1fr 1fr 75px;  
  grid-template-columns: 20% auto 15%;  
}  
.border {  
  border: 1px solid black;  
}
```

```
header {  
  grid-area: header;  
}  
footer {  
  grid-area: footer;  
}  
section#productos {  
  grid-area: productos;  
}  
section#servicios {  
  grid-area: servicios;  
}  
nav {  
  grid-area: nav;  
}  
aside {  
  grid-area: publicidad;  
}
```

GRID POR ÁREAS

De esta forma, se aproxima a lo que queremos inicialmente, sólo nos falta darle unas decoraciones:



Navegacion	Header	
	Section	Aside
	Section	
	Pie de pagina	

Ejemplo
en vivo



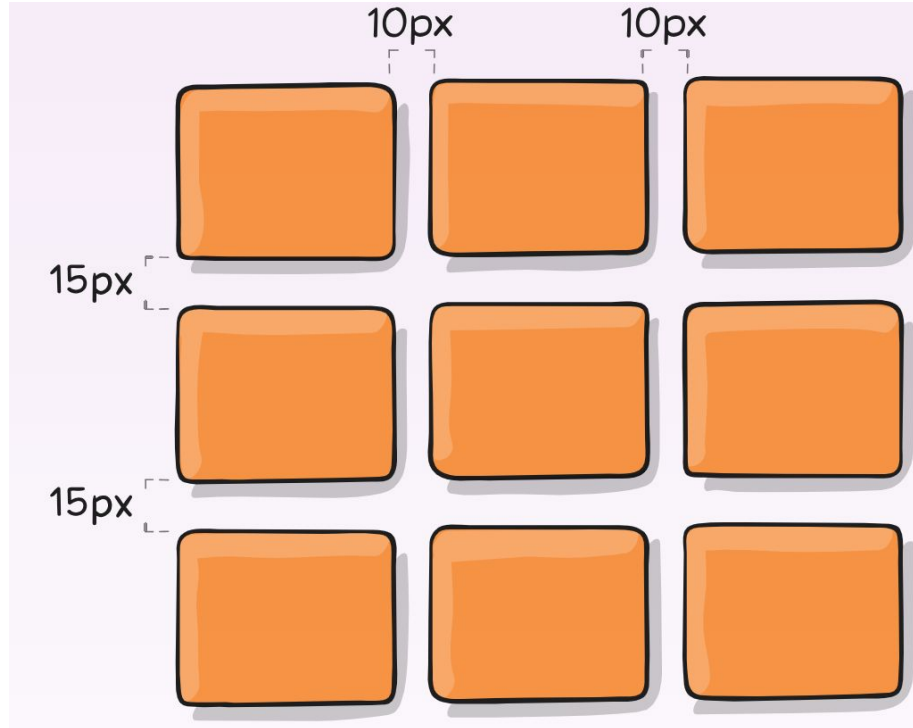
¡VAMOS A PRACTICAR LO VISTO!

GRID ESPACIOS

La cuadrícula tiene todas sus celdas **una a continuación de la otra**. Aunque sería posible darle un margen a las celdas dentro del contenedor, existe una forma más apropiada, evitando los problemas clásicos de los modelos de caja: los **huecos (gutters)**.

```
.grid {  
  grid-column-gap: 10px;  
  grid-row-gap: 15px;  
}
```

GRID ESPACIOS



GRID POR ÁREAS

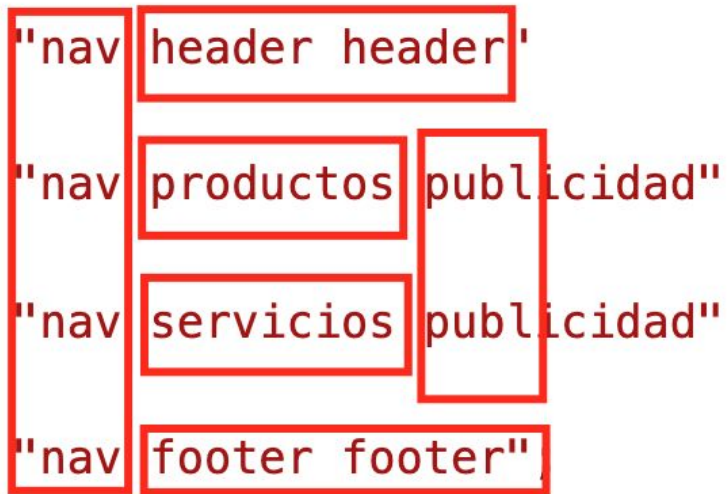
Llevemos a práctica , agregando más estilos al padre (el contenedor) y verificando que tiene las propiedades `grid-row-gap` y `grid-column-gap`, para hacer separaciones entre las columnas o filas, según el caso.

```
.border {  
  border: 1px solid black;  
  background-color: yellow;  
}
```

```
#grilla {  
  display: grid;  
  grid-template-areas:  
    "nav header header"  
    "nav productos publicidad"  
    "nav servicios publicidad"  
    "nav footer footer";  
  grid-template-rows: 100px 1fr 1fr 75px;  
  grid-template-columns: 20% auto 15%;  
  grid-row-gap: 10px;  
  grid-column-gap: 10px;  
  height: 100vh;  
  margin: 0;  
}
```

GRID POR ÁREAS

De esta forma, se aproxima a lo que queremos... sólo nos falta darle unas decoraciones:



Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

POSICIÓN DE HIJOS (DESDE EL PADRE)

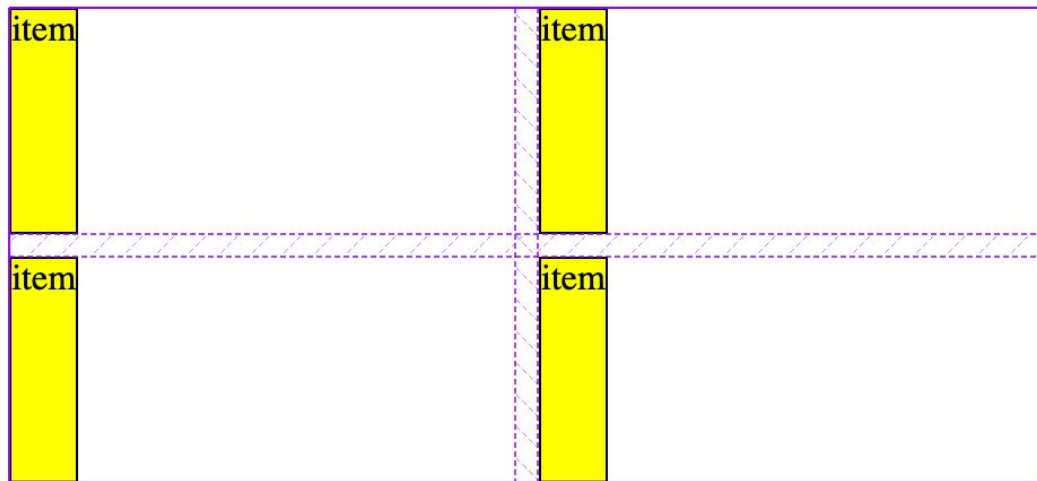
Existen propiedades que se pueden utilizar para colocar los ítems dentro de la cuadrícula. Es posible distribuir los elementos de una forma muy sencilla y cómoda: *justify-items* y *align-items*, que ya conocemos del módulo CSS

Flexbox:

Propiedad	Valores	Descripción
<i>justify-items</i>	start end center stretch	Distribuye los elementos en el eje horizontal.
<i>align-items</i>	start end center stretch	Distribuye los elementos en el eje vertical.

JUSTIFY-ITEMS Y ALIGN-ITEMS

La grilla está, pero las celdas “se achican”, se ajusta. Estas propiedades trabajan sobre la celda:



JUSTIFY-ITEMS Y ALIGN-ITEMS

HTML a usar:

```
<section class="grid">  
  <div>Item 1</div>  
  <div>Item 2</div>  
  <div>Item 3</div>  
  <div>Item 4</div>  
</section>
```

JUSTIFY-ITEMS Y ALIGN-ITEMS

CSS a usar:

```
#padre {  
    justify-items: stretch; /* start | end |  
center | stretch */  
    align-items: stretch; /* start | end |  
center | stretch */  
    display: grid;  
    width: 95%;  
    grid-template-columns: auto auto;  
    grid-column-gap: 10px;  
    grid-template-rows: 100px 100px;  
    grid-row-gap: 10px;  
}
```

```
#padre div {  
    border: solid 1px;  
    font-size: 21px;  
    padding: 5px;  
    background-color: yellow;  
}
```

Ejemplo
en vivo

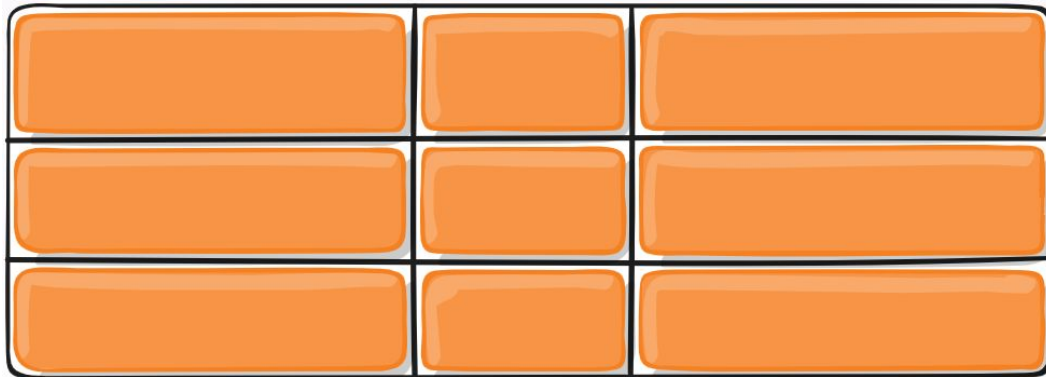


¡VAMOS A PRACTICAR LO VISTO!

JUSTIFY-ITEMS: CENTER

Alineando el
contenido
dentro de las
celdas, de
forma
horizontal.

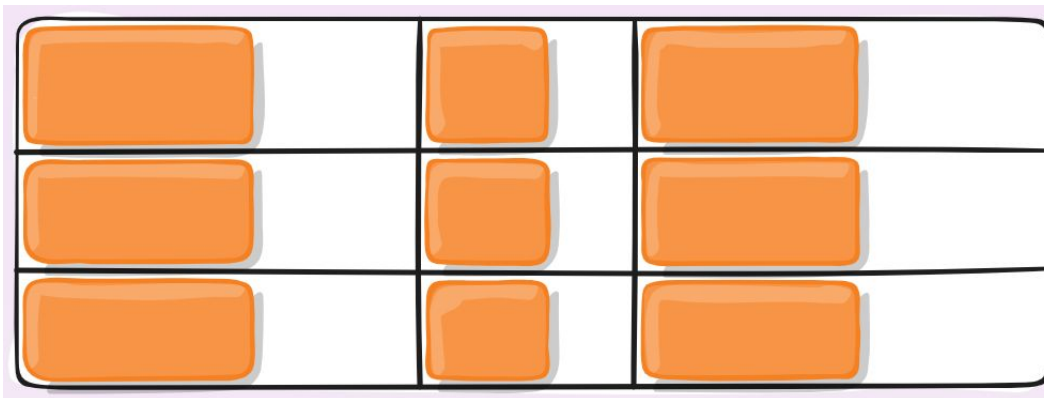
```
.padre {  
    justify-items: stretch;  
    /* predeterminado */  
}
```



JUSTIFY-ITEMS: START

Alineando el
contenido
dentro de las
celdas, de
forma
horizontal.

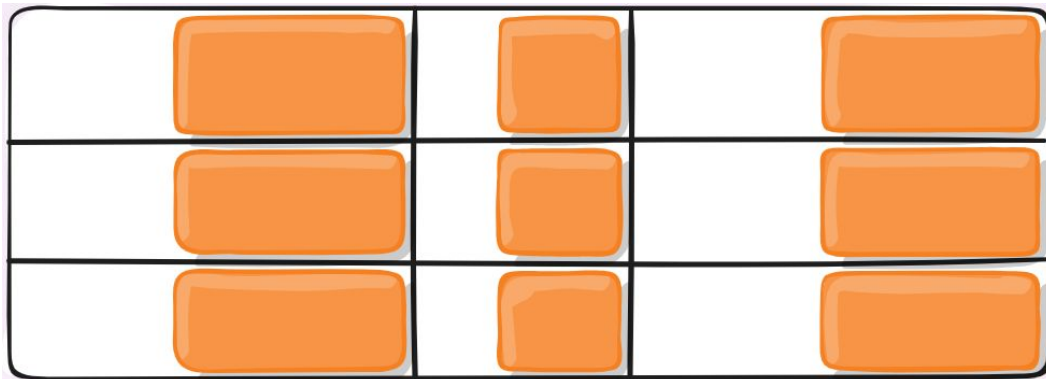
```
.padre {  
    justify-items: start;  
}
```



JUSTIFY-ITEMS: END

Alineando el
contenido
dentro de las
celdas, de
forma
horizontal.

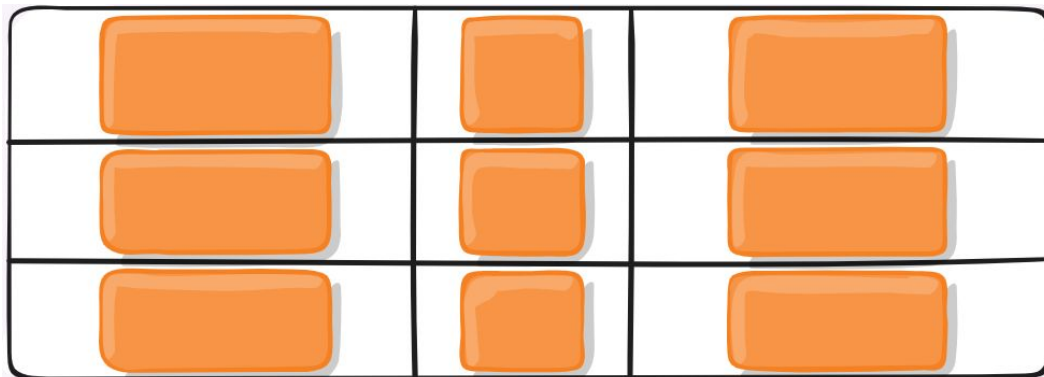
```
.padre {  
    justify-items: end;  
}
```



JUSTIFY-ITEMS: CENTER

Alineando el
contenido
dentro de las
celdas, de
forma
horizontal.

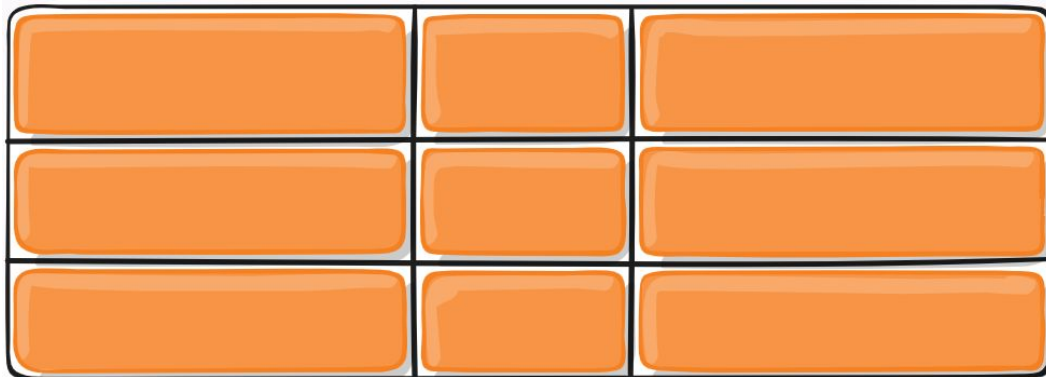
```
.padre {  
    justify-items: center;  
}
```



ALIGN-ITEMS: STRETCH

Alineando el
contenido
dentro de las
celdas, de
forma vertical.

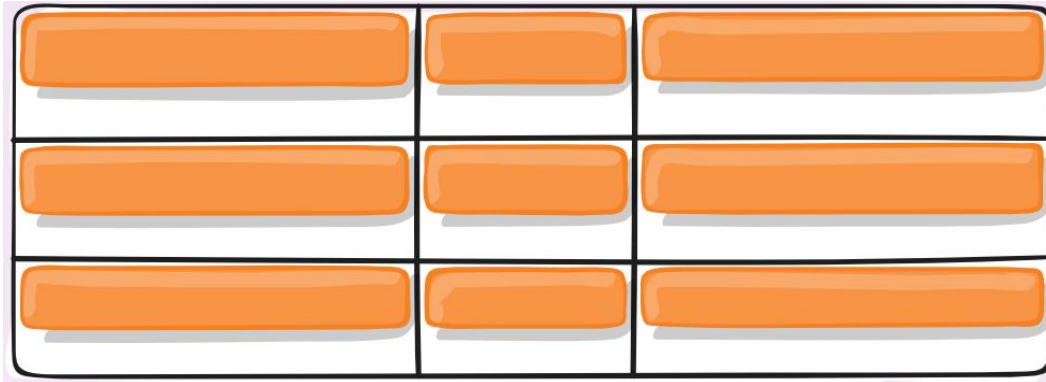
```
.padre {  
    align-items: stretch;  
    /* predeterminado */  
}
```



ALIGN-ITEMS: START

Alineando el
contenido
dentro de las
celdas, de
forma vertical.

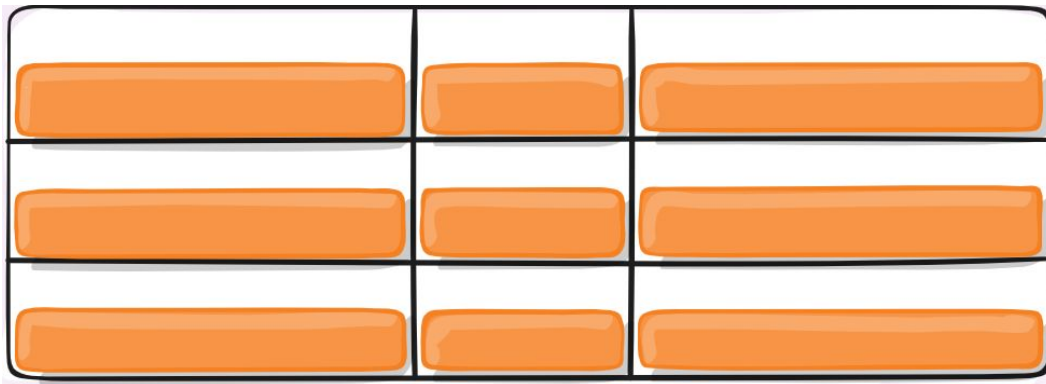
```
.padre {  
    align-items: start;  
}
```



ALIGN-ITEMS: END

Alineando el
contenido
dentro de las
celdas, de
forma vertical.

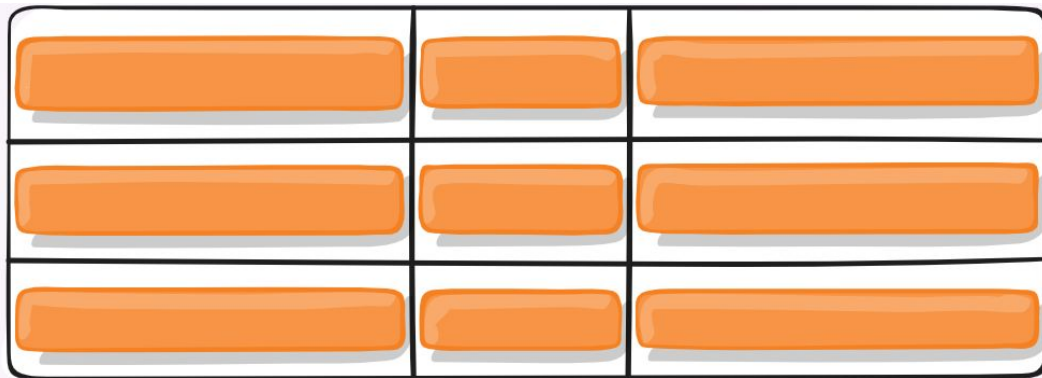
```
.padre {  
  align-items: end;  
}
```



ALIGN-ITEMS: CENTER

Alineando el
contenido
dentro de las
celdas, de
forma vertical.

```
.padre {  
    align-items: center;  
}
```



POSICIÓN DE ELEMENTOS

Es posible utilizar las propiedades *justify-content* o *align-content* para cambiar la distribución de todo el contenido en su conjunto. Puedes hacer pruebas en [este enlace](#).

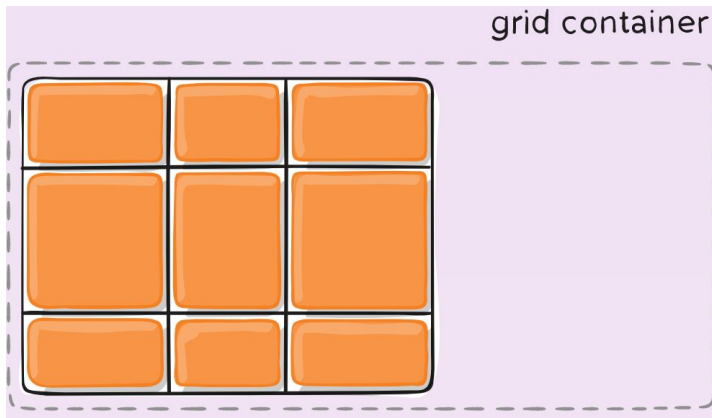
Propiedad	Valores	Afecta a
<i>justify-content</i>	start end center stretch space-around space-between space-evenly	Eje horizontal
<i>align-content</i>	start end center stretch space-around space-between space-evenly	Eje vertical

JUSTIFY-CONTENT: START

```
.padre {  
  justify-content: start;  
}
```

Alineando **todo** el conjunto de celdas, de forma **horizontal**.

Dentro de su “*padre*” (es requisito que tenga **ancho**).

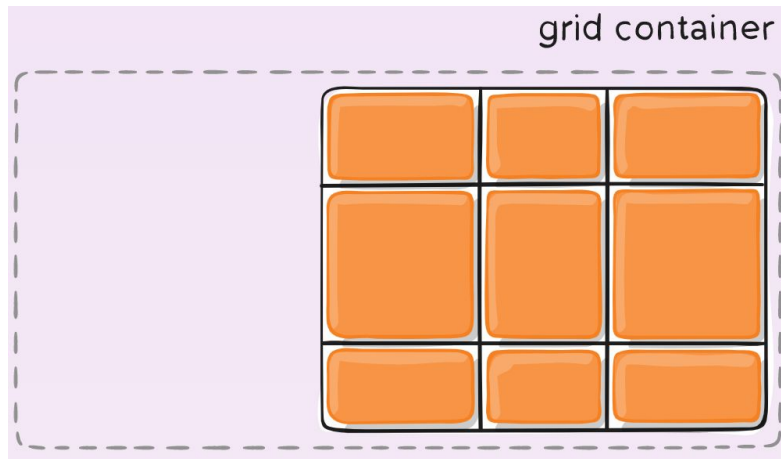


JUSTIFY-CONTENT: END

```
.padre {  
    justify-content: end;  
}
```

Alineando **todo** el conjunto de celdas, de forma **horizontal**.

Dentro de su “*padre*” (es requisito que tenga **ancho**).

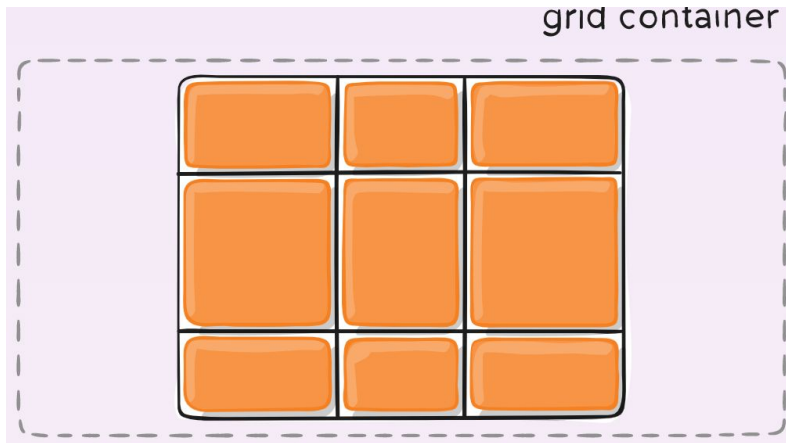


JUSTIFY-CONTENT: CENTER

```
.padre {  
    justify-content: center;  
}
```

Alineando **todo** el conjunto de celdas, de forma **horizontal**.

Dentro de su “*padre*” (es requisito que tenga **ancho**).

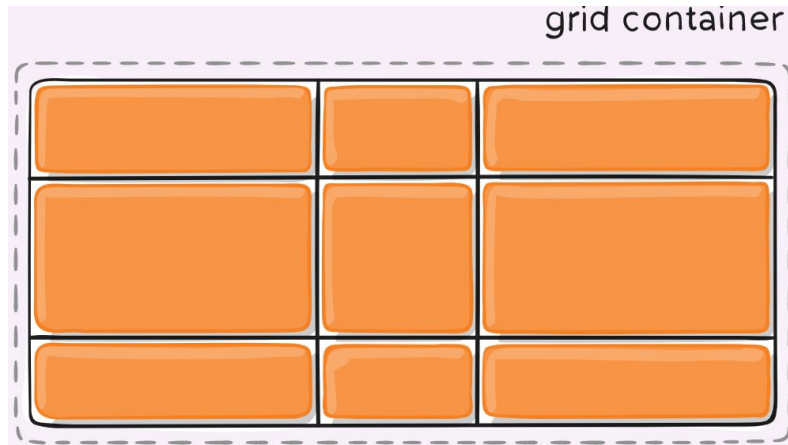


JUSTIFY-CONTENT: STRETCH

```
.padre {  
    justify-content: stretch;  
}
```

Alineando **todo** el conjunto de celdas, de forma **horizontal**.

Dentro de su “*padre*” (es requisito que tenga **ancho**).

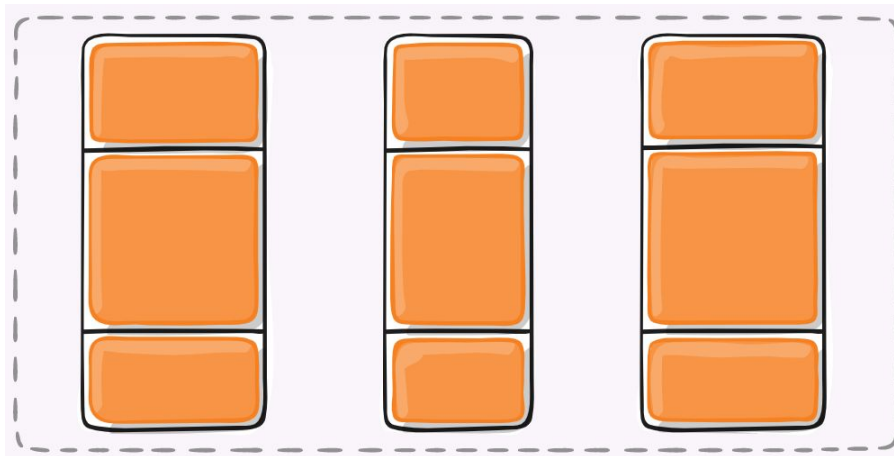


JUSTIFY-CONTENT: SPACE-AROUND

```
.padre {  
  justify-content: space-around;  
}
```

Alineando **todo** el conjunto de celdas, de forma **horizontal**.

Dentro de su “*padre*” (es requisito que tenga **ancho**).

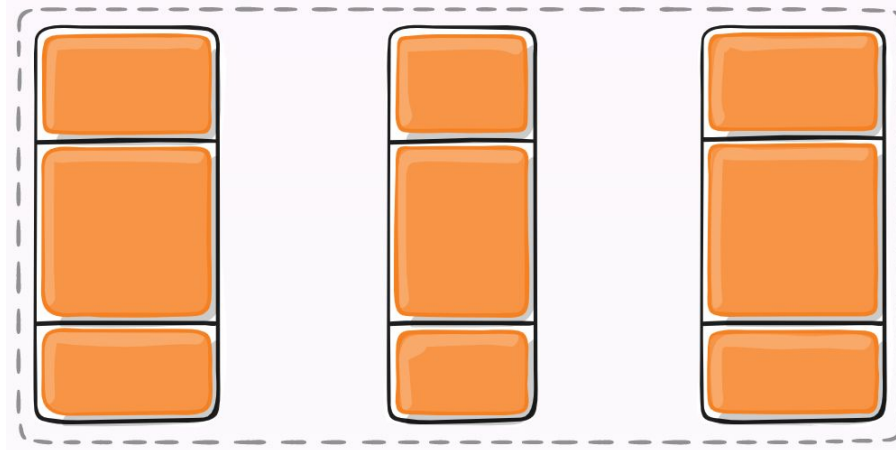


JUSTIFY-CONTENT: SPACE-BETWEEN

```
.padre {  
  justify-content: space-between;  
}
```

Alineando **todo** el conjunto de celdas, de forma **horizontal**.

Dentro de su “*padre*” (es requisito que tenga **ancho**).

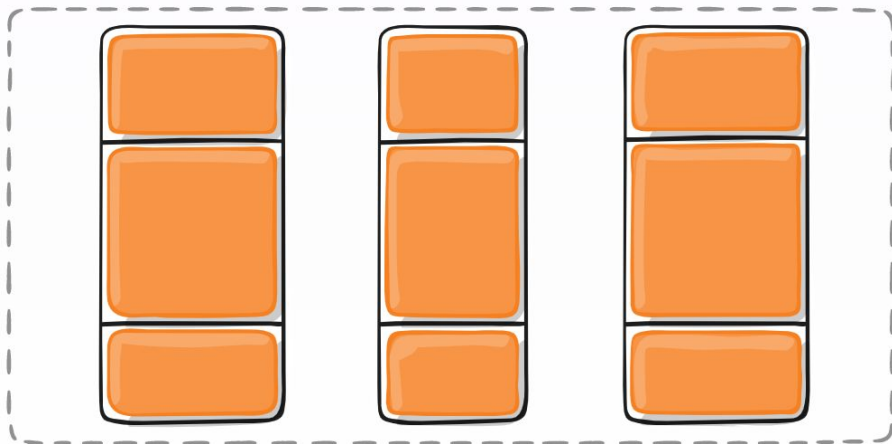


JUSTIFY-CONTENT: SPACE-EVENLY

```
.padre {  
  justify-content: space-evenly;  
}
```

Alineando **todo** el conjunto de celdas, de forma **horizontal**.

Dentro de su “*padre*” (es requisito que tenga **ancho**).

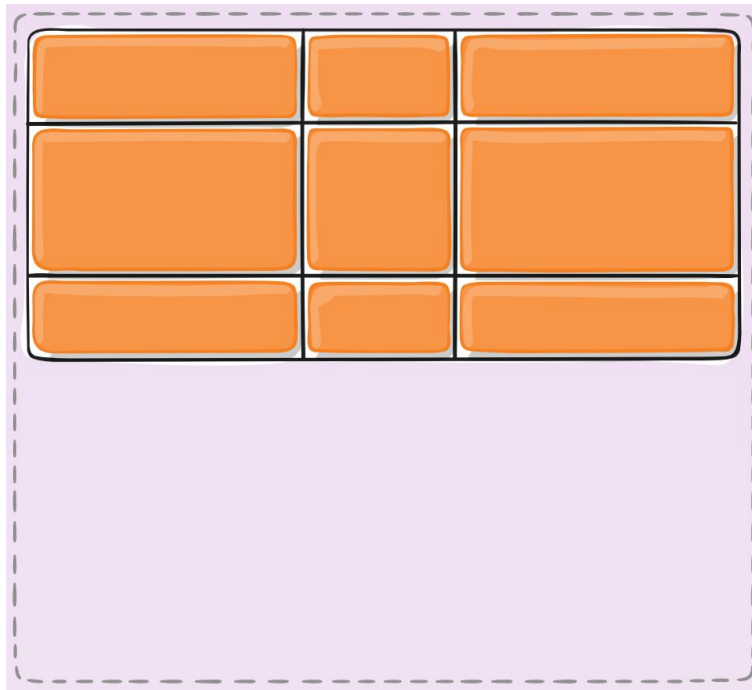


ALIGN-CONTENT: START

Alineando **todo** el conjunto de celdas, de forma **vertical**.

Dentro de su “*padre*” (es requisito que tenga **altura**).

```
.padre {  
  align-content: start;  
}
```

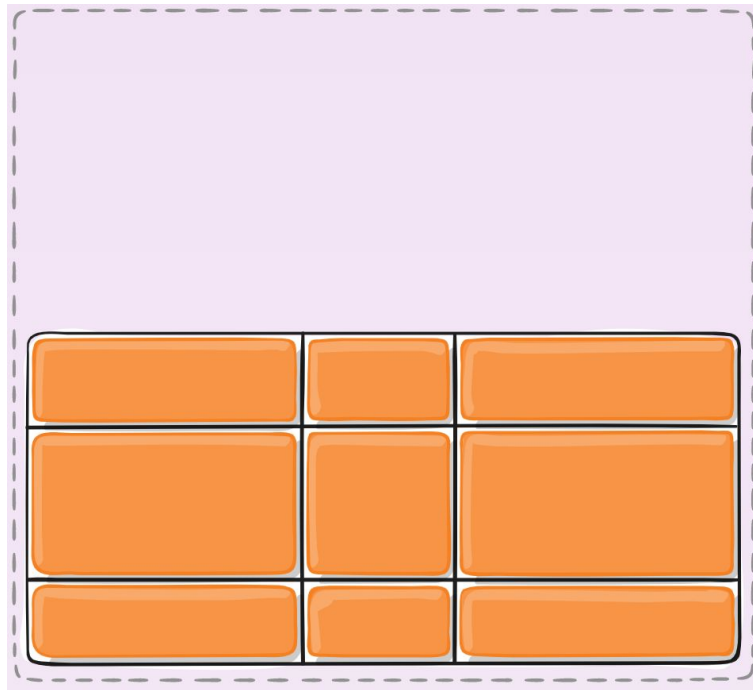


ALIGN-CONTENT: END

Alineando **todo** el conjunto de celdas, de forma **vertical**.

Dentro de su “*padre*” (es requisito que tenga **altura**).

```
.padre {  
  align-content: end;  
}
```

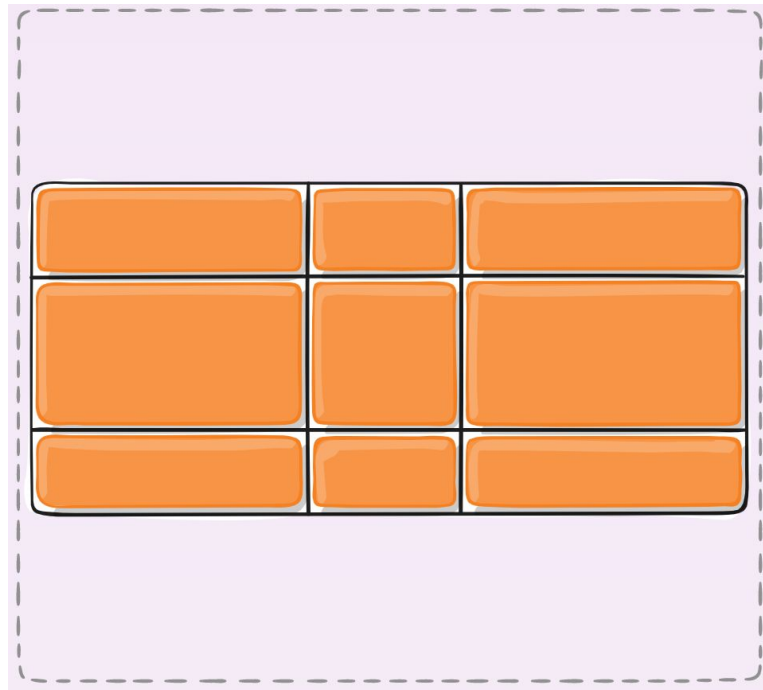


ALIGN-CONTENT: CENTER

Alineando **todo** el conjunto de celdas, de forma **vertical**.

Dentro de su “*padre*” (es requisito que tenga **altura**).

```
.padre {  
  align-content: center;  
}
```

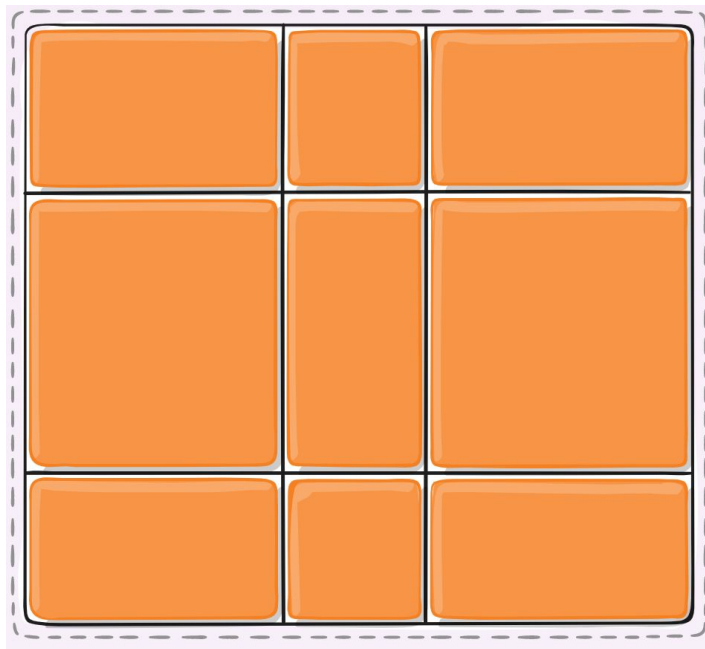


ALIGN-CONTENT: STRETCH

Alineando **todo** el conjunto de celdas, de forma **vertical**.

Dentro de su “*padre*” (es requisito que tenga **altura**).

```
.padre {  
  align-content: stretch;  
}
```

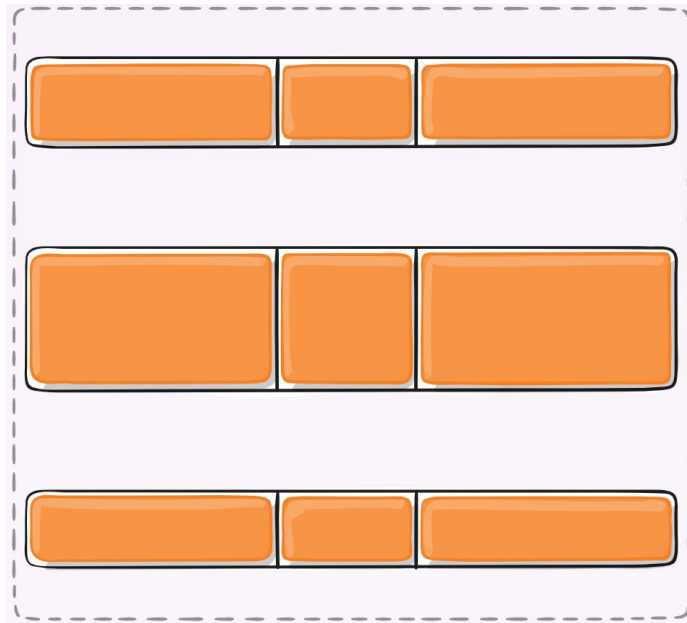


ALIGN-CONTENT: SPACE-AROUND

Alineando **todo** el conjunto de celdas, de forma **vertical**.

Dentro de su “*padre*” (es requisito que tenga **altura**).

```
.padre {  
  align-content: space-around;  
}
```

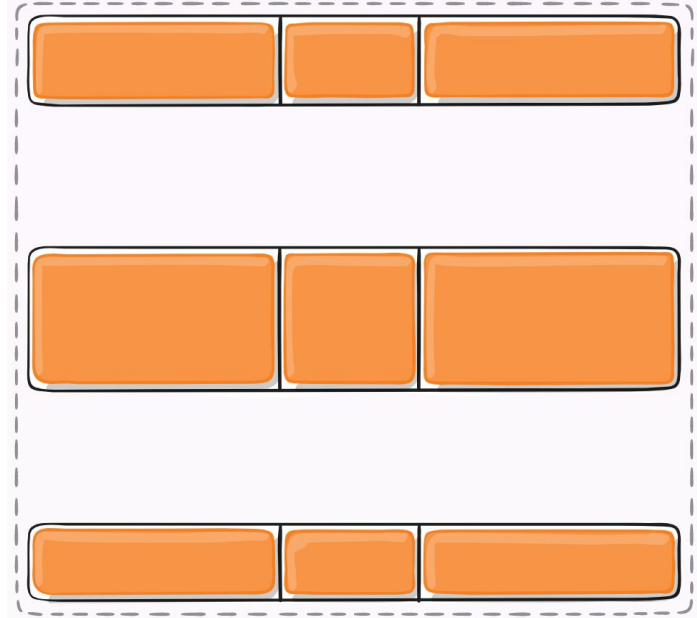


ALIGN-CONTENT: SPACE-AROUND

Alineando **todo** el conjunto de celdas, de forma **vertical**.

Dentro de su “*padre*” (es requisito que tenga **altura**).

```
.padre {  
  align-content: space-between;  
}
```

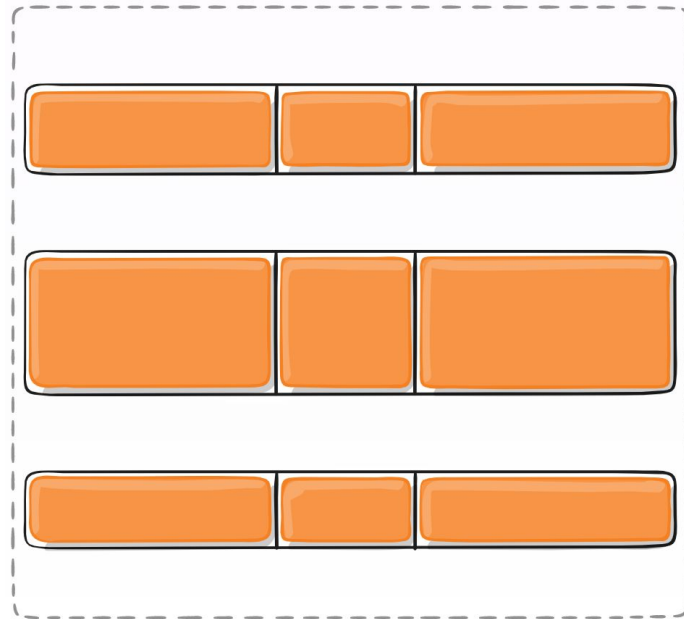


ALIGN-CONTENT: SPACE-EVENLY

Alineando **todo** el conjunto de celdas, de forma **vertical**.

Dentro de su “*padre*” (es requisito que tenga **altura**).

```
.padre {  
  align-content: space-evenly;  
}
```



Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

PROPIEDADES DE LOS HIJOS

Hasta ahora hemos visto propiedades CSS que se aplican solamente al contenedor padre de una cuadrícula. Ahora vamos a ver ciertas propiedades que se aplican a cada ítem hijo de la cuadrícula, para alterar o cambiar el comportamiento específico de dicho elemento.

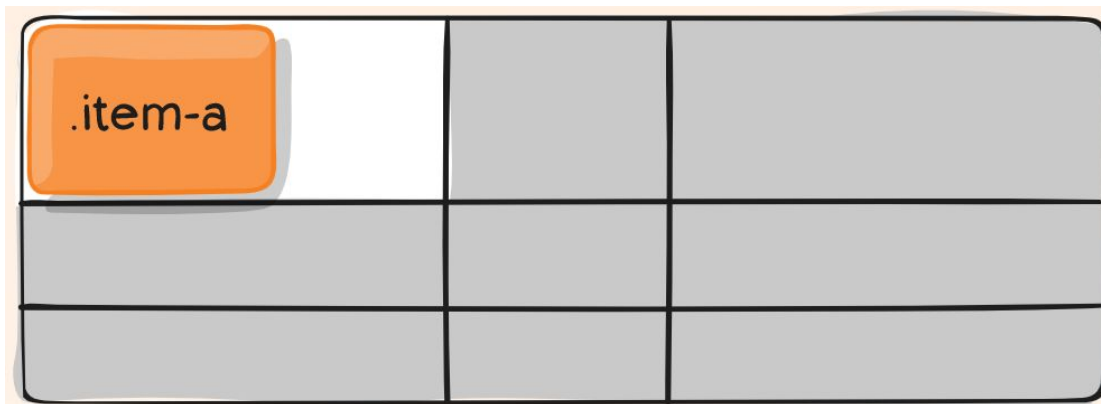
Prueba sus propiedades [aquí](#).

Propiedad	Descripción
<i>justify-self</i>	Altera la justificación del ítem hijo en el eje horizontal.
<i>align-self</i>	Altera la alineación del ítem hijo en el eje vertical.
<i>grid-area</i>	Indica un nombre al área especificada, para su utilización con <i>grid-template-areas</i> .

JUSTIFY-SELF: START

Alinea específicamente a la celda (item, hijo) que necesites, de forma horizontal:

```
.hijo {  
    justify-self: start;  
}
```



JUSTIFY-SELF: END

Alinea específicamente a la celda (item, hijo) que necesites, de forma horizontal:

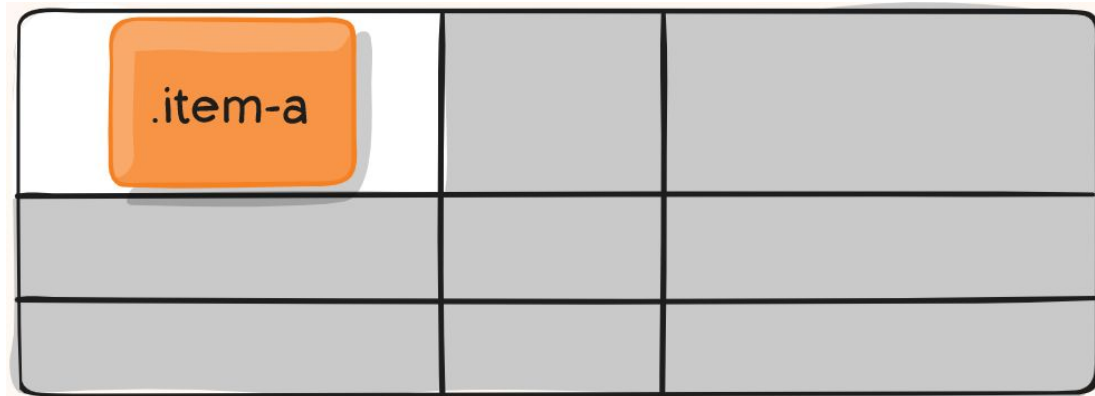
```
.hijo {  
    justify-self: end;  
}
```

<div>.item-a</div>		

JUSTIFY-SELF: CENTER

Alinea específicamente a la celda (item, hijo) que necesites, de forma horizontal:

```
.hijo {  
  justify-self: center;  
}
```



JUSTIFY-SELF: STRETCH

Alinea específicamente a la celda (item, hijo) que necesites, de forma horizontal:

```
.hijo {  
  justify-self: stretch;  
}
```

.item-a		

ALIGN-SELF: START

Alinea específicamente a la celda (item, hijo) que necesites, de forma vertical:

```
.hijo {  
  align-self: start;  
}
```

<div>.item-a</div>		

ALIGN-SELF: END

Alinea **específicamente** a la celda (item, hijo) que necesites, de forma **vertical**:

```
.hijo {  
  align-self: end;  
}
```

<div>.item-a</div>		

ALIGN-SELF: CENTER

Alinea **específicamente** a la celda (item, hijo) que necesites, de forma **vertical**:

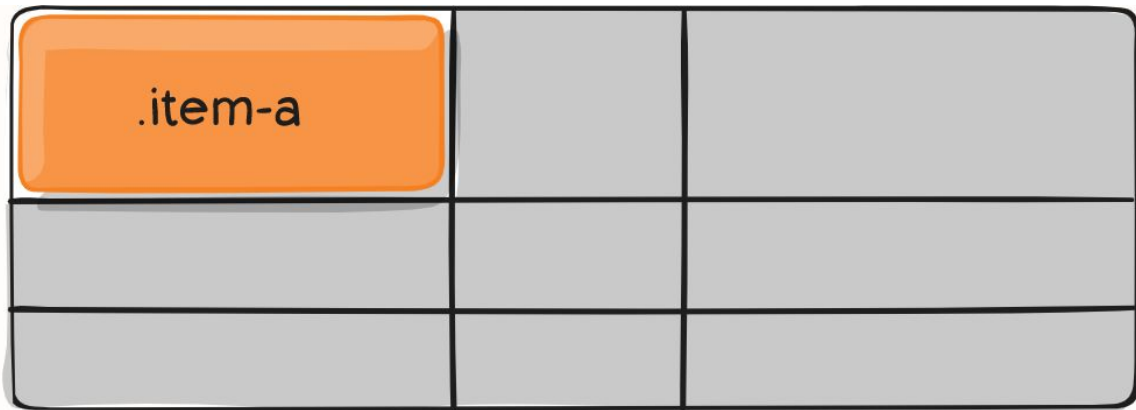
```
.hijo {  
  align-self: center;  
}
```

<div>.item-a</div>		

ALIGN-SELF: STRETCH

Alinea **específicamente** a la celda (item, hijo) que necesites, de forma **vertical**:

```
.hijo {  
  align-self: stretch;  
}
```



Ejemplo
en vivo



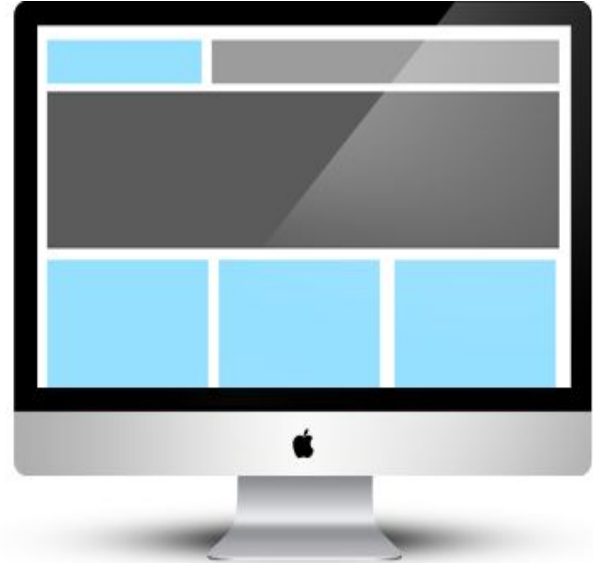
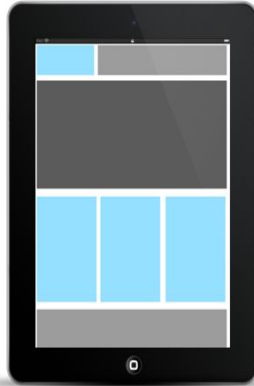
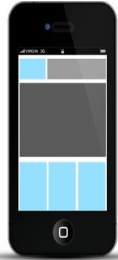
¡VAMOS A PRACTICAR LO VISTO!

GRIDS MOBILE FIRST



Facultad Regional Rosario
Universidad Tecnológica Nacional

MOBILE FIRST



MOBILE FIRST

Antes de hablar de “Mobile-First”, debemos hacer una referencia al llamado diseño responsive. Se refiere a la idea de que un sitio web debería mostrarse igual de bien en todo tipo de dispositivo, desde monitores de pantalla panorámica hasta teléfonos móviles.

Es un enfoque para el diseño y desarrollo web que elimina la distinción entre la versión amigable para dispositivos móviles de un sitio web, y su contraparte de escritorio. Con un diseño responsive, ambos son lo mismo.

MOBILE FIRST

Mobile First significa **crear el código primero para los dispositivos más pequeños** que los usuarios probablemente tengan, como teléfonos o tabletas. Implica trabajar en el dispositivo más pequeño, y luego acumular desde allí todo en el mismo código y el mismo proyecto, en lugar de hacer uno nuevo para cada tamaño de pantalla.

MOBILE FIRST

Se recomienda:

- En primer lugar, **trabajar el código** para que se reproduzca perfectamente en un **teléfono**.
- Luego **ajustar** para que se ejecute en una **tableta**.
- Por último, en un **dispositivo de escritorio**.

MOBILE FIRST

Cualquier estilo dentro del siguiente Media Querie se ejecutará cuando el tamaño de la pantalla sea de al menos 768px de ancho (tablet portrait iPad Mini), pero no cuando el tamaño de la pantalla sea menor:

```
@media only screen and (min-width: 768px)
{
    .body {
        background-color: #000000;
    }
}
```

MEDIA QUERIES

El diseño responsive se logra a través de "Media Queries" de CSS. Pensemos en las Media Queries como una forma de aplicar condicionales a las reglas de CSS. Estas últimas le indican al navegador qué reglas debe ignorar o aplicar, dependiendo del dispositivo del usuario.

BREAK POINTS

Tamaño	Dispositivo
320px	Para dispositivos con pantallas pequeñas, como los teléfonos en modo vertical
480px	Para dispositivos con pantallas pequeñas, como los teléfonos, en modo horizontal
600px	Tabletas pequeñas, como el Amazon Kindle (600×800) y Barnes & Noble Nook (600×1024), en modo vertical
768px	Tabletas de diez pulgadas como el iPad (768×1024), en modo vertical
1024px	Tabletas como el iPad (1024×768), en modo horizontal, así como algunas pantallas de ordenador portátil, netbook, y de escritorio
1200px	Para pantallas panorámicas, principalmente portátiles y de escritorio

GRIDS MOBILE FIRST

1

Estructura
HTML.

2

Asígnale a tu
contenedor la
propiedad de
display: grid;

3

Luego asigna el número
de columnas y filas que
tendrá tu grilla, así como
un espacio de
separación.

4

Define el área que
ocupará cada caja de tu
contenedor. Asígnales
un nombre y un color
característico.

5

Determina cómo
quieres que cada
área sea
acomodada en
tu layout.

TABLET

Siguiendo el ejemplo de las grillas por áreas:

- Para la versión tablet, lo primero que haces es cambiar la disposición de las columnas de tu Grid.
- Luego cambia la disposición de los ítems, esta vez usando el recurso de *grid-row* y *grid-column*, que es el método corto de *grid-row-start/end* *grid-column-start/end*.

```
@media only screen and (min-width: 768px)
{
    #grilla {
        grid-template-columns: repeat(4, 1fr);
    }
    .border {
        border: 4px solid black;
        background-color: blue;
    }
}
```

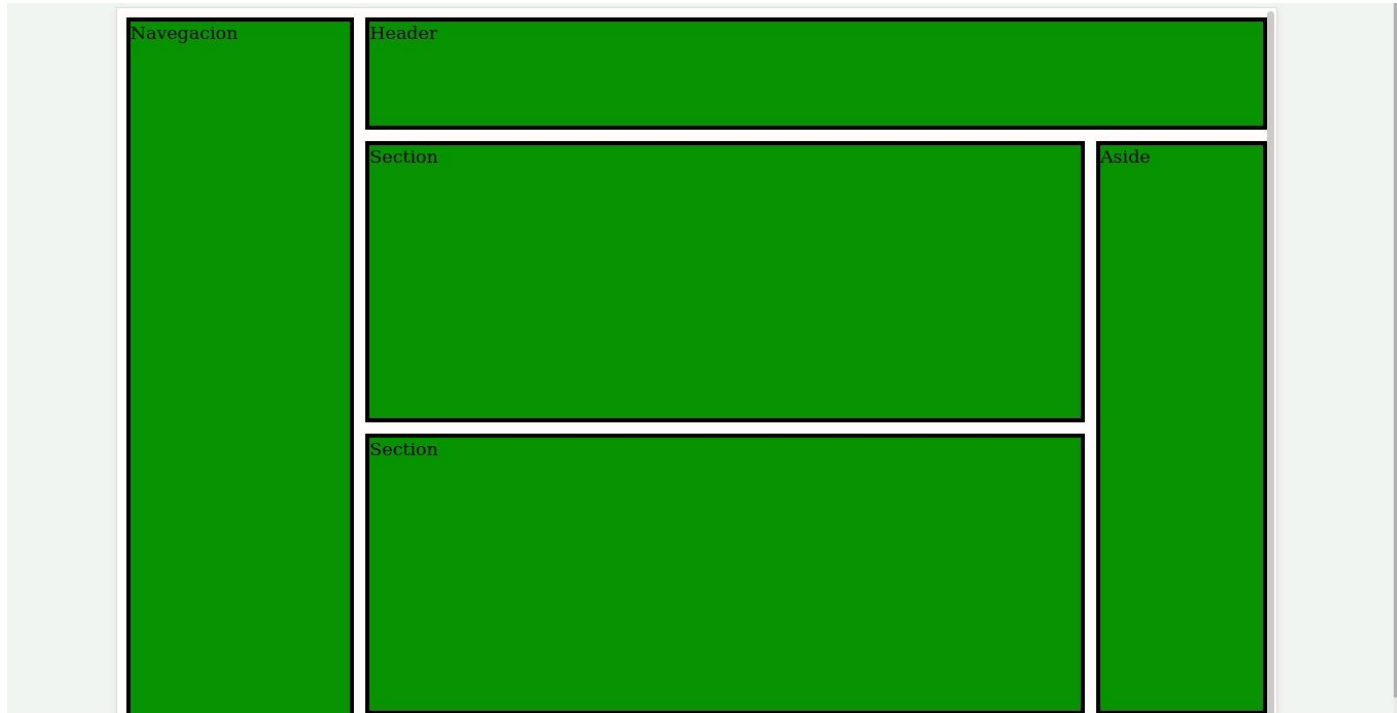
DESKTOP

Siguiendo el ejemplo de las grillas por áreas:

- Cambia la disposición de la grilla.
- Una vez más, cambia la disposición de los ítems.

```
@media only screen and (min-width:
1024px) {
    #grilla {
        grid-template-columns: repeat(3, 1fr);
    }
    .border {
        border: 4px solid black;
        background-color: green;
    }
}
```


RESULTADO



META VIEWPORT

Las páginas optimizadas para diferentes dispositivos deben incluir la etiqueta **<meta> viewport**, en el encabezado del documento HTML. Una etiqueta **<meta> viewport** da al navegador las instrucciones sobre cómo controlar las dimensiones, y el ajuste a escala de la página.

- Usa la etiqueta **<meta> viewport** para controlar el ancho y el ajuste de la ventana de visualización del navegador.
- Incluye **width=device-width** para hacer coincidir el ancho de la pantalla en píxeles independientes del dispositivo.
- Incluye **initial-scale=1** para establecer una relación de 1:1 entre los píxeles CSS, y los píxeles independientes del dispositivo.

META VIEWPORT

```
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
```

El uso del valor de `width=device-width` indica a la página que debe hacer coincidir el ancho de la pantalla en píxeles independientes del dispositivo. Esto permite que la página realice el reprocesamiento del contenido para adaptarlo a diferentes tamaños de pantalla.

META VIEWPORT





APLICANDO GRIDS

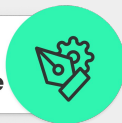
Aplica GRIDS en uno o más HTML que tengas.

APLICANDO GRIDS

Formato: Archivo html y css

Sugerencia: carpeta en formato zip o rar con el/los archivos html y css.

Desafío
entregable



>> Consigna:

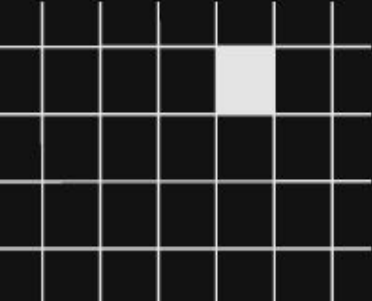
1. Aplicar GRIDS en uno o más HTML que tengas (por ej: integrantes.html)
2. Aplicar Mobile First en el Home de tu proyecto para que sea responsive en tres dispositivos.

¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Grids.
 - Nuevas formas de modelar Grids.
- 



OPINA Y VALORA ESTA CLASE