Desarrollo WEB

CLASE 7

Grids

CODER HOUSE

GRIDS

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**. Destaca por dividir una página en regiones principales, o definir la relación en términos de tamaño, posición y capas entre partes de un control.

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. Por ejemplo, los elementos secundarios de un contenedor de cuadrícula podrían posicionarse de manera que se solapen y se superpongan, similar a los elementos posicionados en CSS.

Fuente:
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout

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

Es importante mencionar que el soporte que actualmente tiene CSS Grid es bueno, sin embargo, **para navegadores viejos o nuestro querido amigo Internet Explorer, es mejor buscar otra alternativa.** *(Fuente:https://caniuse.com/#feat=css-grid)*
En esta Unidad aprenderás a crear tu primer layout con CSS Grid, basado en el paradigma Mobile First . Y podrás ver lo que CSS Grid es capaz de hacer y cómo podrá quitarnos algunos dolores de cabeza. Con esta Unidad como base, podrás extender lo aprendido a tus proyectos.

Implementar Grids

En donde nos encontramos en más dificultades es en el proceso de colocar y distribuir los elementos a lo largo de una página. **Los posicionamientos, floats o elementos de bloque o de línea suelen ser insuficientes** (o a veces un poco complejos) para crear un layout/estructuras para páginas web actuales.

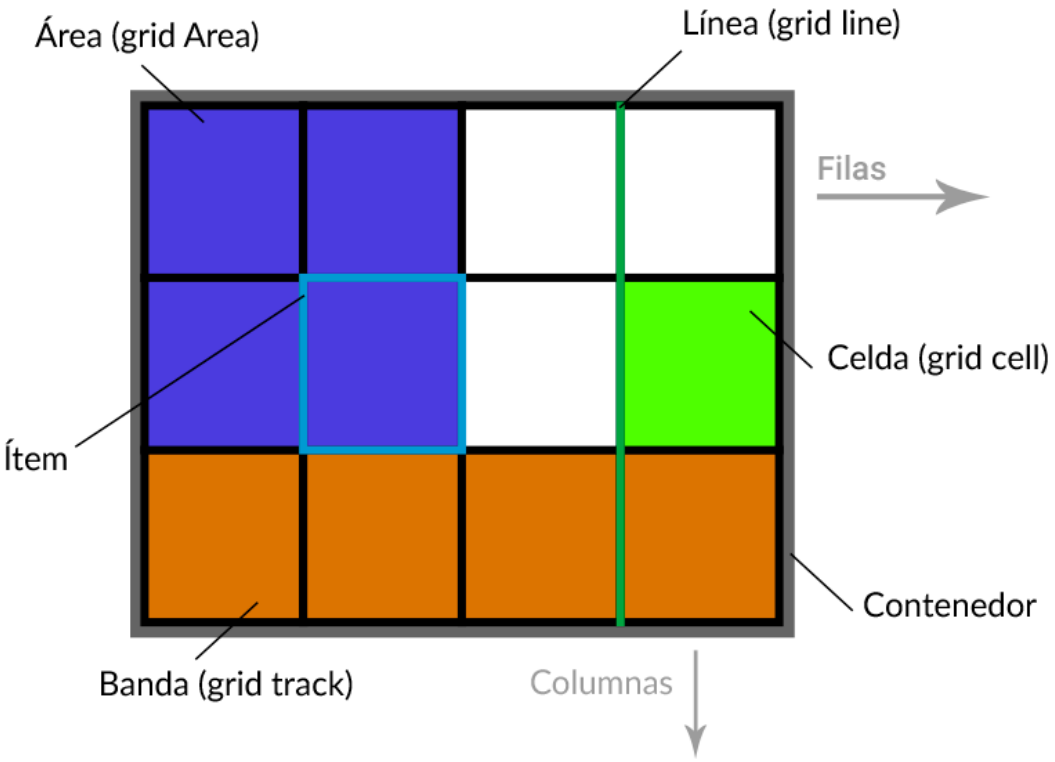
Flexbox fue una gran mejora pero está orientado a estructuras de una sola dimensión por lo que necesitamos algo más potente para estructuras web. Muchos

frameworks y librerías utilizan un sistema grid donde definen una cuadrícula determinada, y cambiando los nombres de las clases de los elementos HTML podemos darle: tamaño, posición o algún tipo de colocación.

Grid CSS surge de la necesidad, y toma las ventajas de ese sistema, sumándole muchas mejoras y características que permiten crear muy rápido cuadrículas sencillas y potentes.

Conceptos de base

Grid toma la filosofía y la base del sistema Flexbox. Para comenzar con Grid necesitamos tener en cuenta una serie de conceptos que utilizaremos:



Contenedor: Elemento padre definirá la cuadrícula.
Ítem: Los hijos que contiene la cuadrícula (Contenedor)
Celda (*grid cell*): Unidad mínima de la cuadrícula.
Área (*grid area*): Región de celdas de la cuadrícula.
Banda (*grid track*): Banda horizontal o vertical deceldas dentro de la cuadrícula.
Línea (*grid line*): Separador entre las celdas de la cuadrícula.

Vamos a empezar la implementación sobre este HTML:

```
<div class="grid"> <!-- contenedor -->
  <div class="a">Item 1</div> <!-- cada uno de los ítems del grid -->
  <div class="b">Item 2</div>
  <div class="c">Item 3</div>
  <div class="d">Item 4</div>
</div>
```

Activamos la cuadrícula Grid utilizando **sobre el elemento contenedor la propiedad display con el valor *grid* o *inline-grid***. Esto influye en cómo se comportará la cuadrícula con el exterior. El primero de ellos permite que la cuadrícula aparezca encima/debajo del contenido exterior (en bloque) y el segundo de ellos permite que la cuadrícula aparezca a la izquierda/derecha (en línea) del contenido exterior.

Filas y columnas explícitas

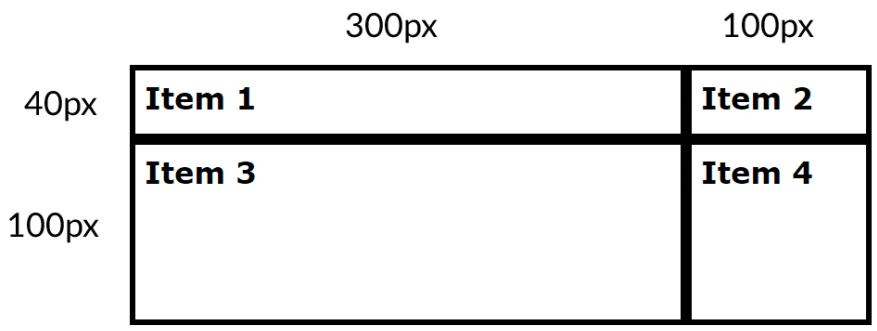
Es posible crear cuadrículas con un tamaño definido. Para ello, sólo tenemos que usar las propiedades CSS *grid-template-columns* y *grid-template-rows*, sirven para indicar las dimensiones de cada celda de la cuadrícula, diferenciando entre columnas y filas.

Propiedad	Descripción
grid-template-columns	Establece el tamaño de las columnas (eje horizontal)
grid-template-rows	Establece el tamaño de las filas (eje vertical)

```
.grid {
  display: grid;
  grid-template-columns: 300px 100px;
```

```
  grid-template-rows: 40px 100px;
}
```

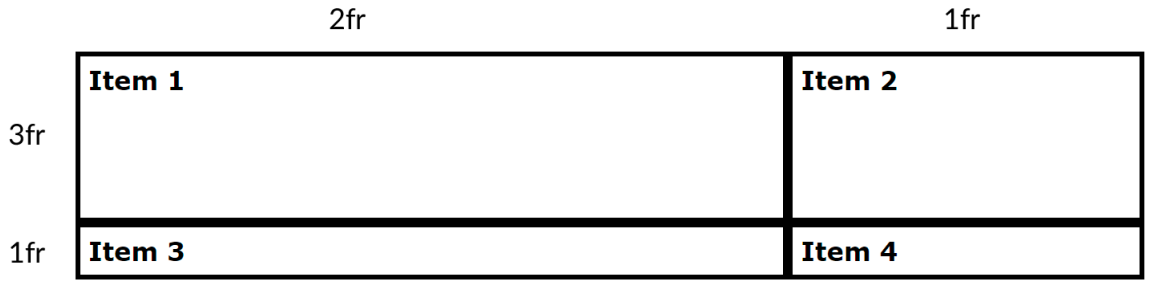
Tendremos una cuadrícula con 2 columnas (la primera con 300px de ancho y la segunda con 100px de ancho) y con 2 filas (la primera con 40px de alto y la segunda con 100px de alto).



Hemos utilizado el pixel como unidades de las celdas de la cuadrícula, sin embargo, también **podemos utilizar otras unidades y combinarlas,** como porcentajes, la palabra clave auto (que obtiene el tamaño restante) o la unidad especial fr (fraction), que simboliza una fracción de espacio restante en el grid.

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

En este nuevo ejemplo se crea una 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):



De esta forma, podemos tener un mejor control del espacio restante de la cuadrícula, y como utilizarlo.

Filas y columnas repetitivas

En las propiedades *grid-template-columns* y *grid-template-rows* podemos indicar expresiones de repetición, indicando celdas que repiten un mismo patrón de celdas varias veces. **repeat(*[núm de veces]*, *[valor o valores]*)**

```
.grid {
  display: grid;
  grid-template-columns: repeat(2, 100px);
  grid-template-rows: repeat(2, 50px 100px);
}
```

En este último ejemplo se crea una cuadrícula de 2x2 donde las columnas van a tener un ancho de 100px. Las filas van adoptar una altura de 50px para la primera y 100 para la segunda.

Que sería lo equivalente a:

```
.grid {
  display: grid;
  grid-template-columns: 100px 100px;
  grid-template-rows: 50px 100px;
}
```

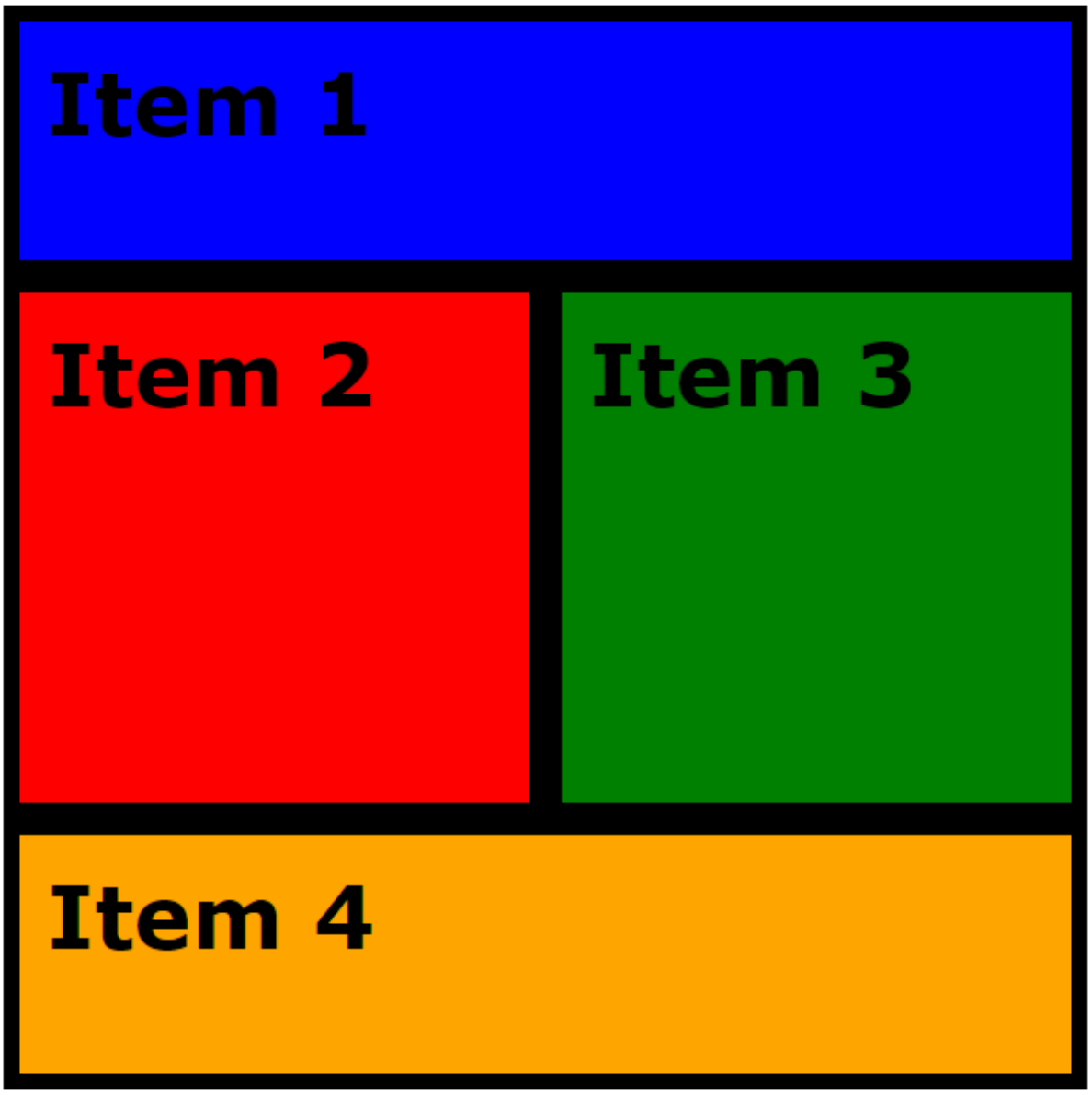
Grid por áreas

Es posible indicar el nombre y la posición concreta de cada área de la cuadrícula. Utilizar la propiedad *grid-template-áreas*, donde debemos especificar el orden de las áreas. Luego, en cada ítem hijo, utilizamos la propiedad *grid-area* para indicar el nombre del área del que se trata

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

```
.grid {
  display:grid;
  grid-template-areas: "head head" "menu main" "foot foot";
}
.a { grid-area:head; background:blue }
.b { grid-area:menu; background:red }
.c { grid-area:main; background:green }
.d { grid-area:foot; background:orange }
```

El **Ítem 1**, la cabecera (head), ocuparía toda la parte superior.
El **Ítem 2**, el menú (menú), ocuparía el área izquierda de la cuadrícula, debajo de la cabecera.
El **Ítem 3,** el contenido (main), ocuparía el área derecha de la cuadrícula, debajo de la cabecera.
El **Ítem 4**, el pie de cuadrícula (footer), ocuparía toda la zona inferior de la cuadrícula.



En la propiedad *grid-template-áreas* también podemos indicar una palabra clave especial:

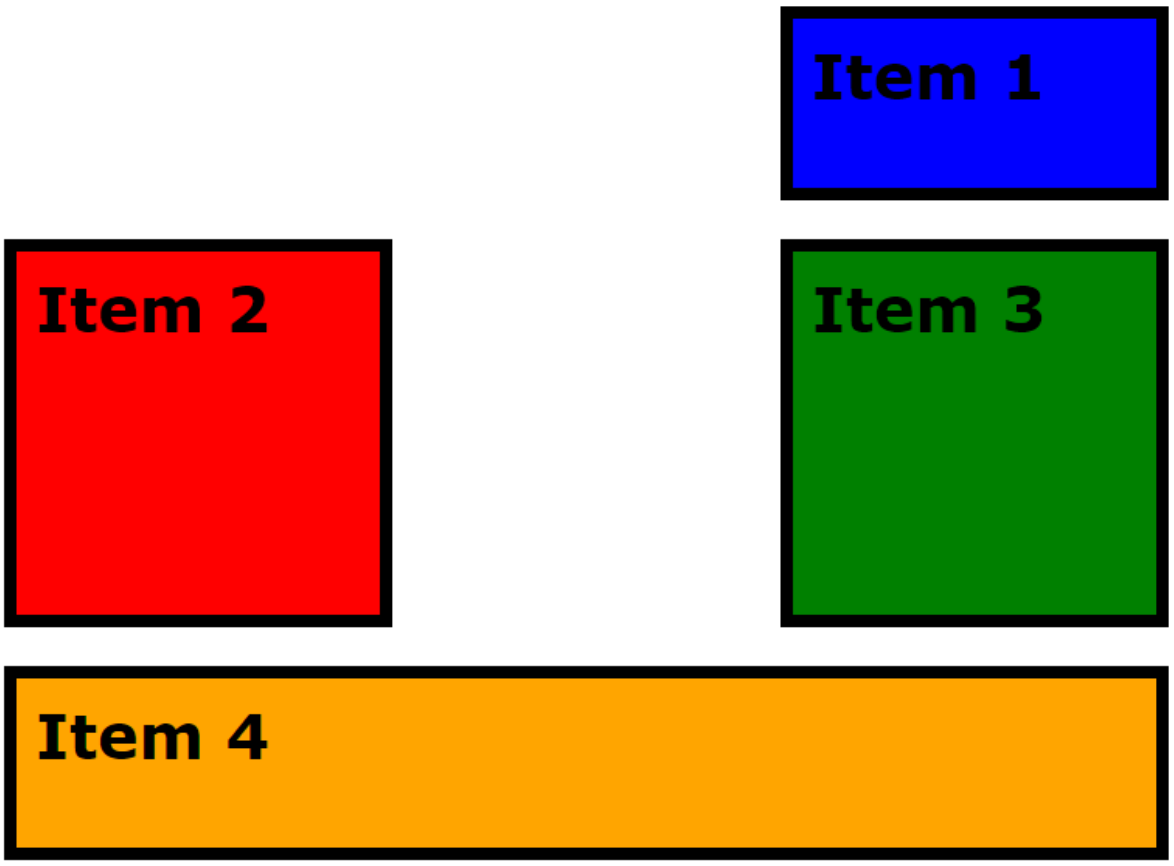
- La palabra clave *none*: Indica que no se colocará ningunacelda en esta posición.
- Uno o más puntos *(.)*: Indica que se colocará una celdavacía en esta posición.

Grid espacios

La cuadrícula tiene todas sus celdas una a continuación de la otra. Aunque sería posible darle un margin 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: 100px;
  grid-row-gap: 10px;
}
```

El resultado sería el siguiente, indicando espacios entre columnas de 100px y espacios entre filas de 10px:



Posición de elementos

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

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

Podemos utilizar las propiedades *justify-content* o *align-content* para cambiar la distribución de todo el contenido en su conjunto:

Propiedad	Valores
justify-content	start | end | center | stretch | space-around | space-between | space-evenly
align-content	start | end | center | stretch | space-around | space-between | space-evenly

Items propiedades

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

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

Existen algunas propiedades más, referentes en este caso, a la posición de los hijos de la cuadrícula donde va a comenzar o terminar una fila o columna. Las propiedades son las siguientes:

Propiedad	Descripción
grid-column-start	Indica en qué columna empezará el ítem de la cuadrícula.
grid-column-end	Indica en qué columna terminará el ítem de la cuadrícula.
grid-row-start	Indica en qué fila empezará el ítem de la cuadrícula.
grid-row-end	Indica en qué fila terminará el ítem de la cuadrícula.

Podemos indicar en el siguiente código CSS sobre el primer ítem de una cuadrícula de 4 ítems:

```
.grid {
  display:grid;
}
.a {
  grid-column-start: 1;
  grid-row-end: 2;
}
```

La utilización de Grids está bajo borrador y su utilización se desaconseja hasta que la especificación se cierre por completo y los navegadores comienzan a soportarlo.
El soporte es el siguiente:

Grids Mobile First

1 - Estructura HTML

```
<div class="container">
 <div class="box-1">
  Header
 </div>
 <div class="box-2">
  Nav
 </div>
 <div class="box-3">
  Main
 </div>
 <div class="box-4">
  Aside
 </div>
 <div class="box-5">
  Footer
 </div>
</div>
```

2 - Lo primero es asignarle a nuestro contenedor la propiedad de *display: grid;*

```
.container {
   display: grid;
}
```

3 - Luego número de columnas y filas que tendrá nuestra grilla, y un espacio de separación.

```
.container {
    display: grid;
    grid-template-columns: 100px auto 100px;
    grid-template-rows: repeat(6, 100px);
    grid-gap:1rem;
}
```

4- Definir el área que ocupará cada caja de nuestro contenedor, primero le asignaremos un nombre y un color característico.

```
.box-1 {
grid-area: box1;
background-color:red
}
.box-2 {
grid-area: box2;
background-color:violet
}
.box-3 {
grid-area: box3;
background-color:green
}
.box-4 {
grid-area: box4;
background-color:yellow
}
.box-5 {
grid-area: box5;
background-color:lightgray;
```

```
}
```

5 - Definir cómo queremos que cada área sea acomodada en nuestro layout

```
.container {
   display: grid;
   grid-template-columns: 100px auto 100px;
   grid-template-rows: repeat(6, 100px);
   grid-gap:1rem;
   grid-template-areas: "box1 box1 box1"  "box2 box3 box4" ". box5 .";
}
```

Con esto terminamos la versión móvil de nuestro proyecto. Ahora sigamos con la versión tablet.

1 - Para la versión tablet lo primero que hacemos es cambiar la disposición de las columnas de nuestro Grid

```
@media only screen and (min-width: 768px) {
        .container {
                grid-template-columns: repeat(4, 1fr);
        }
}
```

2- Luego cambiar 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) {
  .container {
     grid-template-columns: repeat(4, 1fr);
  }
  .box-2 {
    grid-row: 1 / 3;
```

```
    grid-column: 1 / 3;
  }
    .box-1 {
    grid-row: 1 / 2;
    grid-column: 3 / 5;
  }
    .box-3 {
    grid-column: 3 / 5;
    grid-row: 2 / 3;
  }

  .box-4 {
    grid-column: 1 / 2;
    grid-row: 3 / 4;
  }

  .box-5 {
    grid-column: 2 / 4;
    grid-row: 3 / 4;
  }
}
```

Con esto terminamos la versión para tablet de nuestro proyecto. Ahora sigamos con la versión para Desktop.

1- Cambiamos la disposición de la grilla

```
@media only screen and (min-width: 1024px) {
  .container {
      grid-template-columns: repeat(3, 1fr);
  }
```

```
}
```

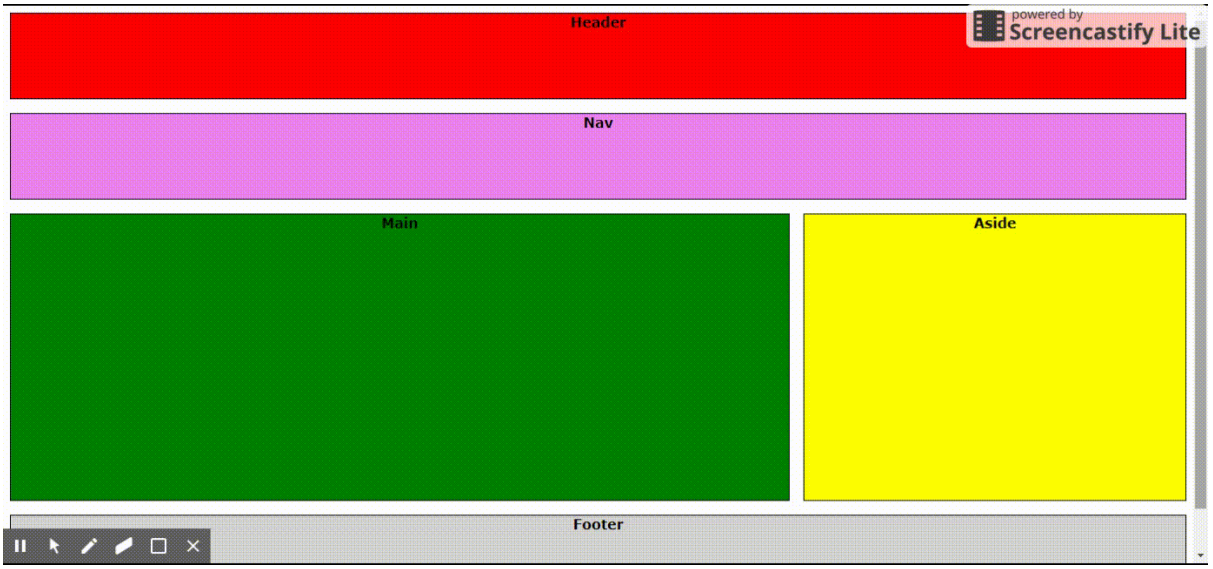
2- Y ahora una vez más cambiamos la disposición de los ítems

```
@media only screen and (min-width: 1024px) {
  .container {
     grid-template-columns: repeat(3, 1fr);
  }
  .box-1 {
    grid-column: 1 / span 3;
    grid-row: 1 / 2;
  }
  .box-2 {
    grid-column: 1 / span 3;
    grid-row: 2 / 3;
  }
    .box-3 {
    grid-column: 1 / 3;
    grid-row: 3 / 6;
  }

  .box-4 {
    grid-column: 3 / 4;
    grid-row: 3 / 6;
  }

  .box-5 {
    grid-column: 1 / 4;
    grid-row: 6 / 7;
  }
}
```

Y como resultado deberíamos de obtener algo como esto:

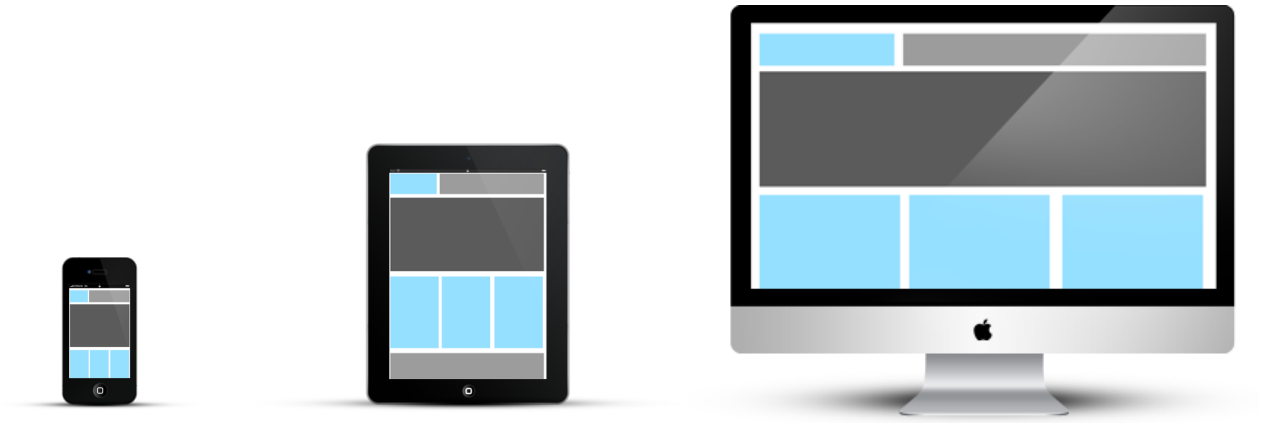


Enlaces de Interés

http://cssgridgarden.com/#es
https://gridbyexample.com/examples/
https://github.com/rachelandrew/gridbugs
https://caniuse.com/#feat=css-grid
https://mozilladevelopers.github.io/playground/css-grid
https://css-tricks.com/getting-started-css-grid/
https://escss.blogspot.com.ar/2015/12/guia-css-grid-layout.html
https://neliosoftware.com/es/blog/css-grid-futuro-diseno-web-ya-esta-aqui/
https://css-tricks.com/video-screencasts/153-getting-started-with-css-grid/

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.



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 dicen al navegador qué reglas debe ignorar o aplicar dependiendo del dispositivo del usuario.**

Los Media Queries **nos permiten presentar el mismo contenido HTML con diseños basados en CSS distintos**. Por lo tanto, en lugar de mantener un sitio web para teléfonos inteligentes y un sitio totalmente independiente para computadoras portátiles o de escritorio, podemos usar el mismo HTML (y servidor web) para ambos. Esto significa que cada vez que agreguemos un nuevo artículo o editemos un error tipográfico en nuestro HTML, esos cambios se reflejarán automáticamente en los diseños tanto móviles como de pantalla ancha. Esta es también una de las razones por la que el contenido siempre tiene que ir separado del estilo. **Nada que se pueda hacer con CSS debe estar hecho por HTML.**

Veremos que básicamente los Media Queries no son más que un envoltorio alrededor de nuestro código CSS.

Comenzaremos de forma pequeña simplemente actualizando el color de fondo en el **<body>** en función del ancho del dispositivo. Esta es una buena manera de asegurarse de que nuestras Media Queries realmente estén funcionando antes de entrar en diseños complicados.

Diferenciamos entre diseños estrechos, medios y anchos agregando lo siguiente:

```
/*  Mobile Styles */
@media only screen and (max-width: 400px) {
  body {
    background-color: #F09A9D; /*  Red */
  }
}
/* Tablet Styles */
@media only screen and (min-width: 401px) and (max-width: 960px) {
  body {
    background-color: #F5CF8E; /* Yellow */
  }
}
/*  Desktop Styles */
@media only screen and (min-width: 961px) {
  body {
    background-color: #B2D6FF; /*  Blue */
  }
}
```

Cuando cambia el tamaño del navegador, debería ver tres colores de fondo diferentes: azul cuando es mayor que *960px* de ancho, amarillo cuando está entre *401px* y *960px*, y rojo cuando es menor que *400px*.

Los Media Queries siempre comienzan con *@media* seguida de algún tipo de instrucción condicional, y luego las llaves. Dentro de las llaves, se insertan un montón de

reglas de CSS comunes. **El navegador solo presta atención a esas reglas si se cumple la condición.**

Mas información acerca de Media Queries:

- https://www.w3.org/TR/css3-mediaqueries/
- https://mediaqueri.es/
- https://caniuse.com/#search=css3%20media
- https://css-tricks.com/snippets/css/media-queries-for-standard-devices/
- https://css-tricks.com/css-media-queries/

Breakpoints

Ethan Marcotte en su libro *Responsive Design* (*http://www.reprosol.be/sites/reprosol.beta.the-aim.be/files/responsive-webdesign(ethan-marcotte).pdf*) nos propone la siguiente lista de puntos de interrupción según la resolución del dispositivo:

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

Importante: Convierte tus Breakpoints a EMs

Aunque los pixeles se consideran una unidad de medida absoluta, en realidad son unidades relativas a la resolución de pantalla del dispositivo que lo visualiza. Si dicho dispositivo tiene una densidad mayor a la normal (densidad de pixeles), entonces la proporción de los pixeles cambiará. Por ello **es importante que los breakpoints de los Media Queries se conviertan a EM,**que sí son unidades relativas y proporcionales.

A continuación un par de enlaces para que amplies acerca del tema:

- https://zellwk.com/blog/media-query-units/
- https://cloudfour.com/thinks/the-ems-have-it-proportional-media-queries-ftw/
- https://css-tricks.com/why-ems/

Meta viewport

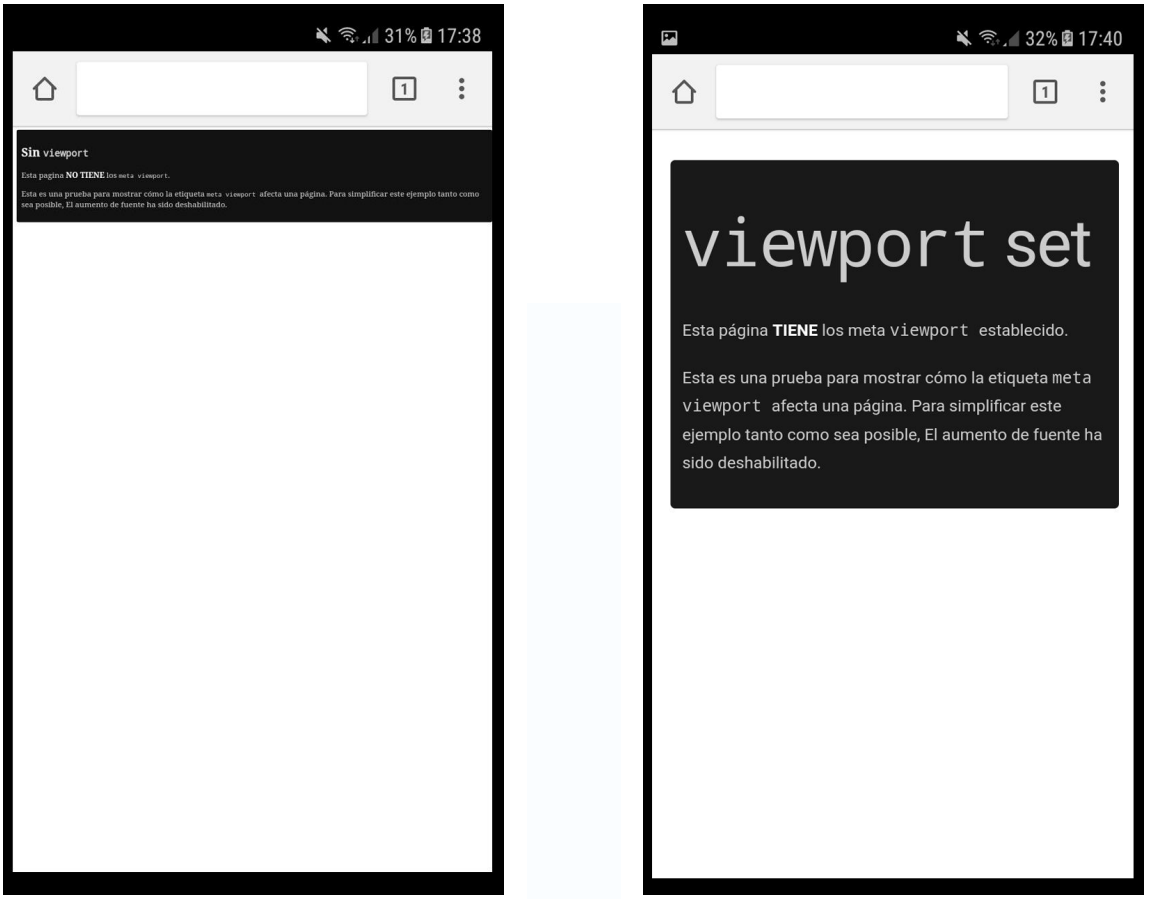
Las páginas optimizadas para diferentes dispositivos deben incluir la etiqueta **<meta>** *viewport* en el encabezado del documento HTML. Una etiqueta **<meta>** *viewport* da al navegador las instrucciones sobre cómo controlar las dimensiones y el ajuste a escala de la página.

- 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.

Para ofrecer la mejor experiencia posible, los navegadores de dispositivos móviles muestran la página con el ancho de una pantalla de escritorio (por lo general, alrededor de 980px, aunque esto varía entre dispositivos) y luego se intenta mejorar el aspecto del contenido aumentando los tamaños de las fuentes y modificando la escala del contenido para que se ajuste a la pantalla. Esto significa que los tamaños de fuente pueden parecer inconsistentes para los usuarios, quienes tal vez tengan que presionar dos veces o pellizcar para hacer zoom e interactuar con el contenido.

```
<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.



La imagen izquierda superior nos muestra una prueba de un sitio web sin los **<meta>** *viewport*, la imagen de al lado es otra prueba pero esta vez nos muestra como la etiqueta **<meta>** *viewport* afecta una página en un dispositivo mobile.

Aplicando Mobile First

"Mobile First" significa **crear el código primero para los dispositivos más pequeños** que los usuarios probablemente tengan, como teléfonos o tabletas. Trabajar en el dispositivo que es el más pequeño, y luego acumular desde allí, todo en el mismo código y el mismo proyecto, y no un nuevo proyecto para cada tamaño de pantalla. Por lo tanto primero trabajar el código para que se reproduzca perfectamente en un teléfono y luego ajustar para que se ejecute en una tableta y luego en un dispositivo de escritorio.

Cualquier estilo dentro del siguiente Media Querie se ejecutará tan pronto como el tamaño de la pantalla sea de 768 px 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;
  }
}
```

Para ajustar la pantalla de nuevo cuando el usuario tiene el iPad Mini en orientación horizontal, simplemente agregar otro Media Querie en su CSS para 1024px de ancho (el ancho de un iPad Mini en horizontal):

```
@media only screen and (min-width: 1024px) {
  body {
    background-color: #FFFFFF;
  }
}
```

Esto es sencillamente un acercamiento al paradigma, si quieres profundizar aún más te recomendamos los siguientes artículos:

- http://www.headfirstlabs.com/books/hf-mw/hfmw_ch2.pdf
- http://gilbaneboston.com/11/presentations/Tom-Wentworth-mobilefirst.pdf
- https://carlosazaustre.es/es-tu-web-realmente-mobile-first/
- https://www.interaction-design.org/literature/article/mobile-first-and-the-power-of-mobile-computing
- https://mayvendev.com/blog/mobilefirst

- https://searchengineland.com/designing-content-mobile-first-index-280071
- http://fredericgonzalo.com/en/2017/03/01/understanding-the-difference-between-mobile-first-adaptive-and-responsive-design/
- https://blog.prototypr.io/mobile-first-desktop-worst-f900909ae9e2
- https://blog.intercom.com/why-mobile-first-may-already-be-outdated/
- https://www.entrepreneur.com/article/285444
- http://www.uxbooth.com/articles/where-do-we-go-from-mobile-first/

- **Diferencias entre Flexbox y Grillas**
- Aprendé jugando
- Ejemplos de grid (por Rachel Andrew)
- Bugs de Grid
- Soporte navegadores
- Introduction To Css Grid Layout
- Getting started css grid
- Guia css grid layout
- CSS grid futuro diseno web ya está aquí
- Getting started with css grid
- Complete guide to Grid