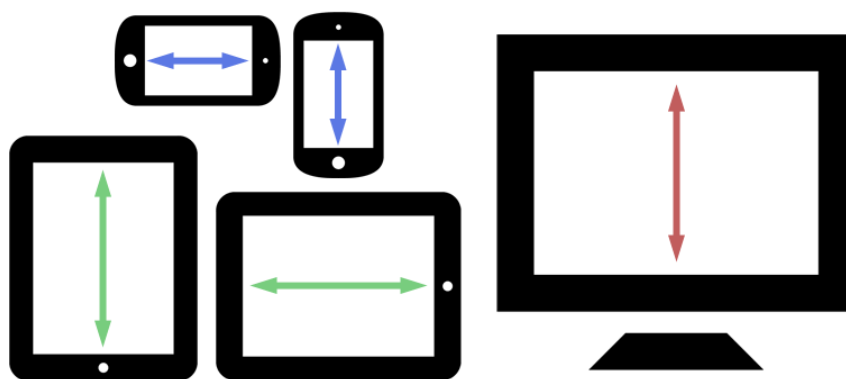


## Responsive Web Design

Fuente: [lenguajecss.com](http://lenguajecss.com)

En la actualidad, el uso de todo tipo de **dispositivos móviles** se ha disparado, no sólo de «smartphones», sino también de tablets, «smartwatches», lectores de ebooks y múltiples tipos de dispositivos con capacidad de conexión a Internet.

Cada vez es más frecuente acceder a Internet con diferentes tipos de dispositivos, que a su vez tienen **diferentes pantallas y resoluciones**, con distintos tamaños y formas, que hacen que se consuman las páginas webs de formas diferentes, apareciendo por el camino también diferentes necesidades, problemas y soluciones.



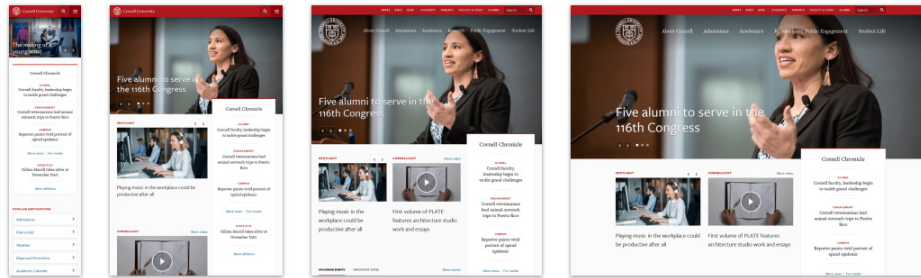
Por lo tanto, en la actualidad, cuando diseñamos una web, esta debe estar preparada para verse correctamente en diferentes resoluciones, cosa que, a priori no es sencilla.

Antiguamente, se llegó al punto de preparar una web diferente dependiendo del dispositivo o navegador que utilizaba el usuario, pero era algo que se terminó descartando, ya que no era práctico.

Por suerte, esos tiempos han quedado atrás, y la máxima que se sigue hoy es **diseñar una sola web, que se adapte visualmente al dispositivo utilizado**.

Hoy en día se le denomina **Responsive Web Design** (o *RWD*) a los diseños web que tienen la capacidad de adaptarse al tamaño y formato de la pantalla en la que se visualiza el contenido, respecto a los diseños tradicionales en los que las páginas web estaban diseñadas sólo para un tamaño o formato específico, y no tenían esa capacidad de adaptación.

Cornell University

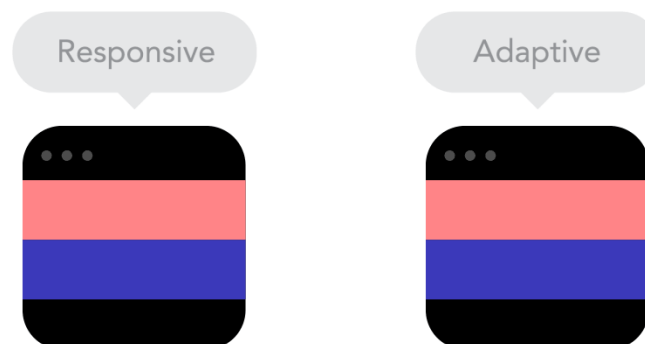


Aunque en principio el concepto de **web adaptativa** es muy sencillo de comprender, aplicarlo puede ser todo un quebradero de cabeza si no se conocen bien las bases y se adquiere experiencia. En [MediaQueri.es](https://mediaqueri.es) puedes encontrar algunos ejemplos de páginas que utilizan Responsive Web Design para tener clara la idea.

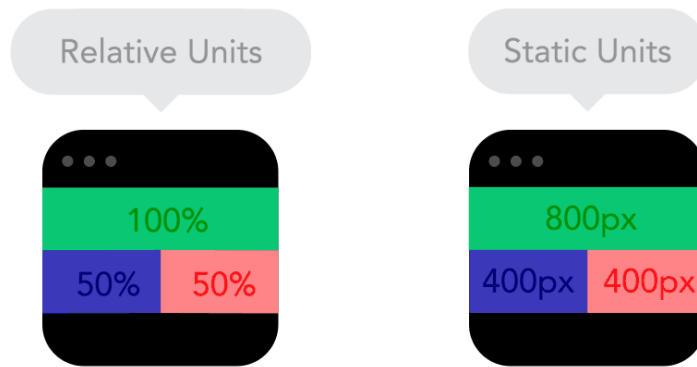
## Conceptos básicos

En el excelente artículo [9 basic principles of responsive web design](#), de Froont, hay una estupenda explicación visual de algunos conceptos básicos necesarios para entender correctamente el **Responsive Web Design**. Son los siguientes:

El primero de ellos es la diferencia entre **diseño responsivo** y **diseño adaptativo**. Como se puede ver en la imagen a continuación, un diseño **responsive** responde (*valga la redundancia*) en todo momento a las dimensiones del dispositivo, mientras que un diseño adaptable es aquel que se adapta, pero no necesariamente responde en todo momento:

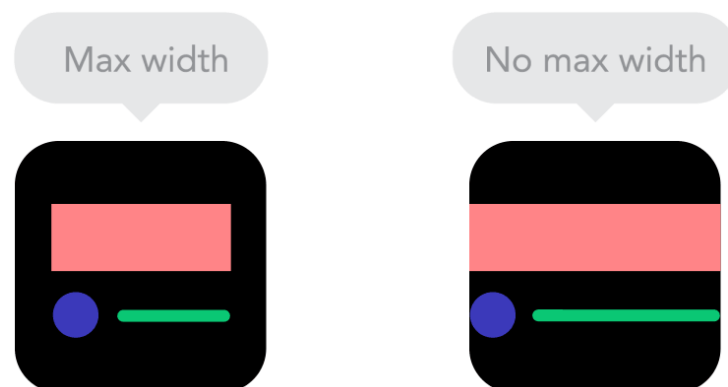


Por otro lado, para trabajar correctamente en diseños **responsive** hay que tener en cuenta que debemos trabajar con unidades relativas e intentar evitar las unidades fijas o estáticas, las cuales no responden a la adaptación de nuestros diseños flexibles:



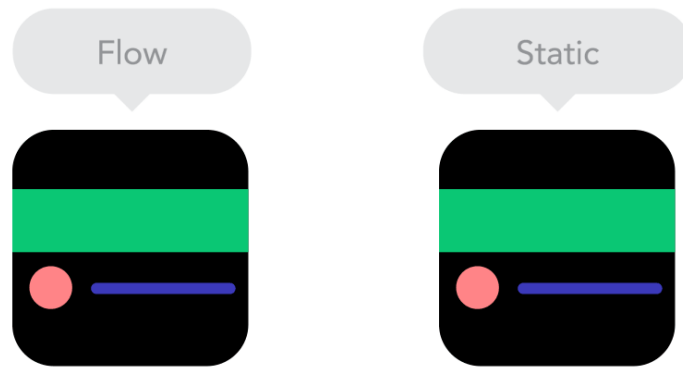
Otra forma interesante de trabajar esa respuesta de los diseños **responsive** es utilizar propiedades como **min-width** o **max-width**, donde definimos tamaños mínimos o máximos, para que los elementos de nuestra página puedan ampliarse o reducirse según sea necesario dependiendo de la pantalla del dispositivo utilizado.

Con estas propiedades podemos crear diseños que aprovechen al máximo toda la pantalla de dispositivos pequeños (*como móviles o tablets*), mientras que establecemos unos máximos en pantallas de dispositivos grandes, para crear unos espacios visuales que hacen que el diseño sea más agradable:

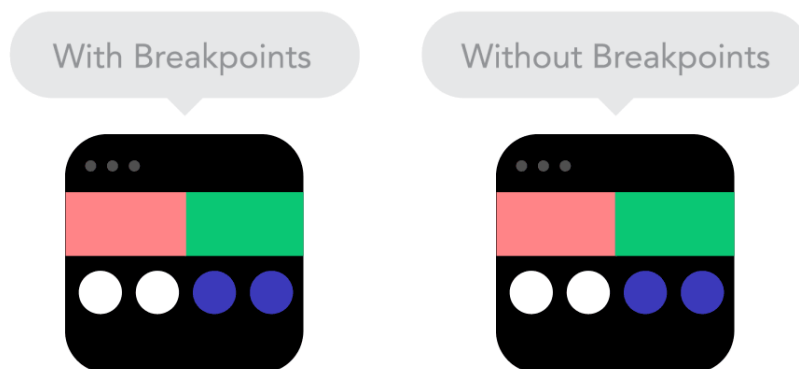


Otro concepto, que a la misma vez es una característica muy atractiva en nuestros diseños responsive es la de mantener el flujo de los elementos cuando cambian de tamaño y evitar que estos se solapen unos con otros.

Si estamos habituados a trabajar en diseños más estáticos que no están preparados para móviles, suele ser duro hacer ese cambio. Sin embargo, una vez lo conseguimos, todo resulta mucho más fácil y conseguiremos transmitir una buena respuesta y fluidez visual:



Esto último va muy de la mano del sistema habitual de recolocación de elementos que se suele seguir en los diseños **Responsive Design**. Como se puede ver en la siguiente imagen, en un diseño responsive se utilizan ciertos «puntos de control». Por ejemplo, se suele pensar que en una resolución de escritorio queremos mostrar la información dentro de una cuadrícula (*grid*) de 4 ó 5 celdas de ancho, mientras que en la versión de tablet será sólo de 3 celdas de ancho (*el resto se desplazará a la siguiente fila*) y en móviles será una sola celda de ancho, mostrándose el resto de celdas haciendo scroll hacia abajo:



Esta forma de trabajar nos proporciona múltiples ventajas:

- Es mucho más sencillo mostrar la misma información desde diseños de pantalla grande.
- Ayuda a evitar la mala práctica de ocultar bloques de información en dispositivos móviles.
- Incentiva a diseñar siguiendo buenas prácticas para facilitar la creación responsive.

## Preparación previa

Antes de comenzar a crear un diseño web preparado para móviles, es importante tener claro ciertos detalles:

- A priori, ¿Cuál es tu público objetivo? ¿móvil o escritorio? ¿ambos?
- Debes conocer las resoluciones más utilizadas por tu público potencial
- Debes elegir una estrategia acorde a los datos anteriores

En primer lugar, es importante **conocer los formatos** de pantalla más comunes con los cuales nos vamos a encontrar. Podemos consultar páginas como [MyDevices](#), la cual tiene un apartado de [comparación de dispositivos](#), donde se nos muestra un listado de dispositivos categorizados en smartphones, tablets u otros dispositivos con las características de cada uno: dimensiones de ancho, alto, radio de píxels, etc...

## Estrategias de diseño

Por último, es aconsejable decidirse por una estrategia de diseño antes de comenzar. Aunque existen otras estrategias, las dos vertientes principales más populares son las siguientes:

Estrategia	Descripción
<b>Mobile first</b>	Primero nos enfocamos en dispositivos móviles y luego pensamos en otros.
<b>Desktop first</b>	Primero nos enfocamos en dispositivos de escritorio, y luego pensamos en otros.

La estrategia **Mobile-first** es la que utilizan los diseñadores de sitios webs en las que su público objetivo es mayoritariamente usuario de móvil. Ejemplos como una web para comprar billetes de transporte, la web de un juego o aplicación móvil o una web para pedir cita en un restaurante podrían ser, a priori, una buena elección para utilizar **Mobile-first**.

Esta estrategia hace que el desarrollo en escritorio sea muy sencillo, ya que se reduce a tener un diseño de móvil en escritorio e ir añadiendo nuevas secciones o partes para «completar» el diseño en resoluciones grandes.

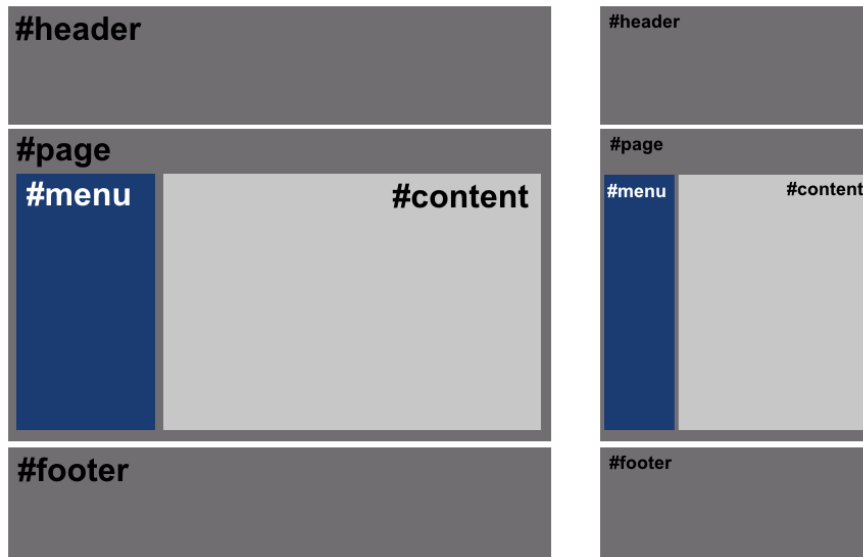
Por otro lado, la estrategia **Desktop-first** suele interesar más a los diseñadores de sitios webs en las que el público objetivo son usuarios de escritorio. Por ejemplo, una página de una aplicación para PC/Mac o similares, podría ser una buena opción para la estrategia **Desktop-first**. En ella, hacemos justo lo contrario que en la anterior, lo primero que diseñamos es la versión de escritorio, y luego vamos descargando detalles o colocando información hasta tener la versión para dispositivos móviles.

Como explicamos en el capítulo anterior, hay ciertos conceptos que hay que tener claros antes de comenzar con el **Responsive Design**. En esta sección vamos a ver cómo llevarlos a la práctica con código.

## Diseño con porcentajes

El primer paso para crear un diseño que se adapte correctamente, es comenzar a familiarizarse con un tipo de unidades relativas: **los porcentajes**. Recordemos que los porcentajes son relativos al contenedor padre, por lo que si especificamos un porcentaje a un elemento, el navegador va a tomar dicho porcentaje del contenedor.

Podemos comenzar usando porcentajes con las propiedades **width** en un ejemplo sencillo. Si establecemos un ancho de **100%** (*valor por defecto en elementos de tipo block, no hace falta indicarlo*) a los elementos grises que vemos a continuación (**#header**, **#page** y **#footer**), un **30%** al azul (**#menu**) y un **70%** al gris claro (**#content**) podríamos obtener este diseño:



El código utilizado sería algo parecido a lo siguiente:

```
<div id="header"></div>
<div id="page">
  <div id="menu"></div>
  <div id="content"></div>
</div>
<div id="footer"></div>
```

Nótese que los elementos **#menu** y **#content** se encuentran dentro de **#page**. Tengan en cuenta que, estamos utilizando **id** en este ejemplo, aunque habíamos comentado que lo ideal quizás sería utilizar clases.

Por su parte, el código CSS tendría esta pinta:

```
div {
  /* Ponemos un alto mínimo, por defecto es 0 */
  min-height: 200px;
  /* Dibujamos un borde para ver límites */
  border: 2px solid black;
}

#header, #page, #footer {
  background: grey;
}

#menu, #content {
  /* Necesario para que los elementos estén en horizontal */
```

```

display: inline-block;
}

#menu {
  background: blue;
  width: 30%;
}

#content {
  background: lightgrey;
  width: 70%;
}

```

Sin embargo, utilizar porcentajes no nos garantiza un diseño adaptativo de calidad, hay que comprender otros detalles. El primer problema que encontraremos será que si sumamos el tamaño de los elementos ( $70\% + 30\%$ ) junto a los bordes (*2px por cada lado*), la suma es superior al 100% del contenedor padre, por lo que **no cabe en su interior** y el segundo elemento se desplaza a la zona inferior, descuadrando todo el diseño. Lo mismo puede ocurrir si intentamos añadir **margin** o **padding**. Esto es algo muy habitual en CSS. Y frustrante al principio.

Hay varias formas de solucionar esto:

- Eliminar los bordes y reducir los porcentajes hasta que quepan en el 100% del padre.
- Usar **box-sizing: border-box** para cambiar el modo en el que se gestionan los tamaños.
- Utilizar un sistema moderno como **Flexbox** o **Grid** (*recomendado*).

Una forma simple de solucionar el problema en el ejemplo anterior, es hacer los siguientes cambios en el CSS del documento:

```

/* Eliminamos este bloque */
#menu, #content {
  display: inline-block;
}

/* Añadimos este */
#page {
  display: flex;
}

```

De esta forma, conseguimos que nuestro diseño se adapte de forma adecuada a la página, sin necesidad de tener que ajustar los márgenes, rellenos, bordes o tamaño de los contenidos.



## Tamaños máximos y mínimos

Si buscamos un cierto grado de control aún mayor, podríamos recurrir a las propiedades **max-width** y **min-width** , con las que podemos indicar el ancho de un elemento como máximo y el ancho de un elemento como mínimo respectivamente, consiguiendo así garantizar cierto control del diseño:

```
.picture {  
  min-height: 200px; /* Por defecto, height es 0 */  
  background: grey; /* Simplemente, para verlo visualmente */  
  
  max-width: 1024px;  
  min-width: 800px;  
}
```

En este caso, el elemento tiene un tamaño máximo de 1024 píxeles, y un tamaño mínimo de 800 píxeles, por lo que si ajustamos el ancho de la ventana del navegador, dicho elemento iría variando en un rango de 800 a 1024 píxeles, nunca haciéndose más pequeño de 800 o más grande de 1024.

Con las imágenes, videos y contenidos multimedia, se puede hacer lo mismo, consiguiendo así que las imágenes se escalen y adapten al formato especificado o incluso al tamaño de pantalla de los diferentes dispositivos utilizados:

```
img,  
video,  
object,  
embed {  
  max-width: 100%;  
  height: auto;  
}
```

## El viewport

En muchos casos puede que oigas hablar del **viewport** del navegador. Esa palabra hace referencia a la **región visible del navegador**, o sea, la parte de la página que está visualizándose actualmente en el navegador. Los usuarios podemos redimensionar la ventana del navegador para reducir el tamaño del viewport y simular que se trata de una pantalla y dispositivo más pequeño.

Si queremos editar ciertos comportamientos del viewport del navegador, podemos editar el documento HTML para especificar el siguiente campo meta, antes de la parte del **</head>**:

```
<meta name="viewport" content="initial-scale=1, width=device-width">
```

Con esta etiqueta **<meta>**, estamos estableciendo unos parámetros de comportamiento para el **viewport** del navegador. Veamos que significan y cuales más existen:

Propiedades	Valor	Significado
width	device-width	Indica un ancho para el viewport.
height	device-height	Indica un alto para el viewport.
initial-scale	1	Escala inicial con la que se visualiza la página web.
minimum-scale	0.1	Escala mínima a la que se puede reducir al hacer zoom.
maximum-scale	10	Escala máxima a la que se puede aumentar al hacer zoom.
user-scalable	no/fixed   <b>yes/zoom</b>	Posibilidad de hacer zoom en la página web.

Las propiedades **initial-scale** , **minimum-scale** y **maximum-scale** permiten valores desde el **0.1** al **10** , aunque ciertos valores se traducen automáticamente a ciertos números determinados:

- yes = 1
- no = 0.1
- device-width = 10
- device-height = 10

Por otra parte, **user-scalable** permite definir si es posible que el usuario pueda «pellizcar» la pantalla para ampliar o reducir el zoom.

Una vez nos adentramos en el mundo del **Responsive Design**, nos damos cuenta de que hay situaciones en las que determinados aspectos o componentes visuales deben aparecer en un tipo de dispositivos, o deben existir ciertas diferencias.

Por ejemplo, una zona donde se encuentra el buscador de la página puede estar colocada en un sitio concreto en la versión de escritorio, pero en móvil quizás nos interesa que ocupe otra zona (*o que tenga otro tamaño o forma*) para aprovechar mejor el poco espacio que tenemos en la versión móvil de la página.

Para ello, utilizaremos un concepto denominado **media queries**, con los que podemos hacer esas excepciones para que sólo se apliquen a un tipo de diseño concreto.

### ¿Qué son las media queries?

Las reglas **media queries** (*también denominadas **MQ** a veces*) son un tipo de reglas de CSS que permiten crear un bloque de código que sólo se procesa en los dispositivos que cumplan los criterios especificados como condición:

```
@media screen and (*condición*) {  
  /* reglas CSS */  
  /* reglas CSS */  
}
```

```
@media screen and not (*condición*) {  
  /* reglas CSS */  
  /* reglas CSS */  
}
```

Con este método, especificamos que queremos aplicar los estilos CSS para tipos de medios concretos (**screen**: *sólo en pantallas, en este caso*) que cumplan las condiciones especificadas entre paréntesis. De esta forma, una estrategia aconsejable es crear reglas CSS generales (*como hemos hecho hasta ahora*) aplicadas a todo el documento: colores, tipo de fuente, etc. y luego, las particularidades que se aplicarían sólo en el dispositivo en cuestión.

Aunque suele ser menos habitual, también se pueden indicar reglas **@media** negadas mediante la palabra clave **not**, que aplicará CSS siempre y cuando no se cumpla una determinada condición. También pueden separarse por comas varias condiciones de media queries.

Existen los siguientes **tipos de medios**:

Tipo de medio	Significado
screen	Monitores o pantallas de ordenador. Es el más común.
print	Documentos de <b>medios impresos</b> o pantallas de previsualización de impresión.
speech	Lectores de texto para invidentes (Antes <b>aural</b> , el cuál ya está obsoleto).
all	Todos los dispositivos o medios. El que se utiliza <b>por defecto</b> .

Otros tipos de medios como **braille**, **embossed**, **handheld**, **projection**, **tty** o **tv** aún son válidos, pero están marcados como obsoletos a favor de utilizar tipos de medios de la lista anterior y restringir sus características posteriormente.

Recordemos que con el siguiente fragmento de código HTML estamos indicando que el nuevo ancho de la pantalla es **el ancho del dispositivo**, por lo que el aspecto del viewport se va a adaptar consecuentemente:

```
<meta name="viewport" content="initial-scale=1, width=device-width">
```



Con esto conseguiremos preparar nuestra web para dispositivos móviles y prepararnos para la introducción de reglas **media query** en el documento CSS.

## Ejemplos de media queries

Veamos un ejemplo clásico de **media queries** en el que definimos diferentes estilos dependiendo del dispositivo que estamos utilizando. Obsérvese que en el código existen 3 bloques **@media** donde se definen estilos CSS para cada uno de esos tipos de dispositivos.

El código sería el siguiente:

```
@media screen and (max-width: 640px) {  
  .menu {  
    background: blue;  
  }  
}  
  
@media screen and (min-width: 640px) and (max-width: 1280px) {  
  .menu {  
    background: red;  
  }  
}  
  
@media screen and (min-width: 1280px) {  
  .menu {  
    background: green;  
  }  
}
```

El ejemplo anterior muestra un elemento (*con clase [menu](#)*) con un color de fondo concreto, dependiendo del tipo de medio con el que se visualice la página:

- **Azul** para resoluciones menores a **640 píxeles** de ancho (*móviles*).
- **Rojo** para resoluciones entre **640 píxeles** y **1280 píxeles** de ancho (*tablets*).
- **Verde** para resoluciones mayores a **1280 píxeles** (*desktop*).

El número de bloques de reglas [@media](#) que se utilicen depende del desarrollador web, ya que no es obligatorio utilizar un número concreto. Se pueden utilizar desde un sólo media query, hasta múltiples de ellos a lo largo de todo el documento CSS.

Hay que tener en cuenta que los **media queries** también es posible indicarlos desde HTML, utilizando la etiqueta [<link>](#):

```
<link rel="stylesheet" href="mobile.css"  
      media="screen and (max-width: 640px)">  
  
<link rel="stylesheet" href="tablet.css"  
      media="screen and (min-width: 640px) and (max-width: 1280px)">  
  
<link rel="stylesheet" href="desktop.css"  
      media="screen and (min-width: 1280px)">
```

Estos estilos quedarán separados en varios archivos diferentes. Ten en cuenta que todos serán descargados al cargar la página, sólo que no serán aplicados al documento hasta que cumplan los requisitos indicados en el atributo **media**.

## Tipos de características

En los ejemplos anteriores solo hemos utilizado **max-width** y **min-width** como tipos de características a utilizar en condiciones de media query. Sin embargo, tenemos una lista de tipos de características que podemos utilizar:

Tipo de característica	Valores	¿Cuándo se aplica?
<b>width</b>		Si el dispositivo tiene el tamaño indicado exactamente.
<b>min-width</b>		Si el dispositivo tiene un tamaño de ancho mayor al indicado.
<b>max-width</b>		Si el dispositivo tiene un tamaño de ancho menor al indicado.

<b>aspect-ratio</b>	<u>aspect-ratio</u>	Si el dispositivo encaja con la proporción de aspecto indicada.
<b>orientation</b>	landscape   portrait	Si el dispositivo está colocado en modo vertical o apaisado.

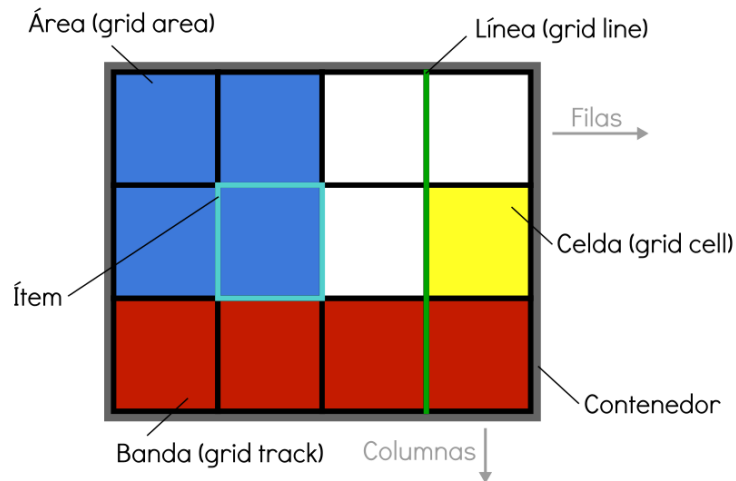
Existen otras características minoritarias que en algunos casos límite pueden ser interesantes, como por ejemplo **scan**, **resolution**, **monochrome**, **grid**, **color-index**, **color**, etc...

## CSS Grid

El sistema **flexbox** es una gran mejora, sin embargo, está orientado a estructuras de una sola dimensión, por lo que aún necesitamos algo más potente para estructuras web más específicas o complejas. Con el paso del tiempo, muchos frameworks y librerías utilizan un **sistema grid** donde definen una cuadrícula determinada, y modificando los nombres de las clases de los elementos HTML, podemos darle tamaño, posición o colocación.

## Conceptos





- **Contenedor:** El elemento padre contenedor que definirá la cuadrícula o rejilla.
- **Ítem:** Cada uno de los hijos que contiene la cuadrícula (*elemento contenedor*).
- **Celda (grid cell):** Cada uno de los cuadritos (*unidad mínima*) de la cuadrícula.
- **Área (grid area):** Región o conjunto de celdas de la cuadrícula.
- **Banda (grid track):** Banda horizontal o vertical de celdas de la cuadrícula.
- **Línea (grid line):** Separador horizontal o vertical de las celdas de la cuadrícula.

Para utilizar cuadrículas **Grid CSS**, trabajaremos bajo el siguiente escenario:

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

Para activar la cuadrícula **grid** hay que utilizar sobre el elemento contenedor la propiedad **display** y especificar el valor **grid** o **inline-grid**. Este valor influye en cómo se comportará la cuadrícula con el contenido 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.

## Tipo de elemento

## Descripción

inline-grid	Establece una cuadrícula con ítems en línea, de forma equivalente a inline-block.
grid	Establece una cuadrícula con ítems en bloque, de forma equivalente a block.

Una vez elegido uno de estos dos valores, y establecida la propiedad **display** al elemento contenedor, hay varias formas de configurar nuestra cuadrícula grid. Comencemos con las propiedades que se aplican al elemento contenedor (*padre*).

### Grid con filas y columnas

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

## Propiedad

## Valor

## Descripción

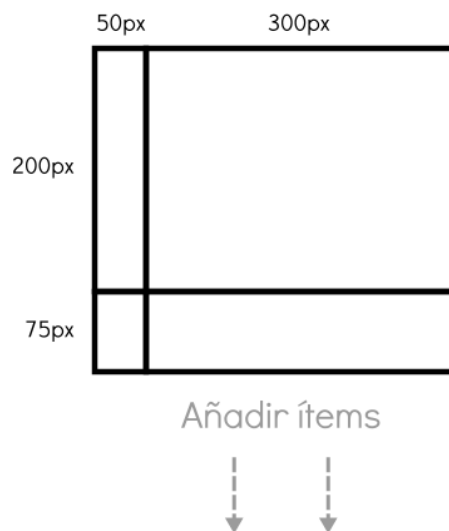
<b>grid-template-columns</b>	<u>[c1]</u> <u>[c2]</u> ...	Establece el ancho de cada columna ( <u>col 1</u> , <u>col 2...</u> ).
------------------------------	-----------------------------	------------------------------------------------------------------------

<b>grid-template-rows</b>	<b>[f1] [f2] ...</b>	Establece el ancho de cada fila ( <u>fila 1</u> , <u>fila 2...</u> ).
---------------------------	----------------------	-----------------------------------------------------------------------

Conociendo estas dos propiedades, asumamos el siguiente código CSS:

```
.grid {
  display: grid;
  grid-template-columns: 50px 300px;
  grid-template-rows: 200px 75px;
}
```

Esto significa que tendremos una cuadrícula con **2 columnas** (la primera con 50px de ancho y la segunda con 300px de ancho) y con **2 filas** (la primera con 200px de alto y la segunda con 75px de alto). Ahora, dependiendo del número de ítems (*elementos hijos*) que tenga el contenedor **grid**, tendremos una cuadrícula de 2x2 elementos (4 *ítems*), 2x3 elementos (6 *ítems*), 2x4 elementos (8 *ítems*) y así sucesivamente. Si el número de ítems es impar, la última celda de la cuadrícula se quedará vacía.



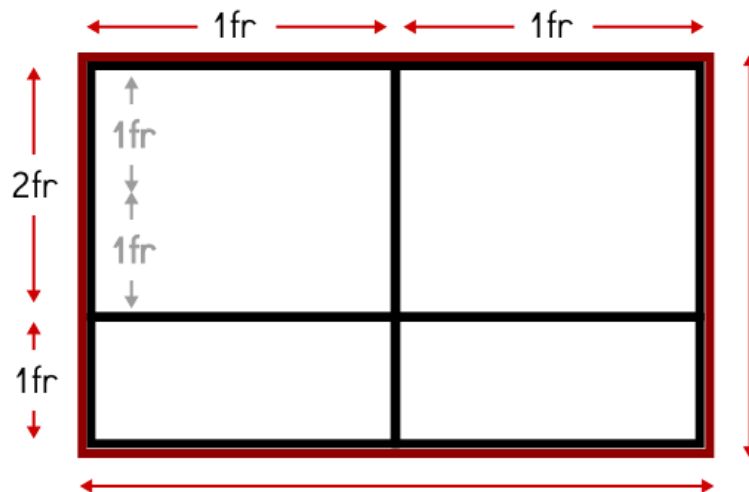
## Unidad fracción restante (fr)

En el ejemplo anterior he utilizado **píxeles** como unidades de las celdas de la cuadrícula, sin embargo, también podemos utilizar otras unidades (*o incluso combinarlas*) como porcentajes, la palabra clave **auto** (*que obtiene el tamaño restante*) o la unidad especial de

Grid CSS **fr** (*fraction*), que simboliza una **fracción de espacio restante en el grid**. Veamos un código de ejemplo en acción:

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

Este nuevo ejemplo, se crea una cuadrícula de 2x2, donde el tamaño de ancho de la cuadrícula se divide en **dos columnas** (*mismo tamaño de ancho para cada una*), y el tamaño de alto de la cuadrícula se divide en **dos filas**, donde la primera ocupará el doble (*2 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 algunos casos, en las propiedades **grid-template-columns** y **grid-template-rows** podemos necesitar indicar las mismas cantidades un número alto de veces, resultando repetitivo y molesto. Se puede utilizar la expresión **repeat()** para indicar repetición de valores, indicando el número de veces que se repiten y el tamaño en cuestión.

La expresión a utilizar sería la siguiente: **repeat([número de veces], [valor o valores])**:

```
.grid {
  display: grid;
  grid-template-columns: 100px repeat(2, 50px) 200px;
  grid-template-rows: repeat(2, 50px 100px);
}
```

Asumiendo que tuviéramos un contenedor grid con 8 ítems hijos (*o más*), el ejemplo anterior crearía una cuadrícula con **4 columnas** (*la primera de 100px de ancho, la segunda y tercera de 50px de ancho y la cuarta de 200px de ancho*). Por otro lado, tendría **2 filas** (*la primera de 50px de alto, y la segunda de 100px de alto*). En el caso de tener más ítems hijos, el patrón se seguiría repitiendo.

El ejemplo anterior sería equivalente al código CSS siguiente:

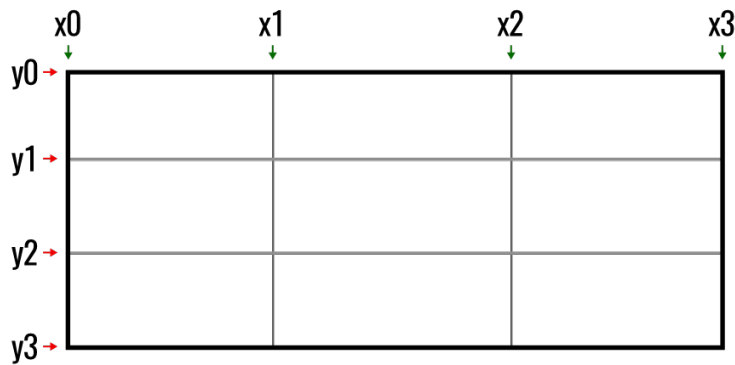
```
.grid {
  display: grid;
  grid-template-columns: 100px 50px 50px 200px;
  grid-template-rows: 50px 100px 50px 100px;;
}
```

## Grid con líneas nombradas

Con Grid CSS también tenemos la posibilidad de usar «**linenames**», o lo que es lo mismo, ponerle nombre a las líneas de nuestro sistema grid. Vamos a verlo con un ejemplo, donde probablemente se vea mucho mejor. Partamos de esta estructura HTML:

```
<div class="grid">
  <div class="header">Header</div>
  <div class="sidebar">Sidebar</div>
  <div class="content">Content</div>
  <div class="footer">Footer</div>
</div>
```

Los nombres de las clases ya dan una idea del contenido que tendrán. Ahora, mediante Grid CSS lo que haremos es darle una estructura definida. Para ello, vamos a considerar los siguientes nombres para las líneas de nuestro grid, utilizando **X** para las posiciones en el **eje X** y utilizando **Y** para las posiciones en el **eje Y**:



Teniendo esto en cuenta, lo único que tenemos que hacer es indicar estos nombres entre corchetes, justo antes de la medida que establecimos como vimos en apartados anteriores. Obsérvese que también se coloca un nombre de línea final sin medida a continuación, que representa la línea final:

```
.grid {  
  display: grid;  
  grid-template-columns: [x0] 1fr [x1] 1fr [x2] 1fr [x3];  
  grid-template-rows: [y0] 1fr [y1] 1fr [y2] 1fr [y3];  
}
```

En este caso, los nombres utilizados son tan sólo un ejemplo didáctico. Si se considera más adecuado, se podrían utilizar otros nombres quizás más amigables como **top-line**, **top-medium-line**, **bottom-medium-line** y **bottom-line**, por ejemplo, en lugar de **y0**, **y1**, **y2** y **y3** respectivamente.

Ahora, teniendo los nombres, sólo quedaría delimitar que zonas del grid queremos que ocupe cada uno de nuestros bloques **<div>** del grid. Para ello utilizaremos las propiedades **grid-column-start**, **grid-column-end** y **grid-row-start**, **grid-row-end**. También podríamos utilizar sus propiedades de atajo **grid-column** y **grid-row**.

```
.header {  
  background: darkred;  
  grid-column-start: x0;  
  grid-column-end: x3;  
  
  /* Equivalente a */  
  grid-column: x0 / x3;  
}
```

```
.sidebar {  
  background: black;  
  grid-row: y1 / y2;  
  color: white;
```

```

}

.content {
  background: orange;
  grid-column: x1 / x3;
  grid-row: y1 / y3;
}

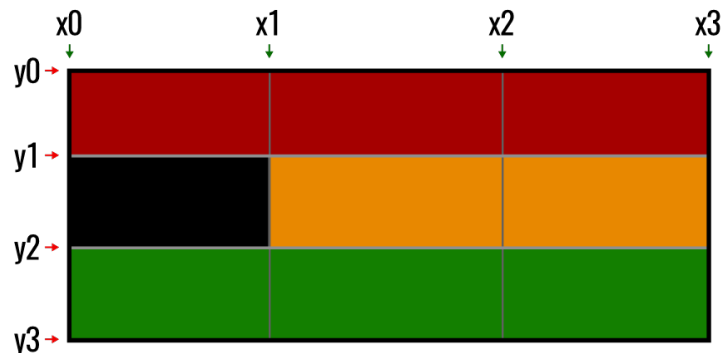
.footer {
  background: green;
  grid-column: x0 / x3;
  grid-row: y2;
}

```

Hemos aplicado la siguiente estructura:

- Zona **.header** desde la columna **x0** a **x3**.
- Zona **.sidebar** desde la fila **y1** a **y2**.
- Zona **.content** desde la columna **x1** a **x3** y desde la fila **y1** a **y3**.
- Zona **.footer** desde la columna **x0** a **x3** en la fila **y2**.

Por lo que nuestra estructura grid quedaría así:



## Grid por áreas

Mediante los **grids CSS** es posible indicar el nombre y posición concreta de cada área de una cuadrícula. Para ello utilizaremos la propiedad **grid-template-areas**, donde debemos especificar el orden de las áreas en la cuadrícula. Posteriormente, en cada ítem hijo, utilizamos la propiedad **grid-area** para indicar el nombre del área del que se trata:

Propiedad	Descripción
<b>grid-template-areas</b>	Indica la disposición de las áreas en el grid. Cada texto entre comillas simboliza una fila.
<b>grid-area</b>	Indica el nombre del área. Se usa sobre ítems hijos del grid.

De esta forma, es muy sencillo crear una cuadrícula altamente personalizada en apenas unas cuantas líneas de CSS, con mucha flexibilidad en la disposición y posición de cada área:

```
.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 }
```

Aplicando este código, conseguimos una cuadrícula donde:

- El **Item 1**, la cabecera (*head*), ocuparía toda la parte superior.
- El **Item 2**, el menú (*menu*), ocuparía el área izquierda del grid, debajo de la cabecera.
- El **Item 3**, el contenido (*main*), ocuparía el área derecha del grid, debajo de la cabecera.
- El **Item 4**, el pie de cuadrícula (*foot*), ocuparía toda la zona inferior del grid.





En la propiedad **grid-template-areas**, en lugar de indicar el nombre del área a colocar, también podemos indicar una palabra clave especial:

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

## Huecos en grid

Por defecto, la cuadrícula tiene todas sus celdas pegadas a sus celdas contiguas. Aunque sería posible darle un **margin** a las celdas dentro del contenedor, existe una forma más apropiada, que evita los problemas clásicos de los modelos de caja: los huecos (*gutters*).

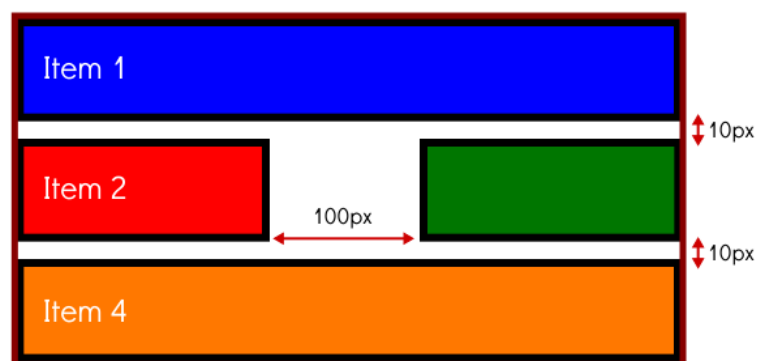
Para especificar los **huecos** (*espacio entre celdas*) podemos utilizar las propiedades **grid-column-gap** y/o **grid-row-gap**. En ellas indicaremos el tamaño de dichos huecos:

Propiedad	Descripción
<b>grid-column-gap</b>	Establece el de los huecos entre columnas ( <i>líneas verticales</i> ).
<b>grid-row-gap</b>	Establece el de los huecos entre filas ( <i>líneas horizontales</i> ).

Tomemos el ejemplo anterior como base. En él, le indicamos estas propiedades para colocar **huecos** entre las celdas de la cuadrícula. El código a añadir al ejemplo anterior sería el siguiente:

```
.grid {  
  grid-column-gap: 100px;  
  grid-row-gap: 10px;  
}
```

Con esto, obtendremos un resultado como el siguiente, indicando huecos entre columnas de 100px y huecos entre filas de 10px:



### Atajo: Grid con huecos

Existe una propiedad de atajo para las propiedades **grid-column-gap** y **grid-row-gap**, permitiéndonos la posibilidad de no tener que indicarla por separado.

La propiedad en cuestión sería **grid-gap** y se utilizaría de la siguiente forma:

```
.grid {  
  /* grid-gap: <row-gap> <column-gap> */  
  grid-gap: 20px 80px;  
  
  /* Equivalente a grid-gap: 40px 40px; */  
  /* grid-gap: <rowcolumn-gap> */
```

```
grid-gap: 40px;  
}
```

## Posición en el grid

Existen una serie de propiedades que se pueden utilizar para colocar los ítems dentro de la cuadrícula. Con ellas podemos distribuir los elementos de una forma muy sencilla y cómoda. Dichas propiedades son **justify-items** y **align-items**, que ya conocerás del módulo CSS **flexbox**:

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

Estas propiedades se aplican sobre el elemento contenedor padre, pero afectan a los ítems hijos, por lo que actúan sobre la distribución de cada uno de los hijos. En el caso de que queramos que uno de los ítems hijos tengan una distribución diferente al resto, aplicamos la propiedad **justify-self** o **align-self** sobre el ítem hijo en cuestión, sobrescribiendo su distribución.

Estas propiedades funcionan exactamente igual que sus análogas **justify-items** o **align-items**, sólo que en lugar de indicarse en el elemento padre contenedor, se hace sobre un elemento hijo. Las propiedades sobre ítems hijos las veremos más adelante.

También podemos utilizar las propiedades **justify-content** o **align-content** para modificar la distribución de todo el contenido en su conjunto, y no sólo de los ítems por separado:

Propiedad	Valores
<b>justify-content</b>	start   end   center   stretch   space-around   space-between   space-evenly
<b>align-content</b>	start   end   center   stretch   space-around   space-between   space-evenly

De esta forma, podemos controlar prácticamente todos los aspectos de posicionamiento de la cuadrícula directamente desde los estilos CSS de su contenedor padre:



## Atajo de posiciones grid

Si vamos a crear estructuras grid donde necesitamos utilizar las cuatro propiedades anteriores, quizás sería mejor utilizar un atajo donde simplificaríamos mucho el código resultante. Es el caso de las siguientes propiedades:

Propiedad	Valores	Descripción
<b>place-items</b>	<u>[align-items]</u> <u>[justify-items]</u>	Propiedad de atajo para <b>*-items</b>
<b>place-content</b>	<u>[align-content]</u> <u>[justify-content]</u>	Propiedad de atajo para <b>*-content</b>

Con ellas conseguiremos que nuestro código sea menos verboso.

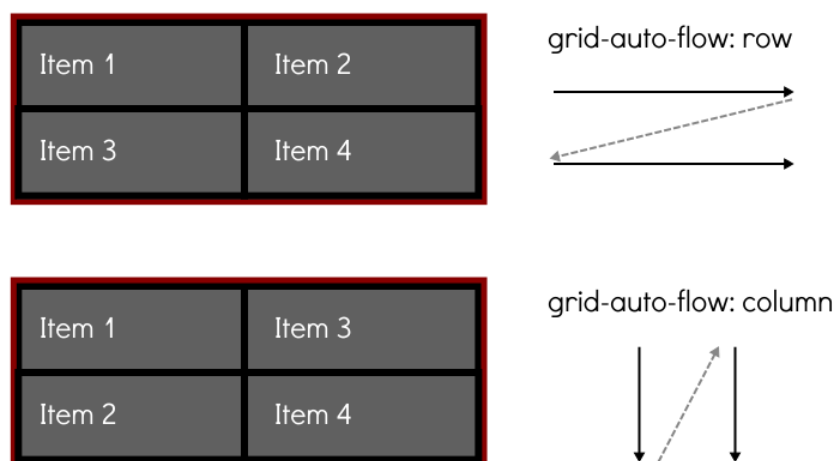
### Ajuste automático de celdas

Es posible utilizar las propiedades **grid-auto-columns** y **grid-auto-rows** para darle un tamaño automático a las celdas de la cuadrícula. Para ello, sólo hay que especificar el tamaño deseado en cada una de las propiedades. Además, también podemos utilizar **grid-auto-flow** para indicar el flujo de elementos en la cuadrícula, y especificar por donde se irán añadiendo. Las propiedades son las siguientes:

Propiedad	Valores	Descripción
<b>grid-auto-columns</b>		Indica el tamaño automático de ancho que tendrán las columnas.
<b>grid-auto-rows</b>		Indica el tamaño automático de alto que tendrán las filas.

<b>grid-auto-flow</b>	row   column   dense	Utiliza un algoritmo de autocolocación (intenta rellenar huecos).
-----------------------	----------------------------	-------------------------------------------------------------------

Un ejemplo de como se insertarían los elementos en una cuadrícula de 2x2 utilizando **grid-auto-flow** por columnas o por filas:



## Atajo: Grid

Por último, existe una propiedad **grid** que sirve de atajo para la mayoría de propiedades CSS relativas a cuadrículas. Su esquema de utilización sería el siguiente, junto a algunos ejemplos:

```
.grid {
  /* grid: <grid-template> <grid-auto-flow> <grid-auto-rows> / <grid-auto-columns> */

  grid: 100px 20px;
  grid: 200px repeat(2, 100px) 300px;
  grid: row;
  grid: column dense;
  grid: row 200px;
  grid: row 400px / 150px;
}
```



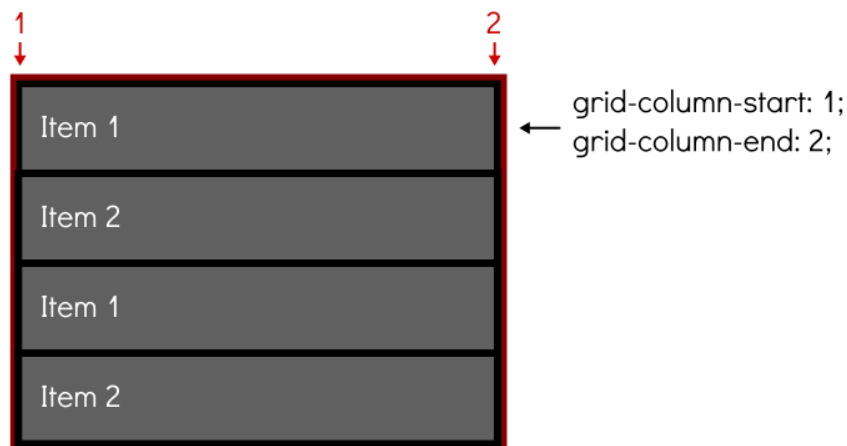
<b>grid-column-start</b>	Indica en que columna empezará el ítem de la cuadrícula.
<b>grid-column-end</b>	Indica en que columna terminará el ítem de la cuadrícula.
<b>grid-row-start</b>	Indica en que fila empezará el ítem de la cuadrícula.
<b>grid-row-end</b>	Indica en que fila terminará el ítem de la cuadrícula.

Con dichas propiedades, podemos indicar el siguiente código CSS sobre el primer ítem de una cuadrícula de 4 ítems:

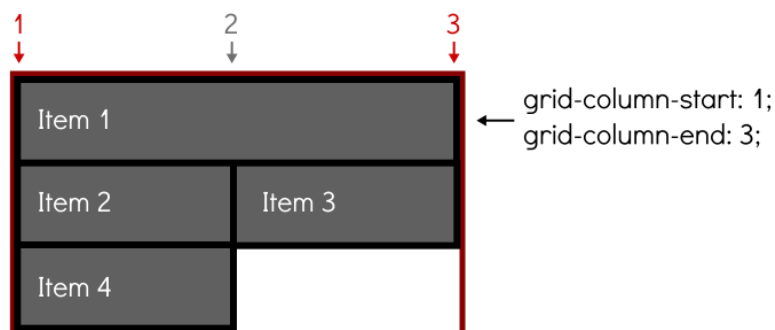
```
.grid {  
  display:grid;  
}  
  
.a {  
  grid-column-start: 1;  
  grid-row-end: 2;  
}
```

De esta forma, tenemos una cuadrícula de 4 elementos, en el que indicamos la posición del ítem 1 (*elemento con clase **.a***): comenzando en la columna 1 y acabando en el inicio de la columna 2:

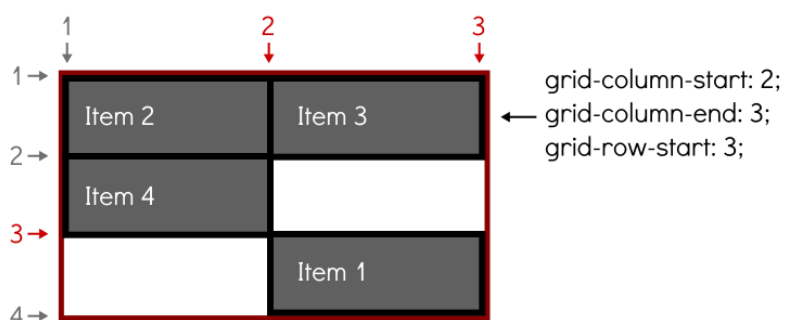




Ese sería el funcionamiento normal. Donde se ve la utilidad de estas propiedades, es si variamos los valores de forma que tomen posiciones diferentes, como por ejemplo, si indicamos que el ítem 1 debe comenzar en la columna 1, pero acabar en la columna 3 (ocupando la hipotética primera y segunda celda):



En este nuevo ejemplo, comenzamos el primer ítem en la columna 2 y lo acabamos al principio de la columna 3, por lo que la celda permanecerá en la posición de la segunda columna. Además, añadimos la propiedad **grid-row-start** que hace lo mismo que hasta ahora, pero con las filas. En este caso, le indicamos que comience en la fila 3, por lo que el **ítem 1** se desplaza a una nueva fila de la cuadrícula, dejando en la anterior el **ítem 4**:



También es posible utilizar la palabra clave **span** seguida de un número, que indica que abarque hasta esa columna o celda.