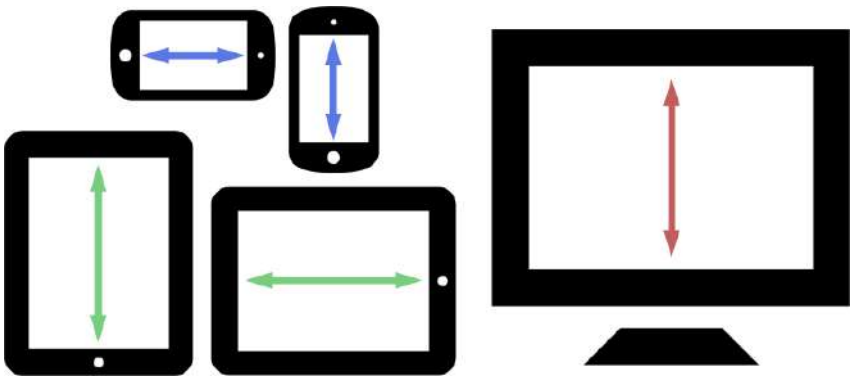
Responsive Web Design

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

Cada vez es más frecuente acceder a Internet con diferentes tipos de dispositivos, que a suvez tienen **diferentes pantallas y resoluciones**, con distintos tamaños y formas, que hacen que se consuman las páginas webs de formas diferentes, apareciendo por el caminotambién 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 noera práctico.

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

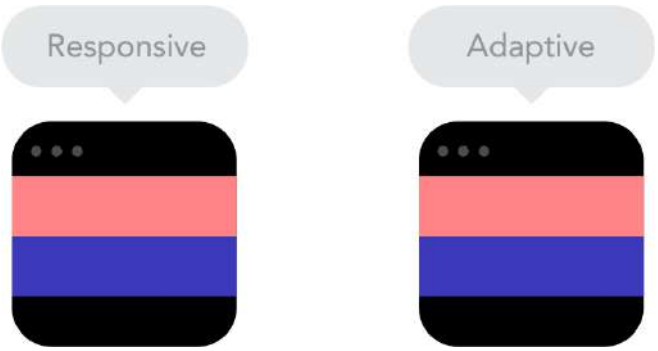
Hoy en día se le denomina **Responsive Web Design** (*o RWD*) a los diseños web que tienen la capacidad de adaptarse al tamaño y formato de la pantalla en la que se visualiza elcontenido, respecto a los diseños tradicionales en los que las páginas web estaban diseñadas sólo para un tamaño o formato específico,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

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 seadquiere experiencia. En MediaQueri.es puedes encontrar algunos ejemplos de páginasque utilizan Responsive Web Design para tener clara la idea.

Conceptos básicos

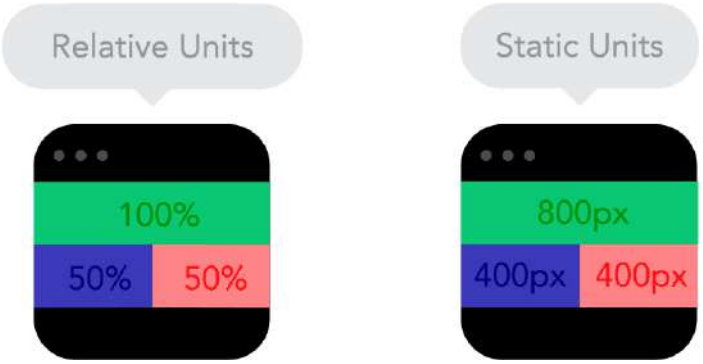
En el excelente artículo 9 basic principles of responsive web design, de Froont https://blog.froont.com/9-basic-principles-of-responsive-web-design/, hay una estupenda explicación visual de algunos conceptos básicos necesarios para entender correctamente el **Responsive Web Design**. Son los siguientes:

El primero de ellos es la diferencia entre **diseño responsivo** y **diseño adaptativo**. Comose 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 diseñoadaptable es aquel que se adapta, pero no necesariamente responde en todo momento:



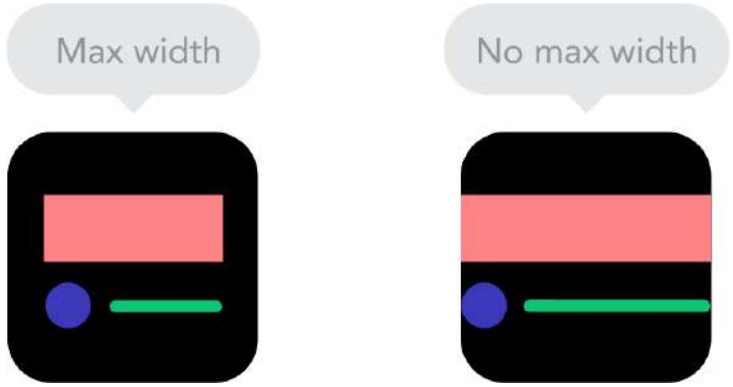
Nota: podrá ver este y los siguientes GIFs animados en el archivo **rwd_gifs_animados.zip**.

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:



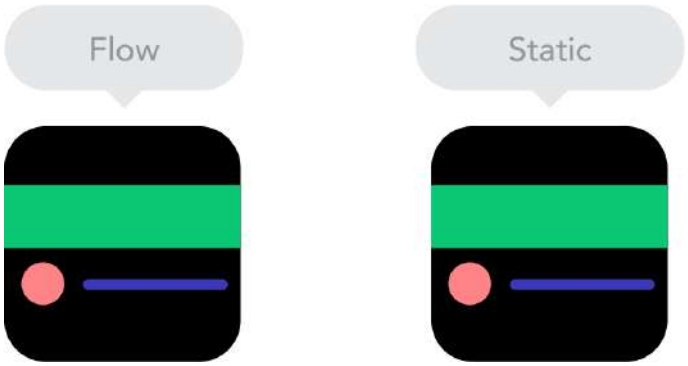
Otra forma interesante de trabajar esa respuesta de los diseños **responsive** es utilizar propiedades como **min-width** o **max-width**, donde definimos tamaños mínimos o máximos,para que los elementos de nuestra página puedan ampliarse o reducirse según sea necesario dependiendo de la pantalla del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

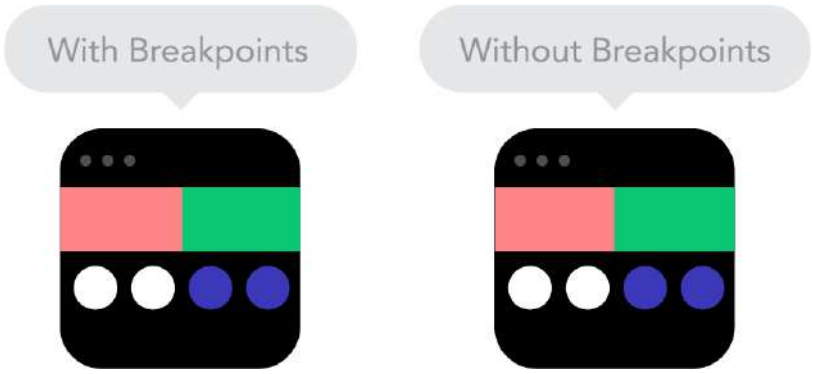


Otro concepto, que a la misma vez es una característica muy atractiva en nuestros diseños responsive es la de mantener el flujo de los elementos cuando cambian de tamaño y evitarque estos se solapen unos con otros.

Si estamos habituados a trabajar en diseños más estáticos que no están preparados paramóviles, 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 deuna 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 pantallagrande.

- Ayuda a evitar la mala práctica de ocultar bloques de información en dispositivosmóviles.

- 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 tenerclaro 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 https://www.mydevice.io/, la cual tieneun apartado de comparación de dispositivos https://www.mydevice.io/#compare-devices, 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.

Fuente: lenguajecss.com

El viewport

En muchos casos puede que oigas hablar del **viewport** del navegador. Esa palabra hacereferencia 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 eldocumento 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 comportamientopara 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 hacerzoom.

maximum-scale	10	Escala máxima a la que se puede aumentar al hacerzoom.
user-scalable	no/fixed | **yes/zoom**	Posibilidad de hacer zoom en la página web.

Las propiedades **initial-scale** , **minimum-scale** y **maximum-scale** permiten valores desdeel **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 apareceren 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 ocupeotra 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 podemoshacer esas excepciones para que sólo se apliquen a un tipo de diseño concreto.

Fuente: lenguajecss.com

¿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 quecumplan 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 mediosconcretos (***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. Tambien pueden separarse por comas varias condiciones de mediaqueries.

Existen los siguientes **tipos de medios**:

Tipo de medio	**Significado**

screen	Monitores o pantallas de ordenador. Es el más común.
print	Documentos de medios impresos o pantallas de previsualización deimpresión.
speech	Lectores de texto para invidentes (Antes **aural**, el cual ya está obsoleto).
all	Todos los dispositivos o medios. El que se utiliza **por defecto**.

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 listaanterior 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 viewportse 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 parala 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. Observese 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, yaque 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 quecumplan 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 detipos de características que podemos utilizar:

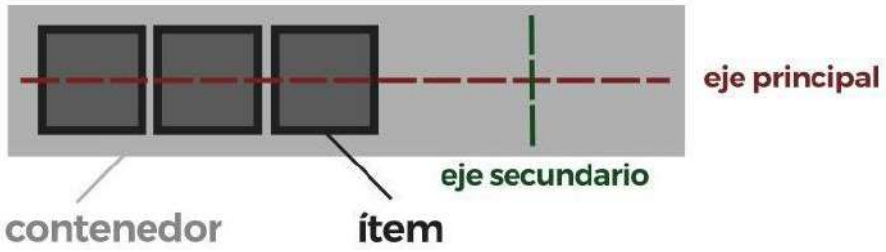
Tipo de característica	**Valores**	**¿Cuándo se aplica?**
width		Si el dispositivo tiene el tamaño indicado exactamente.
min-width		Si el dispositivo tiene un tamaño de ancho mayoral indicado.
max-width		Si el dispositivo tiene un tamaño de ancho menoral indicado.
aspect-ratio	*aspect-ratio*	Si el dispositivo encaja con la proporción deaspecto indicada.
orientation	landscape | portrait	Si el dispositivo está colocado en modo vertical oapaisado.

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

Fuente: lenguajecss.com

Flexbox

Flexbox es un sistema de **elementos flexibles** que llega con la idea de acostumbrarnos a una mecánica más potente, limpia y personalizable, en la que los elementos HTML se adaptan y colocan automáticamente y es más fácil personalizar los diseños. Está especialmente diseñado para crear, mediante CSS, estructuras de una sóla dimensión.



Conceptos

Para empezar a utilizar **flexbox** lo primero que debemos hacer es conocer algunos de los elementos básicos de este nuevo esquema, que son los siguientes:

- Contenedor: Existe un elemento padre que es el contenedor que tendrá en su interior cada uno de los ítems flexibles y adaptables.
- Eje principal: Los contenedores flexibles tendrán una orientación principal específica. Por defecto, es en horizontal (fila).
- Eje secundario: De la misma forma, los contenedores flexibles tendrán una orientación secundaria, perpendicular a la principal. Si la principal es en horizontal, la secundaria será en vertical, y viceversa.
- Ítem: Cada uno de los hijos flexibles que tendrá el contenedor en su interior.

Imaginemos el siguiente escenario:

```
<div id="contenedor">    <!-- contenedor flex -->
<div class="item item-1">1</div>    <!-- cada uno de los ítems flexibles -->
<div class="item item-2">2</div>
<div class="item item-3">3</div>
</div>
```

Para activar el modo **flexbox** hay que utilizar sobre el elemento contenedor la propiedad display, y especificar el valor **flex.**

Por defecto, y sólo con esto, observaremos que los elementos se disponen todos sobre una misma línea. Esto ocurre porque estamos utilizando el modo **flexbox** y estaremos trabajando con ítems flexibles básicos, garantizando que no se desbordaran.

Dirección de los ejes

Existen dos propiedades principales para manipular la dirección y comportamiento de los ítems a lo largo del eje principal del contenedor. Son las siguientes:

Propiedad	Valor	Significado
flex-direction:	**row** | row-reverse | column | column-reverse	Cambia la orientación del eje principal.
flex-wrap:	**nowrap** | wrap | wrap-reverse	Evita o permite el desbordamiento (multilínea).

Mediante la propiedad flex-direction podemos modificar la dirección del eje principal del contenedor para que se oriente en horizontal (por defecto) o en vertical. Además, también podemos incluir el sufijo -reverse para indicar que coloque los ítems en orden inverso.

Valor	Descripción
row	Establece la dirección del eje principal en horizontal.
row-reverse	Establece la dirección del eje principal en horizontal (invertido).
column	Establece la dirección del eje principal en vertical.
column-reverse	Establece la dirección del eje principal en vertical (invertido).

Esto nos permite tener un control muy alto sobre el orden de los elementos en una página. Veamos la aplicación de estas propiedades sobre el ejemplo anterior, para modificar el flujo del eje principal del contenedor:

```
#contenedor {
background: #CCC;
display: flex;
flex-direction: column;
}
```

```
.item {
background: #777;
}
```

Por otro lado, existe otra propiedad llamada flex-wrap con la que podemos especificar el comportamiento del contenedor respecto a evitar que se desborde (nowrap, valor por defecto) o permitir que lo haga, en cuyo caso, estaríamos hablando de un contenedor flexbox multilinea.

Valor	**Descripción**
nowrap	Establece los ítems en una sola línea (no permite que se desborde el contenedor).
wrap	Establece los ítems en modo multilínea (permite que se desborde el contenedor).
wrap-reverse	Establece los ítems en modo multilínea, pero en dirección inversa.

Teniendo en cuenta estos valores de la propiedad flex-wrap, podemos conseguir cosas como la siguiente:

```
#contenedor {
background: #CCC;
display: flex;
width: 200px;
flex-wrap: wrap; /* Comportamiento por defecto: nowrap */
}
.item {
background: #777;
width: 50%;
}
```

En el caso de especificar **nowrap** (u omitir la propiedad flex-wrap) en el contenedor, los 3 ítems se mostrarían en una misma línea del contenedor. En ese caso, cada ítem debería tener un 50% de ancho (o sea, 100px de los 200px del contenedor). Un tamaño de **100px** por ítem, sumaría un total de **300px**, que no cabrían en el contenedor de **200px**, por lo que flexbox reajusta los ítems flexibles para que quepan todos en la misma línea, manteniendo las mismas proporciones.

Sin embargo, si especificamos wrap en la propiedad flex-wrap, lo que permitimos

es que el contenedor se pueda desbordar, pasando a ser un contenedor **multilínea,** que mostraría el **ítem 1 y 2** en la primera linea (con un tamaño de 100px cada uno) y el **ítem 3** en la línea siguiente, dejando un espacio libre para un posible **ítem 4.**

Atajo: Dirección de los ejes

Existe una propiedad de atajo (short-hand) llamada flex-flow, con la que podemos resumir los valores de las propiedades flex-direction y flex-wrap, especificándolas en una sola propiedad y ahorrándonos utilizar las propiedades concretas:

```
#contenedor {
/* flex-flow: <flex-direction> <flex-wrap>; */
flex-flow: row wrap;
}
```

Propiedades de alineación de ítems

Ahora que tenemos un control básico del contenedor de estos ítems flexibles, necesitamos conocer las propiedades existentes dentro de flexbox para disponer los ítems dependiendo de nuestro objetivo. Vamos a echar un vistazo a cuatro propiedades interesantes para ello:

Propiedad	Valor	Actúa sobre
justify-content:	**flex-start** | flex-end | center | space-between |space-around	Eje principal
align-content:	flex-start | flex-end | center | space-between | space-around | **stretch**	Eje principal
align-items:	flex-start | flex-end | center | **stretch** | baseline	Eje secundario
align-self:	**auto** | flex-start | flex-end | center | stretch |baseline	Eje secundario

De esta pequeña lista, nos centraremos en la primera y la tercera propiedad, que son las más importantes (las otras dos son casos particulares que explicaremos más adelante):

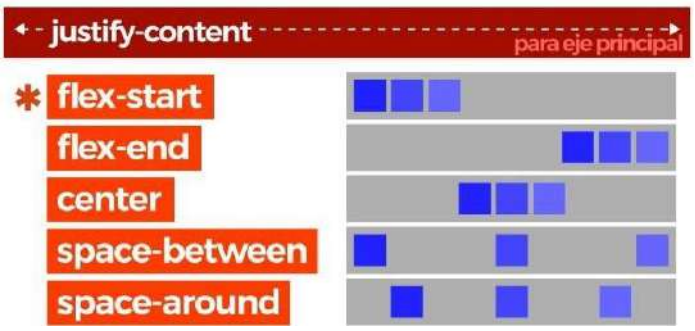
justify-content: Se utiliza para alinear los ítems del eje principal (por defecto, el horizontal).
align-items: Usada para alinear los ítems del eje secundario(por defecto, el vertical).

Sobre el eje principal

La primera propiedad, justify-content, sirve para colocar los ítems de un contenedor mediante una disposición concreta a lo largo del eje principal:

Valor	Descripción
flex-start	Agrupa los ítems al principio del eje principal.
flex-end	Agrupa los ítems al final del eje principal.
center	Agrupa los ítems al centro del eje principal.
space-between	Distribuye los ítems dejando (el mismo) espacio entre ellos.
space-around	Distribuye los ítems dejando (el mismo) espacio a ambos lados decada uno de ellos.

Con cada uno de estos valores, modificaremos la disposición de los ítems del contenedor donde se aplica, pasando a colocarse como se ve en la imagen siguiente (nótense las diferentes tonalidades azules para indicar las posiciones decada ítem):

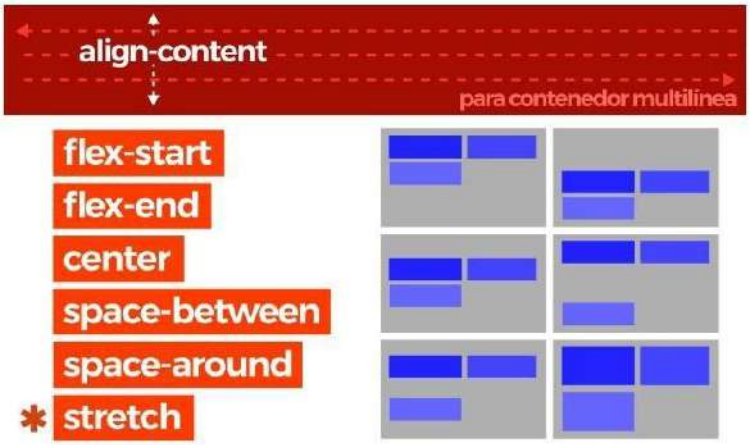


Una vez entendido este caso, debemos atender a la propiedad align-content, quees un caso particular del anterior. Nos servirá cuando estemos tratando con un contenedor flex multilínea, que es un contenedor en el que los ítems no caben en elancho disponible, y por lo tanto, el eje principal se divide en múltiples líneas.

De esta forma, align-content servirá para alinear cada una de las líneas delcontenedor multilinea. Los valores que puede tomar son los siguientes:

Valor	Descripción
flex-start	Agrupa los ítems al principio del eje principal.
flex-end	Agrupa los ítems al final del eje principal.
center	Agrupa los ítems al centro del eje principal.
space-between	Distribuye los ítems desde el inicio hasta el final.
space-around	Distribuye los ítems dejando el mismo espacio a los lados de cadauno.
stretch	Estira los ítems para ocupar de forma equitativa todo el espacio.

Con estos valores, vemos cómo cambiamos la disposición en vertical (porque partimos de un ejemplo en el que estamos utilizando flex-direction: row, y el eje principal es horizontal) de los ítems que están dentro de un contenedor multilínea.



En el ejemplo siguiente, veremos que al indicar un contenedor de 200 píxels de altocon ítems de 50px de alto y un flex-wrapestablecido para tener contenedores multilinea, podemos utilizar la propiedad align-content para alinear los ítems de forma vertical de modo que se queden en la zona inferior del contenedor:

#contenedor { background: #CCC;display: flex;
width: 200px; height: 200px; flex-wrap: wrap;

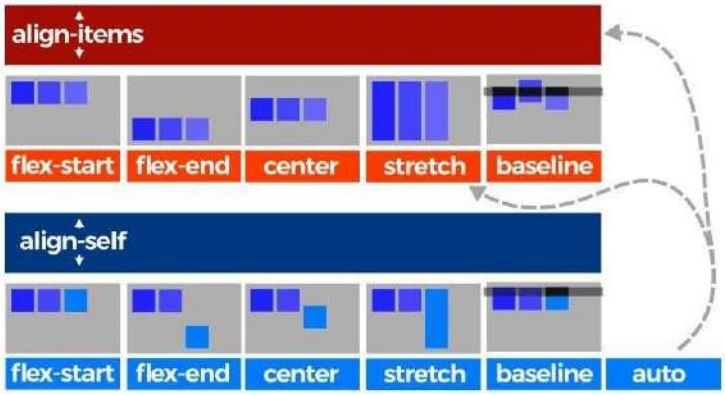
```
align-content: flex-end;
}
.item {
background: #777;
width: 50%; height: 50px;
}
```

Sobre el eje secundario

La otra propiedad importante de este apartado es align-items, que se encarga de alinear los ítems en el eje secundario del contenedor. Hay que tener cuidado de no confundir align-content con align-items, puesto que el primero actúa sobre cada una de las líneas de un contenedor multilinea (no tiene efecto sobre contenedores de una sola línea), mientras que align-items lo hace sobre la línea actual. Los valores que puede tomar son los siguientes:

Valor	Descripción
flex-start	Alinea los ítems al principio del eje secundario.
flex-end	Alinea los ítems al final del eje secundario.
center	Alinea los ítems al centro del eje secundario.
stretch	Alinea los ítems estirándolos de modo que cubran desde el inicio hasta elfinal del contenedor.
baseline	Alinea los ítems en el contenedor según la base del contenido de losítems del contenedor.

Por otro lado, la propiedad align-self actúa exactamente igual que align-items, sin embargo es la primera propiedad de flexbox que vemos que se utiliza sobre un ítem hijo específico y no sobre el elemento contenedor. Salvo por este detalle, funciona exactamente igual que align-items.



Gracias a ese detalle, align-self nos permite cambiar el comportamiento de align-items y sobreescribirlo con comportamientos específicos para ítems concretos que no queremos que se comporten igual que el resto. La propiedad puede tomar los siguientes valores:

Valor	Descripción
flex-start	Alinea los ítems al principio del contenedor.
flex-end	Alinea los ítems al final del contenedor.
center	Alinea los ítems al centro del contenedor.
stretch	Alinea los ítems estirándolos al tamaño del contenedor.
baseline	Alinea los ítems en el contenedor según la base de los ítems.
auto	Hereda el valor de align-items del padre (o si no lo tiene, stretch).

Si se especifica el valor auto a la propiedad align-self, el navegador le asigna el valor de la propiedad align-items del contenedor padre, y en caso de no existir, el valor por defecto: stretch.

Propiedades de ítems hijos

A excepción de la propiedad align-self, todas las propiedades que hemos visto hasta ahora se aplican sobre el elemento contenedor. Las siguientes propiedades, sin embargo, se aplican sobre los ítems hijos. Echemos un vistazo:

Propiedad	Valor	Descripción
flex-grow:	**0** | *[factor de crecimiento]*	Número que indica el crecimiento delítem respecto al resto.
flex-shrink:	**1** | *[factor de decrecimiento]*	Número que indica el decrecimiento delítem respecto al resto.
flex-basis:	**content**	Tamaño base de los ítems antesde aplicar variación.
order:	**0** | *[número]*	Número (peso) que indica el orden deaparición de los ítems.

En primer lugar, tenemos la propiedad **flex-grow** para indicar el factor de crecimiento de los ítems enel caso de que no tengan un ancho específico. Por ejemplo, si con flex-grow indicamos un valor de 1 a todos sus ítems, tendrían el mismo tamaño cada uno de ellos. Pero si colocamos un valor de 1 a todos los elementos, salvo a uno de ellos, que le indicamos 2, ese ítem será más grande que los anteriores. Los ítems a los que no se le especifique ningún valor, tendrán por defecto valor de 0.

En segundo lugar, tenemos la propiedad **flex-shrink** que es la opuesta a flex-grow. Mientras que la anterior indica un factor de crecimiento, flex-shrink hace justo lo contrario, aplica unfactor de decrecimiento. De esta forma, los ítems que tengan un valor numérico más grande, serán más pequeños, mientras que los que tengan un valor numérico más pequeño serán más grandes, justo al contrario de cómo funciona la propiedad flex-grow.

Por último, tenemos la propiedad **flex-basis**, que define el tamaño por defecto (de base) que tendránlos ítems antes de aplicarle la distribución de espacio.
Generalmente, se aplica un tamaño (unidades, porcentajes, etc...), pero también se puede aplicar lapalabra clave content que ajusta automáticamente el tamaño al contenido del ítem, que es su valor por defecto.

Atajo: Propiedades de ítems hijos

Existe una propiedad llamada flex que sirve de atajo para estas tres propiedades delos ítems hijos. Funciona de la siguiente forma:

```
.item {
/* flex: <flex-grow> <flex-shrink> <flex-basis> */
flex: 1 3 35%;
```

}

Orden de los ítems

Por último, y quizás una de las propiedades más interesantes, es order, que modificar y establece el orden de los ítems según una secuencia numérica.

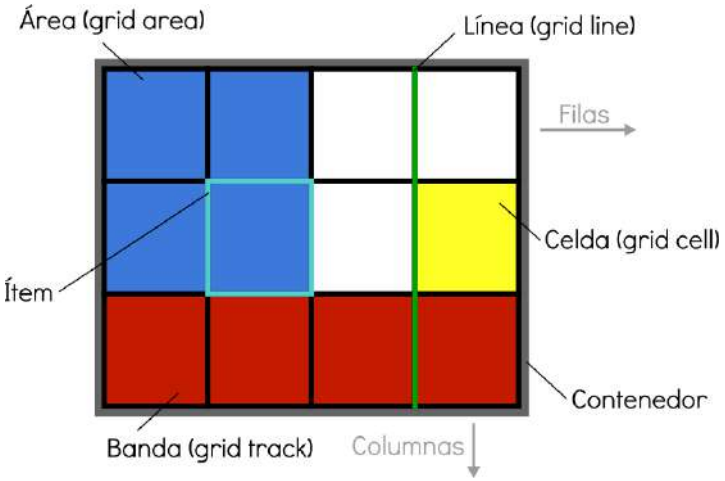
Por defecto, todos los ítems flex tienen un order: 0 implícito, aunque no se especifique. Si indicamos un order con un valor numérico, irá recolocando los ítems según su número, colocando antes los ítems con número más pequeño (incluso valores negativos) y después los ítems con números más altos.

Fuente: lenguajecss.com

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.
Area (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 cuadriculas **Grid CSS**, trabajaremos bajo el siguiente escenario:

```
<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>
```

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 lacuadrícula 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 equivalentea 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:

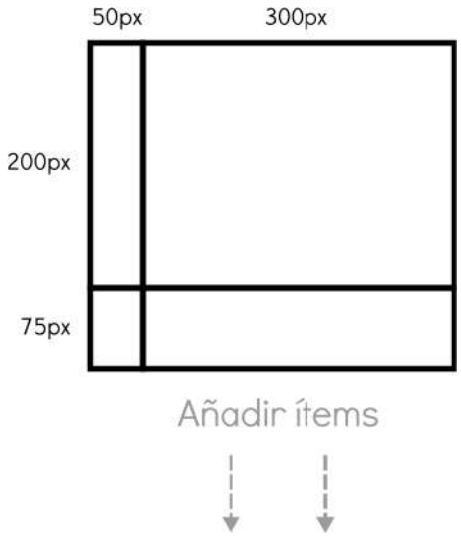
Propiedad	Valor	Descripción
grid-template-columns	[*c1*] [*c2*] ...	Establece el ancho de cada columna (*col 1,col 2...*).

grid-template-rows	[*f1*] [*f2*] ...	Establece el ancho de cada fila (*fila 1, fila2...*).

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.



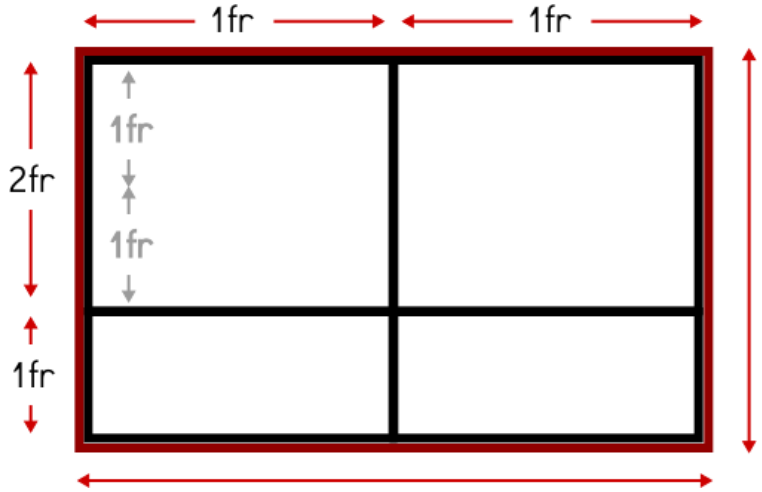
Unidad fracción restante (fr)

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 (*2fr*) 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([núm 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 anteriorcrearía 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:

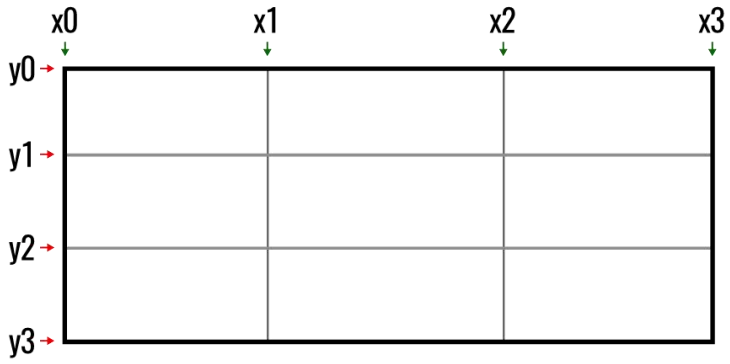
```
.grid  {
display: grid;
grid-template-columns: 100px 50px 50px 200px;
grid-template-rows: 50px 100px 50px 100px;;
}
```

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 lossiguientes 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 una 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 ** ** 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**.

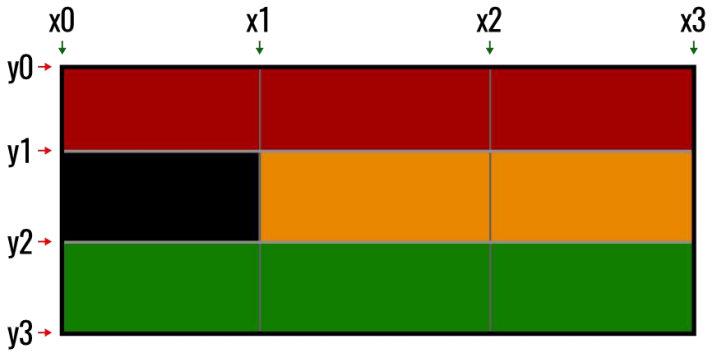
```
.header {
background: darkred;
grid-column-start: x0;
grid-column-end: x3;
/* Equivalente a */
grid-column: x0 / x3;
}
.sidebar {
background: black;
grid-row: y1 / y2;
```

```
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

grid-template-areas	Indica la disposición de las áreas en el grid. Cada texto entre comillas simboliza una fila.
grid-area	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 lacabecera.
El **Item 3**, el contenido (*main*), ocuparía el área derecha del grid, debajo de lacabecera.
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*).

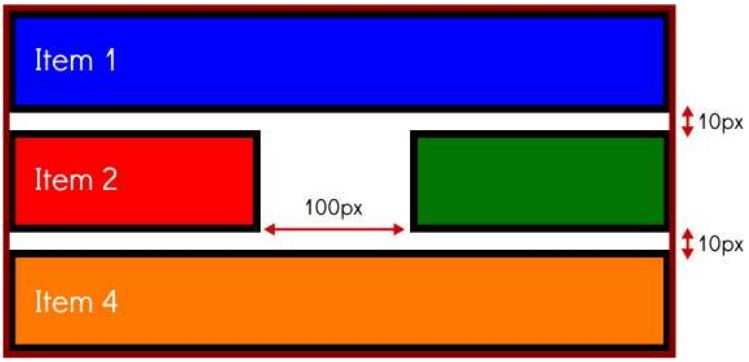
Para especificar los **huecos** (*espacio entre celdas*) podemos utilizar las propiedades **grid-column-gap** y/o **grid-row-gap**. En ellas indicaremos el tamaño de dichos huecos:

Propiedad	Descripción
grid-column-gap	Establece el de los huecos entre columnas (*líneas verticales*).
grid-row-gap	Establece el de los huecos entre filas (*líneas horizontales*).

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 columnasde 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 indicarlas 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 móduloCSS **flexbox**:

Propiedad	Valores	Descripción

justify-items	start | end | center | **stretch**	Distribuye los elementos en el ejehorizontal.
align-items	start | end | center | **stretch**	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, sobreescribiendo 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 ladistribución de todo el contenido en su conjunto, y no sólo de los ítems por separado:

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

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

Nota: podrá ver este GIF animado en el archivo **rwd_gifs_animados.zip**.

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
place-items	[*align-items*] [*justify-items*]	Propiedad de atajo para ***-items**
place-content	[*align-content*] [*justify-content*]	Propiedad de atajo para ***-content**

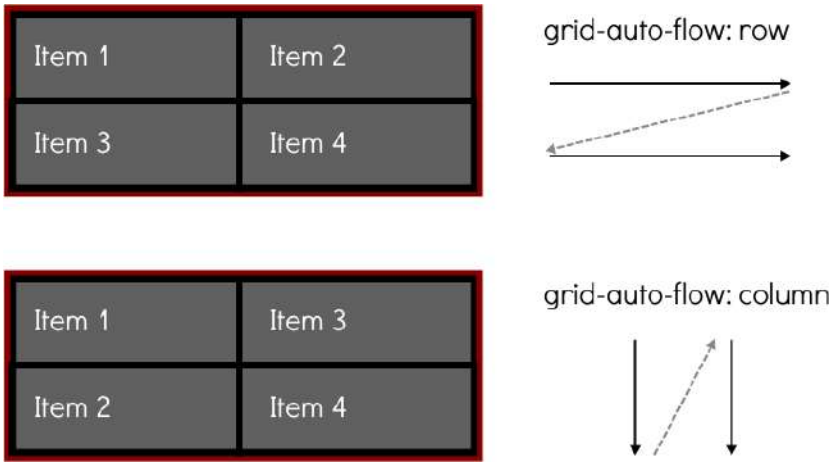
Con ellas conseguiremos que nuestro código sea menos verboso.

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
grid-auto-columns		Indica el tamaño automático de ancho que tendrán las columnas.
grid-auto-rows		Indica el tamaño automático de alto que tendrán las filas.
grid-auto-flow	row | column |dense	Utiliza un algoritmo de autocolocación (intentar rellenar huecos).

Un ejemplo de cómo 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;
}
```

Propiedades para ítems hijos

Hasta ahora, salvo algunas excepciones como **justify-self**, **align-self** o **grid-area**, hemos visto propiedades CSS que se aplican solamente al contenedor padre de una cuadrícula. A continuación, vamos a ver ciertas propiedades que en su lugar, se aplican a cada ítem hijo de la cuadrícula, para alterar o cambiar el comportamiento específico de dicho elemento, que no se comporta como la mayoría.

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-areas**.

Sin embargo, 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:

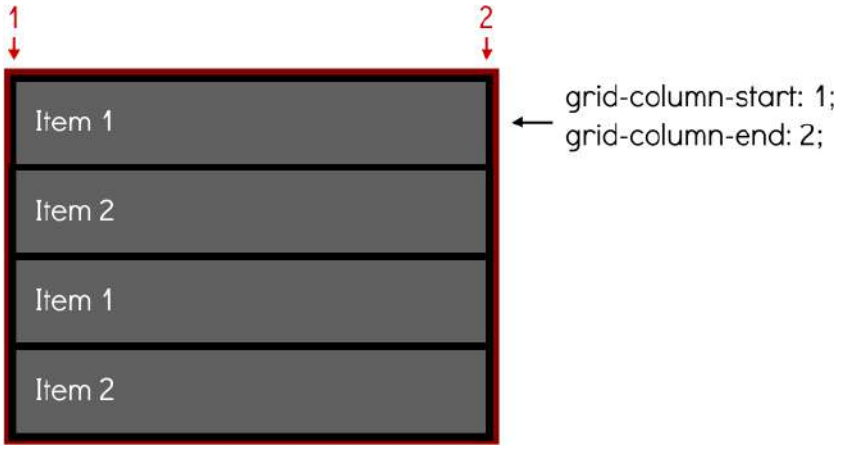
Propiedad	Descripción
grid-column-start	Indica en que columna empezará el ítem de la cuadrícula.
grid-column-end	Indica en que columna terminará el ítem de la cuadrícula.
grid-row-start	Indica en que fila empezará el ítem de la cuadrícula.
grid-row-end	Indica en que fila terminará el ítem de la cuadrícula.

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

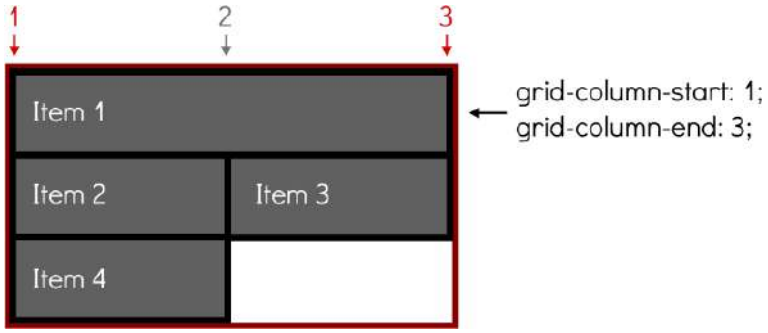
```
.grid { display:grid;
}

.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:

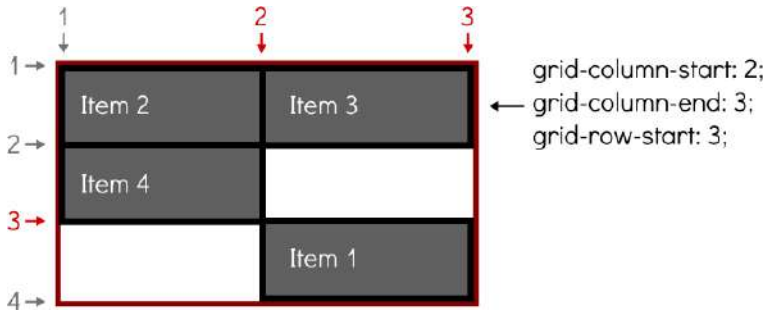


Ese sería el funcionamiento normal. Donde se ve la utilidad de estas propiedades, es si variamos los valores de forma que tomen posiciones diferentes, como por ejemplo, si indicamos que el ítem 1 debe comenzar en la columna 1, pero acabar en la columna 3 (*ocupando la hipotética primera y segunda celda*):



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.