CONSTRUCCIÓN DE PÁGINAS WEB

@mardeasa

CREACIÓN DE PÁGINAS WEB CON EL LENGUAJE DE MARCAS

Flexbox y Grid

@mardeasa

Flexbox

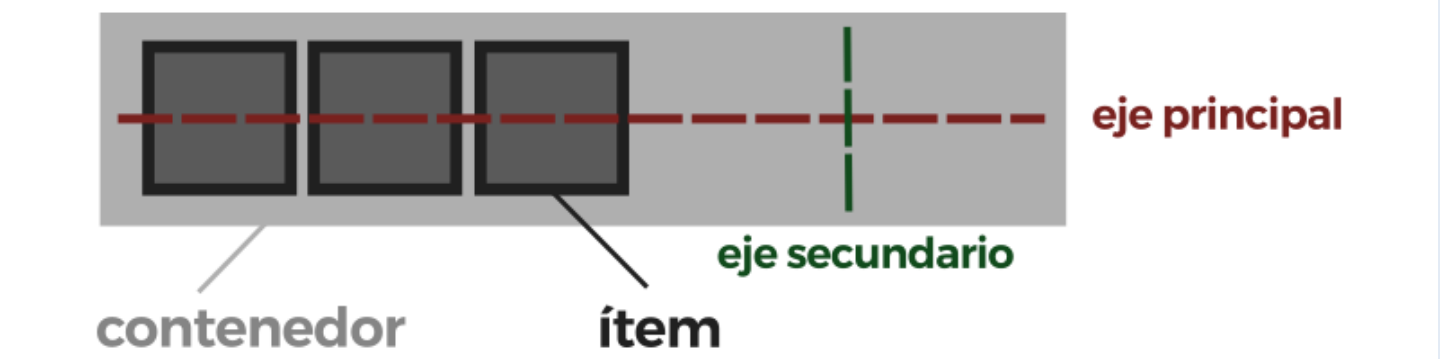
El Módulo de Caja Flexible, comúnmente llamado Flexbox, fue creado como un modelo **unidimensional** de diseño, y como un método que pudiese ayudar a distribuir el espacio entre los ítems de una interfaz y mejorar las capacidades de alineación, principalmente para suplantar al modelo de cajas flotantes (float).

Decimos que Flexbox esta diseñado como modelo unidimensional, y destacamos el hecho que maneja el diseño en una sola dimensión a la vez — ya sea como fila o como columna. Esto contrasta con el modelo **bidimensional de Grid Layout** de CSS, el cual controla columnas y filas a la vez.

Flexbox

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.

Ítem: Cada uno de los hijos flexibles que contendrá el contenedor en su interior.

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.

Flexbox

Los dos ejes de flexbox

Cuando trabajamos con flexbox necesitamos pensar en términos de dos ejes — **el eje principal** y **el eje cruzado**.

El eje principal está definido por la propiedad **flex-direction**, y el eje cruzado es perpendicular a este.

Todo lo que hacemos con flexbox está referido a estos dos ejes, por lo que vale la pena entender cómo trabajan desde el principio.

Propiedades de Flexbox

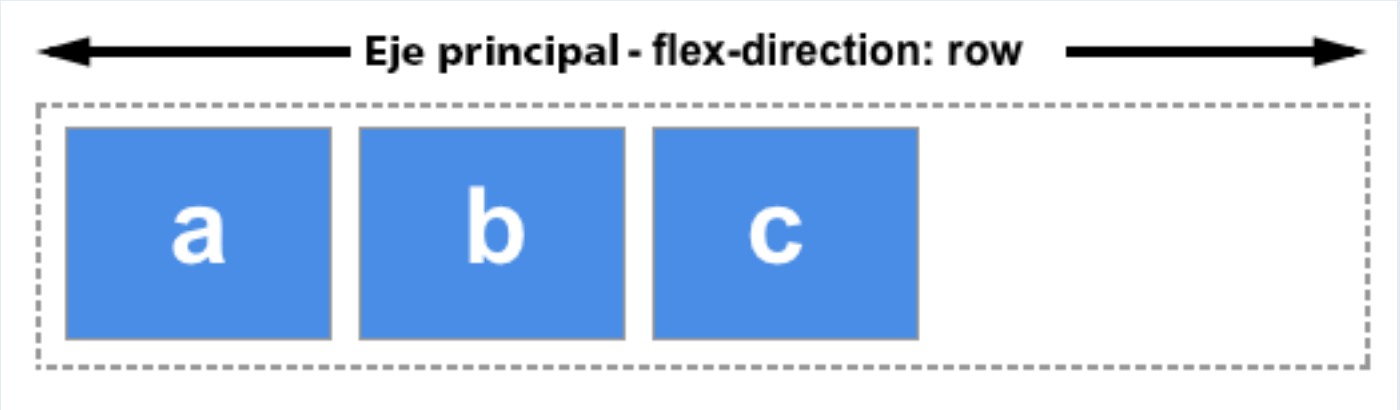
- align-content
- align-items
- align-self
- flex
- flex-basis
- flex-direction
- flex-flow
- flex-grow
- flex-shrink
- flex-wrap
- justify-content
- order
- place-content

El eje principal o Main Axis

El eje principal está definido por **flex-direction**, que posee cuatro posibles valores:

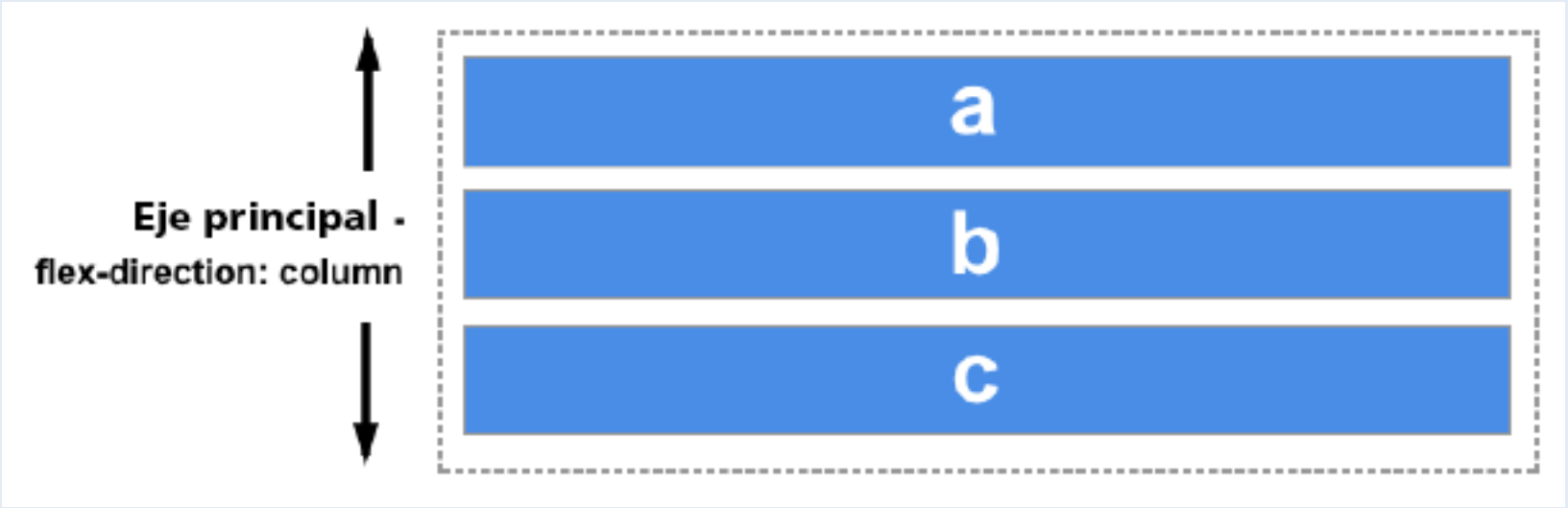
- row
- row-reverse
- column
- column-reverse

Si elegimos row o row-reverse, el eje principal correrá a lo largo de la fila según la dirección de la línea.



El eje principal o Main Axis

Al elegir column o column-reverse el eje principal irá desde el borde superior del bloque hasta el final o desde abajo hacia arriba respectivamente.

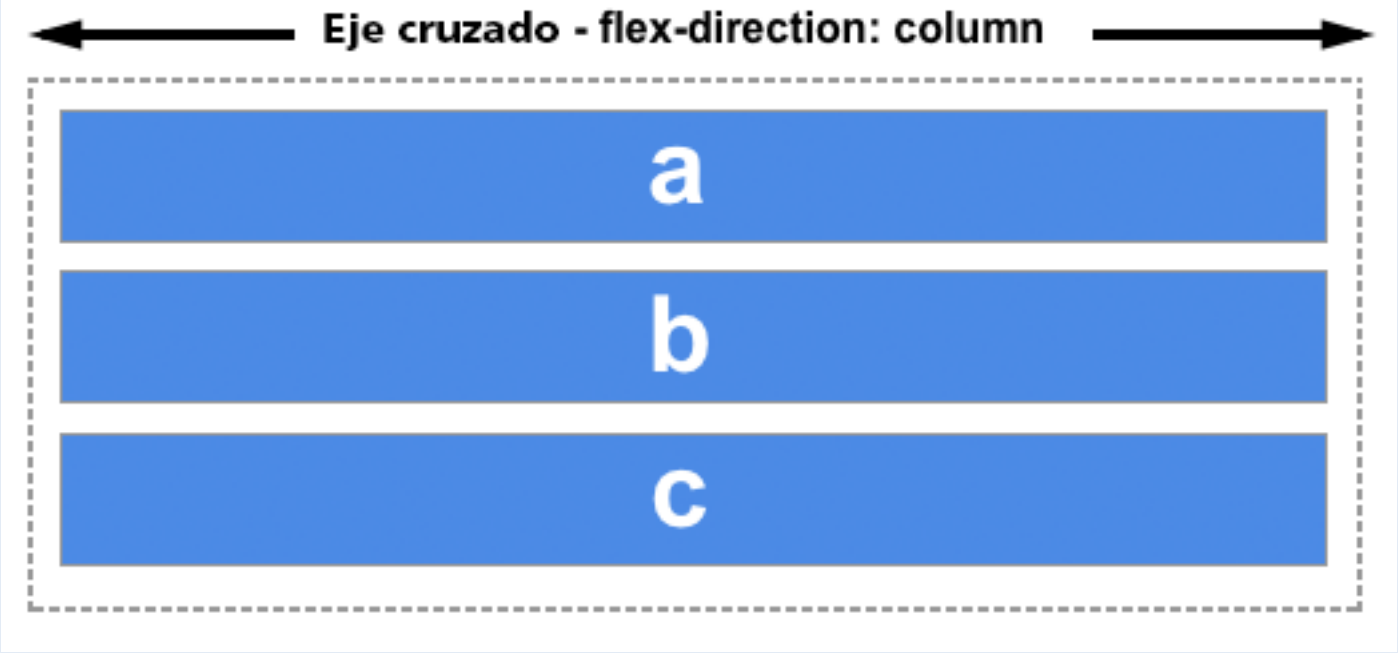


El eje cruzado o Cross Axis

El eje cruzado va perpendicular al eje principal, y por lo tanto si flex-direction (del eje principal) es row o row-reverse el eje cruzado irá por las columnas.

El eje cruzado o Cross Axis

Si el eje principal es column o column-reverse entonces el eje cruzado corre a lo largo de las filas.



Identificar cuál es cada uno de de los ejes, es importante cuando empezamos a mirar la alineación y justificación flexible de los ítems; flexbox posee propiedades que permiten alinear y justificar el contenido sobre un eje o el otro.

El contenedor flex

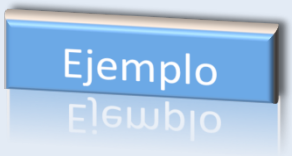
Un área del documento que contiene un flexbox es llamada contendedor flex. Para crear un contenedor flex, establecemos la propiedad del área del contenedor **display** como **flex** o **inline-flex**.

Tan pronto como hacemos esto, los hijos directos de este contenedor se vuelven ítems flex. Como con todas las propiedades de CSS, se definen algunos valores iniciales, así que cuando creemos un contenedor flex todos los ítems flex contenidos se comportarán de la siguiente manera.

- Los ítems se despliegan sobre una fila (la propiedad flex-direction por defecto es row).
- Los ítems empiezan desde el margen inicial sobre el eje principal.
- Los ítems no se ajustan en la dimensión principal, pero se pueden contraer.
- Los ítems se ajustarán para llenar el tamaño del eje cruzado.
- La propiedad flex-basis es definida como auto.
- La propiedad flex-wrap es definida como nowrap.

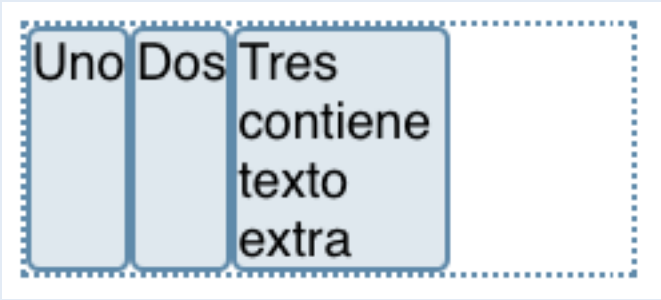
El contenedor flex

El resultado es que todos los ítems se alinearán en una sola fila, usando el tamaño del contenedor como su tamaño en el eje principal. Si hay más ítems de los que caben en el contenedor, estos no pasarán más abajo sino que sobrepasarán el margen. Si hay ítems más altos que otros, todos los ítems serán ajustados en el eje cruzado, para igualar al item mayor.




```
.box {display: flex;}
```

```
<div class="box">
        <div>Uno</div>
        <div>Dos</div>
        <div>Tres
            <br>contiene
            <br>texto
            <br>extra
        </div>
</div>
```



El contenedor flex

flex-direction

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).

Ejemplo

```
.box {display: flex;
      flex-direction: row-reverse;
     }
```

```
<div class="box">
          <div>Uno</div>
          <div>Dos</div>
          <div>Tres</div>
</div>
```

EJERCICIO

- Prueba los otros valores — row, column y column-reverse — para ver qué sucede con el contenido.

Contenedores flex Multi-línea con flex-wrap

flex-wrap

Aunque flexbox es un modelo unidimensional, es posible lograr que los ítems flex sean repartidos en varías líneas. Haciendo esto, se deberá considerar cada línea como un nuevo contenedor flex. Cualquier distribución del espacio solo sucederá dentro de esa línea, sin referenciar las líneas colaterales.

Para lograr repartirse en varias líneas hay que añadir la propiedad **flex-wrap** con el valor **wrap**. Cuando los ítems sean demasiados para desplegarlos en una línea, serán repartidos en la línea siguiente.

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.

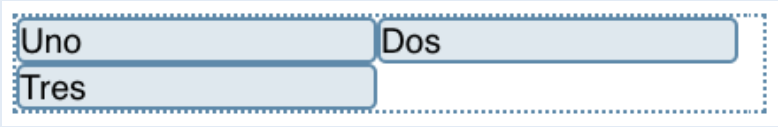
Contenedores flex Multi-línea con flex-wrap

El ejemplo siguiente contiene ítems que se les ha asignando un ancho, donde el ancho total de los ítems excede al del contenedor flex. Cuando **flex-wrap** se coloca como **wrap**, los ítems se repartirán. Al colocarlo como nowrap, el cual es el valor inicial, estos se contraerán para ajustar con el contenedor ya que usan los valores iniciales de flexbox que permiten que los ítems se contraigan. Al usar nowrap los ítems podrían salirse del margen si estos no pudieran contraerse, o no contraerse lo suficiente para entrar.

Ejemplo

```
.box {
        display: flex;
        flex-wrap: wrap;
    }
```

```
<div class="box">
          <div>Uno</div>
          <div>Dos</div>
          <div>Tres</div>
</div>
```



La abreviatura flex-flow

Se pueden combinar las propiedades flex-direction y flex-wrap en la abreviatura flex-flow . El primer valor especificado es flex-direction y el segundo valor es flex-wrap.

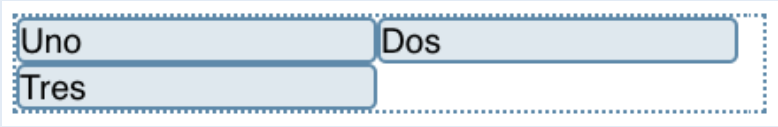
EJERCICIO

- En el ejemplo, intenta cambiar el primer valor por uno de los valores permitidos para flex-direction: row, row-reverse, column o column-reverse, y cambia también el segundo valor por wrap y nowrap.

Ejemplo

```
.box {
        display: flex;
        flex-flow: row wrap;
      }
```

```
<div class="box">
          <div>Uno</div>
          <div>Dos</div>
          <div>Tres</div>
</div>
```



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

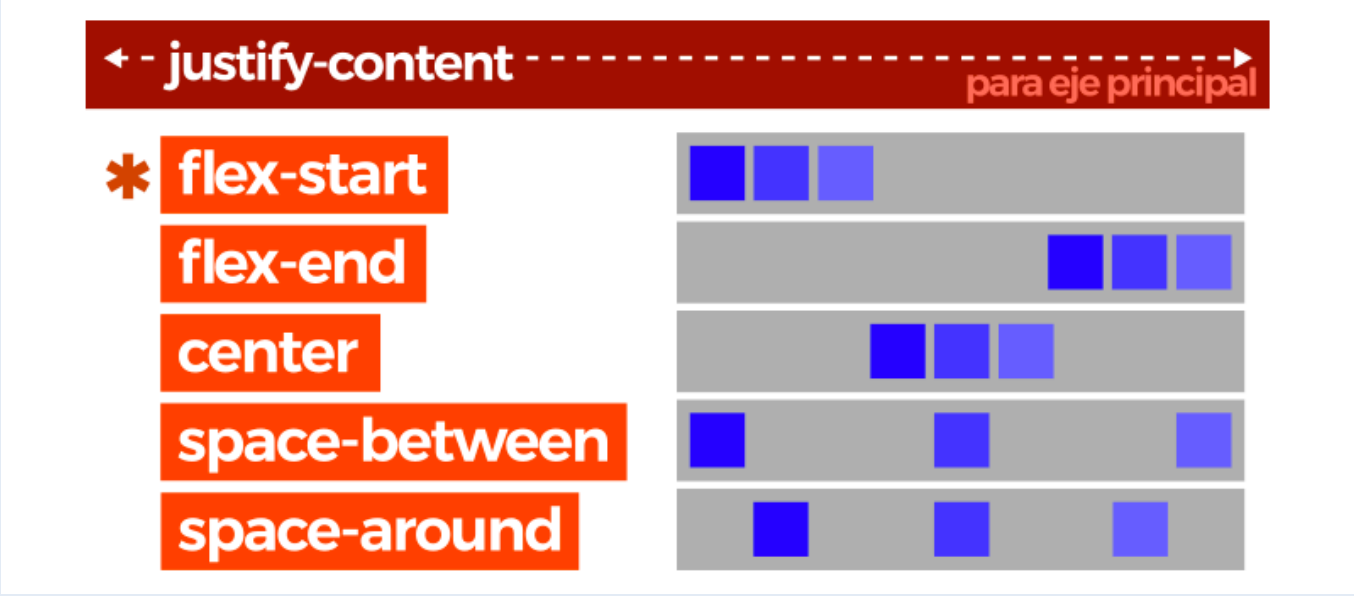
justify-content: Se utiliza para alinear los ítems del eje principal (por defecto, el horizontal).
align-content: actúa sobre cada una de las líneas de un contenedor multilinea.
align-items: Usada para alinear los ítems del eje secundario (por defecto, el vertical).
align-self: actúa exactamente igual que align-items, sin embargo se utiliza sobre un ítem hijo específico y no sobre el elemento contenedor.

Propiedades de alineación de ítems

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 de cada uno de ellos.

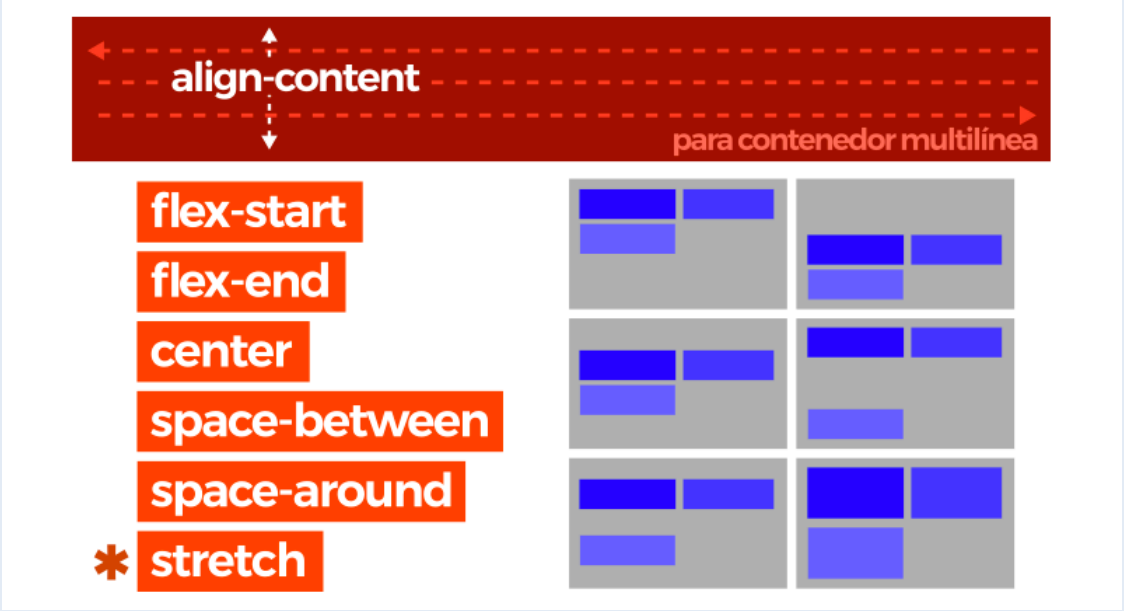


Propiedades de alineación de ítems

align-content:

Sirve para alinear cada una de las líneas del contenedor 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 cada uno.
stretch	Estira los ítems para ocupar de forma equitativa todo el espacio.



Propiedades de alineación de ítems

align-content:

En el ejemplo siguiente, veremos que al indicar un contenedor de 200 píxels de alto con ítems de 50px de alto y un flex-wrap establecido 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:

Ejemplo

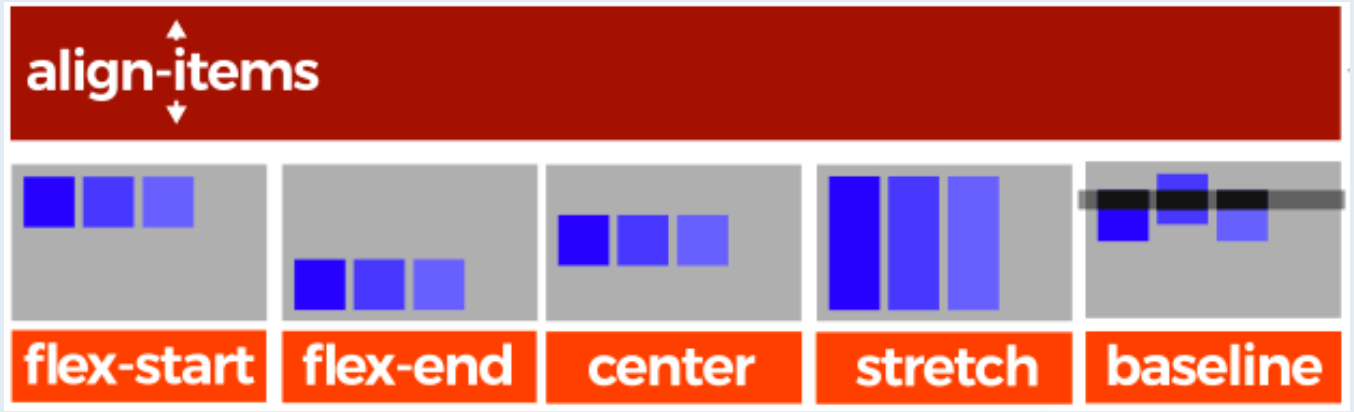
```
#contenedor {
  background: #CCC;
  display: flex;
  width: 200px;
  height: 200px;

  flex-wrap: wrap;
  align-content: flex-end;
}

.item {
  background: #777;
  width: 50%;
  height: 50px;
}
```

Propiedades de alineación de ítems

align-items:

Se encarga de alinear los ítems en el eje secundario del contenedor. Actúa sobre cada una de las líneas de un contenedor multilinea (no tiene efecto sobre contenedores de una sola línea). 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 el final del contenedor.
baseline	Alinea los ítems en el contenedor según la base del contenido de los ítems del contenedor.



Propiedades aplicadas a los ítems flex

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).

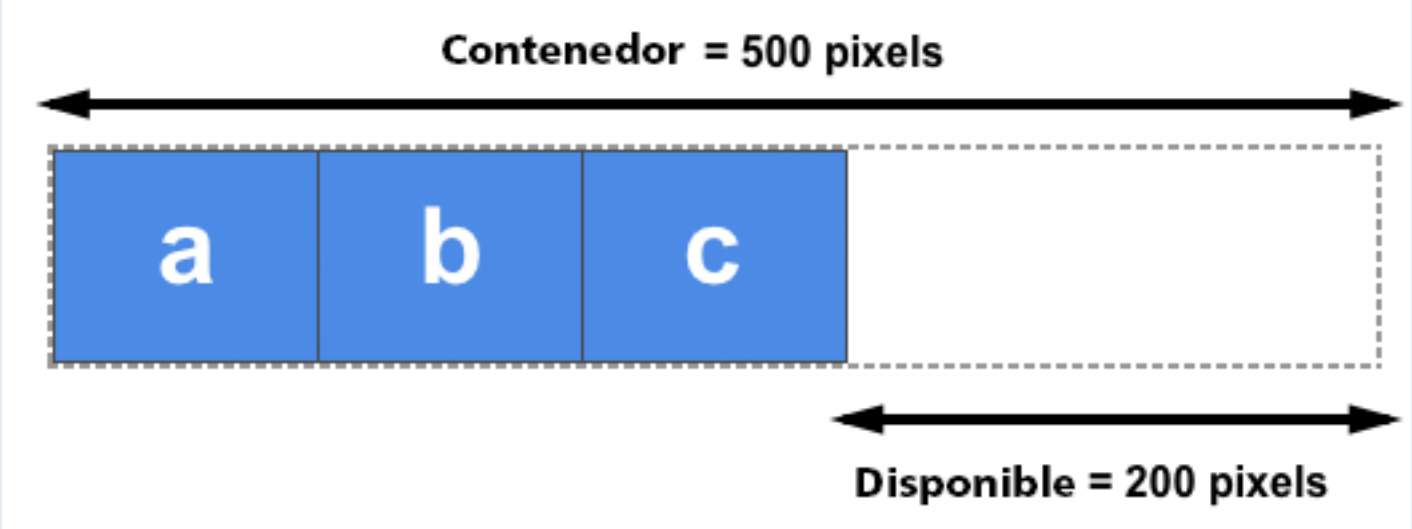
Propiedades aplicadas a los ítems flex

align-self:

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 aplicadas a los ítems flex

Si tenemos tres ítems con un ancho de 100 pixeles en un contenedor de 500 pixeles de ancho, entonces el espacio que se necesita para colocar nuestros ítems es de 300 pixeles. Esto deja 200 pixeles de espacio disponible. Si no cambiamos los valores iniciales entonces flexbox colocará ese espacio después del último ítem.



Si en cambio quisiéramos que los ítems crecieran para llenar ese espacio, entonces necesitaríamos un método para distribuir el espacio sobrante entre los ítems. Es justo lo que harán las propiedades flex que aplicaremos a dichos ítems.

Propiedades aplicadas a los ítems flex

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 del ítem respecto al resto.
flex-shrink:	**1** | *[factor de decrecimiento]*	Número que indica el decrecimiento del ítem respecto al resto.
flex-basis:	*[tamaño base]* | **content**	Tamaño base de los ítems antes de aplicar variación.
order:	*[número]*	Número que indica el orden de aparición de los ítems.

Propiedades aplicadas a los ítems flex

flex-grow

Indica el factor de crecimiento de los ítems en el 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.

Propiedades aplicadas a los ítems flex

flex-shrink

Es la complementaria a flex-grow. Mientras que la anterior indica un factor de crecimiento, flex-shrink hace justo lo contrario, aplica un factor 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 como funciona la propiedad flex-grow.

Propiedades aplicadas a los ítems flex

flex-basis

Define el tamaño por defecto (de base) que tendrán los ítems antes de aplicarle la distribución de espacio. Generalmente, se aplica un tamaño (unidades, porcentajes, etc...), pero también se puede aplicar la palabra clave content que ajusta automáticamente el tamaño al contenido del ítem, que es su valor por defecto.

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

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

Propiedades aplicadas a los ítems flex

order

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.

De esta forma podemos recolocar fácilmente los ítems incluso utilizando media queries o responsive design.

Ejercicios Flexbox

https://webdesign.tutsplus.com/es/tutorials/exercises-in-flexbox-simple-web-components--cms-28049
https://desarrolloweb.com/articulos/flexbox-align-items.html
https://mape.neocities.org/flexbox.html

Grid

Uno de los procesos más problematicos y frustrantes de CSS, sobre todo para principiantes, es el proceso de colocar y distribuir los elementos a lo largo de una página. Mecanismos como posicionamiento, floats o elementos en bloque o en línea, suelen ser insuficientes (o muy complejos) para crear un layout o estructuras para páginas web actuales.

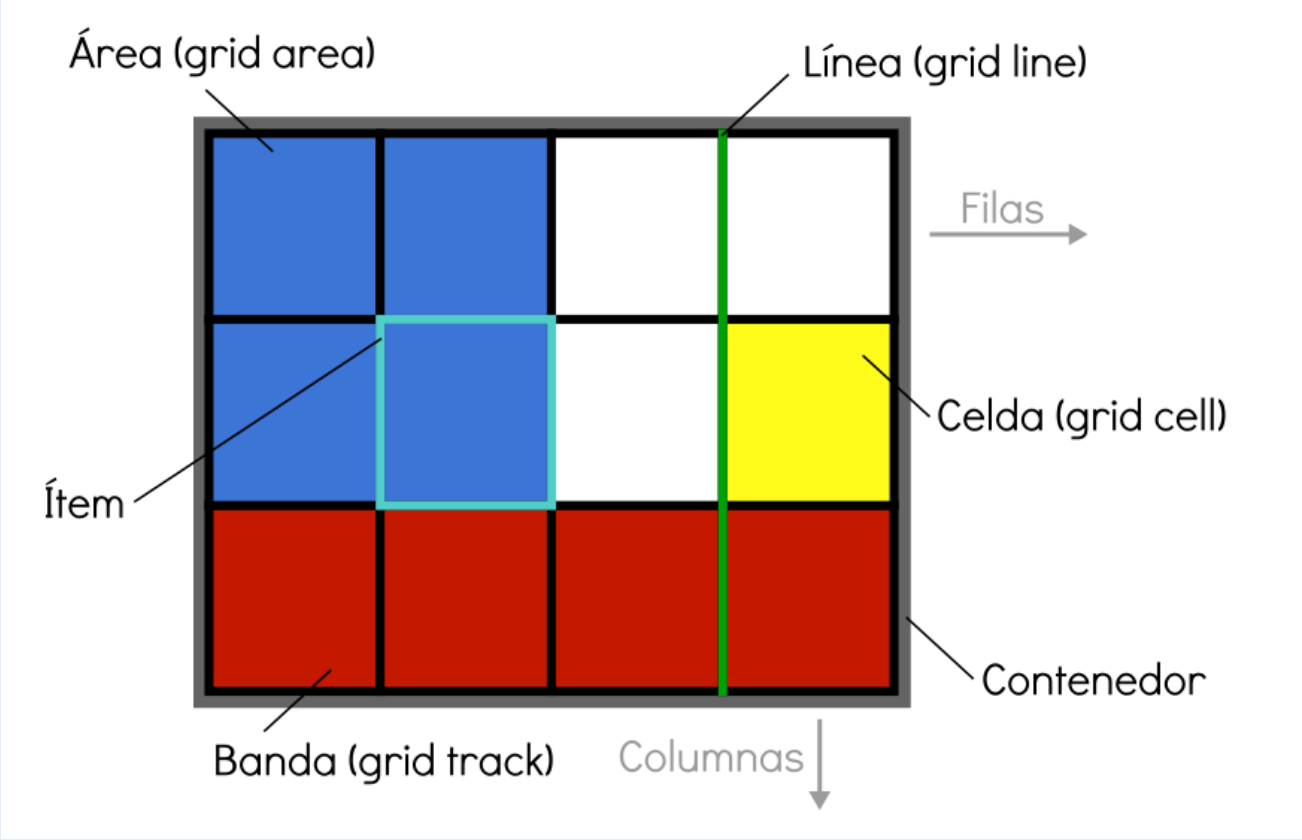
El sistema flexbox es una gran mejora, sin embargo, está orientado a estructuras de una sola dimensión, por lo que aún necesitamos algo más potente para estructuras web. Con el paso del tiempo, muchos frameworks y librerías utilizan un sistema grid donde definen una cuadrícula determinada, y modificando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

Grid CSS nace de esa necesidad, y recoge las ventajas de ese sistema, añadiendole numerosas mejoras y características que permiten crear rápidamente cuadrículas sencillas y potentes.

Grid

Conceptos

Para utilizar Grid CSS necesitaremos tener en cuenta una serie de conceptos que utilizaremos a partir de ahora y que definiremos a continuación:



Grid

- **Contenedor**: Existe un elemento padre que es el 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.

Grid

Para utilizar cuadriculas Grid CSS, trabajaremos bajo este código HTML:

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>
```

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 como 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.

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.

Propiedades del elemento contenedor

Grid con filas y columnas explícitas

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

Conociendo estas dos propiedades, escribamos el siguiente código CSS:

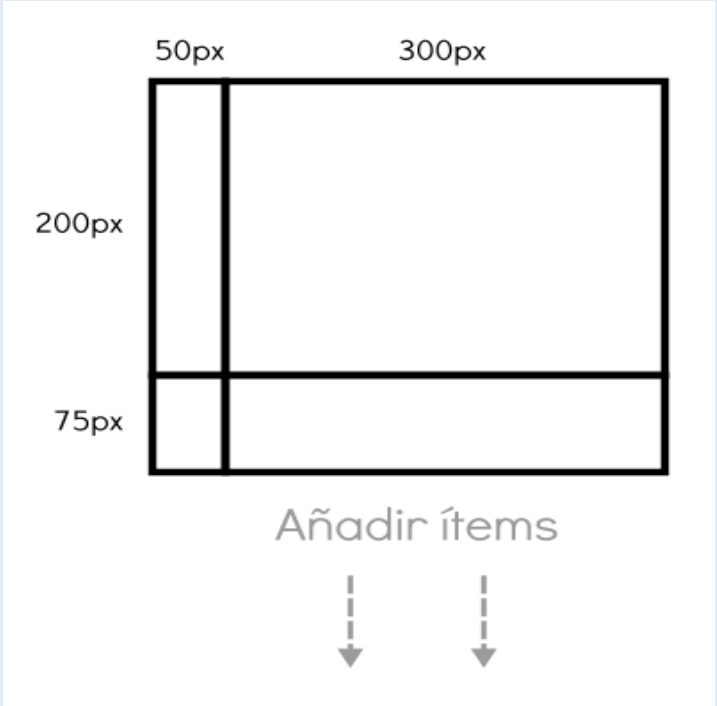
CSS

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

Propiedades del elemento contenedor

Grid con filas y columnas explícitas

Esto significa que tendremos una cuadricula 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.



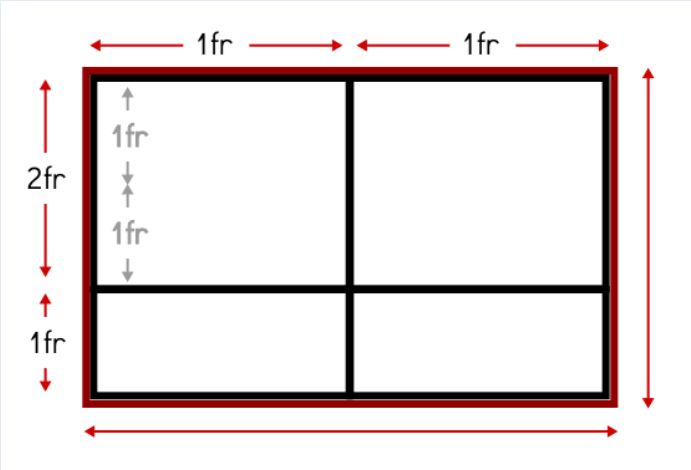
Propiedades del elemento contenedor

Grid con filas y columnas explícitas

En este ejemplo he utilizado píxels como unidades de las celdas de la cuadrícula, sin embargo, también podemos utilizar otras unidades (e incluso 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. Veamos un código de ejemplo en acción:

CSS

```
.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.

Propiedades del elemento contenedor

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. La expresión a utilizar sería la siguiente: repeat([núm de veces], [valor o valores]). Veamos un ejemplo:

CSS

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

Asumiendo que tuvieramos 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.

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:

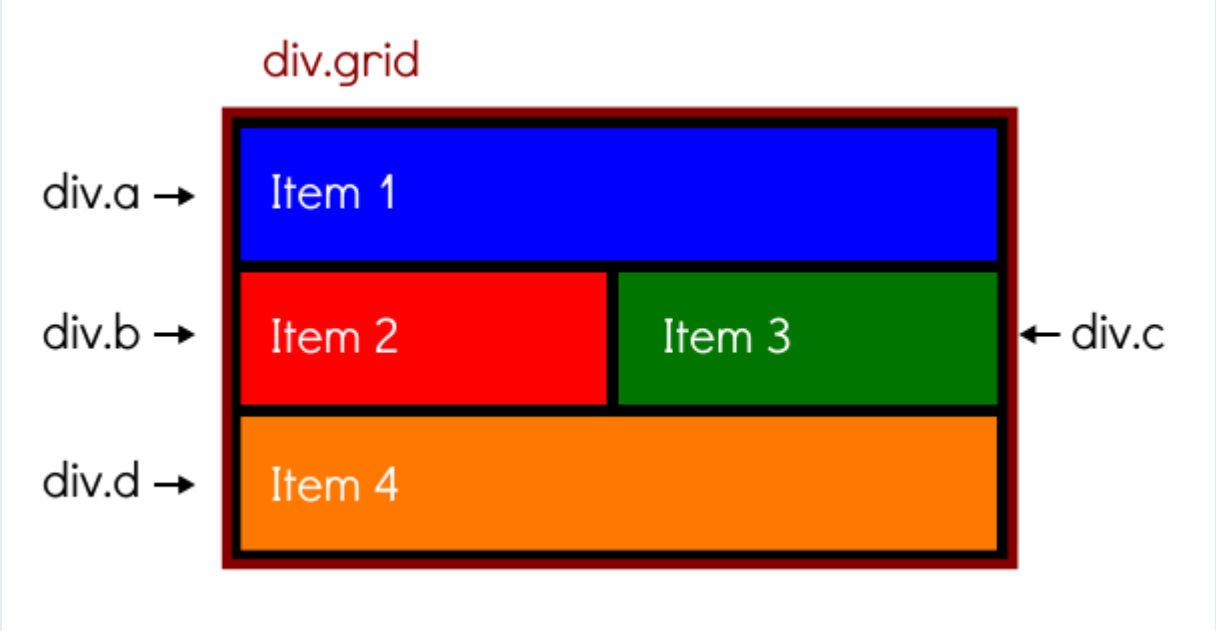
CSS

```
.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 }
```

Grid por áreas

Aplicando este código, conseguiríamos una cuadrícula donde:

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



Grid por áreas

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.

Grid con huecos

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

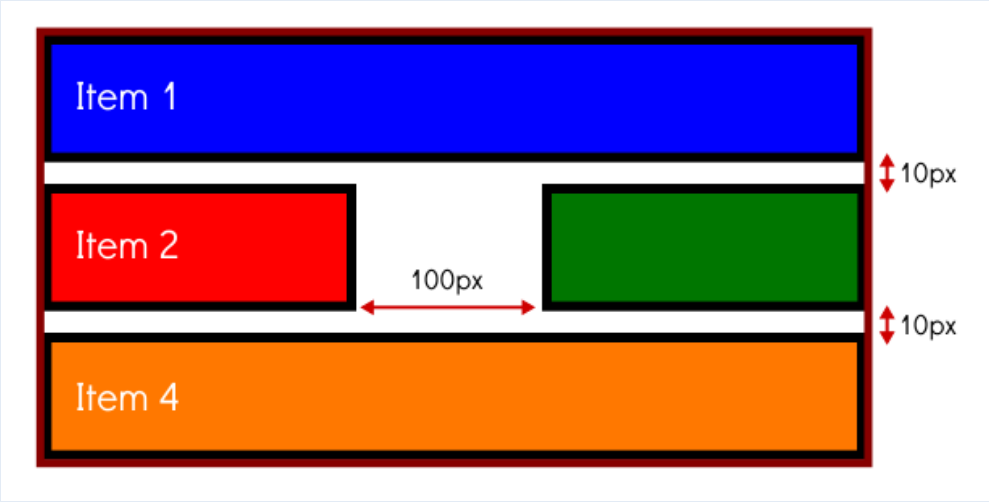
Grid con huecos

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:

CSS

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

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



Grid con huecos

Atajo: Grid con huecos

Existe una propiedad de atajo para las propiedades grid-column-gap y grid-row-gap y no tener que indicarlas por separado. La propiedad en cuestión sería grid-gap y se utiliza de la siguiente forma:

CSS

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

Posición de los elementos 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 conocemos de 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.

Estas propiedades se aplican sobre el elemento contenedor padre, pero afectan a los ítems hijos, por lo que actuan 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.

Posición de los elementos en el grid

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
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.

Posición de los elementos en el grid

```
15    justify-items:stretch;
16    align-items:stretch;
17
18    justify-content:stretch;
19    align-content:stretch;
20  }
21
22  .a { grid-area: head; background:blue }
23  .b { grid-area: menu; background:red }
24  .c { grid-area: main; background:green }
25  .d { grid-area: foot; background:orange }
```

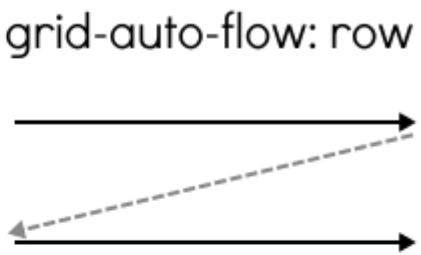
Ajuste automático de celdas

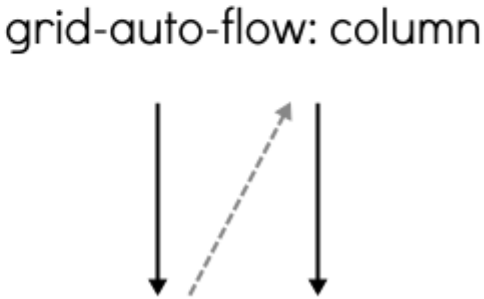
Es posible utilizar las propiedades grid-auto-columns y grid-auto-rows para darle un tamaño automático a las celdas de la cuadrícula. Para ello, sólo hay que especificar el tamaño deseado en cada una de las propiedades. Además, también podemos utilizar grid-auto-flow para indicar el flujo de elementos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

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

Ajuste automático de celdas

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





Ajuste automático de celdas

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:

CSS

```
.grid {
  /* grid: <grid-template> <grid-auto-flow> <grid-auto-rows> / <grid-auto-columns> */
  /* EJEMPLOS*/
  grid: 100px 20px;
  grid: 200px repeat(2, 100px) 300px;
  grid: row;
  grid: column dense;
  grid: row 200px;
  grid: row 400px / 150px;
}
```

Propiedades para ítems grid hijos

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.

Algunas de las propiedades vistas hasta ahora son las siguientes:

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:

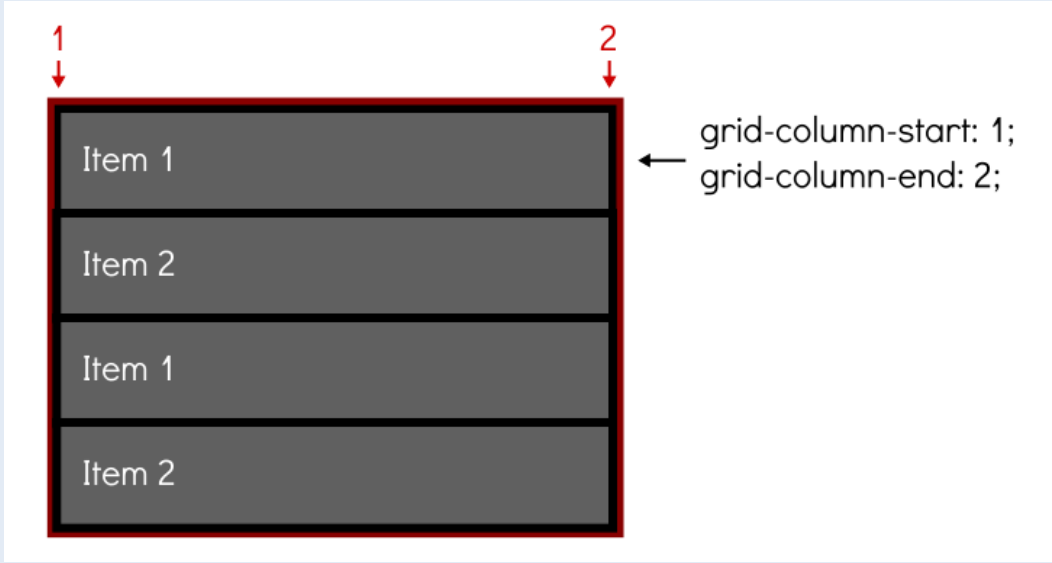
Propiedad	Descripción
grid-column-start	Indica en que columna empezará el ítem de la cuadrícula.
grid-column-end	Indica en que columna terminará el ítem de la cuadrícula.
grid-row-start	Indica en que fila empezará el ítem de la cuadrícula.
grid-row-end	Indica en que fila terminará el ítem de la cuadrícula.

Propiedades para ítems grid hijos

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

CSS

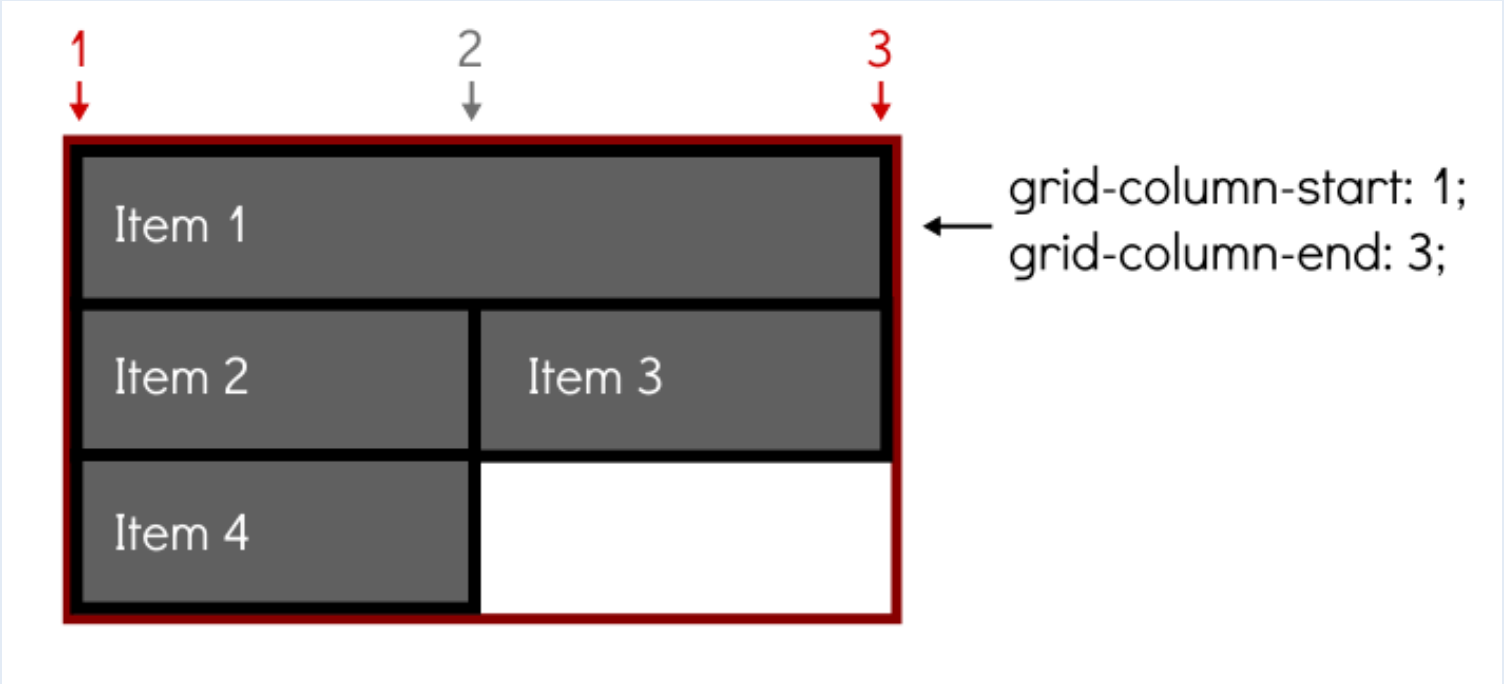
```
.grid {
  display:grid;
}

.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 HTML con clase .a): comenzando en la columna 1 y acabando en el inicio de la columna 2:

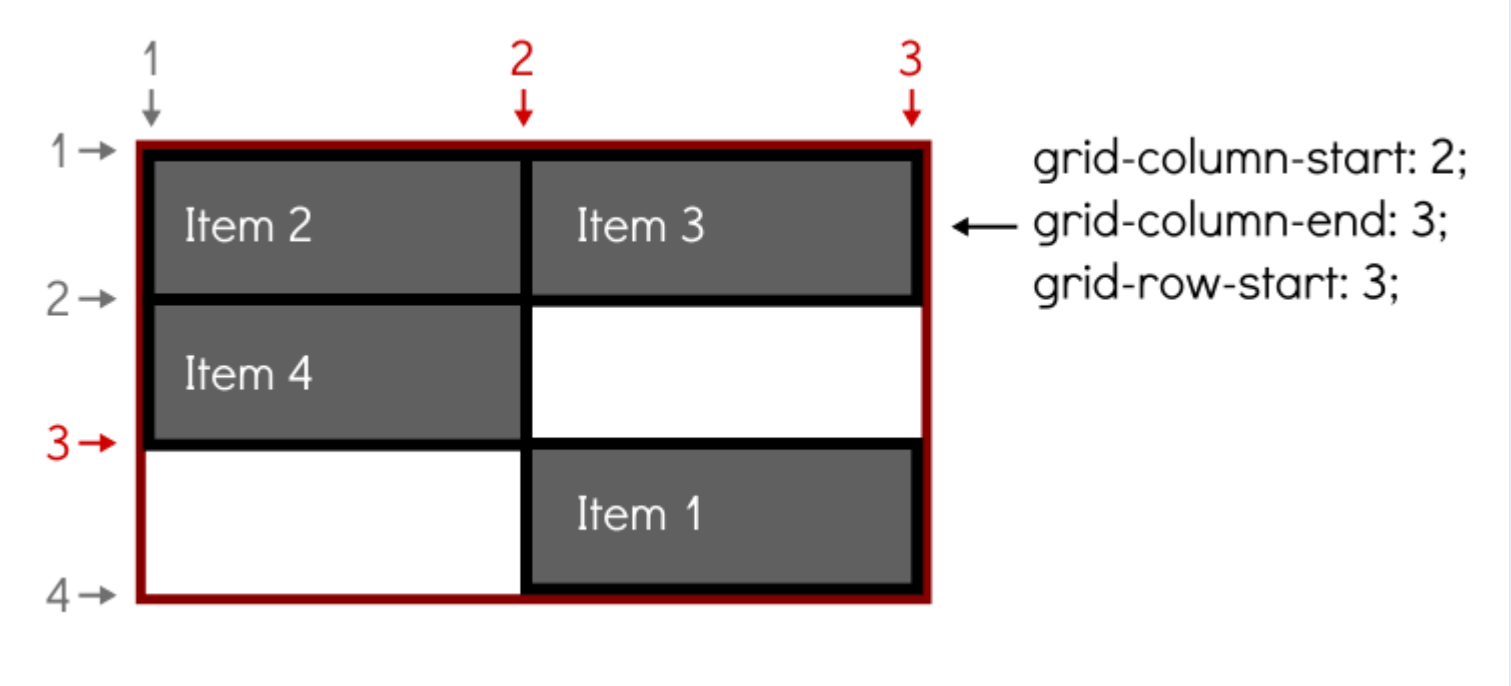
Propiedades para ítems grid hijos

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):



Propiedades para ítems grid hijos

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.

Propiedades para ítems grid hijos

Atajo: grid-column y grid-row

El módulo grid de CSS proporciona las propiedades de atajo grid-column y grid-row donde se nos permite escribir en un formato abreviado las propiedades anteriores. Su sintaxis sería la siguiente:

CSS

```
.grid {
  display: grid;
}
.a {
  /* grid-column: <grid-column-start> <grid-column-end> */
  /* grid-row: <grid-row-start> <grid-row-end> */
  grid-column: auto;
  grid-column: 4 / 6;
  grid-column: span 3;
  grid-column: span 3 / 6;
}
```

Ejercicios Grid CSS:
https://webdesign.tutsplus.com/es/tutorials/solving-problems-with-css-grid-and-flexbox-the-card-ui--cms-27468
https://gridbyexample.com/examples/

Bibliografía Flexbox y Grid

* Toda la información de este manual ha sido recopilada de varios artículos publicados en internet, destacando:
https://lenguajecss.com/p/css/propiedades/grid-css
https://lenguajecss.com/p/css/propiedades/flexbox
https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Conceptos_Basicos_de_Flexbox

Videotutoriales
Rock' n' Grid - Diana Aceves WeCodeFest 2018
Taller de Flexbox --- Diana Aceves