**Universidad Mariano Gálvez de Guatemala**

**Ingeniería en sistemas**

**Desarrollo Web.**

**Sección A.**

**Conny Barrios 0901-15-4555.**

**FlexBox y Grid CSS**

# Introducción

Tradicionalmente, en CSS se ha utilizado el posicionamiento (static, relative, absolute...), los elementos en línea o en bloque (y derivados) o los float, lo que a grandes rasgos no dejaba de ser un sistema de creación de diseños bastante tradicional que no encaja con los retos que tenemos hoy en día: sistemas de escritorio, dispositivos móviles, múltiples resoluciones, etc...

Uno de los procesos más problemáticos y frustrantes de CSS, sobre todo para novatos o 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.

Índice

[Introducción 2](#_Toc18505699)

[FlexBox 4](#_Toc18505700)

[Conceptos 4](#_Toc18505701)

[Dirección de los ejes 5](#_Toc18505702)

[Atajo: Dirección de los ejes 7](#_Toc18505703)

[Propiedades de alineación de ítems 7](#_Toc18505704)

[Sobre el eje principal 8](#_Toc18505705)

[Sobre el eje secundario 11](#_Toc18505706)

[Propiedades de ítems hijos 12](#_Toc18505707)

[Atajo: Propiedades de ítems hijos 13](#_Toc18505708)

[Orden de los ítems 14](#_Toc18505709)

[Práctica 14](#_Toc18505710)

[Grid CSS 15](#_Toc18505711)

[Conceptos 15](#_Toc18505712)

[Grid con filas y columnas 16](#_Toc18505713)

[fr: Unidad fracción restante 18](#_Toc18505714)

[Filas y columnas respectivas 19](#_Toc18505715)

[Grid por áreas 20](#_Toc18505716)

[Grid con huecos 22](#_Toc18505717)

[Atajo: Grid con huecos 23](#_Toc18505718)

[Posición en el grid 23](#_Toc18505719)

[Ajuste automático de celdas 24](#_Toc18505720)

[Atajo: Grid 25](#_Toc18505721)

[Propiedades para ítems hijos 26](#_Toc18505722)

[Atajo: grid-column y grid-row 28](#_Toc18505723)

[Práctica 29](#_Toc18505724)

[Conclusión 30](#_Toc18505725)

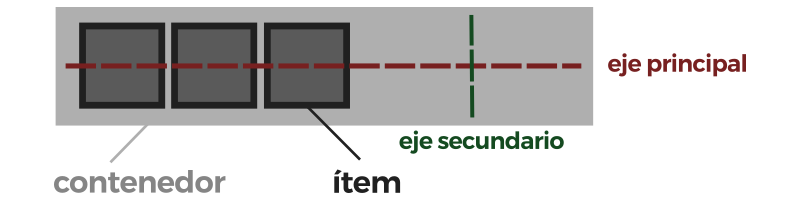
[Bibliografía 31](#_Toc18505726)

# FlexBox

Es un sistema de elementos flexibles que llega con la idea de olvidar estos mecanismos y 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 sola dimensión.

## Conceptos

Para utilizar flexbox debemos 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 horizontal (fila).
  + Eje secundario: De la misma forma, los contenedores flexibles tendrán una orientación secundaria, perpendicular a la principal. Si la principal es horizontal, la secundaria será vertical, y viceversa.
* Ítem: Cada uno de los hijos flexibles que tendrá el contenedor en su interior.

Para activar este modo hay que utilizar sobre el elemento contenedor la propiedad display y especificar el valor flex o inline-flex dependiendo de cómo queramos que se comporte el contenedor: si como un elemento en línea, o como un elemento en bloque.

|  |  |
| --- | --- |
| Tipo de elemento | Descripción |
| inline-flex | Establece un contenedor de ítems flexible en línea, de forma equivalente a inline-block. |
| flex | Establece un contenedor de ítems flexible en bloque, de forma equivalente a block. |

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 desbordarán ni mostrarán los problemas que, por ejemplo, tienen los porcentajes sobre elementos que no utilizan flexbox.

## 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 y 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 (multilinea). |

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. 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.

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. |

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 línea (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

Recuerda que 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:

/\* 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. Las cuatro propiedades son las siguientes:

|  |  |  |
| --- | --- | --- |
| 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:

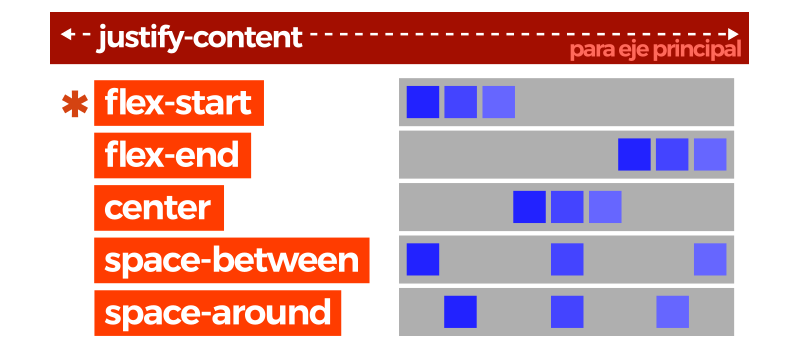
* justify-content: Se utiliza para alinear los ítems del eje principal (por defecto, el horizontal).
* align-items: Usada para alinear los ítems del eje secundario (por defecto, el vertical).

### 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 de cada uno de ellos. |

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

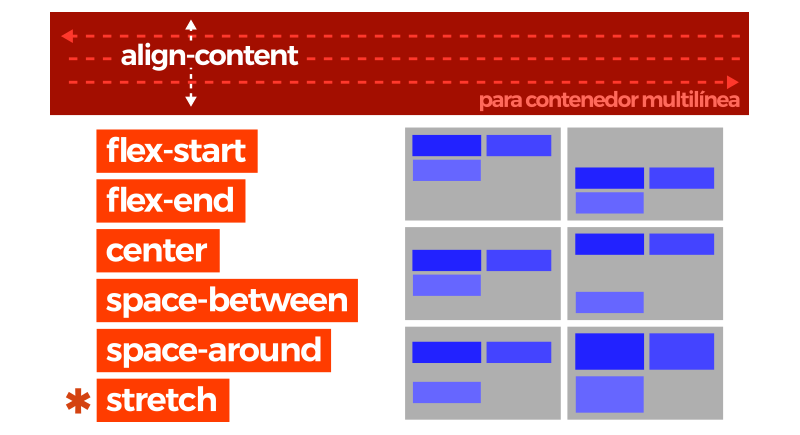


Una vez entendido este caso, debemos atender a la propiedad align-content, que es un caso particular del anterior. Nos servirá cuando estemos tratando con un contenedor flex multilinea, que es un contenedor en el que los ítems no caben en el ancho disponible, y por lo tanto, el eje principal se divide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

De esta forma, align-content servirá para alinear cada una de las líneas 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. |

Con estos valores, vemos como cambiamos la disposición en vertical de los ítems que están dentro de un contenedor multilinea.

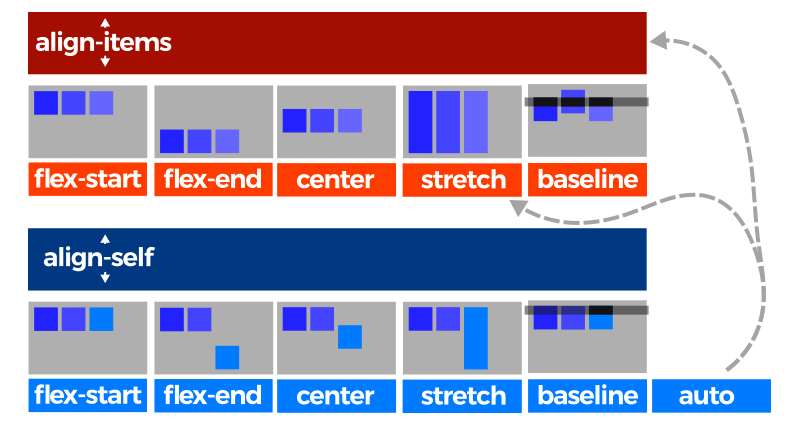


### Sobre el eje secundario

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, 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 el final del contenedor. |
| baseline | Alinea los ítems en el contenedor según la base del contenido de los ítems 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, lo hereda de 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.

|  |  |  |
| --- | --- | --- |
| 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 | content | Tamaño base de los ítems antes de aplicar variación. |
| order: | 0 | [número] | Número (peso) que indica el orden de aparición de los ítems. |

En primer lugar, tenemos la propiedad flex-grow para indicar el factor de crecimiento de los ítems en el caso de que no tengan un ancho específico.

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 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 cómo funciona la propiedad flex-grow.

Por último, tenemos la propiedad flex-basis, que 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.

### Atajo: Propiedades de ítems hijos

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

/\* 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.

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

## Práctica

Se dejara el link del repositorio en GitHub.

<https://github.com/Connybm/Laboratorios/tree/master/Flexbox>

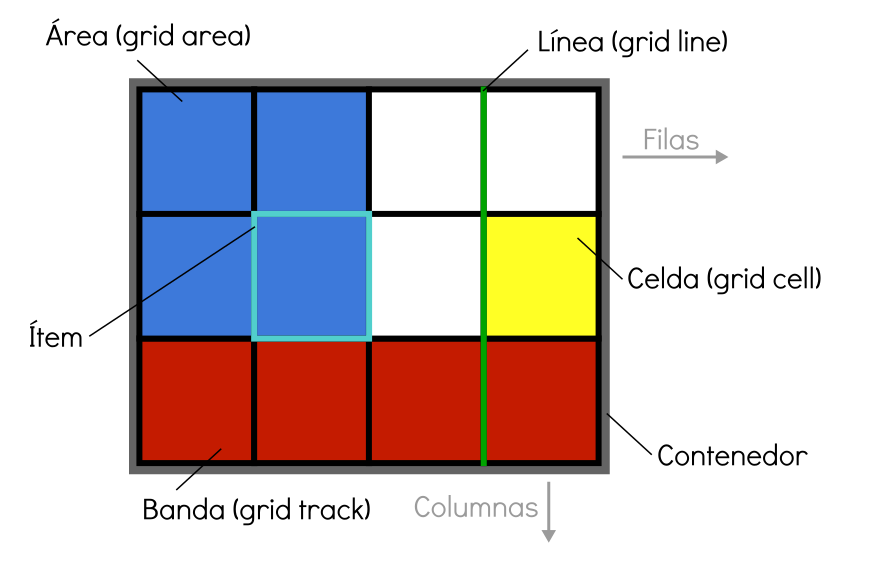
# Grid CSS

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 de forma prácticamente instantánea.

## Conceptos

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



* 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.

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 o debajo del contenido exterior (en bloque) y el segundo de ellos permite que la cuadrícula aparezca a la izquierda o 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 |
| grid-template-columns | [c1] [c2] ... | Establece el TAMAÑO de cada columna (col 1, col 2...). |
| grid-template-rows | [f1] [f2] ... | Establece el TAMAÑO de cada fila (fila 1, fila 2...). |

### fr: Unidad fracción restante

Las unidades de medida pueden ir desde píxeles como unidades de las celdas de la cuadrícula, 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.

#### Nota

Se pueden combinar varias unidades diferentes, pudiendo utilizar píxeles (px) y fracciones restantes (fr), porcentajes (%) y fracciones restantes (fr) o combinaciones similares.

## Filas y columnas respectivas

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 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. |

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). |

### 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-gap: <row-gap> <column-gap> \*/

grid-gap: 20px 80px;

/\* grid-gap: <row-column-gap> \*/

grid-gap: 40px;

/\* Equivalente a grid-gap: 40px 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 | Valor | 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.

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

## 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 | Valor | 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). |

## Propiedades para ítems hijos

Las propiedades que 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. |

### 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.

La especificación de Grid CSS ya tiene buen soporte en la actualidad y puede utilizarse en producción.

## Práctica

Se dejara el link del repositorio en GitHub.

<https://github.com/Connybm/Laboratorios/tree/master/GridCSS>

# Conclusión

En conclusión, el uso de Flexbox para creación de layouts es una técnica muy flexible y eficaz. Sin embargo, es más aconsejable para diseños a pequeña escala, mientras que el uso de Grid, está más enfocado a diseños a mayor escala.

La especificación de diseño de CSS Grid nos permite crear diseños de diseño web complejos con facilidad, mientras escribimos código CSS más simple y fácil de mantener. Ya no tenemos que usar flotante u otras propiedades similares al crear diseños complejos. Las posibilidades son infinitas.

# Bibliografía

* <https://lenguajecss.com/p/css/propiedades/flexbox>
* <https://lenguajecss.com/p/css/propiedades/grid-css#top>