

2023/24

# U.D.3. USO DE ESTILOS.

## 3.4 Maquetación de contenido y preprocesadores CSS.

```
343 .widget-area-sidebar { font-size: 13px; }
344 .widget-area-sidebar input { font-size: 13px; }
345 .widget-area-sidebar input[type="text"] { font-size: 13px; }
346 .widget-area-sidebar input[type="button"] { font-size: 13px; }
347 .widget-area-sidebar button { font-size: 13px; }
348 .widget-area-sidebar font-size: 13px;
349 }
350
351 /* =Menu */
352
353 #access {
354 display: inline-block;
355 height: 69px;
356 float: right;
357 margin: 11px 28px 0px 0px;
358 max-width: 800px;
359 }
360
361 #access ul {
362 font-size: 13px;
363 list-style: none;
364 margin: 0 0 0 -0.8125em;
365 padding-left: 0;
366 z-index: 99999;
367 text-align: right;
368 }
369
370 #access li {
371 display: inline-block;
372 float: right;
373 margin-left: 0;
```





## ÍNDICE

<b>1. Flexbox .....</b>	<b>4</b>
1.1 Conceptos básicos y terminología.....	4
1.2 Propiedades Flexbox.....	6
1.2.1 Propiedades del padre o ‘flex container’ .....	7
1.2.1.1 Display .....	7
1.2.1.2 flex-direction.....	10
1.2.1.3 flex-wrap .....	12
1.2.1.4 flex-flow.....	13
1.2.1.5 justify-content.....	14
1.2.1.6 align-items.....	16
1.2.1.7 align-content .....	18
1.2.1.8 gap.....	20
1.2.2 Propiedades de los hijos o ‘flex items’.....	21
1.2.2.1 order.....	21
1.2.2.2 flex-grow .....	22
1.2.2.3 flex-shrink.....	23
1.2.2.4 flex-basis.....	24
1.2.2.5 flex .....	25
1.2.2.6 align-self.....	25
1.2.3 EJEMPLO PRÁCTICO .....	27
<b>2. CSS Grid .....</b>	<b>29</b>
2.1 Terminología importante de CSS Grid.....	30
2.2 Propiedades del padre o grid container .....	33
2.2.1 ‘display’ .....	34
2.2.2 ‘grid-template-columns’ y ‘grid-template-rows’ .....	36
2.2.3 ‘grid-template-areas’ .....	39
2.2.4 ‘grid-template’ .....	40
2.2.5 ‘row-gap’ y ‘column-gap’.....	42
2.2.6 ‘gap’ .....	42
2.2.7 ‘justify-items’ .....	43



2.2.8	'align-items' .....	45
2.2.9	'place-items' .....	47
2.2.10	'justify-content'.....	48
2.2.11	'align-content' .....	50
2.2.12	'place-content' .....	52
2.2.13	'grid-auto-columns' y 'grid-auto-rows' .....	53
2.2.14	'grid-auto-flow' .....	55
2.2.15	'grid'.....	56
2.3	Propiedades de los hijos o grid items .....	57
2.3.1	'grid-column-start', 'grid-column-end', y 'grid-column' .....	58
2.3.2	'grid-row-start', 'grid-row-end' y 'grid-row' .....	59
2.3.3	'grid-area' .....	61
2.3.4	'justify-self'.....	62
2.3.5	'align-self' .....	64
2.3.6	'place-self' .....	66
<b>3.</b>	<b>Flexbox vs. CSS Grid.....</b>	<b>67</b>
3.1	Diferencias entre Flexbox y Grid .....	67
3.2	¿Cuál utilizar?.....	70
3.2.1	Grid: usos más frecuentes .....	70
3.2.2	Flexbox: usos más frecuentes.....	72
3.3	Combinar Flexbox y Grid .....	76
<b>4.</b>	<b>Maquetación multicolumna .....</b>	<b>82</b>
<b>5.</b>	<b>Preprocesadores CSS Less y Sass.....</b>	<b>85</b>
5.1	Preprocesadores CSS vs PostCSS.....	85
<b>6.</b>	<b>Anexo I - Centrar horizontal y verticalmente en CSS un elemento.....</b>	<b>86</b>
6.1	Centrar vertical y horizontalmente elementos con Flexbox .....	86
6.2	Centrar vertical y horizontalmente un texto con line-height y text-align.....	87
6.3	Centrar vertical y horizontalmente una imagen o contenedor con propiedad transform y posición absoluta.....	88
6.4	Centrar un elemento vertical y horizontalmente utilizando posición absoluta y margen negativo.....	89
6.5	Centrar un elemento con Grid .....	91
6.6	Alinear un texto verticalmente con una imagen con vertical-align .....	92



6.7	Alinear verticalmente un texto en un div con vertical-align .....	93
6.8	Alinear horizontalmente una imagen con text-align .....	94
6.9	Alinear verticalmente un checkbox con un label.....	95
<b>7.</b>	<b>Anexo II - SaSS.....</b>	<b>96</b>
7.1	Instalación de SaSS .....	96
7.2	Formatos .....	97
7.3	Cómo compilar nuestro código .....	99
<b>8.</b>	<b>Anexo III – Chuleta Flex y Grid.....</b>	<b>101</b>



## 1. Flexbox

El módulo **Flexbox Layout** (Caja Flexible) (Recomendación Candidata del W3C a partir de octubre de 2017) tiene como objetivo proporcionar una forma más eficiente de **disponer, alinear y distribuir el espacio entre los elementos de un contenedor, incluso cuando su tamaño es desconocido y/o dinámico** (de ahí la palabra "flex").

La idea principal detrás del Flex Layout es dar al contenedor la capacidad de alterar la anchura/altura de sus elementos (y el orden) para cubrir mejor el espacio disponible (sobre todo para adaptarse a todo tipo de dispositivos de visualización y tamaños de pantalla). Un contenedor flex **expande** los elementos para llenar el espacio libre disponible o los **encoge** para evitar el desbordamiento (overflow).

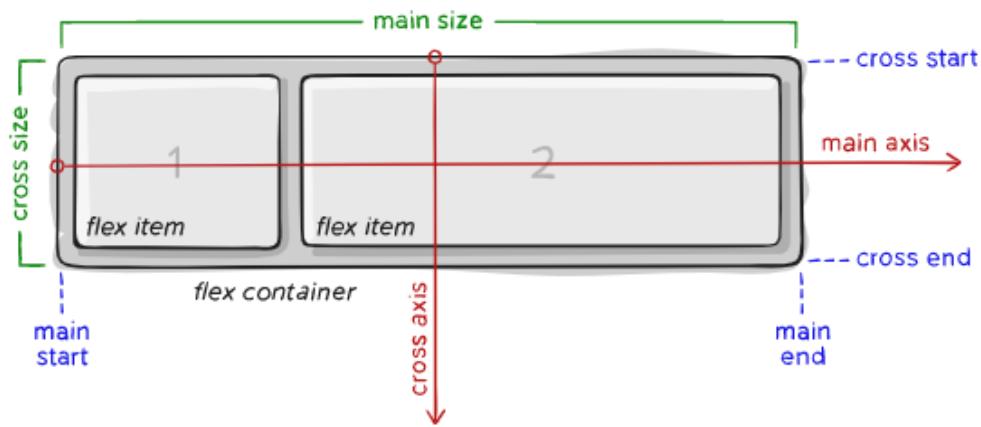
Lo más importante es que el diseño de flexbox es **agnóstico en cuanto a la dirección**, a diferencia de los diseños habituales (block, que se basa en la verticalidad, e inline, que se basa en la horizontalidad). Mientras que estos funcionan bien para las páginas, carecen de flexibilidad para soportar aplicaciones grandes o complejas (especialmente cuando se trata de cambiar la orientación, cambiar el tamaño, estirar, reducir, etc.).

En definitiva, Flexbox **facilita el diseño de una estructura web flexible y responsive** (adaptable) **sin necesidad de usar float o positioning**.

### 1.1 Conceptos básicos y terminología

Dado que Flexbox **es un módulo completo**, y no solo una propiedad, implica conocer todo su conjunto de **propiedades**. Algunas de ellas están pensadas para ser establecidas en el contenedor (elemento padre, conocido como "contenedor flex" o "flex container") mientras que las otras están pensadas para ser establecidas en los hijos (llamados "elementos flex" o "flex items").

Si un layout "normal" se basa en las direcciones de flujo en bloque y en línea, el flex layout **se basa en las "direcciones de flujo flexibles"** o "flex-flow directions".



**Los elementos se dispondrán siguiendo el eje principal o main axis (desde main-start a main-end) o el eje transversal o cross axis (desde cross-start a cross-end).**

- **Main axis o eje principal.** Es el eje principal sobre el cual se disponen los flex items. ¡Cuidado! No es necesariamente horizontal, ya que **depende de la propiedad flex-direction**.
- **Main-start | main-end.** Los elementos o items flex se colocan dentro del contenedor, empezando por main-start y llegando a main-end.
- **Main size.** La anchura (width) o la altura (height) de un elemento o item flex, cualquiera que sea la dimensión principal; es el tamaño principal del elemento flex.
- **Cross axis o eje transversal.** Es el eje perpendicular al eje principal. Su dirección depende de la dirección del eje principal.
- **Cross-start | cross-end.** Las líneas flex se llenan con ítems o hijos que se colocan en el contenedor empezando por cross-start hasta cross-end.
- **Cross size o tamaño transversal.** Es la anchura o altura de un elemento flex, cualquiera que sea la dimensión transversal. Es el tamaño transversal del elemento.

En este apartado se expondrán las principales propiedades de Flexbox. Si quieras conocer más a fondo este sistema puedes acceder a las siguientes guías:

- <https://lenguajecss.com/css/maquetacion-y-colocacion/flex/>
- <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
- [https://www.w3schools.com/css/css3\\_flexbox.asp](https://www.w3schools.com/css/css3_flexbox.asp)



## 1.2 Propiedades Flexbox

Flexbox fue diseñado como un **modelo unidimensional de layout**, y como un método que pueda ayudar a distribuir el espacio entre los ítems de una interfaz y mejorar las capacidades de alineación.

En Flexbox se necesita tener, forzosamente, un **contenedor padre (flex-container)** el cual va a contener **contenedores hijo (flex items)**.



Por tanto, vamos a tener **dos tipos de propiedades**: los que se pueden aplicar al **contenedor padre**, y los que se pueden aplicar a los **contenedores hijo**.

PROPIEDADES DEL CONTENEDOR PADRE	
<b>display</b>	Define un contenedor flex. Puede tomar los valores <i>flex</i> o <i>inline-flex</i> .
<b>flex-direction</b>	Define la dirección en la que se colocan los contenedores hijo.
<b>flex-wrap</b>	Indica qué elementos hijo se deben trasladar cuando no hay suficiente espacio en el contenedor.
<b>flex-flow</b>	Mezcla las propiedades <i>flex-direction</i> y <i>flex-wrap</i> .
<b>justify-content</b>	Define la justificación horizontal de los elementos hijo.
<b>align-items</b>	Alinea los ítems flex o contenedores hijo sobre el eje cruzado o transversal (vertical).
<b>align-content</b>	Alinea las filas de un contenedor flex cuando hay espacio extra en el eje transversal (o vertical). Es similar a <i>align-items</i> pero <b>no tiene efecto en cajas o ítems flex de una sola línea</b> .
<b>gap</b>	Controla el espacio que hay entre los ítems flex.



## PROPIEDADES DE LOS CONTENEDORES HIJO O FLEX ITEMS.

<b>order</b>	Especifica el orden utilizado para disponer los elementos hijo en su contenedor padre.
<b>flex-grow</b>	Define la capacidad de un item flex para crecer si es necesario.
<b>flex-shrink</b>	Permite definir cuánto se puede hacer más pequeño un ítem flex en proporción a los demás items o elementos hijo.
<b>flex-basis</b>	Especifica el tamaño inicial o por defecto que va a tener un elemento hijo.
<b>flex</b>	Concentra las propiedades de <b>flex-grow</b> , <b>flex-shrink</b> y <b>flex-basis</b> en una sola línea de código.
<b>align-self</b>	Permite anular o cambiar la alineación por defecto, o la especificada por <b>align-items</b> , de forma individual para cada ítem flex.

### 1.2.1 Propiedades del padre o ‘flex container’

A continuación, se describen las propiedades que se pueden aplicar al contenedor padre o ‘flex container’.

#### 1.2.1.1 Display

La propiedad ‘**display**’ define un **contenedor flex**. Este puede tomar los valores de **flex** (block) o **inline-flex**.

Al definir un contendor flex, **sus hijos directos se convierten directamente en ítems flex**.

El **resultado** de aplicar esta propiedad es **que todos los ítems flex se alinearán en una sola fila**.

**Hay que tener en cuenta que las columnas CSS no tienen efecto en un contenedor flex.**

Su sintaxis es la siguiente:

```
.container {  
    display: flex | inline-flex;  
}
```



A continuación, se presentan algunos **ejemplos**:

## 1) DIFERENCIA entre `flex` e `inline-flex`

```
<!DOCTYPE html>
<html>
<head>
<style>

.contenedor1{
    height: 200px;
    background: #212d40;
    display: flex;
}

.contenedor2{
    height: 200px;
    background: #212d40;
    display: inline-flex;
}

.caja {
    width: 100px;
    height: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff; }

</style>
</head>
<body>

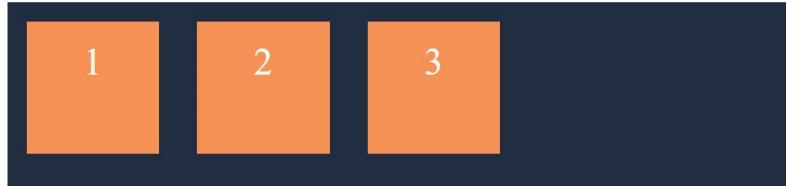
<h1>Display: flex; </h1>

<div class="contenedor1">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>

<h1>Display: inline-flex; </h1>
<div class="contenedor2">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>

</body>
</html>
```

**Display: flex;**



**Display: inline-flex;**



## 2) DIFERENCIA entre `flex` y `float:left`

**HTML:**

```
<h2>Ejemplo para ver diferencias entre display: flex y
float:left</h2>
<h2>Display: flex</h2>
<div id="flex-container">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
</div>
```



```
<h2>Float container</h2>
<div class="float-container">
    <div>1</div>
</div>
<div class="float-container">
    <div>2</div>
</div>
<div class="float-container">
    <div>3</div>
</div>
<div class="float-container">
    <div>4</div>
</div>
<div class="float-container">
    <div>5</div>
</div>
```

## CSS:

```
#flex-p{
    display: flex;
}
#flex-p p{
    margin: 10px;
}
#flex-container{
    display: flex;
}
#flex-container div{
    width: 300px;
    height: 150px;
    color: #FFF;
    text-align: center;
    line-height: 150px;
    margin: 10px;
    font-size: 20px;
    font-family: Arial;
    background-color: #C0392B;
}
```

```
.float-container{
    float: left;
    box-sizing: border-box;
}
.float-container div{
    height: 150px;
    width: 300px;
    margin: 10px;
    background-color: #2980B9;
    color: #FFF;
    text-align: center;
    line-height: 150px;
    font-size: 20px;
    font-family: Arial;
}
```





Ejemplo para ver diferencias entre display: flex y float:left

Display: flex



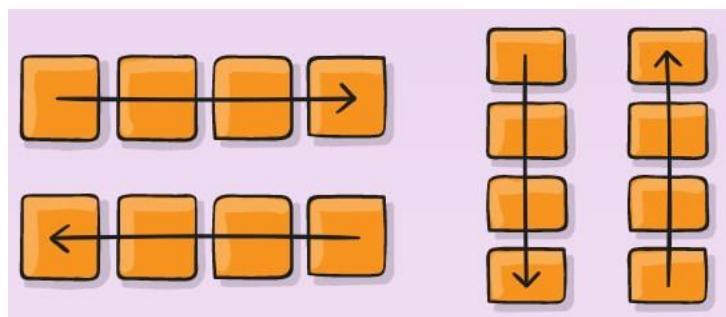
Float container



### 1.2.1.2 flex-direction

La propiedad '**flex-direction**' establece el eje principal, definiendo así la **dirección en la que se colocan los contenedores hijo o flex ítems**.

Flexbox es un **concepto de diseño en una sola dirección**. Piensa que los elementos flex se colocan principalmente en **filas horizontales o en columnas verticales**.



Su **sintaxis** es la siguiente:

```
.container {  
    flex-direction: row | row-reverse | column | column-reverse;  
}
```



- **row**: es el valor que toma por defecto. Los flex ítems se colocan de izquierda a derecha.
- **row-reverse**. Los flex ítems se colocan de derecha a izquierda.
- **column**: Los flex ítems se colocan de arriba abajo.
- **column-reverse**: Los flex-ítems se colocan de abajo a arriba.

A continuación, se presentan algunos **ejemplos**:

```
<!DOCTYPE html>
<html>
<head>
<style>

.contenedor1{
    height: 200px;
    background: #212d40;
    display: flex;
    flex-direction: row-reverse;
}

.contenedor2{
    height: 500px;
    max-width: 200px;
    background: #212d40;
    display: flex;
    flex-direction: column;
}

.contenedor3{
    height: 500px;
    max-width: 200px;
    background: #212d40;
    display: flex;
    flex-direction: column-reverse;
}

.caja {
    width: 100px;
    height: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff; }

</style>
</head>
<body>

<h1>flex-direction: row-reverse; </h1>

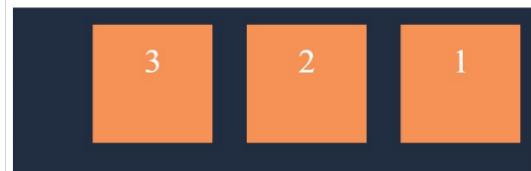
<div class="contenedor1">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>

<h1>flex-direction: column; </h1>
<div class="contenedor2">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>

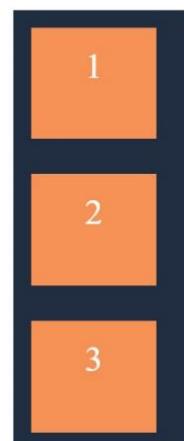
<h1>flex-direction: column-reverse; </h1>
<div class="contenedor3">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>

</body>
</html>
```

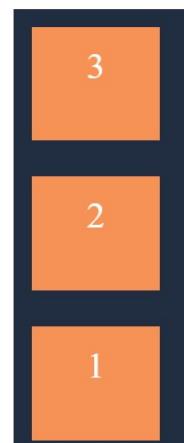
**flex-direction: row-reverse;**



**flex-direction: column;**



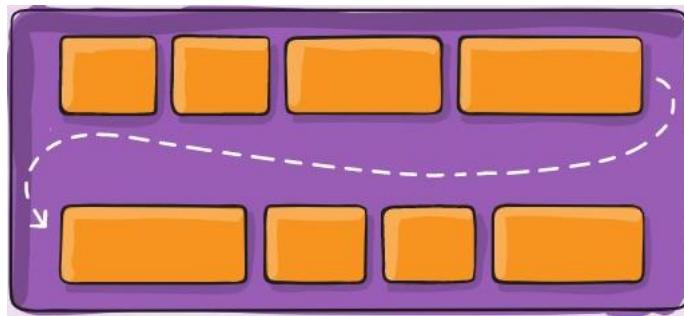
**flex-direction: column-reverse;**





### 1.2.1.3 flex-wrap

Por defecto, los ítems (o contenedores hijo) tratarán de encajar en una línea. Debido a que pueden ser demasiados para una sola línea, podemos **repartirlos** en las líneas siguientes con la propiedad **flex-wrap**.



Su sintaxis es la siguiente:

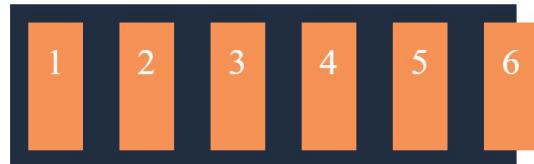
```
.container {  
    flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

- **nowrap** (valor por defecto): todos los ítems flex **se contraerán** para coger en la **misma línea**. Podrían salirse del margen si no pudieran contraerse.
- **wrap**: los ítems flex se distribuirán en varias líneas, de arriba a abajo.
- **wrap-reverse**: los ítems flex se distribuirán en varias líneas, de abajo a arriba.

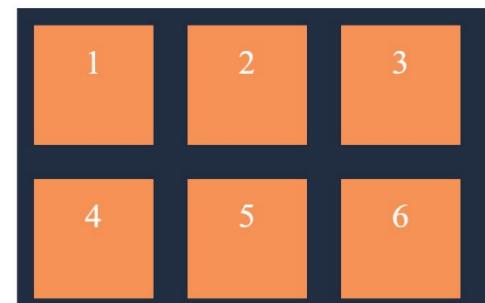
A continuación, se presentan algunos **ejemplos**:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
  
.contenedor1{  
    width: 80%;  
    max-width: 700px;  
    height: 100%;  
    background: #212d40;  
    display: flex;  
    flex-direction: row;  
    flex-wrap: nowrap;  
}  
  
.contenedor2{  
    width: 80%;  
    max-width: 700px;  
    height: 100%;  
    background: #212d40;  
    display: flex;  
    flex-direction: row;  
    flex-wrap: wrap;  
}
```

**flex-wrap: nowrap;**



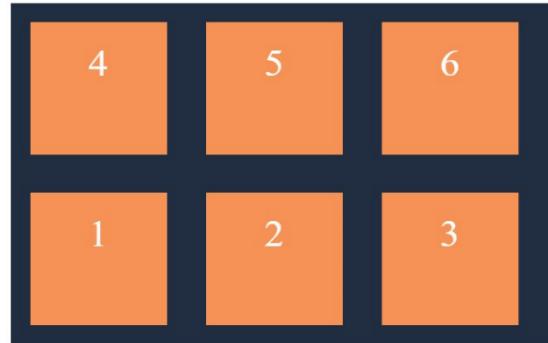
**flex-wrap: wrap;**





```
.contenedor3{  
    width: 80%;  
    max-width: 700px;  
    height: 100%;  
    background: #212d40;  
    display: flex;  
    flex-wrap: wrap-reverse;  
}  
  
.caja {  
    width: 100px;  
    height: 100px;  
    background: #f79256;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 40px;  
    color: #fff;  
}  
  
</style>  
</head>  
<body>  
  
<h1>flex-wrap: nowrap; </h1>  
<div class="contenedor1">  
    <div class="caja">1</div>  
    <div class="caja">2</div>  
    <div class="caja">3</div>  
    <div class="caja">4</div>  
    <div class="caja">5</div>  
    <div class="caja">6</div>  
</div>  
  
<h1>flex-wrap: wrap; </h1>  
<div class="contenedor2">  
    <div class="caja">1</div>  
    <div class="caja">2</div>  
    <div class="caja">3</div>  
    <div class="caja">4</div>  
    <div class="caja">5</div>  
    <div class="caja">6</div>  
</div>  
  
<h1>flex-wrap: wrap-reverse; </h1>  
<div class="contenedor3">  
    <div class="caja">1</div>  
    <div class="caja">2</div>  
    <div class="caja">3</div>  
    <div class="caja">4</div>  
    <div class="caja">5</div>  
    <div class="caja">6</div>  
</div>  
  
</body>  
</html>
```

flex-wrap: wrap-reverse;



#### 1.2.1.4 flex-flow

**flex-flow** es la mezcla de las propiedades **flex-direction** y **flex-wrap**, esto es, podemos definir estas dos últimas propiedades en una sola línea de código. El valor por **defecto** es **row nowrap**.

Su **sintaxis** es la siguiente:

```
.container {  
    flex-flow: flex-direction flex-wrap;  
}
```



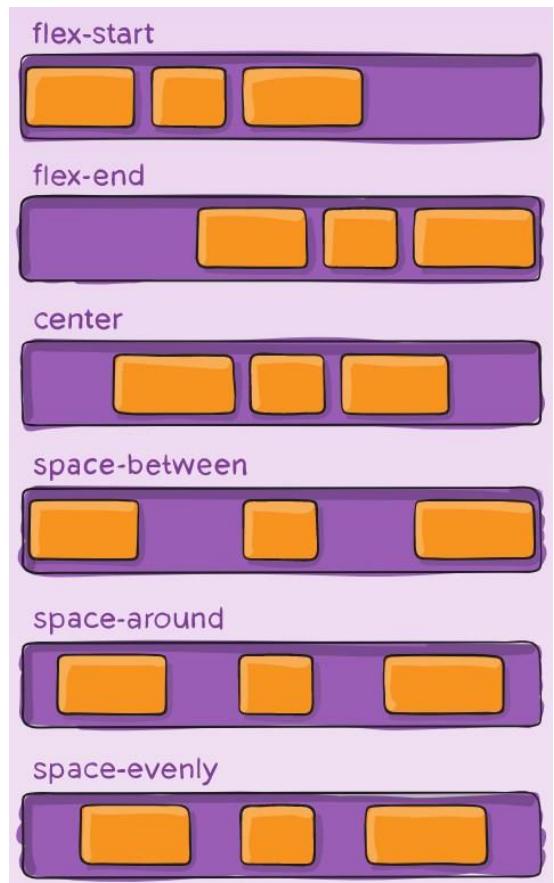
### 1.2.1.5 justify-content

Esta propiedad se utiliza para **alinear los ítems flex** (o contenedores hijo) **a lo largo del eje principal** (horizontal). Ayuda a distribuir el espacio libre sobrante cuando todos los ítems flex de una línea son inflexibles, o son flexibles, pero han alcanzado su tamaño máximo. También ejerce cierto control sobre la alineación de los items cuando desbordan la línea.

Su **sintaxis** es la siguiente:

```
.container {  
    justify-content: flex-start | flex-end | center | space-between |  
    space-around | space-evenly + safe | unsafe;  
}
```

- **flex-start**: alinea los ítems al inicio. Es el valor por **defecto**.
- **flex-end**: alinea los ítems al final.
- **center**: los ítems flex o elementos hijo se centran a lo largo de la línea.
- **space-between**: los ítems se distribuyen **uniformemente** en la línea. Para ello toma todo el espacio sobrante después de que los ítems hayan sido colocados y lo distribuye de forma pareja haciendo que haya un **espacio equitativo (no igual)** entre cada ítem, a excepción del primer y último ítem flex.
- **space-around**: distribuye **uniformemente** los ítems flex sobre la línea, de manera que hay un **espacio equitativo** entre ellos, pero **no igual, incluyendo al primer y último ítem flex**.
- **space-evenly**: los ítems o elementos hijo se **distribuyen** de manera que el **espacio** entre dos elementos cualesquiera (y el espacio hacia los bordes) sea **igual**.





También hay dos palabras clave adicionales que se pueden usar con estos valores: **safe** y **unsafe**. El uso de **safe** cambia el modo de alineación en situaciones de desbordamiento (overflow) para intentar evitar la pérdida de datos.

A continuación, se presenta un **ejemplo**:

```
<!DOCTYPE html>
<html>
<head>
<style>
.contenedor1{
    width: 100%;
    max-width: 700px;
    height: 100%;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: flex-start;
}

.contenedor2{
    width: 100%;
    max-width: 700px;
    height: 100%;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: flex-end;
}

.contenedor3{
    width: 100%;
    max-width: 700px;
    height: 100%;
    background: #212d40;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
}

.contenedor4{
    width: 100%;
    max-width: 700px;
    height: 100%;
    background: #212d40;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
}

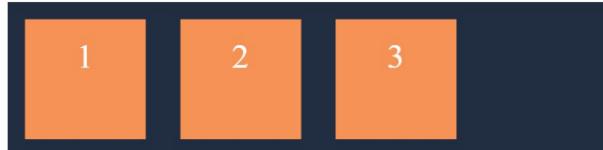
.contenedor5{
    width: 100%;
    max-width: 700px;
    height: 100%;
    background: #212d40;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-evenly;
}

.contenedor6{
    width: 100%;
    max-width: 700px;
    height: 100%;
    background: #212d40;
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
}

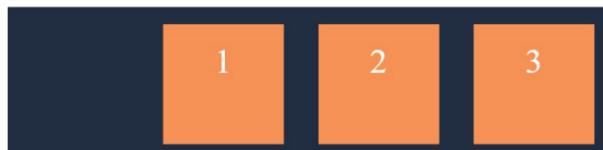
.caja {
    width: 100px;
    height: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
}

</style>
</head>
```

**justify-content: flex-start;**



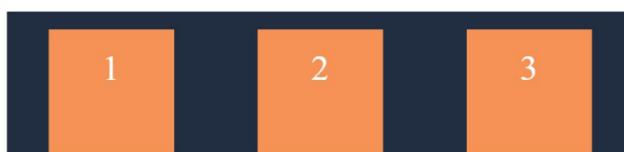
**justify-content: flex-end;**



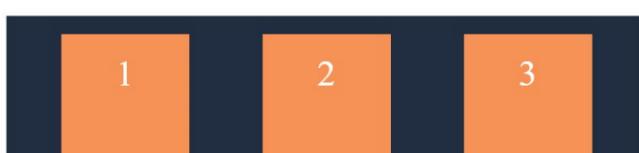
**justify-content: space-between;**



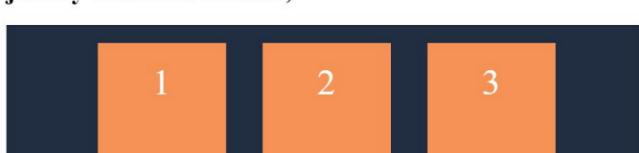
**justify-content: space-around;**



**justify-content: space-evenly;**



**justify-content: center;**



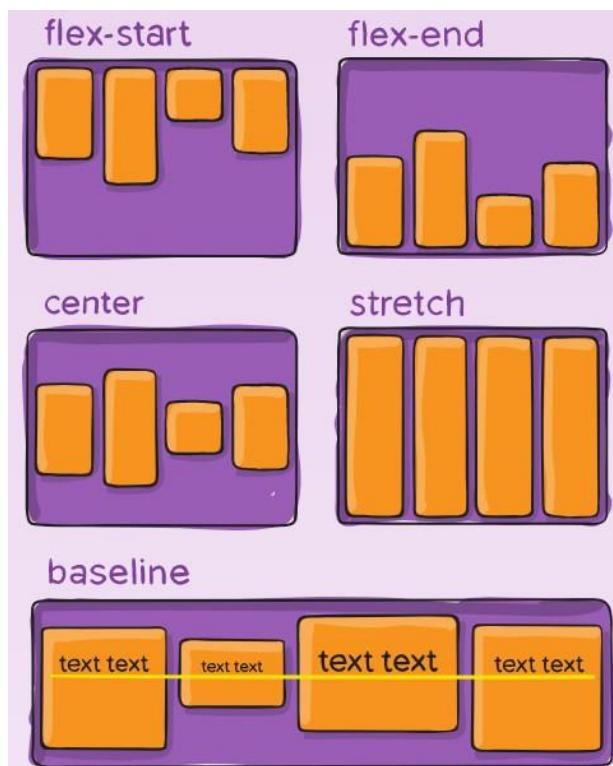


### 1.2.1.6 align-items

**align-items** alinea los ítems flex o contenedores hijo **sobre el eje cruzado o transversal** (vertical). Su **sintaxis** es la siguiente:

```
.container {  
    align-items: stretch | flex-start | flex-end | baseline |  
    center + safe | unsafe;  
}
```

- **flex-start**: los **ítems flex** (o elementos hijo) se colocan **al principio del eje transversal**.
- **flex-end**: **ítems flex** (o elementos hijo) se colocan **al final del eje transversal**.
- **center**: los ítems flex (o elementos hijo) se **centran en el eje transversal**.
- **baseline**: los ítems flex (o elementos hijo) se alinean de tal manera que **sus líneas de base se alinean**. El alineado se hace **en base a la fuente**.
- **stretch**: es el valor por **defecto**. Los ítems flex **se ajustan a la altura de aquel más alto**. Se ajustan para **llenar el contenedor** flex. Por tanto, el ítem o elemento hijo más alto define la altura del contenedor. **Respeta el padding**.  
Si utilizamos este valor, **no tiene sentido y no deberíamos definir la altura al contenedor hijo o item flex**.





A continuación, se presenta un **ejemplo**:

```
<!DOCTYPE html>
<html>
<head>
<style>

.contenedor1{
    width: 80%;
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-items: flex-start;
}

.contenedor2{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-items: flex-end;
}

.contenedor3{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-items: center;
}

.contenedor4{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-items: stretch;
}

.contenedor5{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-items: baseline;
}

.caja {
    width: 100px;
    height: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
}

.caja2 {
    width: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
}

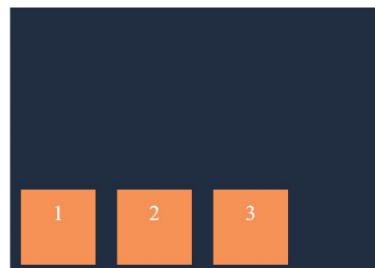
.caja3 {
    width: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 80px;
    color: #fff;
}

</style>
</head>
```

align-items: flex-start;



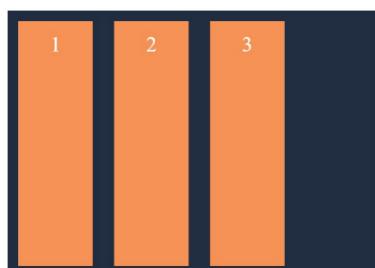
align-items: flex-end;



align-items: center;



align-items: stretch;



align-items: baseline;



```
<body>
<h1> align-items: flex-start; </h1>
<div class="contenedor1">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>

<h1>align-items: flex-end; </h1>
<div class="contenedor2">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>

<h1>align-items: center; </h1>
<div class="contenedor3">
    <div class="caja">1</div>
    <div class="caja">2</div>
    <div class="caja">3</div>
</div>
```

```
<h1>align-items: stretch; </h1>
<div class="contenedor4">
    <div class="caja2">1</div>
    <div class="caja2">2</div>
    <div class="caja2">3</div>
</div>

<h1>align-items: baseline; </h1>
<div class="contenedor5">
    <div class="caja">1</div>
    <div class="caja3">2</div>
    <div class="caja">3</div>
</div>
```





### 1.2.1.7 align-content

Esta propiedad **alinea las líneas o filas de un contenedor flex cuando hay espacio extra en el eje transversal (o vertical)**, de forma similar a como *justify-content* alinea los elementos individuales dentro del eje principal. **No tiene efecto en cajas o ítems flex de una sola línea.**

Esta propiedad sólo tiene efecto en los contenedores flex de varias líneas, en los que *flex-wrap* está configurado como *wrap* o *wrap-reverse*. Un contenedor flex de una sola línea (es decir, donde *flex-wrap* está establecido en su valor por defecto, *no-wrap*) no reflejará *align-content*.

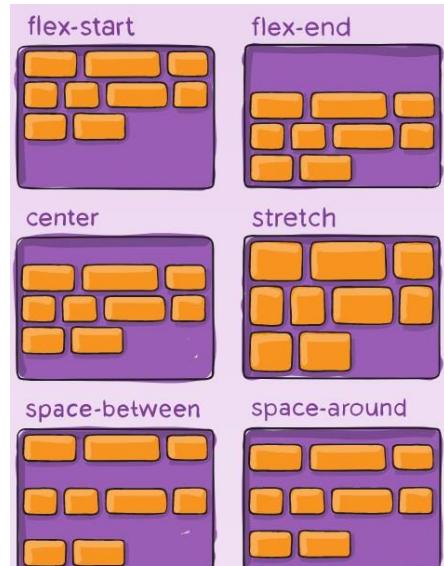
La propiedad *align-content* es similar a *align-items*, por lo que **solo podemos usar una de las dos**.

Su sintaxis es la siguiente:

```
.container {  
    align-content: flex-start | flex-end | center | space-between |  
    space-around | stretch + safe | unsafe;  
}
```

- **flex-start**: los ítems flex (o elementos hijo) se colocan **desde el inicio del eje transversal**.
- **flex-end**: ítems flex (o elementos hijo) se colocan **desde final del eje transversal**.
- **center**: los ítems flex (o elementos hijo) se **centran en el eje transversal**.
- **space-between**: los ítems flex (o elementos hijo) se distribuyen uniformemente, de manera que la primera línea está al principio del contenedor, mientras que la última está al final.
- **space-around**: los ítems flex se distribuyen uniformemente con el mismo espacio alrededor de cada línea.
- **stretch**: los ítems flex se ajustan para **llenar el contenedor flex**. Respeta el padding.

Si utilizamos este valor, **no podemos definir la altura del ítem flex**.





A continuación, se presenta un **ejemplo**:

```
<!DOCTYPE html>
<html>
<head>
<style>

.contenedor1{
    width: 80%;
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-content: flex-start;
}

.contenedor2{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-content: flex-end;
}

.contenedor3{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-content: center;
}

.contenedor4{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-content: space-between;
}

.contenedor5{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-content: space-around;
}

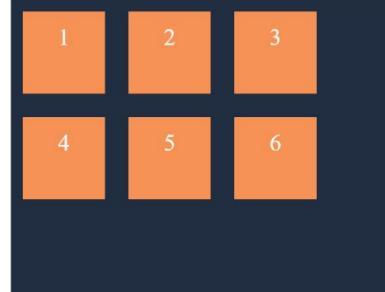
.contenedor6{
    max-width: 700px;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    align-content: stretch;
}

.caja {
    width: 100px;
    height: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
}

.caja2 {
    width: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
}

</style>
</head>
```

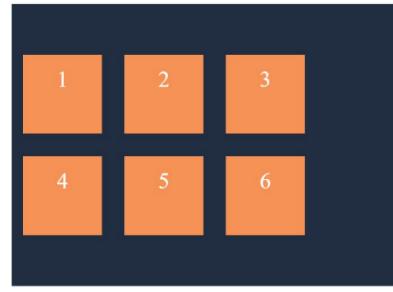
align-content: flex-start;



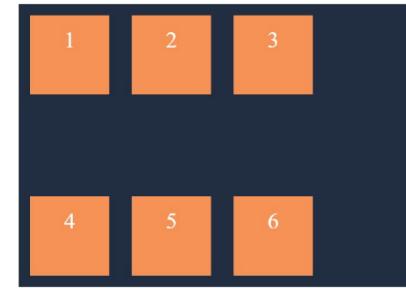
align-content: flex-end;



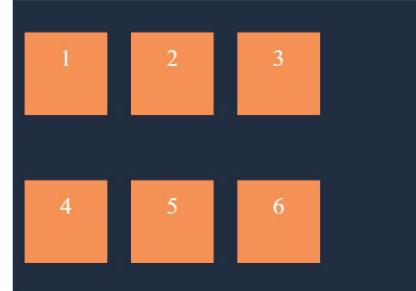
align-content: center;



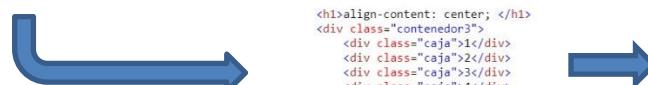
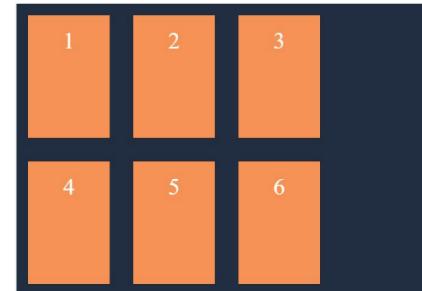
align-content: space-between;



align-content: space-around;



align-content: stretch;



```
<body>
    <h1>align-content: space-between; </h1>
    <div class="contenedor4">
        <div class="caja">1</div>
        <div class="caja">2</div>
        <div class="caja">3</div>
        <div class="caja">4</div>
        <div class="caja">5</div>
        <div class="caja">6</div>
    </div>

    <h1>align-content: space-around; </h1>
    <div class="contenedor5">
        <div class="caja">1</div>
        <div class="caja">2</div>
        <div class="caja">3</div>
        <div class="caja">4</div>
        <div class="caja">5</div>
        <div class="caja">6</div>
    </div>

    <h1>align-content: stretch; </h1>
    <div class="contenedor6">
        <div class="caja">1</div>
        <div class="caja">2</div>
        <div class="caja">3</div>
        <div class="caja">4</div>
        <div class="caja">5</div>
        <div class="caja">6</div>
    </div>
</body>
</html>
```



### 1.2.1.8 gap

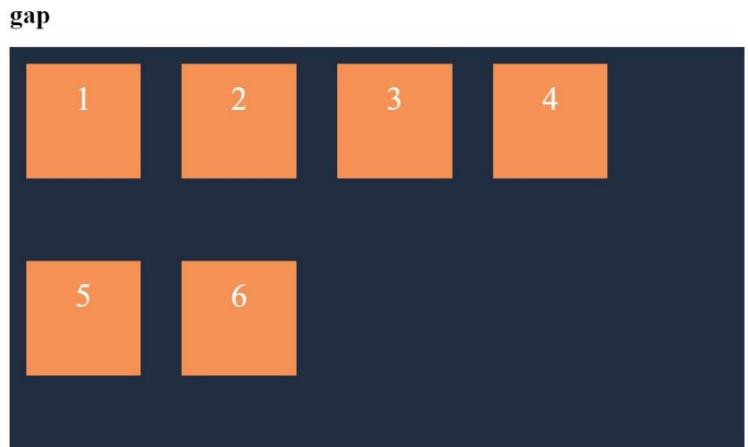
La propiedad **gap** controla explícitamente el espacio que hay entre los ítems flex. Aplica ese espacio sólo entre los elementos, **no en los bordes exteriores**.

```
.container {  
    display: flex;  
    ...  
    gap: 10px;  
    gap: 10px 20px; /* row-gap column gap */  
    row-gap: 10px;  
    column-gap: 20px;  
}
```

**gap no es exclusivo de flexbox**, si no que también funciona en CSS Grid y también en el multi-column layout.

A continuación, se presenta un **ejemplo**:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
  
.contenedor1{  
    width: 100%;  
    height: 500px;  
    background: #212d40;  
    display: flex;  
    flex-direction: row;  
    flex-wrap: wrap;  
    align-content: flex-start;  
    gap: 60px 10px;  
}  
  
.caja {  
    width: 100px;  
    height: 100px;  
    background: #f79256;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 40px;  
    color: #fff; }  
  
</style>  
</head>  
  
<body>  
  
<h1> gap </h1>  
<div class="contenedor1">  
    <div class="caja">1</div>  
    <div class="caja">2</div>  
    <div class="caja">3</div>  
    <div class="caja">4</div>  
    <div class="caja">5</div>  
    <div class="caja">6</div>  
</div>  
  
</body>  
</html>
```





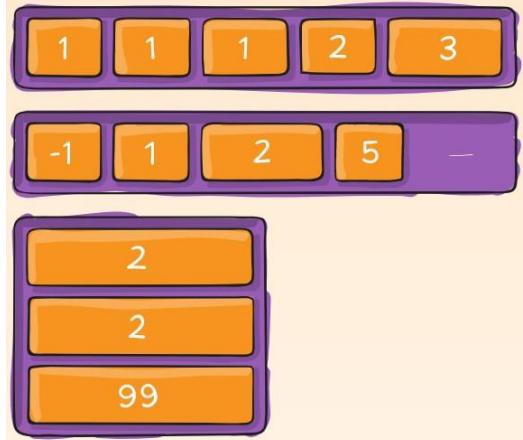
## 1.2.2 Propiedades de los hijos o ‘flex items’

A continuación, se describen las propiedades que se pueden aplicar a los contenedores hijo, elementos hijo o ítems flex.

### 1.2.2.1 order

Esta propiedad **especifica el orden** utilizado para disponer los elementos hijo en su contenedor padre. Los elementos **estarán dispuestos en orden ascendente** según el valor de *order*. Admite valores negativos.

```
.item {  
    order: 5; /* default is 0 */  
}
```



Los elementos hijo con el mismo valor de *order*, se dispondrán en el orden en el cual aparecen en el código fuente. A continuación, se presenta un ejemplo:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
.contenedor1{  
    width: 100%;  
    height: 100%;  
    background: lightgray;  
    display: flex;  
    flex-direction: row;  
    flex-wrap: wrap;  
}  
  
.caja1 {  
    width: 100px;  
    height: 100px;  
    background: pink;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #000;  
    order: 3;}  
  
.caja2 {  
    width: 100px;  
    height: 100px;  
    background: #f79256;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff;  
    order: 1;}  
  
.caja3 {  
    width: 100px;  
    height: 100px;  
    background: purple;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff;  
    order: 2; }  
  
</style>  
</head>  
<body>  
  
<div class="contenedor1">  
    <div class="caja1">1</div>  
    <div class="caja2">2</div>  
    <div class="caja3">3</div>  
</div>  
  
</body>  
</html>
```



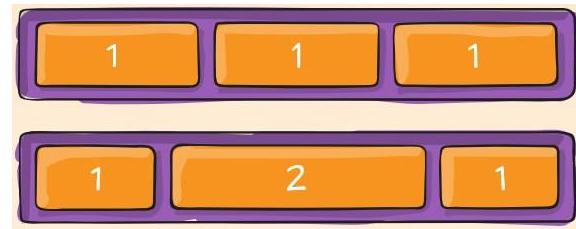


### 1.2.2.2 flex-grow

Esta propiedad define la capacidad de un item flex para **crecer** si es necesario. Por tanto, determina la cantidad de espacio disponible dentro del contenedor padre que debe ocupar el elemento. Acepta un **valor sin unidades** que sirve como **proporción**. **No acepta números negativos**.

```
.item {  
    flex-grow: 4; /* default 0 */  
}
```

Si todos los elementos tienen *flex-grow* establecido en 1, el espacio restante en el contenedor se distribuirá por igual entre todos los hijos. Si uno de los hijos tiene un valor de 2, ese hijo ocupará el doble de espacio que cualquiera de los otros (o lo intentará, al menos).



A continuación, se presenta un **ejemplo**:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
  
.contenedor1{  
    width: 100%;  
    height: 100%;  
    background: lightgray;  
    display: flex;  
    flex-direction: row;  
    flex-wrap: wrap;  
}  
  
.caja1 {  
    width: 100px;  
    height: 100px;  
    background: pink;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #000;  
    flex-grow: 1;  
}  
  
.caja2 {  
    width: 100px;  
    height: 100px;  
    background: #f79256;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff;  
    flex-grow: 2;  
}  
  
.caja3 {  
    width: 100px;  
    height: 100px;  
    background: purple;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff; }  
  
</style>  
</head>  
<body>  
<div class="contenedor1">  
    <div class="caja1">1</div>  
    <div class="caja2">2</div>  
    <div class="caja3">3</div>  
</div>  
</body>  
</html>
```





### 1.2.2.3 flex-shrink

Esta propiedad nos permite definir **cuánto se puede hacer más pequeño un ítem flex en proporción a los demás items o elementos hijo. No admite números negativos.**

```
.item {  
    flex-shrink: 3; /* default 1 */  
}
```

**Si toma el valor 0, no se va a poder hacer más pequeño.** Es un valor muy típico. A continuación, se presenta un **ejemplo:**

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
  
.contenedor1{  
    width: 100%;  
    height: 100%;  
    background: lightgray;  
    display: flex;  
    flex-direction: row; }  
  
.caja1 {  
    width: 100px;  
    height: 100px;  
    background: pink;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #000;  
    flex-shrink: 0; }  
  
.caja2 {  
    width: 100px;  
    height: 100px;  
    background: #f79256;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff; }  
  
.caja3 {  
    width: 100px;  
    height: 100px;  
    background: purple;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff; }  
  
</style>  
</head>  
<body>  
<div class="contenedor1">  
    <div class="caja1">1</div>  
    <div class="caja2">2</div>  
    <div class="caja3">3</div>  
</div>  
</body>  
</html>
```





#### 1.2.2.4 flex-basis

Esta propiedad nos permite especificar el **tamaño inicial o por defecto** que va a tener un elemento hijo. Por tanto, determina el tamaño de una caja de contenidos, a no ser que se hubiera especificado de otra forma, por ejemplo, utilizando *box-sizing*.

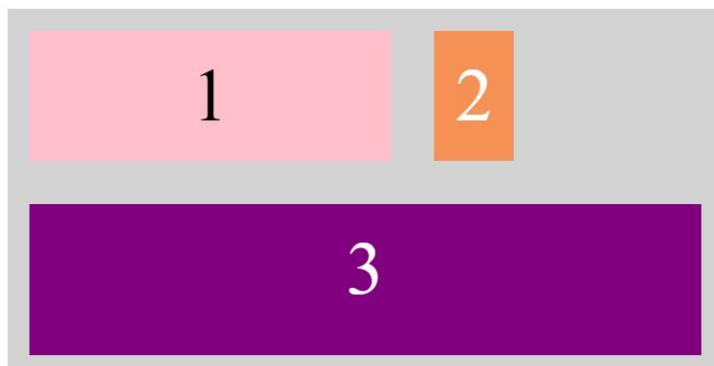
El tamaño de un elemento **se define antes de distribuir el espacio restante**. Puede ser una **longitud** (por ejemplo, 20%, 5rem, etc.) o una **palabra clave**, como puede ser *auto*, la cual significa "mira mi propiedad de anchura o altura".

```
.item {  
    flex-basis: | auto; /* default auto  
}
```

Si *flex-basis* se establece en 0, el espacio extra alrededor del contenido no se tiene en cuenta. **Si se establece en auto, el espacio extra se distribuye en base a su valor *flex-grow*.**

A continuación, se presenta un **ejemplo**:

```
<head>  
<style>  
  
.contenedor1{  
    width: 100%;  
    height: 100%;  
    background: lightgray;  
    display: flex;  
    flex-direction: row;  
    flex-wrap: wrap; }  
  
.caja1 {  
    background: pink;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #000;  
    flex-shrink: 0;  
    flex-basis: 300px; }  
  
.caja2 {  
    background: #f79256;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff;  
    flex-basis: auto; }  
  
.caja3 {  
    width: 100px;  
    height: 100px;  
    background: purple;  
    text-align: center;  
    margin: 20px;  
    padding: 20px;  
    font-size: 70px;  
    color: #fff;  
    flex-grow: 4;  
    flex-basis: auto; }  
  
</style>  
</head>  
<body>  
    <div class="contenedor1">  
        <div class="caja1">1</div>  
        <div class="caja2">2</div>  
        <div class="caja3">3</div>  
    </div>  
</body>  
</html>
```





### 1.2.2.5 flex

Esta propiedad concentra las propiedades de `flex-grow`, `flex-shrink` y `flex-basis` en una sola línea de código. Su **sintaxis** es la siguiente:

```
.item {  
    flex: flex-grow flex-shrink flex-basis;  
}
```

El **segundo y tercer parámetro** (`flex-shrink` y `flex-basis`) son **opcionales**.

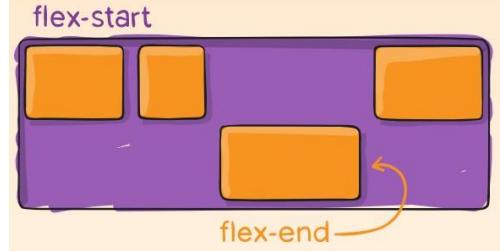
El valor predeterminado es `0 1 auto`, pero si se establece con un solo valor numérico, como por ejemplo `flex: 5;`, esto cambia `flex-basis` a 0%, así que sería como establecer `flex-grow: 5; flex-shrink: 1; flex-basis: 0%;`.

**Se recomienda utilizar esta propiedad abreviada en lugar de establecer las propiedades individuales. La abreviatura establece los otros valores de forma inteligente.**

### 1.2.2.6 align-self

Esta propiedad nos permite anular o cambiar la alineación por defecto, o la especificada por `align-items`, de forma individual para cada ítem flex. Por tanto, **sobreescribirá el valor de align-items**.

Hay que tener en cuenta que `float`, `clear` y `vertical-align` no tienen efecto en un elemento flex.



Su **sintaxis** es la siguiente:

```
.item {  
    align-self: auto | stretch | center | flex-start | flex-end | baseline ;  
}
```

- `auto`: es el valor por defecto. El elemento hereda la propiedad `align-items` de su contenedor padre, o "`stretch`" si no tiene contenedor padre.
- `stretch`: el elemento flex se posiciona para ajustarse al contenedor padre. Para poder utilizar este valor, tenemos que poner la altura del elemento en auto (`height: auto;`)
- `center`: el elemento flex se sitúa en el centro del contenedor padre.
- `flex-start`: el elemento flex se sitúa al inicio del contenedor padre.



- **flex-end**: el elemento flex se sitúa al final del contenedor padre.
- **baseline**: el elemento flex se sitúa en la línea de base del contenedor padre. **El alineado se hace en base a la fuente.**

A continuación, se presenta un **ejemplo**:

```
<!DOCTYPE html>
<html>
<head>
<style>

.contenedor1{
    width: 100%;
    height: 500px;
    background: #212d40;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
}

.caja1 {
    width: 100px;
    height: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
    align-self: flex-start; }

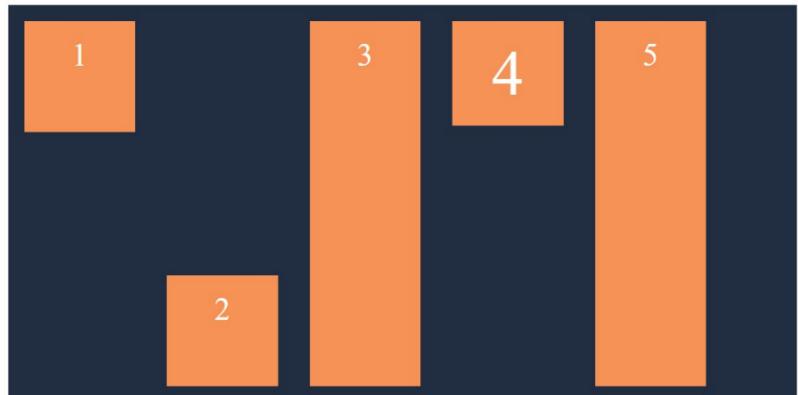
.caja2 {
    width: 100px;
    height: 100px;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
    align-self: flex-end; }

.caja3 {
    width: 100px;
    height: auto;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
    align-self: stretch; }

.caja4 {
    width: 100px;
    height: auto;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 80px;
    color: #fff;
    align-self: baseline; }

.caja5 {
    width: 100px;
    height: auto;
    background: #f79256;
    text-align: center;
    margin: 20px;
    padding: 20px;
    font-size: 40px;
    color: #fff;
    align-self: auto; }

</style>
</head>
```





## 1.2.3 EJEMPLO PRÁCTICO

En este ejemplo, vamos a crear una **caja de información** (o info box) con iconos y texto.

### Código HTML:

```
<!doctype html>
<html>
  <head>
    <script src="https://kit.fontawesome.com/509deda241.js"
           crossorigin="anonymous"></script>
  </head>

  <body>
    <div class="info-box">
      <h2>This is an info-box</h2>
      <div id="flex-p">
        <p class="big"><i class="fas fa-star"></i><br>Lorem ipsum
          dolor sit amet consectetur adipisicing elit.
          Adipisci incident, laudantium nostrum minus fuga aut
          dolore fugit cupiditate qui.</p>

        <p class="big"><i class="fas fa-thumbs-up"></i><br>Vivamus
          sagittis lacus vel augue laoreet rutrum faucibus
          dolor auctor. Integer posuere erat a ante venenatis
          dapibus posuere velit aliquet.</p>

        <p class="big"><i class="fas fa-check"></i><br>Donec sed odio
          dui. Cras mattis consectetur purus sit amet
          fermentum. Nullam id dolor id nibh ultricies vehicula
          ut id elit.</p>
      </div>
    </div>
  </body>
```

### Código CSS:



```
*{  
    margin: 0px;  
    padding: 0 px;  
    font-family: "Roboto", Arial, Tahoma, sans-serif;  
}  
p.big{  
    font-size: 1.1em;  
    line-height: 1.5em;  
}  
.fa-star, .fa-thumbs-up, .fa-check{  
    font-size: 2.7em;  
    display: block;  
    color: black;  
    color: white;  
    margin-bottom: 30px;  
}  
.info-box{  
    background-color: #1C1D22;  
    margin: 0px auto;  
    padding: 110px 0 110px 0;  
    color: white;  
    text-align: center;  
}  
.info-box h2{  
    margin-bottom: 30px;  
}  
#flex-p{  
    justify-content: center;  
    display: flex;  
    flex-wrap: wrap;  
    align-items: center;  
}  
#flex-p p{  
    padding: 30px;  
    max-width: 300px;  
}
```

This is an info-box

★  
Lorem ipsum dolor sit amet  
consectetur adipisicing elit. Adipisci  
incident, laudantium nostrum minus  
fuga aut dolore fugit cupiditate qui.

👍  
Vivamus sagittis lacus vel augue  
laoreet rutrum faucibus dolor auctor.  
Integer posuere erat a ante venenatis  
dapibus posuere velit aliquet.

✓  
Donec sed odio dui. Cras mattis  
consectetur purus sit amet  
fermentum. Nullam id dolor id nibh  
ultricies vehicula ut id elit.

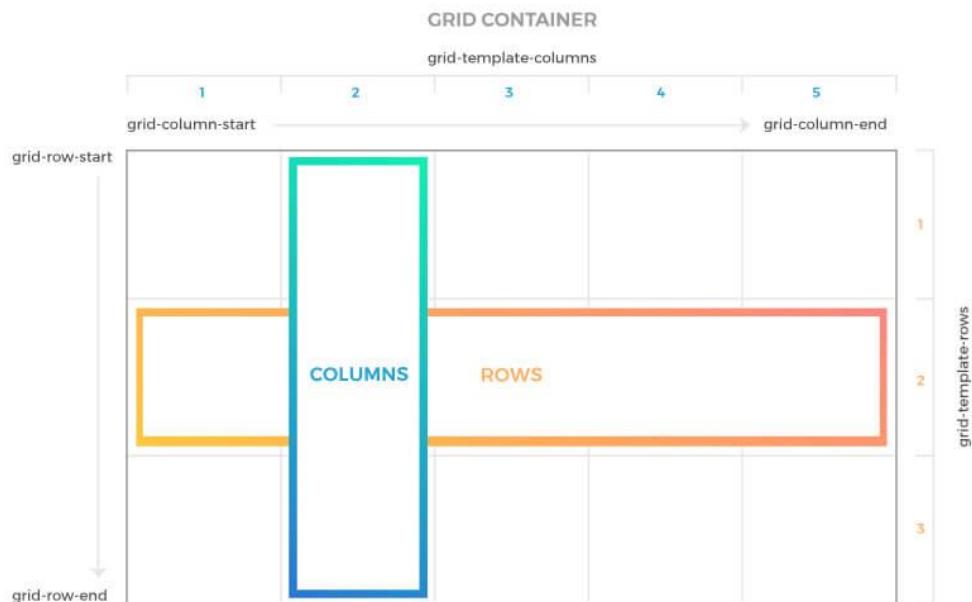


## 2. CSS Grid

**CSS Grid** nos permite maquetar contenido ajustándolo a **cuadrículas o rejillas** (grids) totalmente configurables mediante estilos CSS.

Mediante CSS Grid podemos dividir la página en una rejilla o cuadrícula a partir de la cual se pueden posicionar los diferentes elementos de manera muy sencilla. Gracias a los sistemas de maquetación de CSS Grid y CSS Flexbox se pueden crear estructuras con **menos código** y de una forma más fácil que con los métodos tradicionales.

La **diferencia** básica entre CSS Grid y CSS Flexbox es que **Flexbox** se creó para diseños de una dimensión, en una fila o una columna. En cambio, **CSS Grid Layout** se pensó para el diseño bidimensional, en varias filas y columnas al mismo tiempo.



En este apartado se expondrán las principales propiedades de CSS Grid. Si quieres conocer más a fondo este sistema puedes acceder a las siguientes guías:

- <https://lenguajecss.com/css/maquetacion-y-colocacion/grid-css/>
- <https://css-tricks.com/snippets/css/complete-guide-grid/>



- [https://www.w3schools.com/css/css\\_grid.asp](https://www.w3schools.com/css/css_grid.asp)

## 2.1 Terminología importante de CSS Grid

Antes de sumergirnos en los conceptos de Grid, es importante **entender la terminología**. Dado que todos los términos son conceptualmente similares, es fácil confundirlos entre sí si no se memorizan primero sus significados definidos en la especificación Grid. Estos son los siguientes:

### 1) Contenedor (Grid Container)

El elemento sobre el que se aplica *display: grid*. Es el **padre directo** de todos los elementos de la rejilla. En este ejemplo, el container es el contenedor de la cuadrícula o rejilla.

```
<div class="container">
  <div class="item item-1"> </div>
  <div class="item item-2"> </div>
  <div class="item item-3"> </div>
</div>
```

### 2) Item (o hijos)

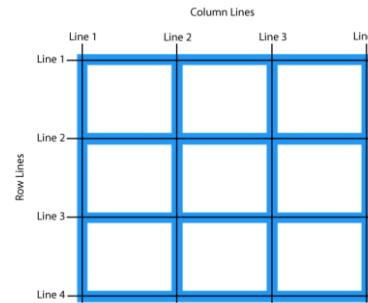
Son los hijos (es decir, **descendientes directos**) del contenedor de grid. En el siguiente ejemplo, los items del elemento son ítems grid, pero los subelementos no lo son.

```
<div class="container">
  <div class="item"> </div>
  <div class="item">
    <p class="sub-item"> </p>
  </div>
  <div class="item"> </div>
</div>
```

### 3) Línea (Grid Line)

Son las **líneas divisorias** que conforman la estructura de la rejilla o cuadrícula. Pueden ser **verticales** ("líneas de columnas") u **horizontales** ("líneas de filas") y residir en cualquier lado de una fila o columna.

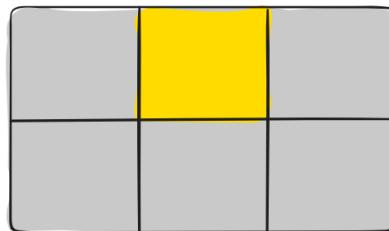
En el siguiente ejemplo, la línea amarilla sería una línea de columnas.



#### 4) Celda (Grid Cell)

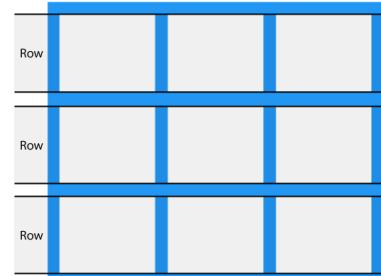
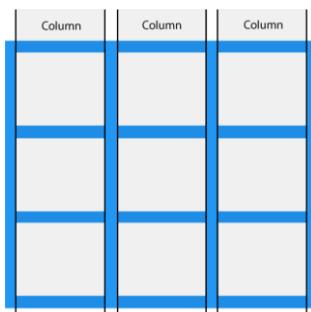
Una celda es el **espacio entre dos filas adyacentes y dos líneas de columnas adyacentes**. Es una única "unidad" de la rejilla o cuadrícula.

En el siguiente ejemplo podemos ver la celda grid entre las líneas 1 y 2 de filas y las líneas 2 y 3 de columnas.

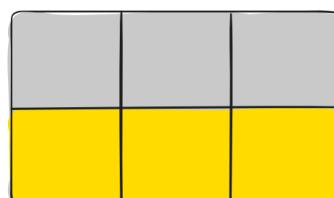


#### 5) Banda (Grid Track) – Filas y columnas.

Es el espacio entre **dos líneas de cuadrícula adyacentes**. Podemos pensar en ellas como las **columnas** o **filas** de la cuadrícula.



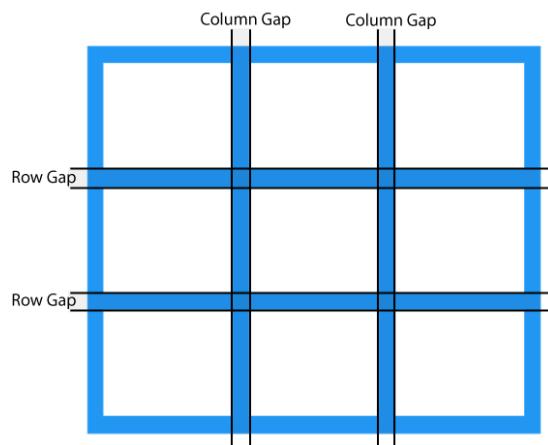
En el siguiente ejemplo podemos ver el recorrido de la cuadrícula entre las líneas de la segunda y tercera fila.





## 6) Grid Gaps

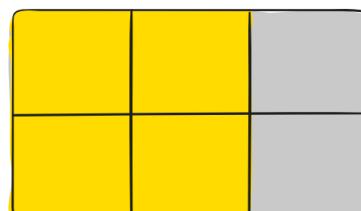
Los gaps son los espacios entre cada columna/fila.



## 7) Área (Grid Area)

Es el **espacio** total rodeado por **cuatro líneas** de la cuadrícula o rejilla. Un área de cuadrícula puede estar compuesto por **cualquier número de celdas**.

En el ejemplo podemos ver el área de la cuadrícula entre las líneas 1 y 3 de filas y las líneas 1 y 3 de columnas.





## 2.2 Propiedades del padre o grid container

PROPIEDADES DEL CONTENEDOR PADRE O GRID CONTAINER	
<b>display</b>	Define el elemento contenedor (grid o inline-grid)
<b>grid-template-columns</b> <b>grid-template-rows</b>	Define el espaciado o tamaño de las columnas/filas
<b>grid-template-areas</b>	Indica el nombre y posición concreta de cada área de la cuadrícula. Define una plantilla de la rejilla haciendo referencia a los nombres de las áreas que se especifican con la propiedad <b>grid-area</b> .
<b>grid-template</b>	Forma abreviada de definir las filas, columnas y áreas de la cuadrícula. Define las tres propiedades anteriores en una línea de código.
<b>row-gap</b> <b>column-gap</b>	Define el espaciado entre las filas/columnas
<b>gap</b>	Define el tamaño del espacio entre las filas y entre las columnas. Define las propiedades <b>row-gap</b> y <b>column-gap</b> en una línea de código.
<b>justify-items</b>	Alinea los <b>elementos</b> de la cuadrícula o rejilla a lo largo del <b>eje horizontal o en línea</b> .
<b>align-items</b>	Alinea los <b>elementos</b> de la cuadrícula o rejilla a lo largo del <b>eje vertical o en bloque</b> .
<b>place-items</b>	Establece las propiedades <b>align-items</b> y <b>justify-items</b> en una única declaración o línea de código.
<b>justify-content</b>	Alinea la <b>cuadrícula</b> a lo largo del <b>eje</b> en línea u <b>horizontal</b> (fila o eje x).
<b>align-content</b>	Alinea la <b>cuadrícula</b> a lo largo del <b>eje de bloque o vertical</b> (columna o eje y).
<b>place-content</b>	Establece las propiedades <b>align-content</b> y <b>justify-content</b> en una única línea de código.
<b>grid-auto-columns</b> <b>grid-auto-rows</b>	Establece el <b>tamaño de las columnas/filas implícitas</b> y cualquier otra columna/fila que no haya sido explícitamente dimensionada.
<b>grid-auto-flow</b>	Controla cómo se insertan los elementos autocolocados en la cuadrícula
<b>grid</b>	Establece todas las siguientes propiedades en una única declaración o sentencia de código: <b>grid-template-rows</b> , <b>grid-template-columns</b> , <b>grid-template-</b>



	areas, grid-auto-rows, grid-auto-columns, y grid-auto-flow.
--	---

## 2.2.1 'display'

Para crear la cuadrícula **grid** hay que definir sobre el elemento contenedor la propiedad **display** y especificar el valor **grid** o **inline-grid**.

Los valores **inline-grid** y **grid** indican **cómo se comporta el contenedor con respecto al contenido exterior**.

Valor	Descripción
<b>inline-grid</b>	Contenedor en <b>línea</b> con respecto al contenido exterior
<b>grid</b>	Contenedor en <b>bloque</b> con respecto al contenido exterior

A continuación, se presenta ejemplos sencillos.

### 1) Disposición de elementos con **display: inline-grid**

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: inline-grid;
  background-color: #2196F3;
  padding: 10px;
}
.grid-item {
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>display: inline-grid</h1>
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
</div>
</body>
</html>
```

### display: inline-grid



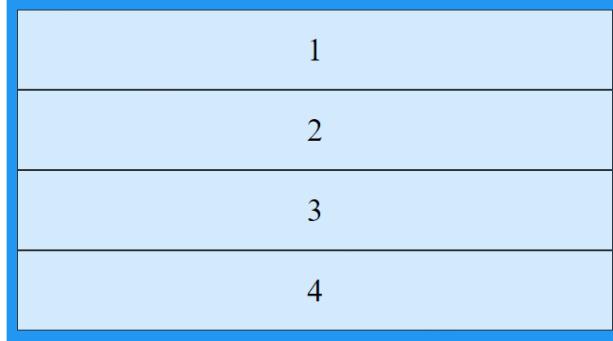


## 2) Disposición de elementos con `display: grid`

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  background-color: #2196F3;
  padding: 10px;
}
.grid-item {
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>display: grid</h1>
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
</div>
</body>
</html>
```

### display: grid



## 3) Ejemplo en donde se utiliza `display:inline-grid` y `display: grid`

### Código HTML:

```
<h2>display: inline-grid;</h2>
<b>Contenido exterior</b>
<div class="contenedor" > <!-- contenedor padre-->
  <div class="rejilla">Item 1</div> <!-- Ítems del grid -->
  <div class="rejilla">Item 2</div>
  <div class="rejilla">Item 3</div>
  <div class="rejilla">Item 4</div>
</div>

<h2>display: grid;</h2>
<b>Contenido exterior</b>
<div class="contenedor2" > <!-- contenedor padre-->
  <div class="rejilla">Item 1</div> <!-- Ítems del grid -->
  <div class="rejilla">Item 2</div>
  <div class="rejilla">Item 3</div>
  <div class="rejilla">Item 4</div>
</div>
```



### Código CSS:

```
.contenedor{  
    display: inline-grid;  
}  
.contenedor2{  
    display: grid;  
}
```

**display: inline-grid;**

Contenido exterior Item 1  
Item 2  
Item 3  
Item 4

**display: grid;**

Contenido exterior  
Item 1  
Item 2  
Item 3  
Item 4

### 2.2.2 ‘grid-template-columns’ y ‘grid-template-rows’

Para definir el espaciado o tamaño de las columnas y las filas usamos las propiedades **grid-template-columns** y **grid-template-rows**.

Propiedad	Valor	Descripción
<b>grid-template-columns</b>	[col1] [col2] ...	Tamaño de cada columna
<b>grid-template-rows</b>	[fila1] [fila2] ...	Tamaño de cada fila

Su **sintaxis** es la siguiente:

```
grid-template-columns: none | auto | max-content | min-content | Length;  
grid-template-rows: none | auto | max-content | min-content | Length;
```

- **auto**: el tamaño de las filas/columnas viene determinado por el tamaño del contenedor y por el tamaño del contenido de los elementos de la fila/columna.
- **max-content**: establece el tamaño de cada fila/columna en función del elemento de mayor tamaño de la fila/columna.

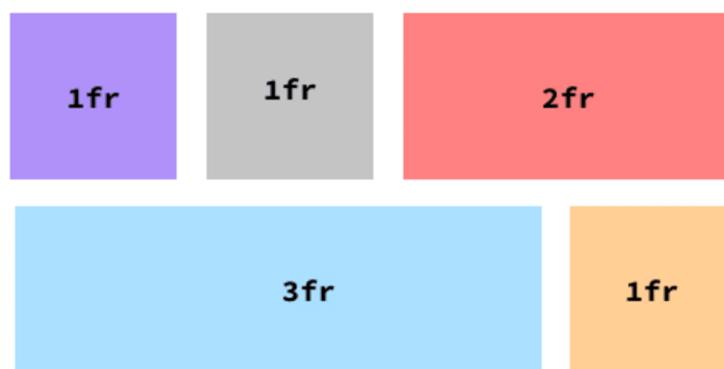


- **min-content**: establece el tamaño de cada fila/columna en función del elemento más pequeño de la fila/columna.
- **length**: establece el tamaño de las filas/columnas, utilizando un valor de longitud legal.

Como vemos a continuación, si definimos los siguientes valores crearemos una cuadrícula de 2 filas por 3 columnas.

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

Podemos usar las unidades que ya conocemos como los píxeles o los porcentajes, o utilizar la unidad **fr** (propia del sistema CSS Grid) que indica la **proporción del espacio que ocupará cada elemento**. Por **ejemplo**, si definimos 3 columnas y le asignamos un valor 1fr a cada una, esto repartirá el espacio a partes iguales entre las 3. En otro caso, si definimos el valor 2fr a una de ellas, esa ocupará el doble de espacio que las otras.



A continuación, se expone un par de ejemplos sencillos, en donde se utilizan las propiedades **grid-template-columns** y **grid-template-rows** con diferentes longitudes y unidades.



## A. Ejemplo 1

### Código HTML:

```
<h2> grid-template-columns: auto auto auto auto auto;</h2>
<div class="contenedor"> <!-- contenedor padre-->
  <div class="rejilla">Item 1</div> <!-- Ítems del grid -->
  <div class="rejilla">Item 2</div>
  <div class="rejilla">Item 3</div>
  <div class="rejilla">Item 4</div>
</div>
```

```
<h2> grid-template-columns: 50px 200px 50px 500px;</h2>
<div class="contenedor2"> <!-- contenedor padre-->
  <div class="rejilla">Item 1</div> <!-- Ítems del grid -->
  <div class="rejilla">Item 2</div>
  <div class="rejilla">Item 3</div>
  <div class="rejilla">Item 4</div>
</div>
<h2> grid-template-columns: 1fr 2fr 1fr 3fr;</h2>
<div class="contenedor3"> <!-- contenedor padre-->
  <div class="rejilla">Item 1</div> <!-- Ítems del grid -->
  <div class="rejilla">Item 2</div>
  <div class="rejilla">Item 3</div>
  <div class="rejilla">Item 4</div>
</div>
```

### Código CSS:

```
.contenedor{
  display: grid;
  grid-template-columns: auto auto auto auto;
}
.contenedor2{
  display: grid;
  grid-template-columns: 50px 200px 300px 500px;
}
.contenedor3{
  display: grid;
  grid-template-columns: 1fr 2fr 1fr 3fr;
}
```



**grid-template-columns: auto auto auto auto;**

Item 1                  Item 2                  Item 3                  Item 4

**grid-template-columns: 50px 200px 50px 500px;**

Item 1 Item 2                  Item 3                  Item 4

**grid-template-columns: 1fr 2fr 1fr 3fr;**

Item 1                  Item 2                  Item 3                  Item 4

## B. Ejemplo 2

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  grid-template-rows: 1fr 3fr;
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
</div>
</body>
</html>
```

1	2	3	4
5	6	7	8

### 2.2.3 ‘grid-template-areas’

Mediante esta propiedad es posible **indicar el nombre y posición concreta de cada área** de una cuadrícula. Por tanto, esta propiedad define una **plantilla de la rejilla** haciendo referencia a los nombres de las áreas que se especifican con la propiedad **grid-area**.

La **repetición del nombre** de un área de la rejilla o grid hace que el contenido abarque esas celdas. Un **punto** significa una celda vacía.

Su **sintaxis** es la siguiente:

```
grid-template-areas: "<grid-area-name> | . | none | ..." "...";
```

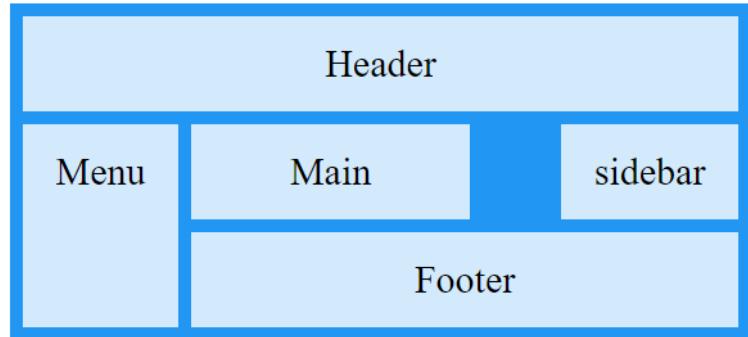


A continuación, se muestra un **ejemplo** sencillo

```
<!DOCTYPE html>
<html>
<head>
<style>
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: sidebar; }
.item5 { grid-area: footer; }

.grid-container {
  display: grid;
  grid-template-areas:
    'header header header header header'
    'menu main main main . sidebar'
    'menu footer footer footer footer';
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>A Webpage Template</h1>
<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">Main</div>
  <div class="item4">Sidebar</div>
  <div class="item5">Footer</div>
</div>
</body>
</html>
```

## A Webpage Template



### 2.2.4 ‘grid-template’

Esta propiedad es una forma abreviada de definir las filas, columnas y áreas de la cuadrícula en una única declaración.

Su **sintaxis** es la siguiente:

```
grid-template: none | grid-template-rows | grid-template-columns |  
grid-template-areas;
```

A continuación, se exponen algunos **ejemplos**:



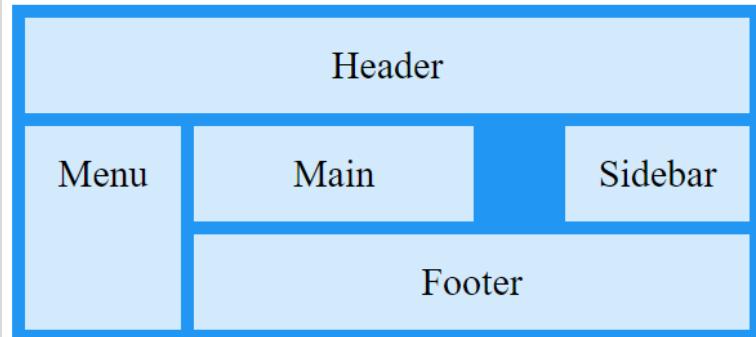
1

```
<!DOCTYPE html>
<html>
<head>
<style>
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: sidebar; }
.item5 { grid-area: footer; }

.grid-container {
  display: grid;
  grid-template:
    'header header header header header'
    'menu main main main sidebar'
    'menu footer footer footer footer footer';
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>A Webpage Template</h1>
<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">Main</div>
  <div class="item4">Sidebar</div>
  <div class="item5">Footer</div>
</div>
</body>
</html>
```

## A Webpage Template



2

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template: 150px / auto auto auto;
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>The grid-template Property</h1>

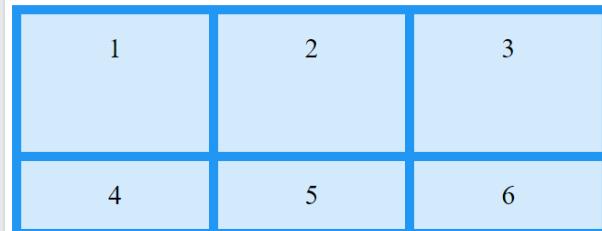
<p>The grid-template property can be used as a shorthand property for the <em>grid-template-rows</em> and the <em>grid-template-columns</em> properties.</p>
<p>This grid layout has three columns, and the first row is 150px high:</p>

<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
</div>
</body>
</html>
```

## The grid-template Property

The `grid-template` property can be used as a shorthand property for the `grid-template-rows` and the `grid-template-columns` properties.

This grid layout has three columns, and the first row is 150px high:





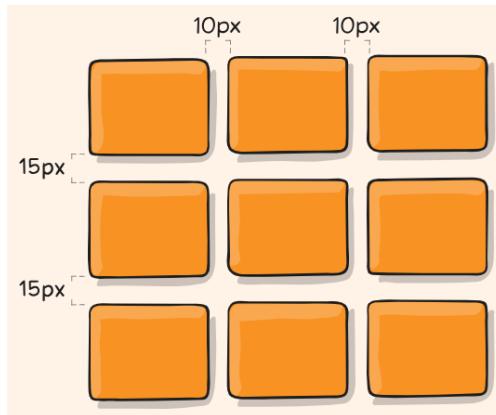
## 2.2.5 ‘row-gap’ y ‘column-gap’

Podemos definir el espaciado entre las rejillas mediante las propiedades `row-gap` y `column-gap`. El espaciado solo se crea entre las columnas/filas, no en los bordes exteriores.

Propiedad	Descripción
<code>column-gap</code>	Espaciado entre columnas
<code>row-gap</code>	Espaciado entre filas

Como vemos a continuación, si definimos los siguientes valores crearemos un espaciado entre columnas de 10px, y de 15px entre filas.

```
.contenedor{  
    display: grid;  
    grid-template-rows: 200px 200px 200px;  
    grid-template-columns: 200px; 200px; 200px;  
    column-gap: 10px;  
    row-gap: 15px; }
```



## 2.2.6 ‘gap’

La propiedad `gap` define el tamaño del espacio entre las filas y entre las columnas. Es una abreviatura de las siguientes propiedades: `row-gap` y `column-gap`.

Su **sintaxis** es la siguiente:

```
gap: row-gap column-gap;
```



A continuación, se presenta un **ejemplo** sencillo:

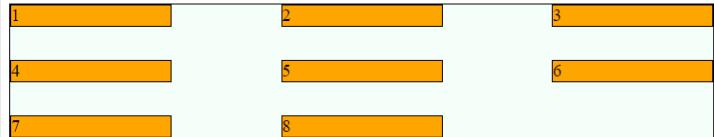
```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  border: 1px solid black;
  background-color: mintcream;
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 30px 100px;
}
.grid-container > div {
  border: 1px solid black;
  background-color: orange;
}
</style>
</head>
<body>

<h1>La propiedad gap</h1>
<p>Espacio de 30px entre filas, y 100px entre columnas</p>

<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
</div>
</body>
</html>
```

## La propiedad gap

Espacio de 30px entre filas, y 100px entre columnas



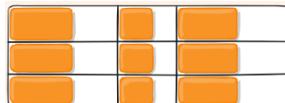
### 2.2.7 'justify-items'

Esta propiedad alinea los elementos de la cuadrícula o rejilla **a lo largo del eje horizontal** (fila o eje x) (a diferencia de **align-items**, que alinea a lo largo del eje en vertical (columna o eje y)). Este valor **se aplica a todos los elementos hijo** de la cuadrícula **dentro del contenedor**.

Su **sintaxis** es la siguiente:

```
.container {
  justify-items: start | end | center | stretch; }
```

- **start**: alinea los elementos hijo para que queden alineados con el borde inicial de su celda.

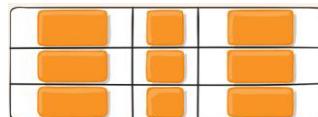


- **end**: alinea los elementos en el borde final de la celda.

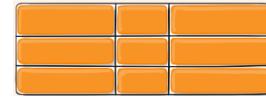




- **center**: alinea los elementos en el centro de la celda.



- **stretch**: rellena todo el ancho de la celda (por defecto).



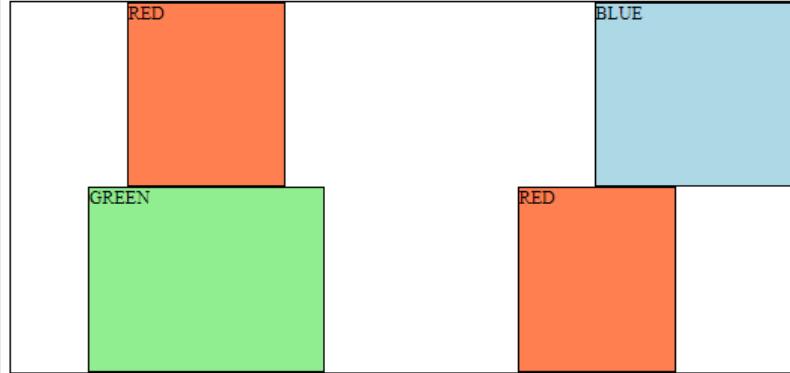
Este comportamiento también puede establecerse en elementos individuales de la rejilla mediante la propiedad **justify-self**.

A continuación, se presentan un par de **ejemplos** sencillos.

1

```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
  width: 50%;
  aspect-ratio: 2/1;
  border: 1px solid black;
  display: grid;
  grid-template-columns: 1fr 1fr;
  justify-items: center;
}
#container > div {
  border: 1px solid black;
}
.blue {
  background-color: lightblue;
  width: 50%;
  justify-self: right;
}
.red {
  background-color: coral;
  width: 40%;
}
.green {
  background-color: lightgreen;
  width: 60%;
}
</style>
</head>
<body>
<h2>justify-items vs. justify-self</h2>
<div id="container">
  <div class="red">RED</div>
  <div class="blue">BLUE</div>
  <div class="green">GREEN</div>
  <div class="red">RED</div>
</div>
</body>
</html>
```

### justify-items vs. justify-self

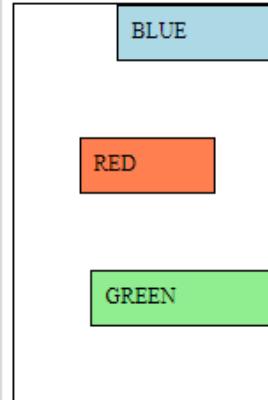




2

```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
    width: 200px;
    aspect-ratio: 2/3;
    border: 1px solid black;
    display: grid;
    position: relative;
    justify-items: right;
}
#container > div {
    border: 1px solid black;
    padding: 10px;
    position: absolute;
}
.blue {
    background-color: lightblue;
    width: 50%;
}
.red {
    background-color: coral;
    width: 40%;
    top: 100px;
    justify-self:center;
}
.green {
    background-color: lightgreen;
    width: 60%;
    top: 200px;
}
</style>
</head>
<body>
<h2>justify-items: right, on positioned items</h2>
<div id="container">
    <div class="red">RED</div>
    <div class="blue">BLUE</div>
    <div class="green">GREEN</div>
</div>
</body>
</html>
```

### justify-items: right, on positioned items



## 2.2.8 ‘align-items’

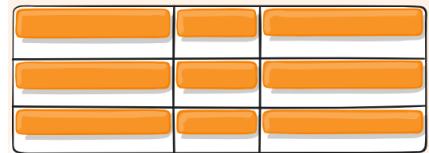
Esta propiedad alinea los elementos de la cuadrícula o rejilla **a lo largo del eje vertical** (columna o eje y) (a diferencia de **justify-items**, que alinea a lo largo del eje en horizontal (fila o eje x)). Este valor **se aplica a todos los elementos hijo** de la cuadrícula **dentro del contenedor**.

Su **sintaxis** es la siguiente:

```
.container {
    align-items: start | end | center | stretch | baseline; }
```



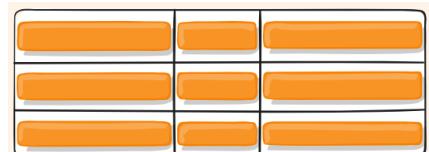
- **start**: alinea los elementos para que queden alineados con el borde inicial de su celda.



- **end**: alinea los elementos para que queden alineados con el borde final de su celda.



- **center**: alinea los elementos en el centro de su celda.



- **stretch**: rellena toda la altura de la celda (por defecto).



- **baseline**: alinea los elementos a lo largo de la línea de base del texto.

A continuación, se presentan un par de **ejemplos** sencillos.

1

```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
  width: 70%;
  aspect-ratio: 2/1;
  border: 1px solid black;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  align-items: start;
}
#container div {
  border: 1px solid black;
}
</style>
</head>
<body>
<h2>align-items: start</h2>
<div id="container">
  <div style="background-color:coral;min-height:30px;">RED</div>
  <div style="background-color:lightblue;min-height:50px;">BLUE</div>
  <div style="background-color:lightgreen;min-height:190px;">GREEN</div>
  <div style="background-color:lightgreen;min-height:190px;">GREEN</div>
  <div style="background-color:coral;min-height:30px;">RED</div>
  <div style="background-color:lightblue;min-height:50px;">BLUE</div>
</div>
</body>
</html>
```

**align-items: start**

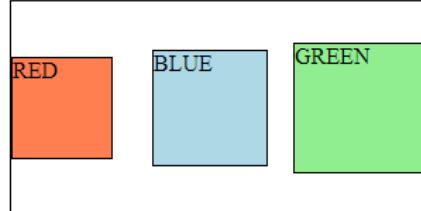
The diagram illustrates the effect of the `align-items: start` property. It shows a grid with three columns. The first column contains two red boxes (30px and 30px). The second column contains two blue boxes (50px and 50px). The third column contains two green boxes (190px and 190px). The green boxes are taller than the blue boxes, which are taller than the red boxes. This visualizes how items are aligned at the top (start) of each column.



2

```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
    width: 300px;
    aspect-ratio: 2/1;
    border: 1px solid black;
    display: grid;
    position: relative;
    align-items: center; }
#container > div {
    border: 1px solid black;
    position: absolute;
    aspect-ratio: 1; }
.red {
    background-color: coral;
    inline-size: 70px;
    left: 0px; }
.blue {
    background-color: lightblue;
    inline-size: 80px;
    left: 100px; }
.green {
    background-color: lightgreen;
    inline-size: 90px;
    left: 200px; }
</style>
</head>
<body>
<h2>align-items: center</h2>
<div id="container">
    <div class="red">RED</div>
    <div class="blue">BLUE</div>
    <div class="green">GREEN</div>
</div>
</body>
</html>
```

### align-items: center



## 2.2.9 ‘place-items’

Esta propiedad establece las propiedades **align-items** y **justify-items** en una única declaración o línea de código.

Su **sintaxis** es la siguiente:

```
.container {
    place-items: <align-items> / <justify-items>; }
```

Si se omite el segundo valor (**<justify-items>**), el primer valor es asignado a ambas propiedades.

A continuación, se muestra un **ejemplo**:



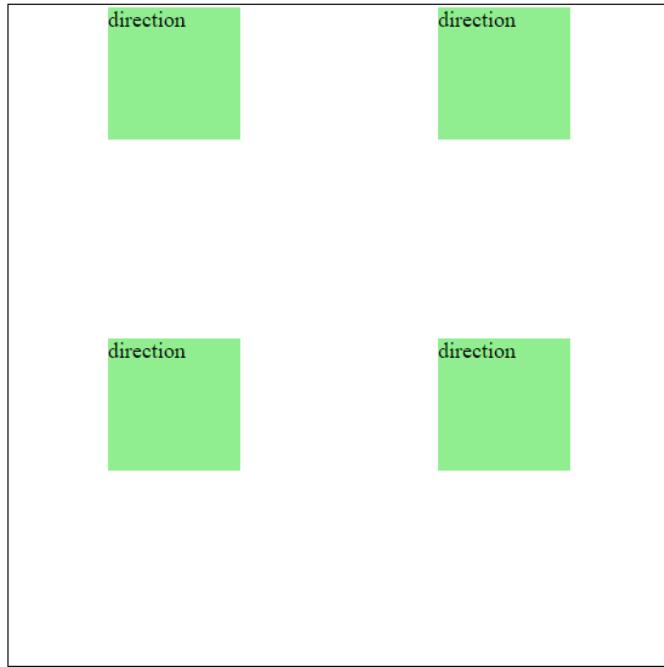
```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
  width: 60%;
  aspect-ratio: 1;
  border: solid black 1px;
  display: grid;
  grid-template-columns: 1fr 1fr;
  place-items: start center;
}

#container > div {
  width: 40%;
  aspect-ratio: 1;
  margin: 2px;
  background-color: lightgreen;
}
</style>
</head>
<body>

<h1>The place-items Property</h1>
<div id="container">
  <div>direction</div>
  <div>direction</div>
  <div>direction</div>
  <div>direction</div>
</div>

</body>
</html>
```

## The place-items Property



### 2.2.10 'justify-content'

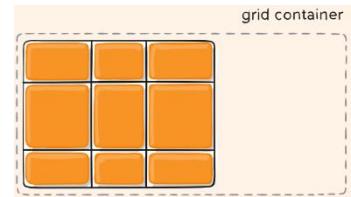
A veces, el tamaño total de la cuadrícula puede ser inferior al tamaño de nuestro contenedor. Esto puede ocurrir si todos los elementos de la rejilla están dimensionados con unidades no flexibles como px. En este caso se puede establecer la alineación de la rejilla dentro del contenedor padre. Por tanto, esta propiedad **alinea la cuadrícula a lo largo del eje en línea (fila o eje x)** (a diferencia de *align-content* que alinea la cuadrícula a lo largo del eje de bloque (columna)).

Su **sintaxis** básica es la siguiente:

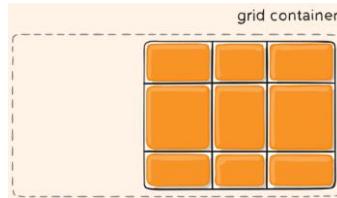
```
.container {
  justify-content: start | end | center | stretch | space-around |
    space-between | space-evenly;
}
```



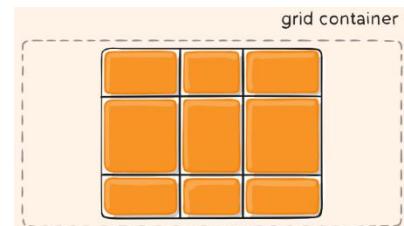
- **start**: alinea la rejilla o cuadrícula con el borde inicial del contenedor.



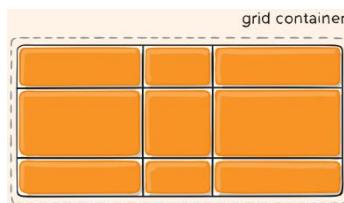
- **end**: alinea la rejilla o cuadrícula con el borde final del contenedor.



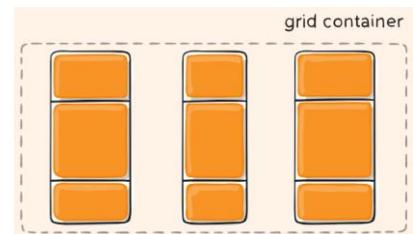
- **center**: alinea la rejilla o cuadrícula en el centro del contenedor.



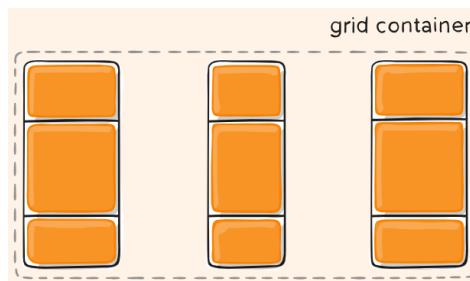
- **stretch**: cambia el tamaño de los elementos de la cuadrícula para que ocupe toda la anchura del contenedor.



- **space-around**: coloca una cantidad uniforme de espacio entre cada elemento de la cuadrícula, con espacios de tamaño medio en los extremos.

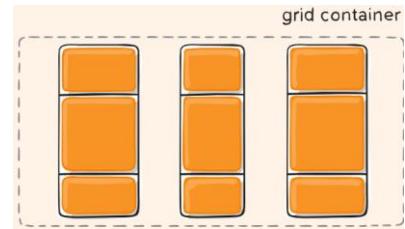


- **space-between**: coloca una cantidad uniforme de espacio entre cada elemento de la cuadrícula, sin espacio en los extremos.





- **space-evenly**: coloca una cantidad uniforme de espacio entre cada elemento de la cuadrícula, incluidos los extremos.



A continuación, se expone un **ejemplo** sencillo:

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  justify-content: space-around;
  grid-template-columns: 50px 50px 50px; /*Make the grid
smaller than the container*/
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>

<h1>The justify-content Property - "space-around"</h1>

<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
</body>
</html>
```

### The justify-content Property - "space-around"



## 2.2.11 'align-content'

A veces, el tamaño total de la cuadrícula puede ser inferior al tamaño de nuestro contenedor. Esto puede ocurrir si todos los elementos de la rejilla están dimensionados con unidades no flexibles como px. En este caso se puede establecer la alineación de la rejilla dentro del contenedor padre. Por tanto, esta propiedad **alinea la cuadrícula a lo largo del eje de bloque (columna o eje y)** (a diferencia de **justify-content** que alinea la cuadrícula a lo largo del eje de línea (fila o eje x)).

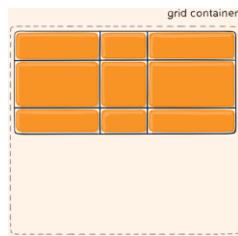
Su **sintaxis** es la siguiente:

```
.container {

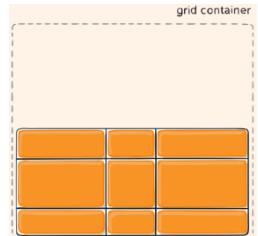
  align-content: start | end | center | stretch | space-around |
  space-between | space-evenly;}
```



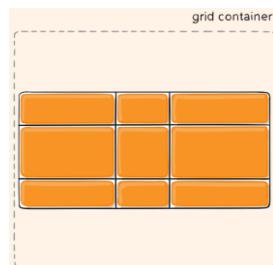
- **start**: alinea la rejilla o cuadrícula con el borde inicial del contenedor.



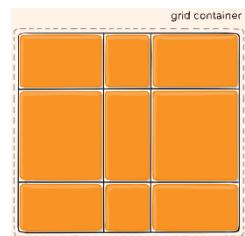
- **end**: alinea la rejilla o cuadrícula con el borde final del contenedor.



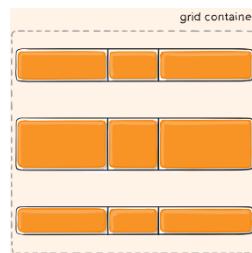
- **center**: alinea la rejilla en el centro del contenedor.



- **stretch**: cambia el tamaño de los elementos de la cuadrícula para que ocupe toda la altura del contenedor de la cuadrícula.



- **space-around**: coloca una cantidad uniforme de espacio entre cada elemento de la cuadrícula, con espacios de tamaño medio en los extremos.

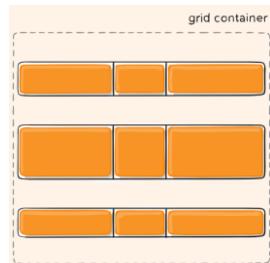


- **space-between**: coloca una cantidad uniforme de espacio entre cada elemento de la cuadrícula, sin espacio en los extremos.





- **space-evenly**: coloca una cantidad uniforme de espacio entre cada elemento de la cuadrícula, incluidos los extremos.



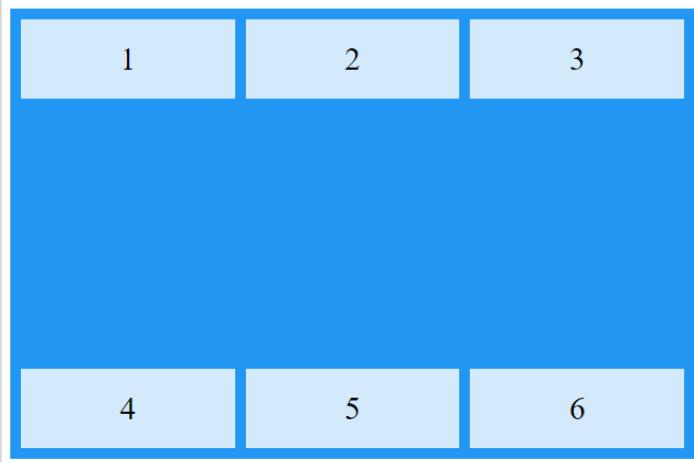
A continuación, se puede ver un **ejemplo** sencillo:

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  height: 400px;
  align-content: space-between;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>

<h1>The align-content Property - "space-between"</h1>
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
</body>
</html>
```

### The align-content Property - "space-between"



## 2.2.12 'place-content'

La propiedad **place-content** establece las propiedades **align-content** y **justify-content** en una única línea de código. Su sintaxis es la siguiente:

```
.container {  
  place-content: <align-content> / <justify-content>; }
```

De manera que si se omite el segundo valor (**<justify-content>**), el primer valor es asignado a ambas propiedades.

Todos los navegadores principales, **excepto Edge**, admiten la propiedad abreviada **place-content**.





Un **ejemplo** sencillo de esta propiedad sería el siguiente:

```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
  height: 350px;
  width: 60%;
  border: solid black 1px;
  display: grid;
  place-content: space-around center;
}

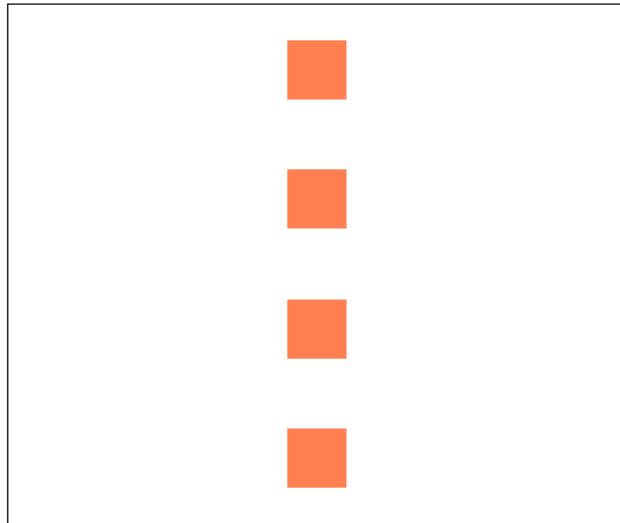
#container > div {
  inline-size: 40px;
  block-size: 40px;
  margin: 2px;
  background-color: coral;
}
</style>
</head>
<body>

<h1>The place-content Property with
grid</h1>

<div id="container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>

</body>
</html>
```

## The place-content Property with grid



### 2.2.13 'grid-auto-columns' y 'grid-auto-rows'

Esta propiedad especifica el tamaño de las columnas/filas de la cuadrícula de aquellas que se crearon sin tener un tamaño explícito. En otras palabras, esta propiedad establece el **tamaño de las columnas/filas implícitas** y cualquier otra columna/fila que no haya sido explícitamente dimensionada en la propiedad de columnas/filas de plantilla de cuadrícula.

Su **sintaxis** es la siguiente:

```
.container {
  grid-auto-columns: auto | max-content | min-content | length;
}
```



```
.container {  
    grid-auto-rows: auto | max-content | min-content | length;  
}
```

- **auto**. Es el valor por defecto. El tamaño de las columnas viene determinado por el tamaño del contenedor. El tamaño de las filas viene determinado por el tamaño del elemento más grande de la fila.
- **max-content**. Establece el tamaño de cada columna/fila según el elemento más grande de la columna/fila.
- **min-content**. Establece el tamaño de cada columna/fila según el elemento más pequeño de la columna/fila.
- **length**. Establece el tamaño de las columnas/filas, utilizando un valor de longitud legal (por ejemplo, %, px, em, etc).

- La propiedad **grid-template-rows** anula a la propiedad **grid-auto-rows**.
- La propiedad **grid-template-columns** anula a la propiedad **grid-auto-columns**.



A continuación, se expone un **ejemplo** sencillo.

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
.item1 { grid-area: 1 / 1 / 2 / 2; }  
.item2 { grid-area: 1 / 2 / 2 / 3; }  
.item3 { grid-area: 1 / 3 / 2 / 4; }  
.item4 { grid-area: 2 / 1 / 3 / 2; }  
.item5 { grid-area: 2 / 2 / 3 / 3; }  
.item6 { grid-area: 2 / 3 / 3 / 4; }  
.grid-container {  
    display: grid;  
    grid-auto-rows: 140px;  
    grid-auto-columns: auto;  
    grid-gap: 10px;  
    background-color: #2196F3;  
    padding: 20px;  
}  
.grid-container > div {  
    background-color: rgba(255, 255, 255, 0.8);  
    text-align: center;  
    padding: 10px 0;  
    font-size: 50px;  
}  
</style>  
</head>  
<body>  


# grid-auto-rows y grid-auto-columns



Establecemos el tamaño de 80px por columna y 140px por fila:



1



2



3



4



5



6



Establecemos el tamaño de 80px por columna y 140px por fila:</p><div class="grid-container"><div class="item1">1</div><div class="item2">2</div><div class="item3">3</div><div class="item4">4</div><div class="item5">5</div><div class="item6">6</div></div>


```

## grid-auto-rows y grid-auto-columns

Establecemos el tamaño de 80px por columna y 140px por fila:

1	2	3
4	5	6



## 2.2.14 'grid-auto-flow'

Si tenemos elementos en la cuadrícula que no hemos colocado explícitamente, la propiedad **grid-auto-flow** se encarga de colocarlos automáticamente. Por tanto, esta propiedad controla cómo se insertan los elementos autocolocados en la cuadrícula. La colocación puede ser por fila o por columna.

Su **sintaxis** es la siguiente:

```
.container {  
    grid-auto-flow: row | column | dense | row dense | column dense;  
}
```

- **row**. Es el valor por defecto. Coloca los elementos rellenando cada fila.
- **column**. Coloca los elementos rellenando cada columna.
- **dense**. Coloca los elementos llenando los huecos de la cuadrícula.
- **row dense**. Coloca los elementos rellenando cada fila, y rellena los huecos de la rejilla.
- **column dense**. Coloca los elementos rellenando cada columna, y rellena los huecos de la rejilla.

A continuación, se expone un **ejemplo** sencillo.

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
.grid-container {  
    display: grid;  
    grid-template-columns: auto auto auto;  
    grid-template-rows: auto auto;  
    grid-gap: 10px;  
    background-color: #2196F3;  
    padding: 10px;  
}  
.grid-container > div {  
    background-color: rgba(255, 255, 255, 0.8);  
    text-align: center;  
    padding: 20px 0;  
    font-size: 30px;  
}  
</style>  
</head>  
<body>  


## grid-auto-flow: column



Insert items column by column:



|   |   |
|---|---|
| 1 | 3 |
| 2 | 4 |



## grid-auto-flow: row



Insert items row by row:



|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 |   |   |

  
</body>  
</html>
```

### grid-auto-flow: column

Insert items column by column:

1	3
2	4

### grid-auto-flow: row

Insert items row by row:

1	2	3
4		



## 2.2.15 'grid'

Esta propiedad nos permite establecer todas las siguientes propiedades en una única declaración o sentencia de código: `grid-template-rows`, `grid-template-columns`, `grid-template-areas`, `grid-auto-rows`, `grid-auto-columns`, y `grid-auto-flow`.

Su sintaxis es la siguiente:

```
.container {  
    grid: none |grid-template-rows/grid-template-columns | grid-template-  
    areas | grid-template-rows/[grid-auto-flow] grid-auto-columns |  
    [grid-auto-flow] grid-auto-rows/grid-template-columns;  
}
```

A continuación, se muestran un par de **ejemplos**.

1

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
.grid-container {  
    display: grid;  
    grid: 150px / auto auto auto;  
    grid-gap: 10px;  
    background-color: #2196F3;  
    padding: 10px;  
}  
  
.grid-container > div {  
    background-color: rgba(255, 255, 255, 0.8);  
    text-align: center;  
    padding: 20px 0;  
    font-size: 30px;  
}  
</style>  
</head>  
<body>  
  
<h1>The grid Property</h1>  
<p>The grid is a shorthand property for all grid container properties.</p>  
<p>This grid layout has three columns, and the first row is 150px  
high:</p>  
  
<div class="grid-container">  
    <div class="item1">1</div>  
    <div class="item2">2</div>  
    <div class="item3">3</div>  
    <div class="item4">4</div>  
    <div class="item5">5</div>  
    <div class="item6">6</div>  
</div>  
</body>  
</html>
```

### The grid Property

The grid is a shorthand property for all grid container properties.

This grid layout has three columns, and the first row is 150px high:

1	2	3
4	5	6

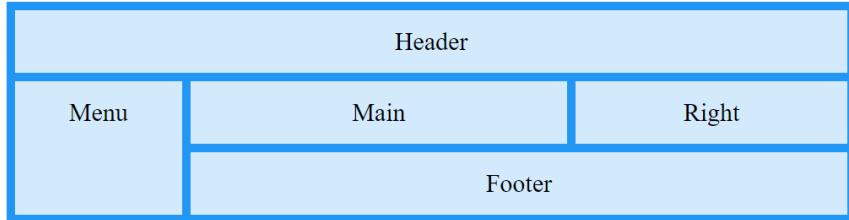


## 2

```
<!DOCTYPE html>
<html>
<head>
<style>
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }
.grid-container {
  display: grid;
  grid:
    'header header header header header'
    'menu main main right right'
    'menu footer footer footer footer';
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>A Webpage Template</h1>
<p>This grid layout contains six columns and three rows:</p>
<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">Main</div>
  <div class="item4">Right</div>
  <div class="item5">Footer</div>
</div>
</body>
</html>
```

### A Webpage Template

This grid layout contains six columns and three rows:



## 2.3 Propiedades de los hijos o grid items

### PROPIEDADES DE LOS CONTENEDORES HIJO O GRID ITEMS.

<b>grid-column-start</b>	Es la línea de columna donde comienza el elemento.
<b>grid-column-end</b>	Es la línea de columna donde termina el elemento.
<b>grid-column</b>	Esta propiedad incluye a las dos anteriores (grid-column-start y grid-column-end) en una línea de código. Especifica en qué columna colocar un elemento y cuántas columnas abarcará.
<b>grid-row-start</b>	Es la línea de fila donde comienza el elemento.
<b>grid-row-end</b>	Es la línea de fila donde termina el elemento.
<b>grid-row</b>	Esta propiedad incluye a las dos anteriores (grid-row-start y grid-row-end) en una línea de código. Especifica en qué fila colocar un elemento y cuántas filas abarcará.
<b>grid-area</b>	Especifica el tamaño y la ubicación de un elemento en un diseño de rejilla. Es una <u>propiedad abreviada</u> para las propiedades: <b>grid-row-start</b> , <b>grid-column-start</b> , <b>grid-row-end</b> y <b>grid-column-end</b>
<b>justify-self</b>	Alinea <b>un elemento</b> o item de la cuadrícula dentro de una celda a lo largo del eje <b>en línea u horizontal (fila o eje x)</b> .



<b>align-self</b>	Alinea <b>un elemento</b> o ítem de la cuadrícula dentro de una celda a lo largo del eje <b>en bloque o vertical (columna o eje y)</b> .
<b>place-self</b>	Establece las propiedades <b>align-self</b> y <b>justify-self</b> en una única línea de código

### 2.3.1 ‘grid-column-start’, ‘grid-column-end’, y ‘grid-column’

Estas propiedades determinan la **ubicación de un elemento hijo** de la cuadrícula haciendo referencia a líneas específicas de la cuadrícula.

- **grid-column-start** es la línea de columna donde comienza el elemento.
- **grid-column-end** es la línea de columna donde termina el elemento.
- **grid-column**. Esta propiedad incluye a las dos anteriores (**grid-column-start** y **grid-column-end**), y por tanto especifica en qué columna colocar un elemento y cuántas columnas abarcará.

Su **sintaxis** es la siguiente:

```
.item {  
    grid-column-start: auto | span n | column-line;  
    grid-column-end: auto | span n | column-line; }
```

```
.item {  
    grid-column: grid-column-start / grid-column-end; }
```

- **auto**. Es el valor por defecto.
  - En el caso de usar **grid-column-start**, el elemento se colocará siguiendo el flujo.
  - En el caso de usar **grid-column-end**, el elemento abarcará una columna.
- **span n**. Especifica el número de columnas que abarcará el elemento.
- **column-line**.
  - En el caso de usar **grid-column-start**, especifica en qué columna comenzar la visualización del elemento.



- En el caso de usar `grid-column-end`, especifica en qué columna debe terminar la visualización del elemento.

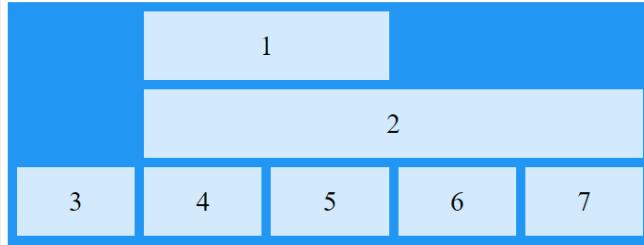
A continuación, se muestra un **ejemplo** sencillo.

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
.item1 {
  grid-column: 2 / 4;
}
.item2 {
  grid-column-start: 2;
  grid-column-end: span 4;
}
</style>
</head>
<body>
<h1>The grid-column Property</h1>


<div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>


</body>
</html>
```

### The grid-column Property



## 2.3.2 'grid-row-start', 'grid-row-end' y 'grid-row'

Estas propiedades determinan la **ubicación de un elemento hijo** de la cuadrícula haciendo referencia a líneas específicas de la cuadrícula.

- `grid-row-start` es la línea de fila donde comienza el elemento.
- `grid-row-end` es la línea de fila donde termina el elemento.
- `grid-row`. Esta propiedad incluye a las dos anteriores (`grid-row-start` y `grid-row-end`), y por tanto especifica en qué fila colocar un elemento y cuántas filas abarcará.

Su **sintaxis** es la siguiente:

```
.item {
  grid-row-start: auto | row-line;
  grid-row-end: auto | row-line | span n; }
```

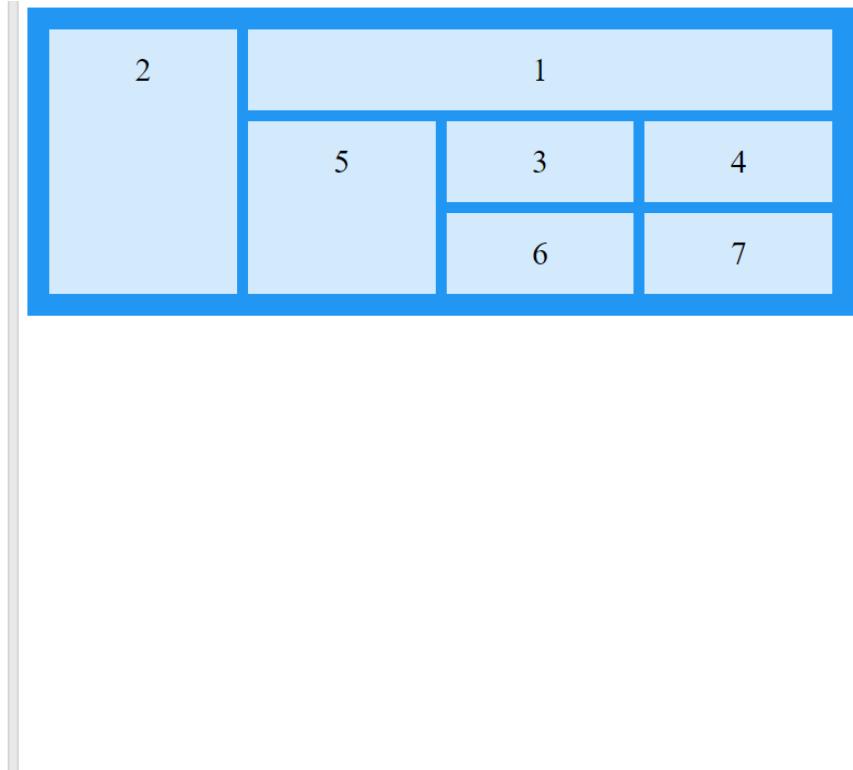
```
.item {
  grid-row: grid-row-start / grid-row-end; }
```



- **auto**. Es el valor por defecto.
  - En el caso de usar `grid-row-start`, el elemento se colocará siguiendo el flujo.
  - En el caso de usar `grid-row-end`, el elemento abarcará una fila.
- **row-line**.
  - En el caso de usar `grid-row-start`, especifica en qué fila debe comenzar la visualización del elemento.
  - En el caso de usar `grid-row-end`, especifica en qué fila finaliza la visualización del elemento.
- **span n**. Especifica el número de filas que abarcará el elemento.

A continuación, se muestra un **ejemplo** sencillo.

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 20px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
.item1 {
  grid-column-start: 2;
  grid-column-end: span 3; }
.item2 {
  grid-row-start: 1;
  grid-row-end: span 3; }
.item5 {
  grid-row: 2 / span 2 ; }
</style>
</head>
<body>
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
</div>
</body>
</html>
```





### 2.3.3 ‘grid-area’

La propiedad `grid-area` especifica el tamaño y la ubicación de un elemento en un diseño de rejilla. Además, es una propiedad abreviada para las siguientes propiedades:

- `grid-row-start`
- `grid-column-start`
- `grid-row-end`
- `grid-column-end`

La propiedad `grid-area` también puede utilizarse para asignar un nombre a un elemento de la rejilla. Los elementos de la rejilla con nombre pueden ser referenciados por la propiedad `grid-template-areas` del contenedor de la rejilla.

Su **sintaxis** es la siguiente:

```
.item {  
    grid-area: grid-row-start / grid-column-start / grid-row-end /  
              grid-column-end | itemname; }
```

A continuación, se muestran un par de ejemplos de los casos mencionados anteriormente:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
.grid-container {  
    display: grid;  
    grid-template-columns: auto auto auto auto;  
    grid-gap: 10px;  
    background-color: #2196F3;  
    padding: 10px;  
}  
.grid-container > div {  
    background-color: rgba(255, 255, 255, 0.8);  
    text-align: center;  
    padding: 20px 0;  
    font-size: 30px; }  
.item1 {  
    grid-area: 2 / 1 / span 2 / span 3;  
}  
</style>  
</head>  

```

1

#### La propiedad grid-area

Se puede usar `grid-area` para especificar donde colocar un item.

La sintaxis es: `grid-row-start / grid-column-start / grid-row-end / grid-column-end`.

El Item1 empezara en la fila 2 y columna 1, y se ocupará 2 filas y 3 columnas:

2	3	4	5
1			6
			7



2

```
<!DOCTYPE html>
<html>
<head>
<style>
.item1 {
  grid-area: myArea;
}
.grid-container {
  display: grid;
  grid-template-areas: 'myArea myArea myArea myArea myArea';
  grid-gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>

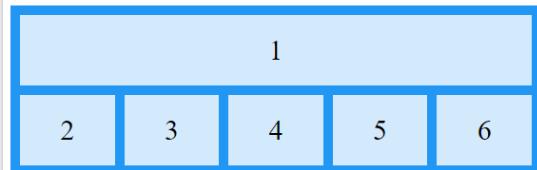
<h1>The grid-area Property</h1>
<p>Se puede usar <em>grid-area</em> para nombrar los items hijo de la cuadrícula.</p>
<p>Nos podemos referirnos al nombre cuando se implementa un grid layout, utilizando para ello la propiedad. <em>grid-template-areas</em></p>
<p>Item1, es llamado como "myArea" y ocuparaá las 5 columnas.</p>
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
</div>
</body>
</html>
```

## The grid-area Property

Se puede usar *grid-area* para nombrar los items hijo de la cuadrícula.

Nos podemos referirnos al nombre cuando se implementa un grid layout, utilizando para ello la propiedad: *grid-template-areas*

Item1, es llamado como "myArea" y ocuparaá las 5 columnas.



### 2.3.4 'justify-self'

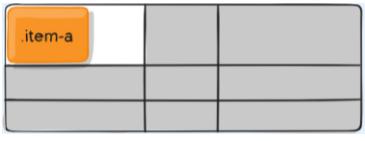
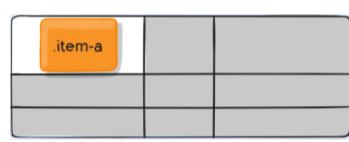
La propiedad **justify-self** alinea un elemento o item de la cuadrícula dentro de una celda a lo largo del eje **en línea (fila o eje x)**, a diferencia la propiedad **align-self**, que alinea a lo largo del eje de bloque (columna o eje y)). Este valor se aplica a un elemento de cuadrícula dentro de **una única celda**.

Su sintaxis es la siguiente:

```
.item {
  justify-self: auto | normal | stretch | positional alignment |
  overflow-alignment | baseline alignment;
}
```

- **auto** (valor por defecto). El valor de la propiedad **justify-self** del contenedor de rejilla se hereda.



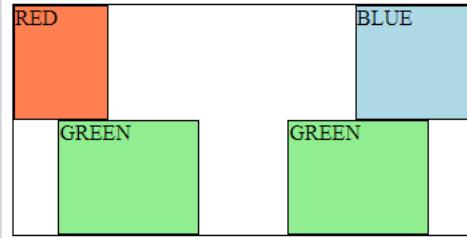
- **normal**. Depende del contexto de diseño, pero es similar a '**stretch**' cuando el tamaño no está establecido. Si se establece el tamaño, el valor de la propiedad se comporta como '**start**'.
- **stretch**. Se estira para llenar la celda de la rejilla si **inline-size (width)** no se ha definido.
- **start**. Alinea los elementos al principio en la dirección inline (fila o eje x).
- **left**. Alinea los elementos a la izquierda en la dirección inline (fila o eje x).
- **center**. Alinea los elementos al centro en la dirección inline (fila o eje x).
- **end**. Alinear los elementos al final en la dirección inline (fila o eje x).
- **right**. Alinear los elementos a la derecha en la dirección inline (fila o eje x).
- **overflow-alignment**
  - '**safe**' establece la alineación del elemento a 'start' si el contenido se desborda.
  - '**unsafe**' mantiene el valor de alineación independientemente de si el contenido del elemento se desborda o no.
- **baseline alignment**. El elemento se alinea con la línea base del parent.

A continuación, se presenta un **ejemplo** sencillo:



```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
  width: 50%;
  aspect-ratio: 2/1;
  border: 1px solid black;
  display: grid;
  grid-template-columns: 1fr 1fr;
  justify-items: center;
}
#container > div {
  border: 1px solid black; }
.blue {
  background-color: lightblue;
  width: 50%;
  justify-self: right;
}
.red {
  background-color: coral;
  width: 40%;
  justify-self: left;
}
.green {
  background-color: lightgreen;
  width: 60%; }
</style>
</head>
<body>
<h2>justify-items vs. justify-self</h2>
<div id="container">
  <div class="red">RED</div>
  <div class="blue">BLUE</div>
  <div class="green">GREEN</div>
  <div class="green">GREEN</div>
</div>
</body>
</html>
```

### justify-items vs. justify-self



### 2.3.5 'align-self'

La propiedad **align-self** alinea un elemento o item de la cuadrícula a lo largo del **eje de bloque (columna o eje y)**, a diferencia de **justify-self** alinea a lo largo del eje en línea (fila o eje x).

Este valor se aplica a un elemento de cuadrícula dentro de **una única celda**.

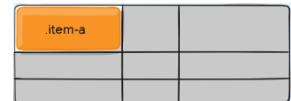
**Nota:** la propiedad **align-self** anula la propiedad **align-items** de css grid o flexbox.

Su **sintaxis** es la siguiente:

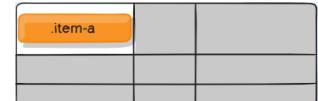
```
.item {
  align-self: auto | stretch | center | start | end | baseline;
```



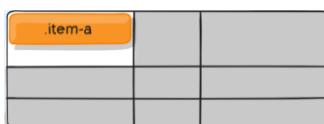
- **auto** (por defecto). El elemento hereda la propiedad align-items de su contenedor padre, o "stretch" si no tiene contenedor padre.
- **stretch**. El elemento se posiciona para ajustarse al contenedor.



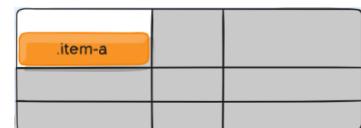
- **center**. El elemento se posiciona en el centro del contenedor.



- **start**. El elemento se sitúa al principio del contenedor.



- **end**. El elemento se posiciona al final del contenedor.

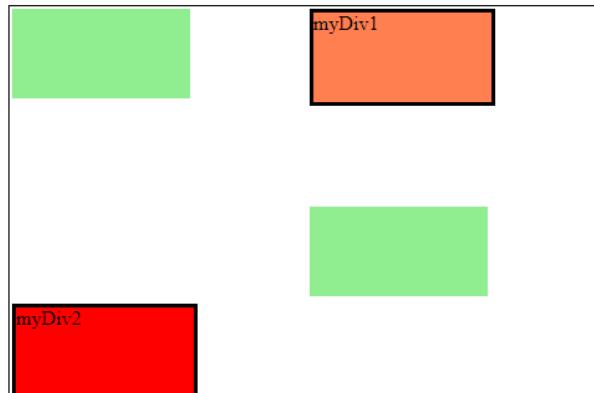


- **baseline**. El elemento se posiciona en la línea base del contenedor.

## Ejemplo sencillo:

```
<!DOCTYPE html>
<html>
<head>
<style>
#container {
    width: 60%;
    aspect-ratio: 3/2;
    border: solid black 1px;
    display: grid;
    grid-template-columns: 1fr 1fr; }
#container > div {
    width: 60%;
    aspect-ratio: 2;
    margin: 2px; }
.normalDiv {
    background-color: lightgreen; }
#myDiv1 {
    border: solid black 3px;
    background-color: coral;
    align-self: start; }
#myDiv2 {
    border: solid black 3px;
    background-color: red;
    align-self: end; }
</style>
</head>
<body>
<h1>The align-self Property</h1>
<div id="container">
    <div class="normalDiv"></div>
    <div id="myDiv1">myDiv1</div>
    <div id="myDiv2">myDiv2</div>
    <div class="normalDiv"></div>
</div>
</body>
</html>
```

## The align-self Property





## 2.3.6 ‘place-self’

Esta propiedad establece las propiedades `align-self` y `justify-self` en una única sentencia o línea de código. Su **sintaxis** es la siguiente:

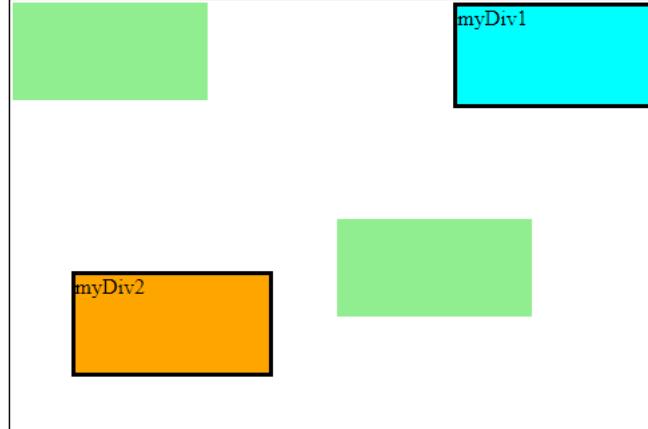
```
.item {  
    place-self: <align-self> / <justify-self>;  
}
```

**Si el segundo valor es omitido, el primer valor se asigna a ambas propiedades.**

A continuación, se muestra un ejemplo sencillo:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
#container {  
    width: 60%;  
    aspect-ratio: 3/2;  
    border: solid black 1px;  
    display: grid;  
    grid-template-columns: 1fr 1fr; }  
#container > div {  
    width: 60%;  
    aspect-ratio: 2;  
    margin: 2px;  
}  
.normalDiv {  
    background-color: lightgreen; }  
#myDiv1 {  
    border: solid black 3px;  
    background-color: aqua;  
    place-self: start end;  
}  
#myDiv2{  
    border: solid black 3px;  
    background-color: orange;  
    place-self: center;  
}  
</style>  
</head>  
<body>  
<h1>The place-self Property</h1>  
<div id="container">  
    <div class="normalDiv"></div>  
    <div id="myDiv1">myDiv1</div>  
    <div id="myDiv2">myDiv2</div>  
    <div class="normalDiv"></div>  
</div>  
</body>  
</html>
```

### The place-self Property





### 3. Flexbox vs. CSS Grid

Los procesos de desarrollo web han ido evolucionando desde la aparición de Internet: los desarrolladores han lanzado modelos de diseño responsive para facilitar la creación y el mantenimiento de las páginas web. **Junto con Bootstrap**, la mayor rivalidad entre los modelos de diseño es la de **CSS Grid vs. Flexbox**.

#### 3.1 Diferencias entre Flexbox y Grid

A continuación, veremos cuáles son las **diferencias** entre CSS Grid y Flexbox.

- 1) **CSS Grid Layout** es un sistema **bidimensional**, lo que significa que puede manejar tanto columnas como filas, a diferencia de **flexbox** que es en gran medida un sistema **unidimensional** (ya sea en una columna o una fila).

#### Flexbox

*One dimension* →

Home    Search    Logout

- Ejemplos:

1

2



## CSS Grid

*Two dimensional* →

Header

Content

Footer

• Ejemplos:

1

2

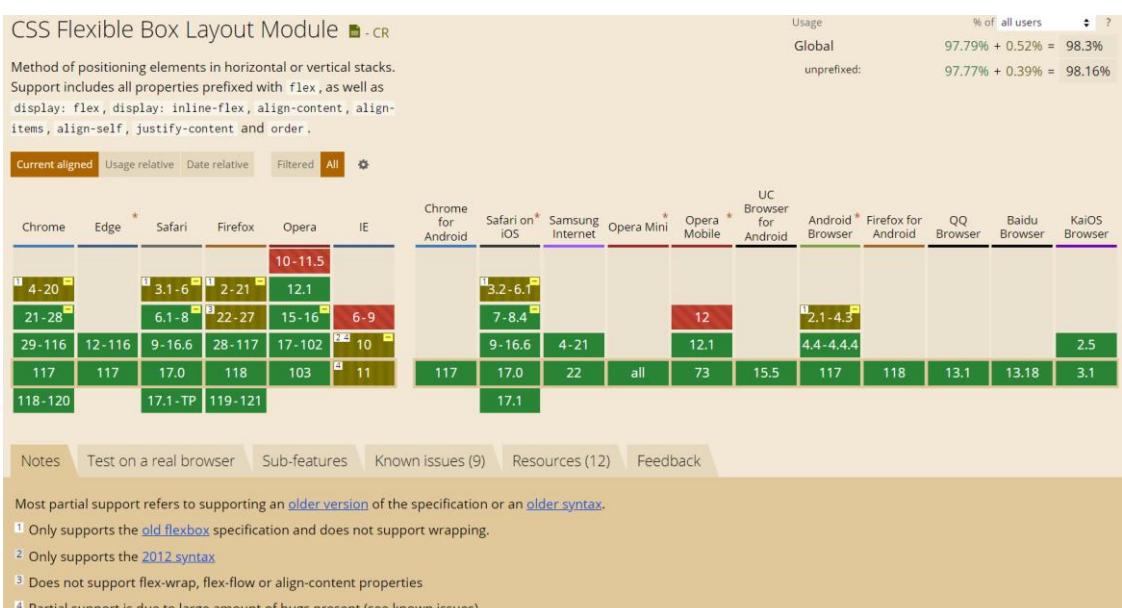
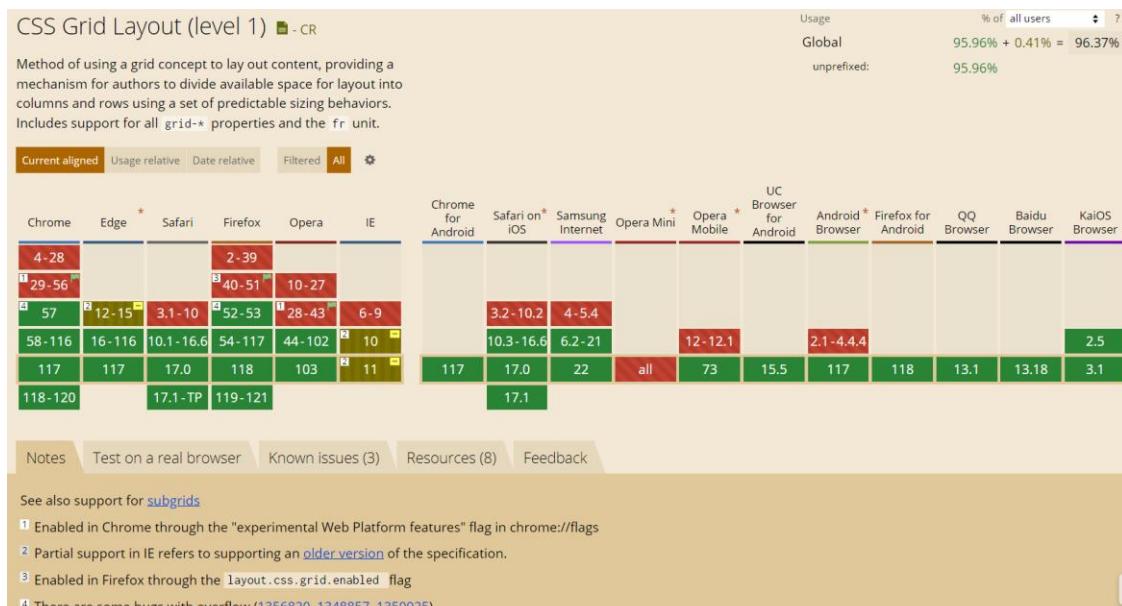
### 2) Una diferencia fundamental entre CSS Grid y Flexbox es su **enfoque**:

- El enfoque de **CSS Grid** es el **diseño primero**. Si lo que quieres es diseñar primero, es decir, crear el diseño y luego colocar los elementos en él, entonces estaremos mejor con CSS Grid.
- El enfoque de **Flexbox** es el **contenido primero**. Si lo primero es el contenido, es decir, si tenemos artículos que queremos colocar en un contenedor y espaciar uniformemente, entonces estaremos mejor con Flexbox o Bootstrap.

Por tanto, si eres consciente de tu contenido antes de hacer el diseño, entonces opta ciegamente por Flexbox y si no, opta por CSS Grid.



- 3) El diseño **Flexbox** es el más apropiado para los componentes de una **aplicación** (ya que la mayoría de ellos son fundamentalmente lineales), y los diseños a pequeña escala, mientras que el diseño **Grid** está pensado para **diseños a mayor escala** que no son lineales en su diseño.
- 4) Si sólo se necesita **definir un diseño como una fila o una columna**, entonces probablemente necesitemos **flexbox**. Si queremos definir una **cuadrícula** y encajar el contenido en ella en dos dimensiones, necesitamos **Grid**.
- 5) Flexbox y CSS Grid tienen distinta **compatibilidad o soporte** con los distintos navegadores web.





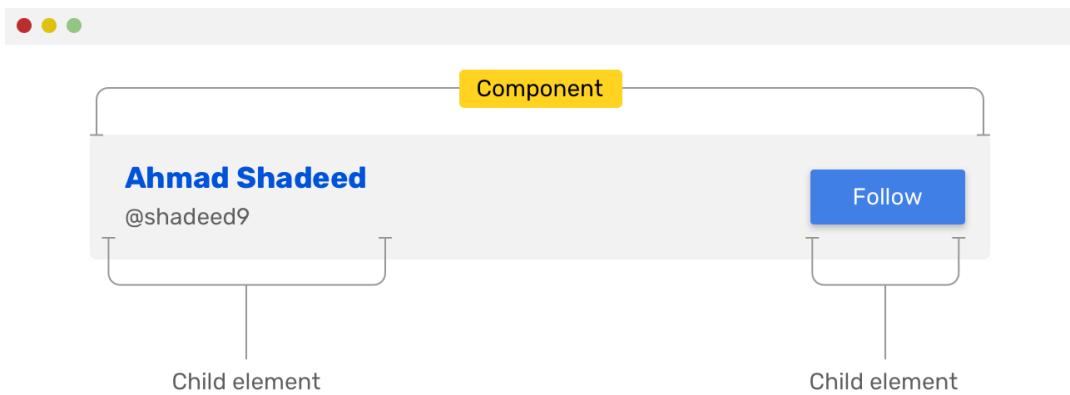
## 3.2 ¿Cuál utilizar?

Decidir entre CSS grid y flexbox puede ser un poco difícil (a veces). Por ello, algunas preguntas iniciales que debes plantearte a la hora de elegir entre ellas son:

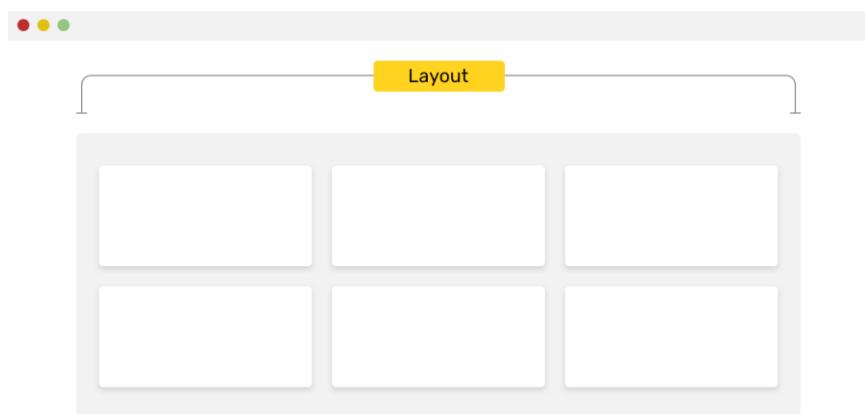
- ¿Cómo se muestran los elementos hijo del componente? ¿En línea o como columnas y filas?
- ¿Cómo se espera que funcione el componente en distintos tamaños de pantalla?

La información completa de este subapartado se recoge en: <https://ishadeed.com/article/grid-layout-flexbox-components/>

La mayoría de las veces, si el componente que estás viendo tiene todos sus elementos secundarios mostrados en línea, lo más probable es que **flexbox** sea la mejor solución.



Sin embargo, si ves columnas y filas, entonces Grid es la solución:

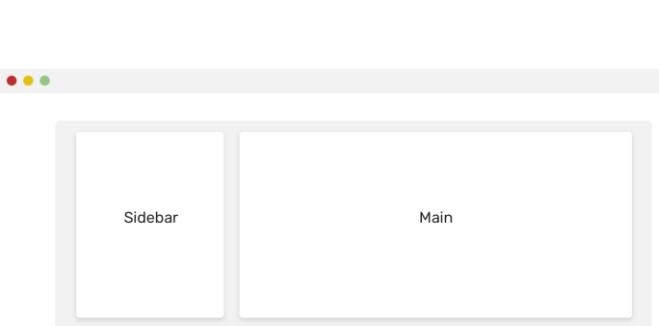


### 3.2.1 Grid: usos más frecuentes

En este apartado se indica en qué **partes de la interfaz web** es más frecuente utilizar Grid.



**1) Main y Sidebar (o barra lateral).** Cuando tienes una barra lateral y una zona principal, normalmente CSS Grid es una solución perfecta para construirlas.



```
<div class="wrapper">
  <aside>Sidebar</aside>
  <main>Main</main>
</div>
```

```
@media (min-width: 800px) {
  .wrapper {
    display: grid;
    grid-template-columns: 200px 1fr;
    grid-gap: 16px;
  }

  aside {
    align-self: start;
  }
}
```

Si no se utilizara `align-self: start` para el elemento `<aside>`, la **altura** de éste sería igual a la del elemento principal, independientemente de la longitud del contenido.

**2) Cards Grid (o rejilla de tarjetas).** Utilizar Grid para diseñar una cuadrícula de tarjetas es un uso perfecto de la misma.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  grid-gap: 16px;
}
```



**3) Diseño de sección (section layout)**

En el siguiente diseño, podemos utilizar la rejilla dos veces, el primer uso es para dividir el área en dos áreas (la barra lateral de contacto, el formulario), y el segundo uso es para la propia rejilla del formulario.



A screenshot of a contact form. The header is dark blue with three colored dots (red, yellow, green) in the top-left corner. On the left side, there's a sidebar with a blue background containing the text "Contact us", "We can't wait to hear from you. Feel free to get in touch with us.", and "Happy ST 000-111-222". The main form area has two input fields: "Full Name" and "Email address". Below them is a larger "Message" input field. At the bottom is a large blue button labeled "Send message".

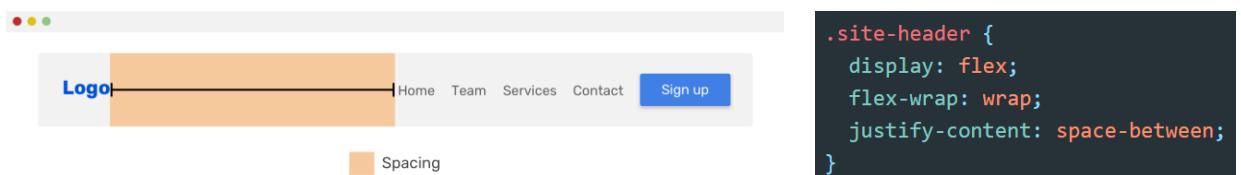


```
@media (min-width: 800px) {  
  .wrapper {  
    display: grid;  
    grid-template-columns: 200px 1fr;  
  }  
  
  .form-wrapper {  
    display: grid;  
    grid-template-columns: 1fr 1fr;  
    grid-gap: 16px;  
  }  
  
  .form-message,  
  .form-button {  
    grid-column: 1 / 3; /* let them take the full width  
  }  
}
```

### 3.2.2 Flexbox: usos más frecuentes

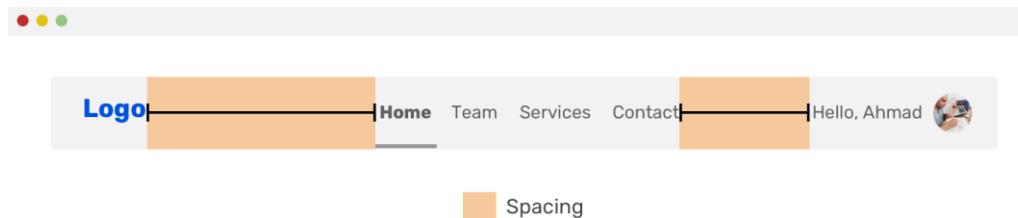
En este apartado se indica en qué **partes de la interfaz web** es más frecuente utilizar Flexbox.

- 1) **Menú de navegación.** El **90% de las veces**, la navegación de un sitio web debe construirse con CSS flexbox. El patrón más común es tener el logo a la izquierda y la navegación a la derecha. Eso es perfecto para flexbox.

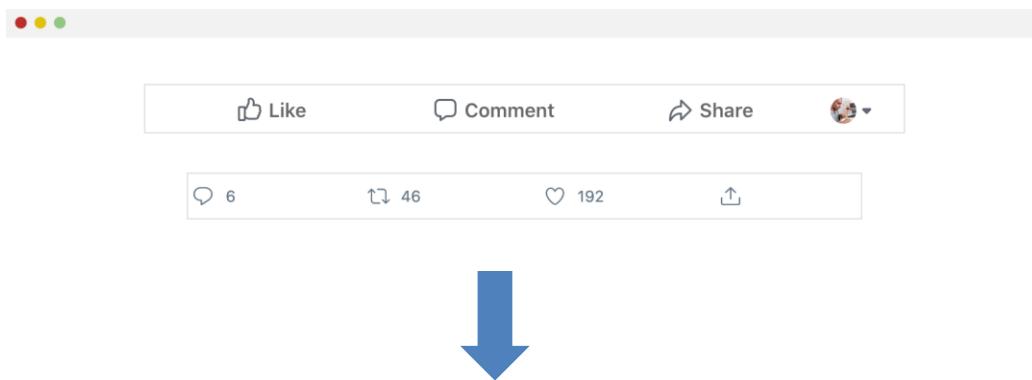




El mismo concepto puede funcionar también en el siguiente diseño. La estructura de navegación es un poco diferente, pero el espaciado entre los elementos se sigue haciendo con la propiedad *justify-content*.

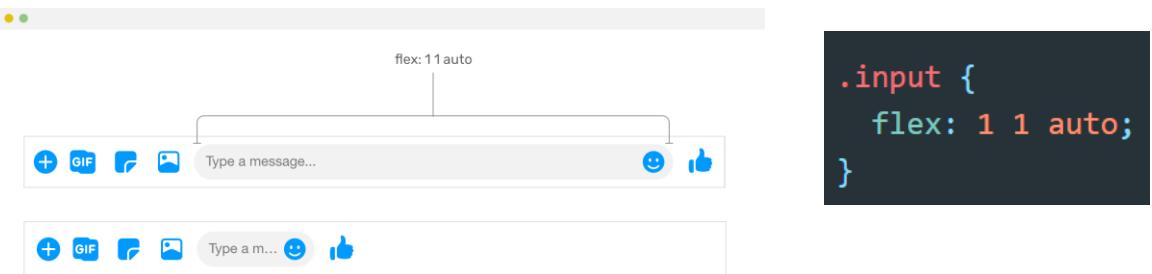


- 2) **Actions List (o listas de acciones).** Una lista de acciones consiste en botones de acción que el usuario puede tomar. Normalmente en este tipo de listas los elementos se muestran unos junto a otros, y se distribuyen horizontalmente.



```
.actions-list {  
    display: flex;  
}  
  
.actions-list__item {  
    flex: 1; /* expand the items to take the available space equally between them */  
}
```

- 3) **Elementos de un formulario.** La combinación de un campo de entrada (*input*) con un botón al lado es un caso de uso perfecto para Flexbox.



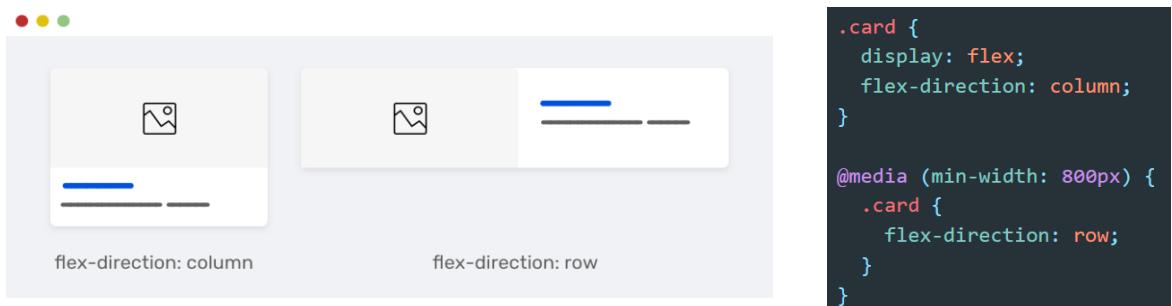
Si no utilizamos `flex: 1 1 auto` en el campo de texto, éste no se expandirá ni llenará el espacio restante.

- 4) **Threads y comentarios.** Otro uso común de flexbox es en el hilo de comentarios, tal y como se muestra en la siguiente imagen:



En este caso tenemos la foto del usuario, y el comentario propiamente dicho. El comentario está tomando el espacio restante de su elemento padre.

- 5) **Cards (o tarjetas).** Un componente de cards tiene muchas variaciones, pero el diseño más común es algo parecido a la maqueta de abajo.



A la izquierda, los elementos hijos de la tarjeta están apilados porque la dirección del `flex wrapper` es columna. Mientras que a la derecha, es lo contrario. La dirección utilizada es fila. Ten en cuenta que la fila es el valor predeterminado para flexbox.



Otra **variación** común para una tarjeta es tener un ícono con una etiqueta de texto debajo. Puede ser un botón, un enlace o simplemente un elemento decorativo. Considera el siguiente diseño:

The image shows two variations of a card design. Variation 1 consists of three colored boxes (red, green, purple) each containing an icon and a label: 'Cars', 'Food', and 'Hot Drinks'. Variation 2 shows the same items but with the text labels moved below the icons. To the right, a snippet of CSS code defines a class '.card' with properties: 'display: flex;', 'flex-direction: column;', 'align-items: center;', and a closing brace '}'.

```
.card {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
}
```

Observa cómo **el ícono y la etiqueta de texto están centrados horizontal y verticalmente**. Gracias a flexbox, esto es fácil de hacer.

- 6) **Tabs o pestañas / Botones del menú.** Cuando se trata de elementos que ocupan todo el ancho de la pantalla y tienen elementos que deben llenar todo el espacio disponible, entonces flexbox es la herramienta perfecta aquí.

The image shows a tabs menu with three items: 'Flights' (selected), 'Hotels', and 'Cars'. The 'Flights' tab has a blue underline. To the right, a snippet of CSS code defines a class '.tabs\_\_item' with a property 'flex-grow: 1;'.

```
.tabs__item {  
  flex-grow: 1;  
}
```

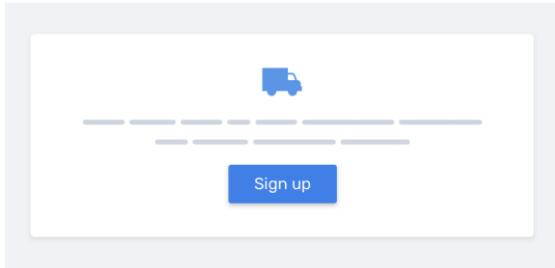
- 7) **Lista de características o features.** La dirección por defecto de flexbox es fila, pero podemos invertirla, creando un efecto como el siguiente:

The image shows a list of three items, each with a small icon and a dashed horizontal line. The items are arranged vertically but have a reversed row direction. To the right, a snippet of CSS code defines a class '.item' with a property 'flex-direction: row-reverse;'. The items are also enclosed in a container with a border.

```
.item {  
  flex-direction: row-reverse;  
}
```



**8) Centrar el contenido de una sección.** Para centrar el contenido de una sección horizontal y verticalmente, es más fácil hacerlo con Flexbox.



```
.hero {  
  display: flex;  
  flex-direction: column;  
  align-items: center; /* centers items horizontally */  
  justify-content: center; /* centers items vertically */  
  text-align: center;  
}
```

### 3.3 Combinar Flexbox y Grid

CSS Grid y Flexbox se pueden combinar. Ahora bien, ¿cómo combinarlos para crear diseños más eficientes e interesantes?

1. **El diseño es importante.** Es conveniente hacer un diseño previo antes de escribir el código, tal que así:



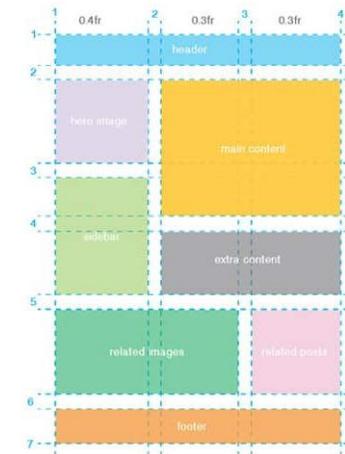


**La planificación es fundamental** en un diseño como éste. Es una buena idea esbozarlo primero y ver cómo se apilan las cosas, literalmente.

Para empezar el código, poner `display: grid;` es esencial; sin él, usar este tipo de layout no funcionará.

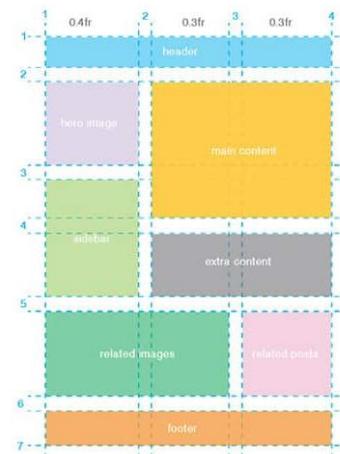
Una cosa a tener en cuenta aquí es que hay espacio entre los bloques de contenido. Esto se logra con `column-gap` y `row-gap`.

```
.container {  
    display: grid;  
    grid-template-columns: 0.4fr 0.3fr 0.3fr;  
    grid-column-gap: 10px;  
    grid-row-gap: 15px;  
}
```



**2. El header.** Debemos empezar por arriba, colocando la cabecera. En este diseño abarca todas las columnas y una fila.

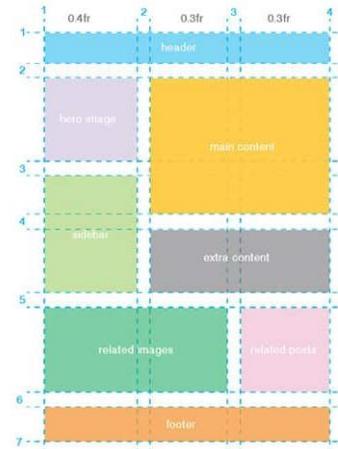
```
.header {  
    grid-column-start: 1;  
    grid-column-end: 4;  
    grid-row-start: 1;  
    grid-row-end: 2;  
    background-color: #d5c9e2;  
}
```



**3. El menú de navegación.** Flexbox es perfecto para colocar los elementos de cabecera, ya que hace el espaciado más fácil para la navegación.



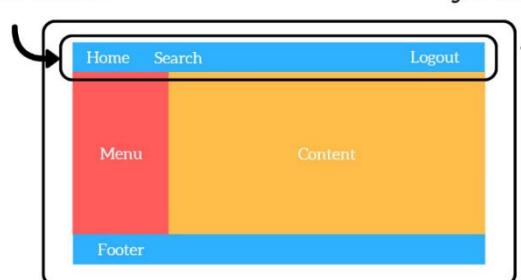
```
.header {  
    grid-column: 1 / 4;  
    grid-row: 1 / 2;  
    color: #9f9c9c;  
    text-transform: uppercase;  
    border-bottom: 2px solid #b0e0ea;  
    padding: 20px 0;  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}
```



Por tanto, para el logo y/o botón en el header, así como los ítems del menú es más fácil utilizar **Flexbox** con la propiedad ***justify-content*** para el espacio entre ellos (y para el ejemplo de diseño que estamos siguiendo).

Flexbox Container

Grid Container



#### 4. Contenido del grid en columnas.

Para aquellas ocasiones que requieren que los elementos se alineen en una dirección, lo que significa que es más "unidimensional", normalmente **Flexbox** es la mejor opción. Además, Flexbox es bueno para **escalar elementos dinámicamente**. Con esa técnica, se forma una bonita línea y es una manera eficiente de asegurarse de que todos los elementos tienen la misma altura.

#### 5. Contenido en filas con texto y botones.

En la sección "contenido extra" para el diseño que estamos cogiendo como referencia, si queremos añadir tres áreas con texto y botones, **flexbox** facilita el mantenimiento de la anchura establecida para tres columnas.



```
.extra {  
    grid-column: 2 / 4;  
    grid-row: 4 / 5;  
    padding: 1rem;  
    display: flex;  
    flex-wrap: wrap;  
    border: 1px solid #ececce;  
    justify-content: space-between; }  
  
.extra h3 {  
    font-weight: 500;  
    margin-bottom: .5rem;  
    flex: 1 0 auto; }  
  
.extra p {  
    font-size: .75rem;  
    margin-bottom: .75rem; }  
  
.extra button {  
    background-color: white;  
    border: 2px solid #50c6db;  
    color: #50c6db;  
    padding: .5rem; }  
  
.content-block-info {  
    flex: 0 1 calc(33% - .5rem); }
```

Ver la ayuda de CSS Grid Ver trucos y consejos para CSS Grid Layouts... <a href="#">Leer</a>	Ver la ayuda de Flexbox Ver trucos y consejos para Flexbox layouts... <a href="#">Leer</a>	Ver la ayuda combinada Ver trucos y consejos para layouts combinados... <a href="#">Leer</a>
--	--	--

Tenéis el **código completo** en el aula virtual, y algunas plantillas en esta url: [Jen Simmons' Lab of Layouts](#)

A continuación, se expone un **ejemplo aún más sencillo** en donde se **combina Grid con Flex**.

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="UTF-8">  
<title>Grid con Flex</title>  
<meta name="viewport" content="width=device-width, user-scalable=no,  
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  
<style>  
    .grid {  
        margin: auto; /*centramos el contenedor*/  
        display: grid;  
        grid-template-columns: repeat(3,1fr);  
        grid-template-rows: auto;  
        gap: 1em;  
        padding: 1em;  
        background: #212d40;  
        width: 95%;  
        height: 90vh;
```



```
/*Alineación global*/
justify-items: center; /*Horizontal*/
align-items: center; /*Vertical*/

/*Autoflow! */
grid-auto-flow: dense; /*row, column, dense*/ }

.grid-item {
/*usamos flex para centrar el contenido de los ítems, tanto
horizontalmente como verticalmente*/
width: 100%;
height: 100%;
background-color: gold;
display: flex;
justify-content: center;
align-items: center; }

.grid-featured-top {
background-color: aquamarine;
grid-column: 1 / span 3;
grid-row: 1; }

.grid-featured-bottom {
background-color: limegreen;
grid-column: 1/ span 3;
grid-row: 6; }

.grid-featured-medium {
background-color: red;
grid-row: 3/ span 3;
grid-column: 2; }

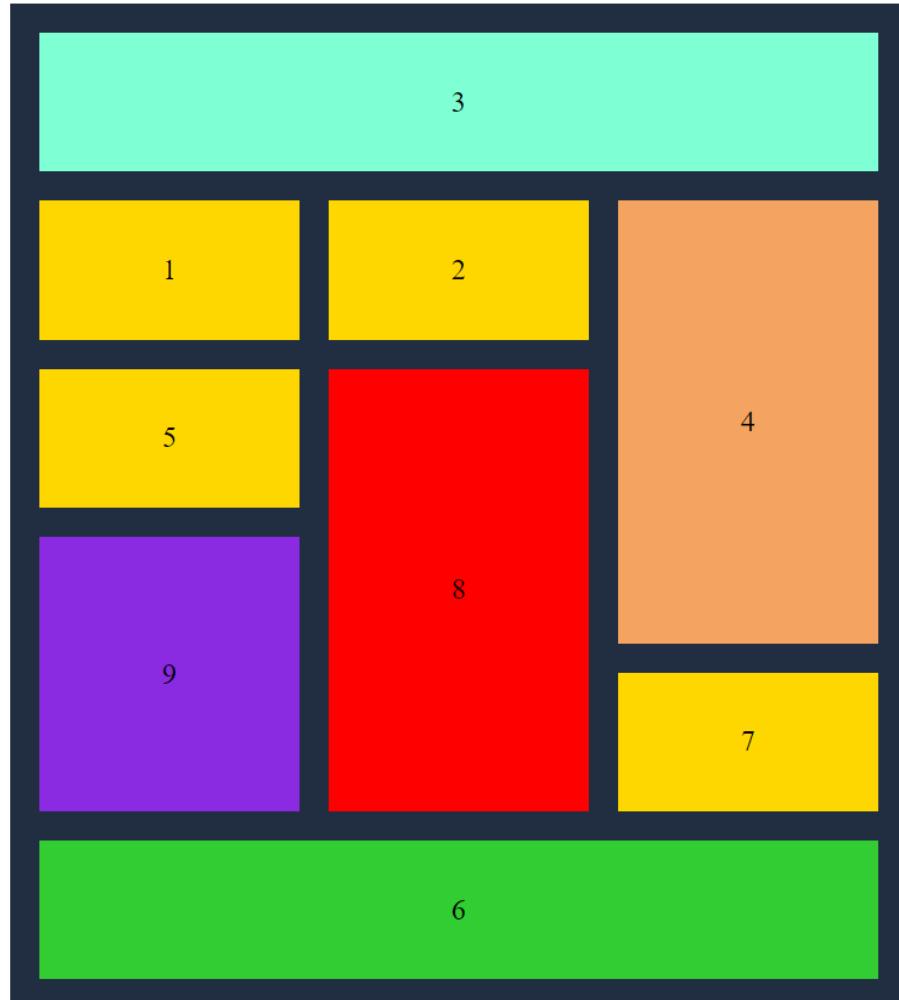
.grid-featured-medium-2 {
background-color: blueviolet;
grid-row: 4/ span 2;
grid-column: 1; }

.grid-featured-medium-3 {
background-color: sandybrown;
grid-row: 2/ span 3;
grid-column: 3; }

</style>
</head>
<body>
<div class="grid">
<div class="grid-item">1</div>
<div class="grid-item">2</div>
<div class="grid-item grid-featured-top">3</div>
<div class="grid-item grid-featured-medium-3">4</div>
<div class="grid-item">5</div>
<div class="grid-item grid-featured-bottom">6</div>
<div class="grid-item">7</div>
```



```
<div class="grid-item grid-featured-medium">8</div>
<div class="grid-item grid-featured-medium-2">9</div>
</div>
</body>
</html>
```





## 4. Maquetación multicolumna

La maquetación **multicolumna** o **multi-column layout** nos permite maquetar **texto** en varias columnas.

Las **propiedades** CSS para controlar las columnas son las siguientes:

- 1) **column-count**. Determina el número de columnas en que debe dividirse un elemento.

`column-count: number / auto;`

- **number**. Número óptimo de columnas en las que fluirá el contenido del elemento.
- **auto**. Valor por defecto. El número de columnas vendrá determinado por otras propiedades, como por ejemplo "**column-width**".

- 2) **column-fill**. Especifica cómo llenar las columnas, equilibradas o no.

`column-fill: balance / auto;`

- **balance**. Valor por defecto. Rellena cada columna con aproximadamente la misma cantidad de contenido, pero no permitirá que las columnas sean más altas que la altura (por lo tanto, las columnas podrían ser más cortas que la altura ya que el navegador distribuye el contenido horizontalmente de manera uniforme).
- **auto**. Rellena cada columna hasta que alcanza la altura, y hace esto hasta que se queda sin contenido (por lo tanto, este valor no llenará necesariamente todas las columnas ni las llenará uniformemente).

- 3) **column-gap**. Determina el **espacio** entre las columnas.

`column-gap: length / normal;`

- **length**. Longitud especificada que fijará el espacio entre las columnas.
- **normal**. Valor por defecto. Especifica un espacio normal entre las columnas. El W3C sugiere un valor de 1em.



4) **column-rule-style**. Define el estilo del borde entre columnas.

```
column-rule-style: none | hidden | dotted | dashed | solid | double  
| groove | ridge | inset | outset;
```

5) **column-rule-width**. Define el ancho del borde entre columnas.

```
column-rule-width: medium | thin | thick | length;
```

6) **column-rule-color**. Define el color del borde entre columnas.

```
column-rule-color: color;
```

7) **column-rule** que define un borde entre columnas. Esta propiedad resume las tres anteriores (**column-rule-style**, **column-rule-width**, **column-rule-color**) en una sola línea de código.

```
column-rule: column-rule-width column-rule-style column-rule-color;
```

8) **column-span**. Especifica cuántas columnas debe abarcar un elemento.

```
column-span: none | all;
```

- **none**. Valor por defecto. El elemento debe abarcar una columna.
- **all**. El elemento debe abarcar todas las columnas.

9) **column-width**. Especifica un ancho óptimo sugerido para las columnas.

```
column-width: auto | length;
```

- **auto**. Valor por defecto. El ancho de la columna será determinado por el navegador.
- **length**. Una longitud que especifica el ancho de las columnas. El número de columnas será el mínimo necesario para mostrar todo el contenido a través del elemento.



10) **columns**. Es una propiedad abreviada para las propiedades **column-width** y **column-count**.

La parte **column-width** definirá la anchura mínima de cada columna, mientras que la parte **column-count** definirá el número máximo de columnas. Utilizando esta propiedad, el diseño multicolumna se dividirá automáticamente en una sola columna con anchos de navegador reducidos, sin necesidad de media queries u otras reglas.

```
columns: auto | column-width column-count;
```

A continuación, se muestra un **ejemplo** sencillo.

```
<!DOCTYPE html>
<html>
<head>
<style>
.newspaper {
    column-count: 3;
    column-gap: 40px;
    column-rule: 4px double #ff00ff;
}

h2 {
    column-span: all;
}

</style>
</head>
<body>
<h1>The column-rule Property</h1>
<div class="newspaper">
<h2>Lorem ipsum dolor sit amet</h2>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.
</div>
</body>
</html>
```

## The column-rule Property

### Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.

eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.

option congue nihil imperiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.



## 5. Preprocesadores CSS Less y Sass.

Los preprocesadores de CSS son lenguajes de programación **que amplían las capacidades predeterminadas de CSS**. Nos permiten utilizar lógica en nuestro código CSS, como variables, anidamiento, herencia, mixins, funciones y operaciones matemáticas. Los preprocesadores de CSS facilitan la automatización de tareas repetitivas, reducen el número de errores y la saturación del código, crean fragmentos de código reutilizables y garantizan la compatibilidad con versiones anteriores.

**Cada preprocesador de CSS tiene su propio ecosistema** (herramientas, frameworks, bibliotecas), **y su propia sintaxis** que compilan en CSS normal para que los navegadores puedan renderizarlo en el lado del cliente.

### 5.1 Preprocesadores CSS vs PostCSS

Los preprocesadores de CSS a veces se equiparan con PostCSS, pero **no son lo mismo**.

**PostCSS** es una popular biblioteca JavaScript que nos permite **automatizar tareas** relacionadas con CSS **una vez que el código CSS está listo para ser publicado**.

La confusión viene del hecho de que algunos preprocesadores CSS, por ejemplo Sass, Less y Stylus, tienen plugins PostCSS dedicados.

Actualmente, los tres preprocesadores de CSS más populares son **Sass** (Syntactically Awesome Style Sheets), **LESS** (Learner Style Sheets) y **Stylus**. Existen más como puede ser Stylable, Clay o CSS Crush.

Conoce más acerca de los **preprocesadores Sass y Less** en los siguientes enlaces:

- [Aprende Sass](#)
- [Aprende Less](#)



## 6. Anexo I - Centrar horizontal y verticalmente en CSS un elemento

Existen **diversas formas de centrar horizontal y verticalmente un elemento en un contenedor mediante CSS**. Cada método es adecuado para una situación concreta y, por ello, es necesario saber cómo funcionan todas las propiedades para reconocer fácilmente **qué técnica utilizar en cada caso**.

En este apartado se expondrán diferentes métodos, sobre qué elementos funcionan, y ejemplos de cómo centrar horizontal y verticalmente elementos CSS.

### 6.1 Centrar vertical y horizontalmente elementos con Flexbox

Este es el **método más útil** porque funciona para **todos los elementos** que se posicen dentro de un contenedor. Por tanto, es **válido para imágenes, textos, vídeos, etc.**

Tal y como hemos visto, para utilizar las propiedades de **flexbox** es necesario crear un contenedor y declararle la propiedad **display: flex**. Para centrar horizontal y verticalmente el contenido se pueden utilizar las propiedades “**justify-content: center**” y “**align-items: center**”.

```
div {  
    display:flex;  
    justify-content: center;  
    align-items: center;  
}
```

A continuación, se muestra un **ejemplo** de cómo centrar texto y una imagen con Flexbox.

#### Código HTML:

```
<div>  
    <!-- Cualquier elemento incluido dentro del div se centrará  
        vertical y horizontalmente con flexbox -->  
    <h1>Centrado con flexbox</h1>  
</div><br>  
<div>  
      
</div>
```



## Código CSS:

```
div{  
    height: 200px;  
    background-color: #EEE;  
    display:flex;  
    justify-content: center;  
    align-items: center;  
}
```



## 6.2 Centrar vertical y horizontalmente un texto con line-height y text-align

Mediante las propiedades `line-height` y `text-align` podemos alinear únicamente **textos**. Así que este método sólo va a ser válido para el centrado de párrafos, encabezados, etc.

```
div {  
    text-align: center;  
}  
div h1 {  
    line-height: 200px;  
}
```

A continuación, se muestra un **ejemplo** sencillo.

## Código HTML:



```
<div>
  <h1>Centrar texto</h1>
</div>
```

### Código CSS:

```
div {
  height: 200px;
  background-color: #EEE;
  text-align: center;
}
div h1 {
  line-height: 200px;
}
```

Centrar texto

## 6.3 Centrar vertical y horizontalmente una imagen o contenedor con propiedad transform y posición absoluta

Mediante este método se **puede alinear tanto vertical como horizontalmente imágenes y contenedores. No es válido para la alineación de textos** ya que no tiene en cuenta el tamaño de la letra. El centrado de elementos se realiza mediante posición absoluta y la propiedad transform.

```
div {
  position: relative;
}
div img{
  position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%); }
```



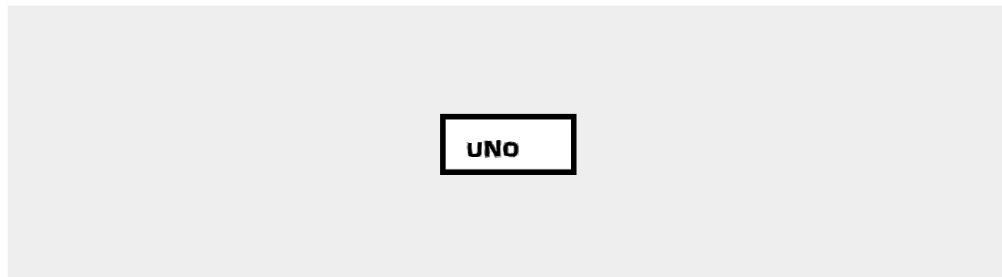
A continuación, se muestra un **ejemplo** sencillo.

### Código HTML:

```
<div>
  
</div>
```

### Código CSS:

```
div {
  height: 300px;
  background-color: #EEE;
  position: relative;
}
div img{
  position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);
  -webkit-transform: translate(-50%, -50%);
}
```



## 6.4 Centrar un elemento vertical y horizontalmente utilizando posición absoluta y margen negativo

Mediante este método se puede **alinear texto dentro de un contenedor**. Para ello realizamos un centrado absoluto con posición absoluta y margen negativo:



```
.container {  
    position: relative;  
}  
  
.centered-element {  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    margin-top: -50px; /* Ajusta el valor según la mitad de la altura  
                        del elemento */  
    margin-left: -50px; /* Ajusta el valor según la mitad del ancho del  
                        elemento */  
}
```

A continuación, se muestra un **ejemplo** sencillo.

#### Código HTML:

```
<div class="container">  
    <div class="centered-element">Contenido centrado</div>  
</div>
```

#### Código CSS:

```
.container {  
    position: relative;  
    width: 400px;  
    height: 300px;  
    background-color: lightgray;  
}  
  
.centered-element {  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    transform: translate(-50%, -50%);  
    background-color: white;  
    padding: 20px;  
}
```



## 6.5 Centrar un elemento con Grid

Para centrar un elemento con grid se debe utilizar `display: grid` y `place-items: center` en el contenedor. De esta forma, el elemento se centrará tanto vertical como horizontalmente dentro del contenedor.

```
.container {  
    display: grid;  
    place-items: center; }
```

En el siguiente ejemplo, el contenedor tiene una altura fija de 300px y un borde para que puedas visualizarlo. El elemento que deseas centrar está dentro del contenedor y tiene un fondo gris claro y un relleno de 20px.

### Código HTML:

```
<div class="container">  
    <div class="centered-element">Elemento centrado</div>  
</div>
```

### Código CSS:

```
.container {  
    display: grid;  
    place-items: center;  
    height: 300px;  
    border: 1px solid #ccc; }  
  
.centered-element {  
    background-color: #f1f1f1;  
    padding: 20px; }
```



## 6.6 Alinear un texto verticalmente con una imagen con vertical-align

Con esta técnica podemos **alinear un texto a una imagen de forma vertical**. Esta propiedad admite los valores *top*, *middle* y *bottom*, entre otros valores.

```
img{  
    vertical-align: middle;  
}
```

A continuación, se muestra un **ejemplo** sencillo.

### Código HTML:

```
 Alinear texto
```

### Código CSS:

```
img{  
    vertical-align: middle;  
}
```



Alinear texto



## 6.7 Alinear verticalmente un texto en un div con vertical-align

Con este método podemos **alinear un texto** en un div con la propiedad **vertical-align**. Hay que tener en cuenta que esta propiedad no se comporta de la misma manera con todos los elementos. En este caso, haremos uso de las reglas de estilo “**display: inline-table**” y “**display: table-cell**”.

```
div{  
    height: 200px; width: 200px;  
    background-color: #EEE;  
    display: inline-table;  
}  
p{  
    display: table-cell;  
    vertical-align: middle;  
}
```

A continuación, se muestra un **ejemplo** sencillo de este método.

### Código HTML:

```
<div class="a">  
    <p>Texto dentro de un div</p>  
</div>
```

### Código CSS:

```
div{  
    height: 200px; width: 200px;  
    background-color: #EEE;  
    display: inline-table;  
    width: 100%;  
}  
p{  
    text-align: center;  
    display: table-cell;  
    vertical-align: middle;  
}
```



## 6.8 Alinear horizontalmente una imagen con text-align

Las imágenes se pueden alinear a la izquierda, derecha o centro insertándolas dentro de un div e indicando la propiedad `text-align` correspondiente.

```
.a{ text-align: left; }
.b{ text-align: center; }
.c{ text-align: right; }
```

A continuación, se muestra un **ejemplo** sencillo de este método.

### Código HTML:

```
<div class="a">
</div>

<div class="b">
</div>

<div class="c">
</div>
```



### Código CSS:

```
.a{ text-align: left; }  
.b{ text-align: center; }  
.c{ text-align: right; }
```



### 6.9 Alinear verticalmente un checkbox con un label

Cuando el tamaño del texto no coincide con el tamaño del *checkbox*, estos dos elementos no quedan alineados verticalmente. Para conseguir esta alineación utilizaremos “vertical-align:middle” y **definiremos el tamaño del texto a las dos etiquetas**, tal y como se muestra a continuación.

```
label, input{  
    font-size: 28px;  
    vertical-align: middle;  
}
```

A continuación, se muestra un **ejemplo** sencillo de este método.

### Código HTML:

```
<label for="texto">TEXTO</label><input type="checkbox" id="texto"/>
```

### Código CSS:

```
label, input{  
    font-size: 28px;  
    vertical-align: middle; }
```

TEXTO



## 7. Anexo II - SaSS

**SaSS** es un preprocesador CSS que traduce un código de hojas de estilo **NO estándar** a un código CSS estándar legible por la mayoría de los navegadores.

SaSS es el preprocesador más utilizado por los desarrolladores web de frontend hoy en día.

La principal utilidad de SaSS es la de hacer **más simple la escritura del código CSS**, además de brindar diversas utilidades que a día de hoy CSS no puede ofrecer.

Entre otras posibilidades, SaSS permite:

- Usar **variables**.
- Realizar **cálculos** con operadores.
- **Anidar** selectores para escribir menos código.
- Usar **funciones** para no tener que repetir el código similar.
- **Separar** el **código** del proyecto en varios ficheros para un mantenimiento más sencillo.

**Tutoriales** de SaSS:

- <https://www.w3schools.com/sass/>
- <https://sass-lang.com/documentation/>
- <https://sass-lang.com/guide/>

### 7.1 Instalación de SaSS

La instalación de SaSS se puede hacer mediante aplicaciones (son de pago) o mediante línea de comandos: <https://sass-lang.com/install/>

En este apartado vamos a centrarnos en la instalación por línea de comandos. Simplemente lo que hay que hacer es descargar el paquete correspondiente al sistema operativo que utilizamos desde [GitHub](#) y añadirla a nuestro [PATH](#).

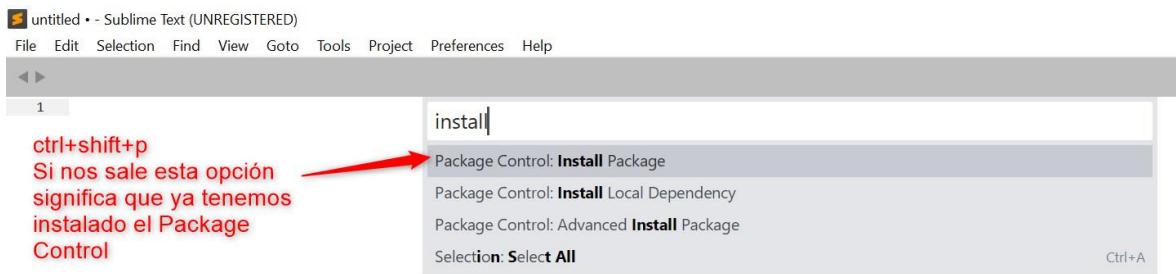
- Si usamos Visual Studio, debemos instalar un plugin que nos permite trabajar con archivos \*.sass y \*.scss y procesarlos en tiempo real.

<https://marketplace.visualstudio.com/items?itemName=ritwickdey.live-sass>

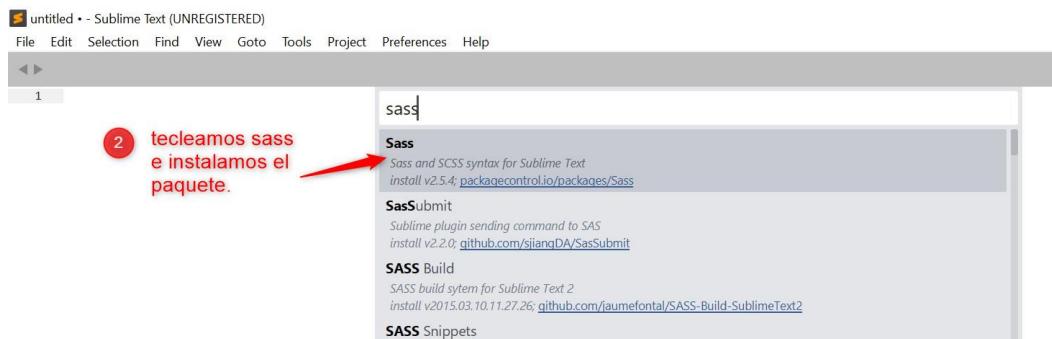
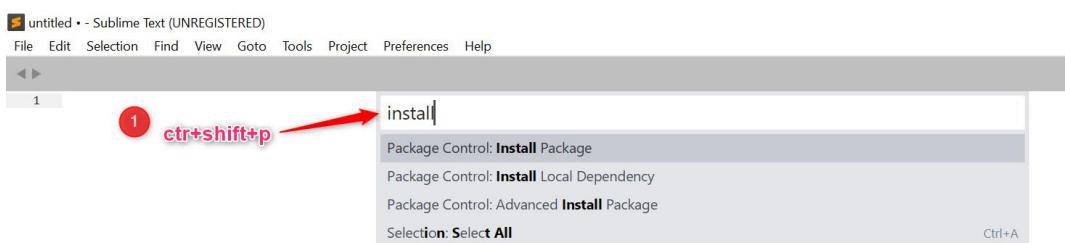


- Si usamos SublimeText debemos instalar lo siguiente:

### 1) Package Control (<https://packagecontrol.io/installation>)



### 2) Paquete de SaSS del Package Control (<https://packagecontrol.io/packages/Sass>)



## 7.2 Formatos

SASS dispone de dos formatos diferentes para la sintaxis, lo que hace se traduce en dos extensiones de fichero diferentes: **.sass** y **.scss**. De ambas extensiones hay que elegir una.

El primero en salir fue **.sass** y se caracteriza, al igual que Stylus y [coffeescript](#) para el lenguaje de programación Javascript, en no hacer uso de llaves ni punto y coma final, en busca de la simplicidad y eliminación de ruido.



```
body  
background: #000  
font-size: 62.5%
```

Los **.scss** salieron con la versión 3 de preprocesador y permiten utilizar llaves e incorporar código de CSS clásico.

```
body {  
background: #000;  
font-size: 62.5%;  
}
```

Tanto la sintaxis de **Sass** como la de **.scss** no puede ser interpretada directamente por los navegadores, por lo se debe compilar para traducir nuestro archivo SASS en un clásico fichero CSS.

Por ejemplo, si tenemos el siguiente código CSS:

```
body {  
    background: #f2f2f2;  
}  
  
.articulo {  
    background: #fff; }  
  
.articulo .titulo {  
    color: #000; }  
  
.articulo .fecha {  
    // estilos }  
  
.articulo p a {  
    text-decoration: none; }
```

Y lo queremos escribir en Sass, este tendría el siguiente aspecto:

#### ✍ .sass:

```
// Esta syntax es mas antigua pero en vez de {} utiliza tabulaciones.  
  
body  
    background: #f2f2f2  
  
.articulo  
    background: #fff  
    .titulo  
        color: #000  
    .fecha  
        // estilos  
p  
    a  
        text-decoration: none
```

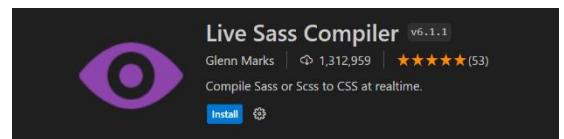


### +.SCSS:

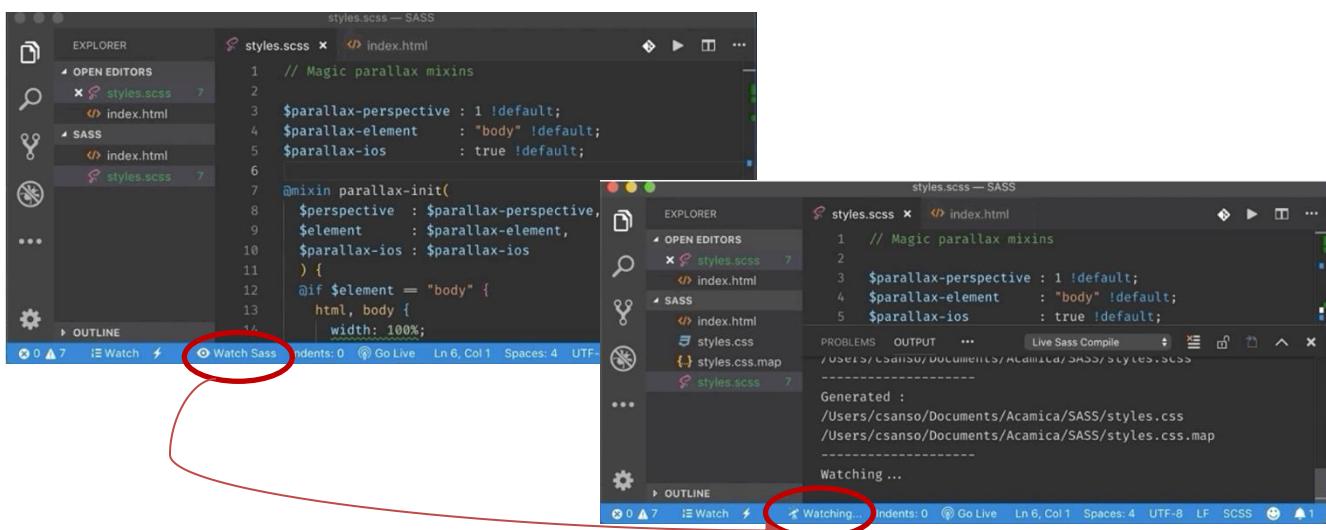
```
// Esta syntax es mas nueva pero utiliza { } y ;  
  
body {  
    background: #f2f2f2;  
}  
  
.articulo {  
    background: #fff;  
.titulo {  
    color: #000;  
}  
.fecha {  
    // estilos  
}  
p {  
    a {  
        text-decoration: none;  
    }  
}
```

## 7.3 Cómo compilar nuestro código

La estrategia más simple para compilar nuestro código sass o scss, es descargar la extensión '**Live Sass Compiler**' de Visual Studio Code que con un simple botón nos va a generar nuestro código CSS.



Una vez descargado y recargado VSCode, se puede observar que en la parte inferior aparece un nuevo botón que dice '**Watch Sass**'. Haciendo click en él, podemos ver como automáticamente nos creará dos archivos nuevos. Uno con la extensión de **.css**, que es el que vamos a usar para dar estilos, y otro es **.css.map**.





El **output** en Visual Studio Code nos mostrará siempre el status de la compilación de nuestro documento.

The screenshot shows the Visual Studio Code interface with the 'OUTPUT' tab selected. The output window displays the following text:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Compiling Sass/Scss Files:
/Users/spokelopez/Documents/Frontend/SASS/Fodesite/src/css/landing.scss
-----
Generated :
/Users/spokelopez/Documents/Frontend/SASS/Fodesite/src/css/landing.css
-----
Watching...
-----
```

At the bottom right of the output window, there is a small icon with the text "Watching..." and a circular progress bar.

Existen más formas para compilar nuestro código sass. Algunos de ellos son:

- 1) Utilizando **consola**. Es la forma más liviana, simple, escalable y controlada (<https://sass-lang.com/guide/#preprocessing>).
- 2) **Programas de terceros** como Prepros (de pago), Koala y Scout-app.
  - <http://koala-app.com/>
  - <https://scout-app.io/>
  - <https://prepros.io/help/sass>
- 3) **Compiladores online** como [SASSmeister](#)



## 8. Anexo III – Chuleta Flex y Grid

PROPIEDADES DEL FLEX CONTAINER	
<b>display</b>	Define un contenedor flex. Puede tomar los valores <i>flex</i> o <i>inline-flex</i> .
<b>flex-direction</b>	Define la dirección en la que se colocan los contenedores hijo.
<b>flex-wrap</b>	Indica qué elementos hijo se deben trasladar cuando no hay suficiente espacio en el contenedor.
<b>flex-flow</b>	Mezcla las propiedades <i>flex-direction</i> y <i>flex-wrap</i> .
<b>justify-content</b>	Define la justificación horizontal de los elementos hijo.
<b>align-items</b>	Alinea los ítems flex o contenedores hijo sobre el eje cruzado o transversal (vertical).
<b>align-content</b>	Alinea las filas de un contenedor flex cuando hay espacio extra en el eje transversal (o vertical). Es similar a <i>align-items</i> pero <b>no tiene efecto en cajas o ítems flex de una sola línea</b> .
<b>gap</b>	Controla el espacio que hay entre los ítems flex.

PROPIEDADES DEL GRID CONTAINER	
<b>display</b>	Define el elemento contenedor (grid o inline-grid)
<b>grid-template-columns</b> <b>grid-template-rows</b>	Define el espaciado o tamaño de las columnas/filas
<b>grid-template-areas</b>	Indica el nombre y posición concreta de cada área de la cuadrícula. Define una plantilla de la rejilla haciendo referencia a los nombres de las áreas que se especifican con la propiedad <i>grid-area</i> .
<b>grid-template</b>	Forma abreviada de definir las filas, columnas y áreas de la cuadrícula. Define las tres propiedades anteriores en una línea de código.
<b>row-gap</b> <b>column-gap</b>	Define el espaciado entre las filas/columnas
<b>gap</b>	Define el tamaño del espacio entre las filas y entre las columnas. Define las propiedades <i>row-gap</i> y <i>column-gap</i> en una línea de código.
<b>justify-items</b>	Alinea los <b>elementos</b> de la cuadrícula o rejilla a lo largo del <b>eje horizontal o en línea</b> .
<b>align-items</b>	Alinea los <b>elementos</b> de la cuadrícula o rejilla a lo largo del <b>eje vertical o en bloque</b> .
<b>place-items</b>	Establece las propiedades <i>align-items</i> y <i>justify-items</i> en una única declaración o línea de código.
<b>justify-content</b>	Alinea la <b>cuadrícula</b> a lo largo del <b>eje</b> en línea u <b>horizontal</b> (fila o eje x).
<b>align-content</b>	Alinea la <b>cuadrícula</b> a lo largo del <b>eje de bloque o vertical</b> (columna o eje y).
<b>place-content</b>	Establece las propiedades <i>align-content</i> y <i>justify-content</i> en una única línea de código.
<b>grid-auto-columns</b> <b>grid-auto-rows</b>	Establece el <b>tamaño de las columnas/filas implícitas</b> y cualquier otra columna/fila que no haya sido explícitamente dimensionada.
<b>grid-auto-flow</b>	Controla cómo se insertan los elementos autocolocados en la cuadrícula
<b>grid</b>	Establece todas las siguientes propiedades en una única declaración o sentencia de código: <i>grid-template-rows</i> , <i>grid-template-columns</i> , <i>grid-template-areas</i> , <i>grid-auto-rows</i> , <i>grid-auto-columns</i> , y <i>grid-auto-flow</i> .



## PROPIEDADES DE LOS FLEX ITEMS.

<b>order</b>	Especifica el orden utilizado para disponer los elementos hijo en su contenedor padre.
<b>flex-grow</b>	Define la capacidad de un item flex para crecer si es necesario.
<b>flex-shrink</b>	Permite definir cuánto se puede hacer más pequeño un ítem flex en proporción a los demás ítems o elementos hijo.
<b>flex-basis</b>	Especifica el tamaño inicial o por defecto que va a tener un elemento hijo.
<b>flex</b>	Concentra las propiedades de <b>flex-grow</b> , <b>flex-shrink</b> y <b>flex-basis</b> en una sola línea de código.
<b>align-self</b>	Permite anular o cambiar la alineación por defecto, o la especificada por <b>align-items</b> , de forma individual para cada ítem flex.

## PROPIEDADES DE LOS GRID ITEMS.

<b>grid-column-start</b>	Es la línea de columna donde comienza el elemento.
<b>grid-column-end</b>	Es la línea de columna donde termina el elemento.
<b>grid-column</b>	Esta propiedad incluye a las dos anteriores (grid-column-start y grid-column-end) en una línea de código. Especifica en qué columna colocar un elemento y cuántas columnas abarcará.
<b>grid-row-start</b>	Es la línea de fila donde comienza el elemento.
<b>grid-row-end</b>	Es la línea de fila donde termina el elemento.
<b>grid-row</b>	Esta propiedad incluye a las dos anteriores (grid-row-start y grid-row-end) en una línea de código. Especifica en qué fila colocar un elemento y cuántas filas abarcará.
<b>grid-area</b>	Especifica el tamaño y la ubicación de un elemento en un diseño de rejilla. Es una <u>propiedad abreviada</u> para las propiedades: <b>grid-row-start</b> , <b>grid-column-start</b> , <b>grid-row-end</b> y <b>grid-column-end</b>
<b>justify-self</b>	Alinea <b>un elemento</b> o ítem de la cuadrícula dentro de una celda a lo largo del eje <b>en línea u horizontal</b> (fila o eje x).
<b>align-self</b>	Alinea <b>un elemento</b> o ítem de la cuadrícula dentro de una celda a lo largo del eje <b>en bloque o vertical</b> (columna o eje y).
<b>place-self</b>	Establece las propiedades <b>align-self</b> y <b>justify-self</b> en una única línea de código