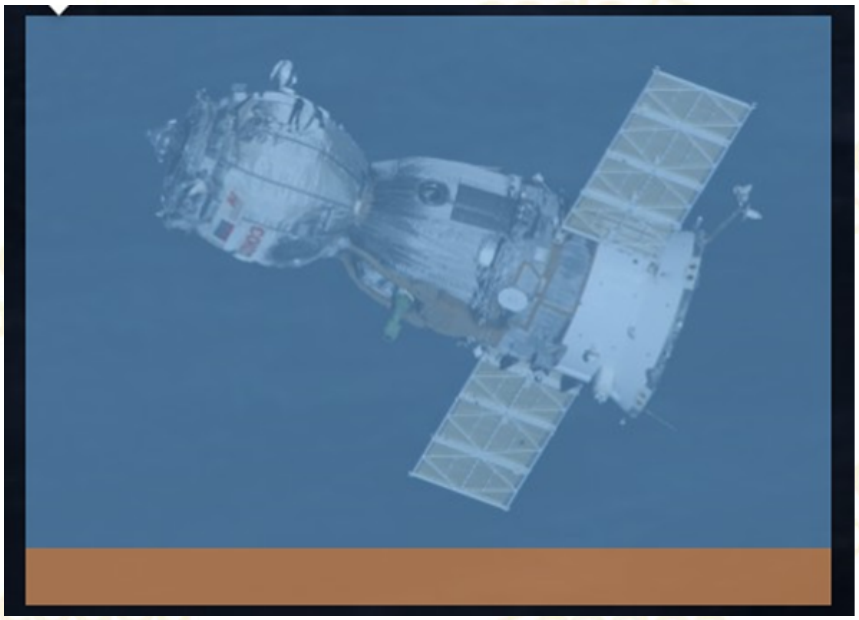
Contenidos a Trabajar

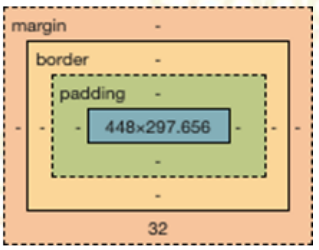
- **Box Modeling**
  - Displays
  - ¿Qué compone el Modelo de Cajas?
- **Normalización**
- **Flexbox**
  - Ejes
  - Propiedades
  - Alineaciones
- **Grid Layout**
  - Propiedades
  - Alineaciones

Box Modeling

Para entender el modelo de caja, debemos comenzar por la premisa de que en la web o mejor dicho, dentro de un navegador, todo es una caja. Al principio este concepto puede parecer un poco raro, pero si analizamos cada elemento de nuestra web y cómo interactúan las etiquetas HTML entre sí, entonces esta idea toma mucho más sentido.

Por ejemplo, si seleccionamos una imagen con transparencia o aplicamos un border-radius a un elemento e inspeccionamos con el inspector de código, veremos que dicho elemento no tendrá los límites en su contorno si no que será cuadrado o rectangular.



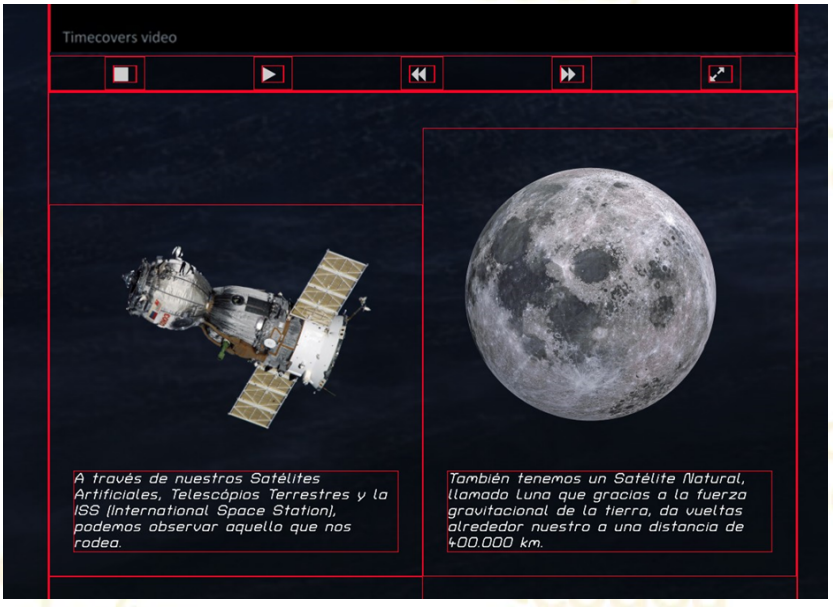


Agencia de Aprendizaje a lo largo de la vida

```
▼<div class="parrafo img1">
  <img src="media/img/satelite resized.png" alt> == $0
  ▶<p>…</p>
 </div>
```

Como vemos en las imágenes anteriores, si bien la foto es transparente, al seleccionar la etiqueta img, esta nos marca su forma como un rectángulo y vemos que el inspector de código nos muestra las propiedades de la caja (margen, relleno, borde y contenido) de la misma manera.

Otro ejemplo es colocar un borde de color a todos los elementos de una web a través del selector universal * y veremos todas las cajas que componen cada elemento de ese sitio, revelando también contenedores padre y sus estructuras.



Esta práctica nos ayuda a entender el modelo de caja y cómo se arman los layouts de los sitios webs a través de su anidado de etiquetas padres e hijas.

Displays

Como vimos en HTML, las etiquetas poseen un comportamiento nativo que puede ser de dos tipos: **en bloque** o **en línea**.

Ahora que sabemos lo que es el "Box Modeling", podemos comprender que estos comportamientos son formas de mostrar (display) una caja y que cada una tiene un comportamiento en particular.

Cajas con display ="block";

Las cajas de bloque, por defecto, ocupan todo el espacio a lo ancho del contenedor y aceptan modificaciones en su morfología a través de las propiedades **width**, **height**, **margin** y **padding**.

Cajas con display="inline";

Las cajas en línea, por defecto, toman el tamaño de contenido y no aceptan modificaciones en su morfología salvo por el **margin** y **padding** pero únicamente a los costados de su contenido.

```
<p>Párrafo de texto con <strong>algunas
palabras</strong> resaltadas</p>

<p>Otro párrafo</p>
```

Párrafo de texto con **algunas palabras** resaltadas

Otro párrafo

Agencia de Aprendizaje a lo largo de la vida

Si bien estas son las opciones pueden adoptar las etiquetas por defecto, existen otros valores posibles para la propiedad display, de los cuales nos concentraremos por el momento en: **inline-block** y **none**.

Cajas con display="inline-block";

Con este valor, tomamos lo mejor de los anteriores y obtenemos un display que ocupa solo el tamaño de su contenido pero donde podemos modificar la morfología del elemento a través de las propiedades **width**, **height**, **margin** y **padding**.

Cajas con display="none";

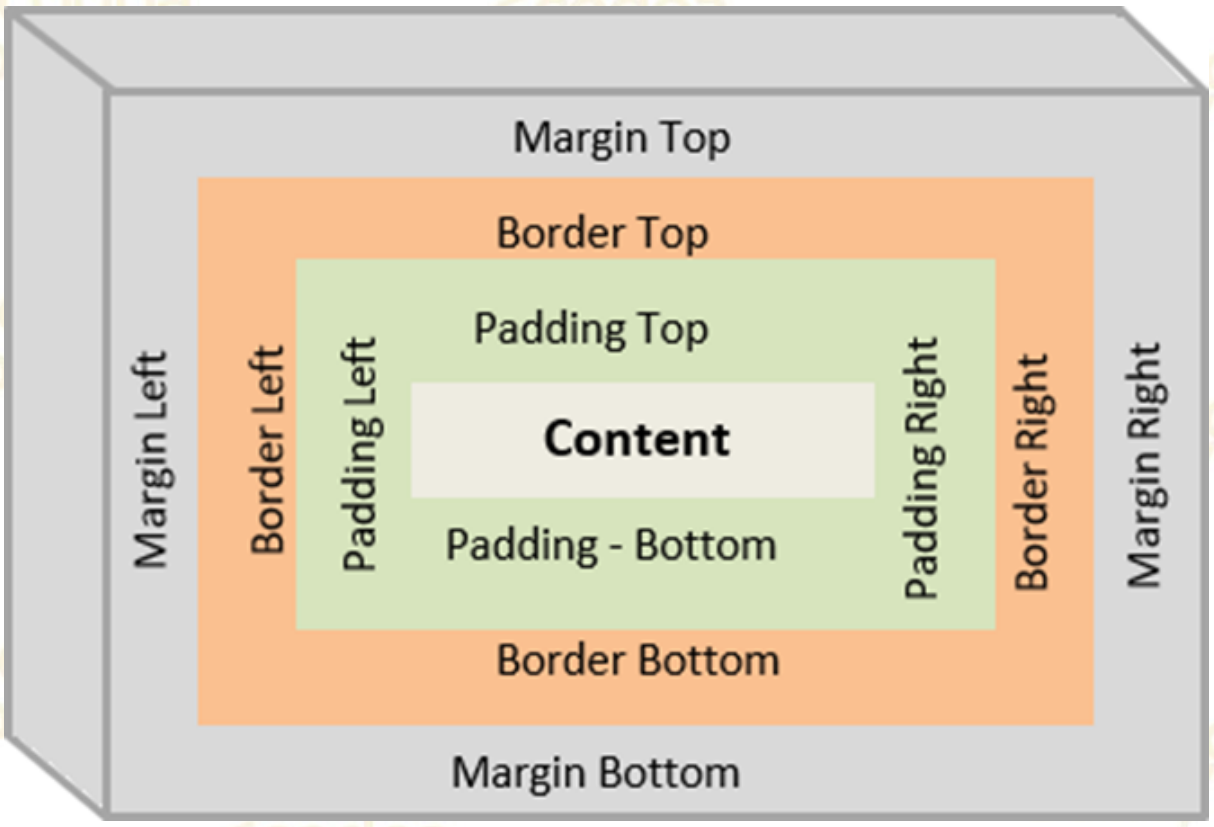
Estas cajas no serán mostradas en nuestro sitio ni ocuparán espacio en el lugar donde deberían ir ubicadas.

Resumen de propiedades:

	WIDTH	HEIGHT	MARGIN	PADDING	VERTICAL-ALIGN	LINE-HEIGHT	TEXT-ALIGN
BLOCK	SI	SI	SI	SI	NO	SI	SI
INLINE	NO	NO	LEFT/RIGHT	LEFT/RIGHT	SI	SI	N/A
INLINE-BLOCK	SI	SI	SI	SI	SI	SI	SI

¿Qué compone al Modelo de Cajas?

Cada elemento HTML es una caja que posee un **ancho** (width) y **alto** (height), **cuatro** lados y se compone de **cuatro** áreas: **content**, **padding**, **border** y **margin**.



Conozcamos cada una de ellas:

Contenido (Content)

El contenido de la caja, donde aparecen texto, imágenes, etc.

El área del **content** (como su nombre lo dice) es el "contenido" principal a mostrar, es decir, un texto, una imagen, un video, etc. El contenido siempre es lo que queremos mostrarle al usuario. Esta área en muchas ocasiones tiene un color o imagen de fondo para hacerla más atractiva.

Propiedades que se pueden aplicar al content:

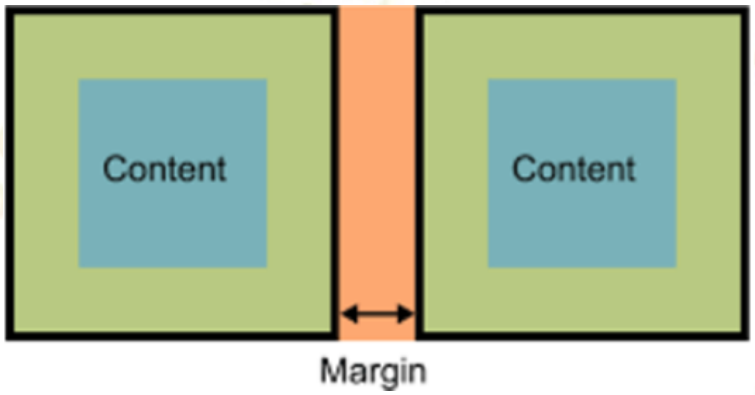
- **width**: modifica el ancho de la caja y su contenido.
- **max-width**: establece un valor de ancho máximo.
- **min-width**: establece un valor de ancho mínimo.
- **height**: modifica el alto de la caja y su contenido.
- **max-height**: establece un valor de alto máximo.
- **min-height**: establece un valor de alto mínimo.

Márgenes (Margin)

Es la separación entre una caja y las cajas adyacentes.

El margen es la última área del Box Model y se puede ver como una separación invisible o transparente que ayuda a separar un elemento de otro. Cuando definimos un color o imagen de fondo, este no se propaga a esta sección.

Los márgenes siempre quedan fuera del modelo de caja, por lo que pueden utilizarse para crear espacio entre los elementos y sus propiedades aceptan valores negativos en caso de ser necesario.

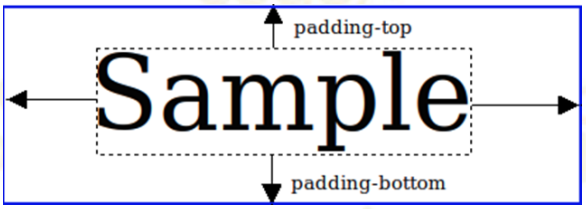


Formas de declarar o modificar el margin:

```
/* top right bottom left */
margin: 10px 20px 10px 20px;
/* top right|left bottom */
margin: 10px 20px 10px;
/* top|bottom right|left */
margin: 10px 20px;
/* top|right|bottom|left */
margin: 10px;
```

Rellenos (padding)

Es el área de relleno del contenido y también es transparente al igual que el margin.



El padding es una separación o espacio interior que existe entre el contenido y el borde de la caja, el cual se utiliza para dar una apariencia estética más atractiva y que el contenido no esté pegado al borde.

Cabe mencionar que el padding sigue siendo parte de los límites de la caja, por ende se tendrá en cuenta para calcular el ancho y alto de la misma junto con el contenido.

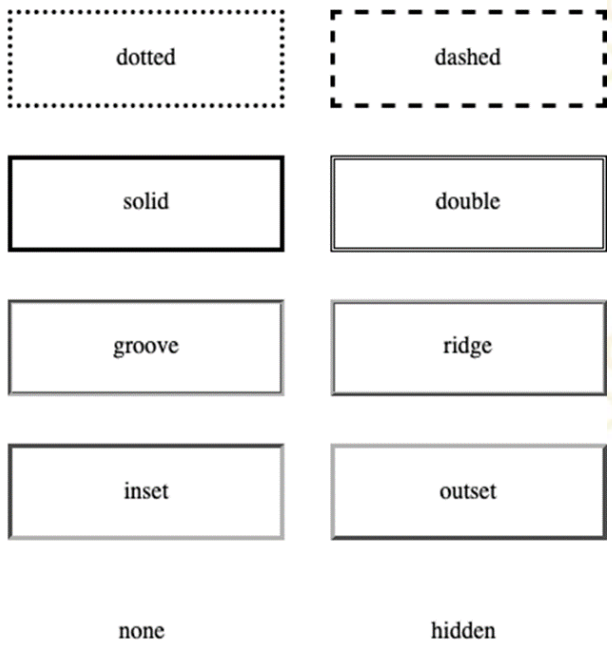
Es importante mencionar que **NO** se permiten valores negativos.

Formas de declarar o modificar el padding:

```
/* top right bottom left */
padding: 10px 20px 10px 20px;
/* top right|left bottom */
padding: 10px 20px 10px;
/* top|bottom right|left */
padding: 10px 20px;
/* top|right|bottom|left */
padding: 10px;
```

Bordes (border)

El borde es la línea que rodea la caja, es como la frontera que rodea al elemento y se utiliza para darle una apariencia estética a la caja ya que permite dibujar una línea con colores y estilos diferentes.



Podemos manejar los bordes con las siguientes propiedades:

border-style: el estilo del borde, los valores posibles son los de la imagen superior.

border-width: define el ancho de la línea del borde.

border-color: nos permite elegir el color de ese borde.

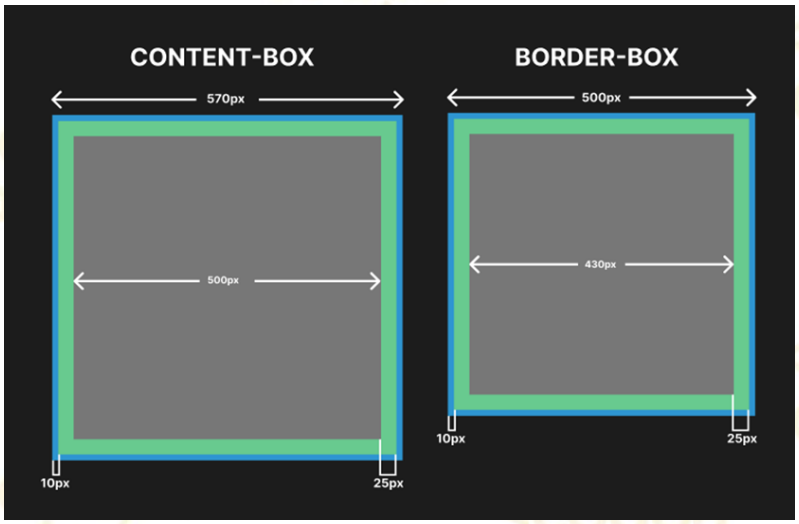
También existe un shorthand property para poder definir estos 3 estilos en una misma propiedad CSS:

```
/* ancho estilo color*/
border: 2px solid #f32872;
```

Box-sizing

La propiedad box-sizing establece cómo se calcula el ancho y alto total de un elemento o caja en HTML/CSS.

Tiene dos valores posibles: **content-box** y **border-box** y la diferencia principal entre ellas radica en que una calcula el ancho de la caja.



box-sizing: content-box;

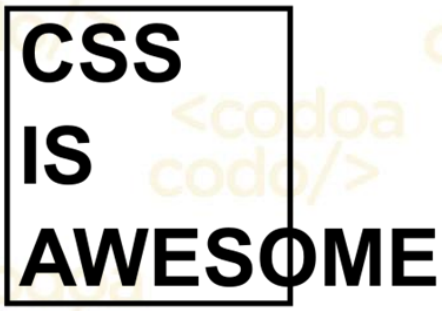
Este valor de propiedad calcula el tamaño total de la caja (con padding y border) SIN modificar el ancho y alto asignado al elemento a través de las propiedades width y height.

box-sizing: border-box;

En cambio, esta propiedad calcula el tamaño total de la caja **SIN** exceder del ancho y alto asignado a través de las propiedades width y height, es decir incluyendo en su tamaño total padding y bordes.

Overflow (desbordamiento)

Esta propiedad nos permite determinar cómo se verá todo el contenido que se desborde de una caja. Normalmente, estas situaciones se dan en casos donde nuestra caja tiene un ancho o alto explícito y el contenido interno es más grande que estos.



Las valores de la propiedad **overflow** para manejar estos casos son:

- **auto**: se colocan barras de desplazamiento sólo cuando sean necesarias.
  hidden: oculta el contenido que sobresale.
- **visible**: muestra el contenido que sobresale (comportamiento por defecto)
- **scroll**: se colocan barras de desplazamiento (horizontales y verticales).

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Magnam fuga odio sequi delectus rem? Tempora accusantium pariatur quaerat repudiandae explicabo laudantium itaque sapiente delectus labore eaque dicta consectetur dignissimos corporis!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Itaque qui consectetur quaerat quo hic quidem illum ipsum dolore modi. Numquam repellat cum iure quisquam veniam praesentium rerum nobis

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Odit labore laudantium nisi aliquid nulla qui quisquam recusandae quis corporis expedita ipsum debitis mollitia ducimus ex enim deleniti

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Animi minus voluptate mollitia optio maiores harum expedita numquam accusantium ut eligendi rem cupiditate illum aspernatur doloremque

Normalización

Para renderizar sin problema cualquier página, todos los navegadores tienen una hoja de estilos por defecto que aplican cuando un elemento no tiene definido un estilo (sea porque el maquetador no lo ha definido o porque no hay estilos definidos en la página).

Si bien estos estilos por defecto, cada vez son más “parecidos” entre los distintos navegadores, aún existe la posibilidad de que cada uno haga su propia interpretación y obtengamos como resultado diferentes estilos según el navegador que esté usando el usuario.

Para poder evitar esto se recomienda normalizar los estilos CSS más comunes como los márgenes, rellenos, viñetas de listas, decoración en los enlaces y comportamientos en los tamaños de las cajas.

Si bien existen hojas de estilos creadas por terceros para este fin como “Normalize.css”, nosotros aprenderemos a hacerlo manualmente y de forma sencilla para los estilos más conflictivos.

Con esta pequeña definición de estilos tendremos una normalización estándar para poder comenzar a trabajar nuestros proyectos sin mayores inconvenientes.

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

html {
    /* Tamaño de fuente relativo al navegador*/
    font-size: 62.5%;
    /* Familia Elegida por defecto */
    font-family: 'Barlow', sans-serif;
    /* Color de texto base */
    color: #2b2b2b;
}

ul, ol {
    list-style-type: none;
}

a {
    text-decoration: none;
}
```

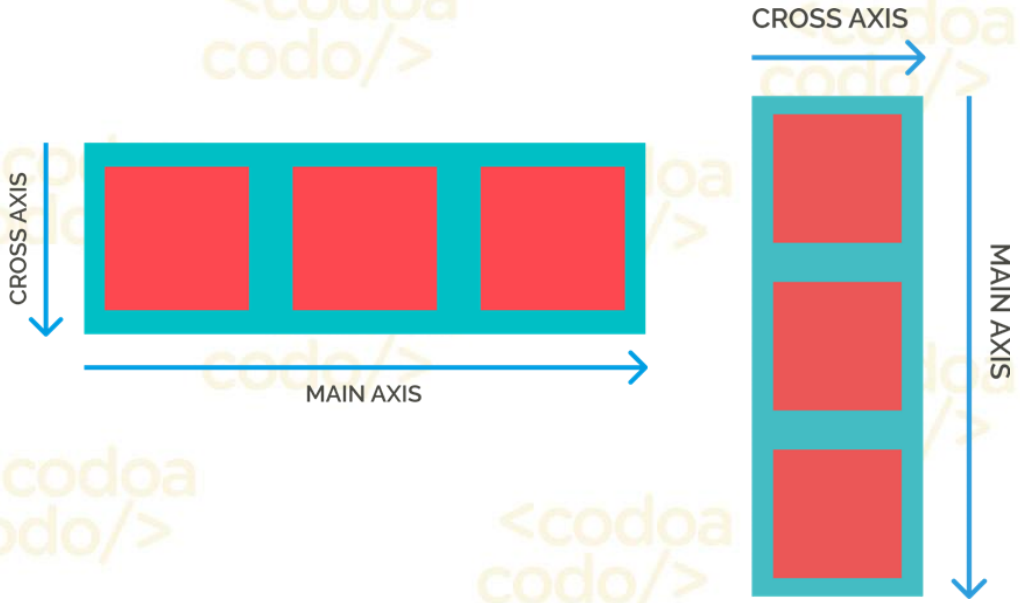
Flexbox

El concepto de Flexbox es un conjunto de propiedades CSS que nos permiten posicionar nuestros elementos HTML con mayor facilidad bajo la premisa de establecer una dirección vertical u horizontal para nuestros elementos hijos desde un elemento padre. Es por eso que algunas propiedades deben declararse en el contenedor (el elemento principal, que llamamos de flex container), mientras que otras deben declararse en los elementos secundarios (el flex ítems).

Esta herramienta se complementa con el concepto de Box Modeling, adoptando soluciones específicas para que nuestras cajas fluyan a través de los distintos tamaños de pantalla de una forma más dinámica y flexible, lo que vuelve a flexbox una excelente alternativa para el diseño responsive. Es por eso que, si bien el diseño web "estándar" del modelo de caja se basa en las direcciones block e inline, el diseño Flex se basa en direcciones de "flex flow" que proponen un flujo en un solo sentido, vertical u horizontal.

Ejes

Flexbox propone distribuir nuestros elementos hijos a través de un eje principal o Main Axis y un eje secundario o Cross Axis.



Bajo esta distribución nuestros ítems se distribuirán en el sentido del eje principal o main axis:

- **Main Axis**: el eje principal de un flex container es el eje primario y a lo largo de él son insertados los flex ítems. El eje principal no es necesariamente horizontal y dependerá de la propiedad flex-direction.
- **Cross Axis**: El eje perpendicular al eje principal se llama eje transversal. Su dirección depende de la dirección del eje principal.

Propiedades

Como se mencionó anteriormente, Flexbox es un conjunto de propiedades CSS donde es muy importante saber qué propiedades se declaran en el elemento principal (contenedor padre o flex container) y cuáles se declararán en los elementos secundarios (elementos hijos o flex items).

A continuación, se muestran las propiedades que deben declararse en el selector css del flex container para posicionar los elementos hijos:

display - con valor flex indica que sus flex items se comportarán de forma flexible.

display: flex | inline-flex;

flex-direction - define la dirección de los elementos flexibles de forma vertical u horizontal.

flex-direction: row | row-reverse | column | column-reverse;

flex-wrap - indica si nuestros elementos flexibles pueden ser multilínea o no.

flex-wrap: nowrap | wrap | wrap-reverse;

flex-flow - shorthand propertie de flex-direction + flex-fow.

flex-flow: row nowrap | row wrap | column nowrap | column wrap;

gap - define la distancia o separación entre los flex-items.

gap: 2rem; /* default: 0*/

A continuación, se muestran las propiedades que deben declararse en el selector css del flex item para posicionarse como hijo del flex container:

order - determina el orden en que aparecerán los elementos.

order: 1;

flex-grow - determina cuánto puede crecer un elemento respecto al espacio sobrante de su flex-container.

flex-grow: 4; /* default 0 */

flex-shrink - determina cuánto puede comprimirse un elemento respecto al espacio faltante de su flex-container.

flex-shrink: 3; /* default 1 */

flex-basis - define un tamaño para ese elemento antes de asignar el espacio sobrante al flex-container. Funciona como un width o un height dependiendo el main axis.

flex-basis: number | auto; /* default auto */

Alineaciones

Las alineaciones son propiedades que se pueden colocar en el selector del flex-container para posicionar a sus flex-items dentro de sí mismo (justify-content | align-items | align-content) o también se pueden aplicar al selector de un flex-item para posicionarse distinto del resto de elementos (align-self).

justify-content - define la alineación de los ítems en el eje principal.

justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly | start | end | left | right;

align-items - define la alineación de los ítems en el eje secundario.

align-items: stretch | flex-start | flex-end | center | baseline | first baseline | last baseline | start | end | self-start | self-end;

align-content - define la alineación de los ítems en el eje secundario.

align-content: flex-start | flex-end | center | space-between | space-around | space-evenly | stretch | start | end | baseline | first baseline | last baseline;

align-self - permite alinear un elemento de forma independiente al alineamiento asignado por su flex-container a sus flex-items.

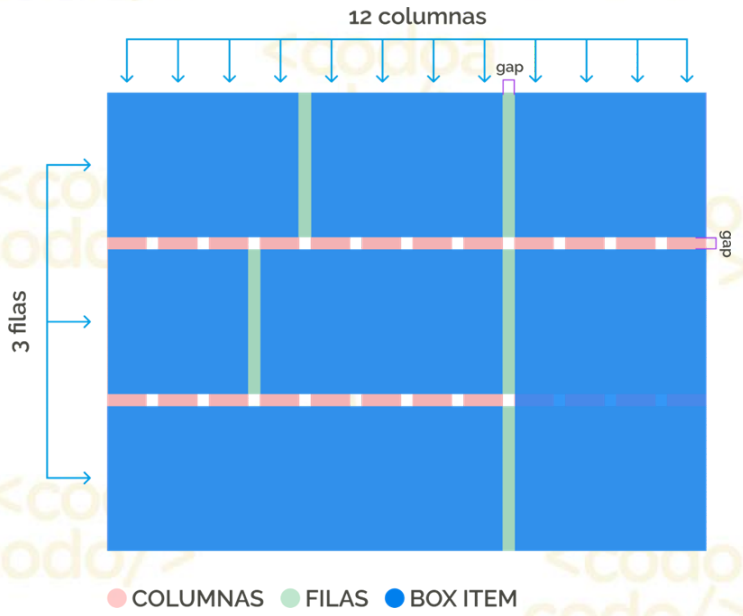
align-self: auto | flex-start | flex-end | center | baseline | stretch;

Grid Layout

El sistema flexbox es una gran mejora, sin embargo, está orientado a estructuras de una sola dimensión, por lo que aún necesitamos algo más potente para estructuras web más específicas o complejas. Es por eso que gracias a GRID ahora podemos tener contenedores donde sus elementos hijos se distribuyan en 2 dimensiones, tomando parte en 2 ejes direccionales. El eje X (horizontal) y el eje Y (vertical).

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

En este sentido, con Grid Layout vamos a tener la posibilidad de crear plantillas de filas y columnas como si de tablas se tratara. Posicionando cada uno de nuestros items hijos en los lugares de la plantilla que necesitemos.



Con un diseño de filas y columnas como el anterior podríamos definir una grilla para ubicar las siguientes imágenes como en una galería:

La ventaja de grid es que todos los elementos hijos de un grid-container pueden posicionarse en 2 dimensiones sin necesidad de crear contenedores intermedios que vayan en una sola dirección como sucedería con flexbox.

Propiedades

Al igual que flexbox, Grid Layout está compuesto por un conjunto de propiedades aplicadas al selector de un elemento padre que determinará una grilla modelo o template con la forma buscada, la cual luego será ocupada por las propiedades aplicadas a los selectores de los elementos hijos.

Para definir nuestra grilla se utiliza las siguientes propiedades:

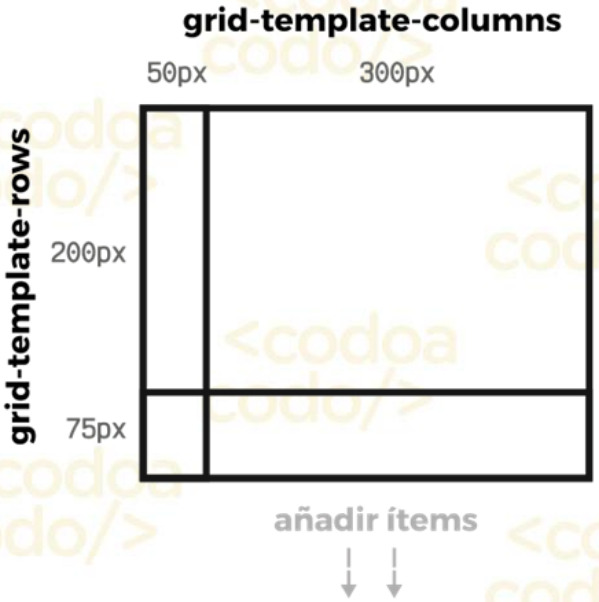
display - con esta propiedad en valor grid o inline-grid iniciamos grid en nuestro contenedor.

display: grid | inline-grid;

grid-template-columns - define tamaño y cantidad de columnas.

grid-template-rows - define tamaño y cantidad de filas.

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



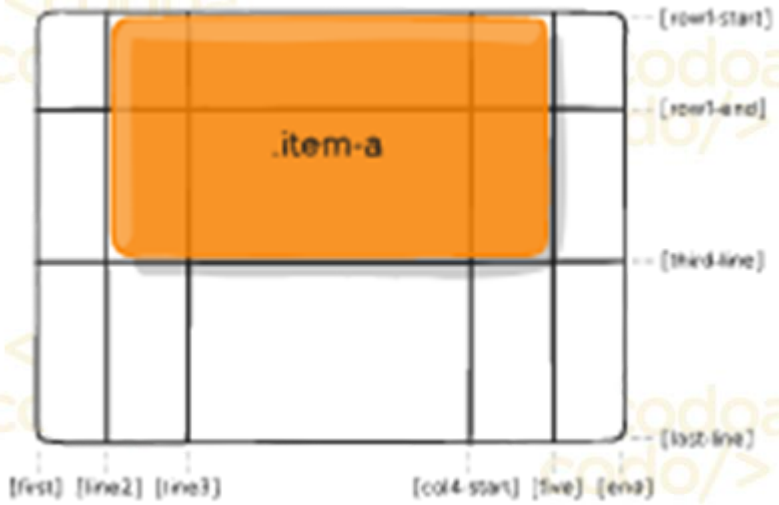
grid-column-start - define en qué columna comienza un item hijo.

grid-column-end - define en qué columna termina un item hijo.

grid-row-start - define en qué fila comienza un item hijo.

grid-row-end - define en qué fila termina un item hijo.

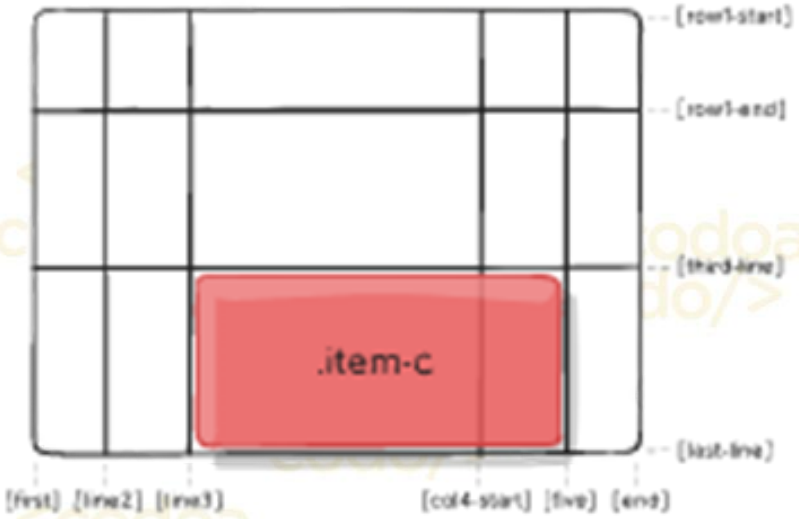
```css
.item-a {
  grid-column-start: 2;
  grid-column-end: 5;
  grid-row-start: 1;
  grid-row-end: 3;
}
```



grid-column - shorthand para declarar columna inicial y final a la vez.

grid-row - shorthand para declarar fila inicial y final a la vez.

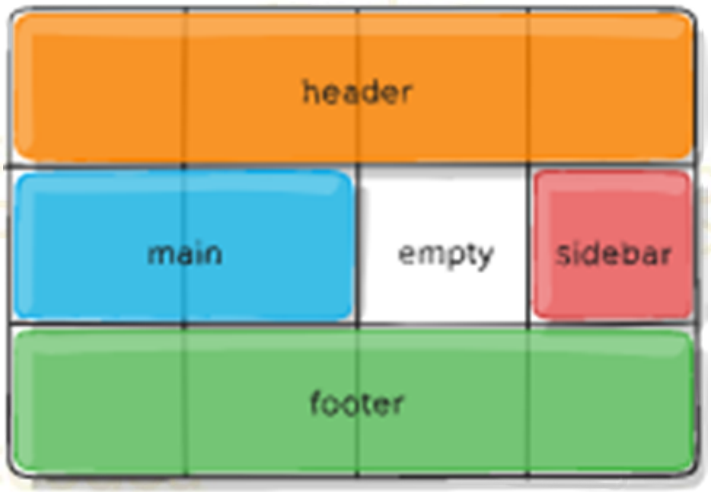
```css
.item-c {
  grid-column: 3 / span 2;
  grid-row: 3 / 4;
}
```



grid-template-areas - define una grilla explícita a través de áreas predeterminadas.

grid-template-areas - define una grilla explícita a través de áreas predeterminadas.

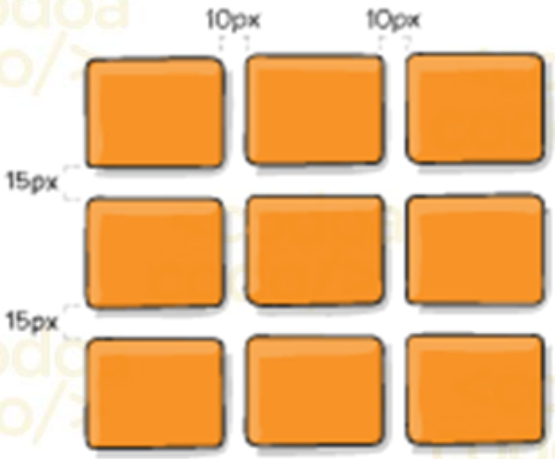
```css
.container {
  display: grid;
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
  "header header header header" "main main . sidebar"
  "footer footer footer footer";
}

.item-a {
  grid-area: header;
}
.item-b {
  grid-area: main;
}
.item-c {
  grid-area: sidebar;
}
.item-d {
  grid-area: footer;
}
```



grid-column-gap - determina el espaciado entre columnas.

grid-row-gap - determina el espaciado entre filas.

grid-gap - determina el espaciado entre filas y columnas a la vez.



Alineación

Para poder alinear nuestros elementos en Grid Layout vamos a utilizar propiedades muy similares a las de flexbox en nombre y comportamiento.

Estas propiedades pueden ser aplicadas al selector del elemento padre (grid-container) o por separado a sus elementos hijos (grid-items) dependiendo la propiedad.

justify-items - define la alineación de los ítems en el eje X.

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

align-items - define la alineación de los ítems en el eje Y.

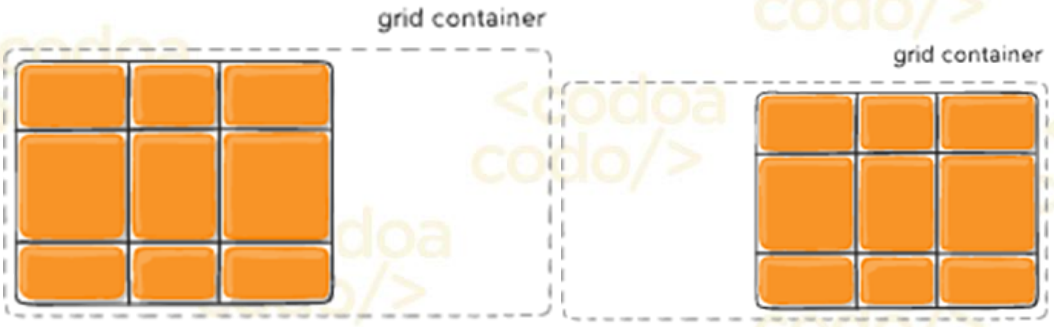
```
.container {
  align-items: start | end | center | stretch;
}
```

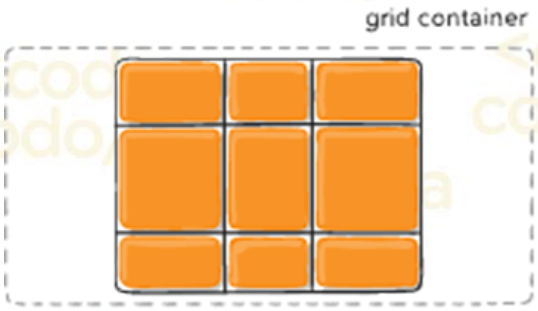
place-items - es un shorthand que define la alineación de los ítems en ambos ejes a la vez.

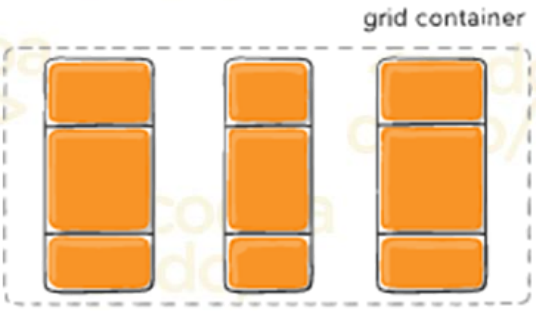
```
.center {
  display: grid;
  place-items: center;
}
```

justify-content - define la posición de toda la grilla en relación al espacio disponible del contenedor, por ejemplo cuando nuestra grilla es más pequeña que la caja que la contiene y necesitamos centrar todo el contenido o distribuir las columnas de forma uniforme.

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

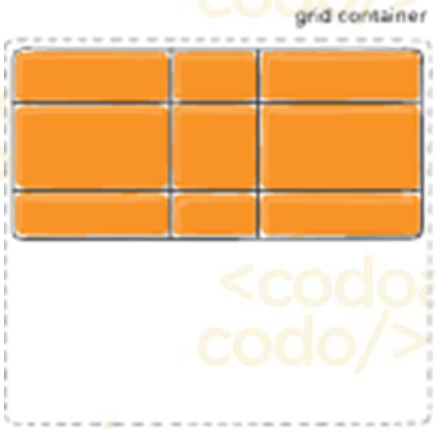


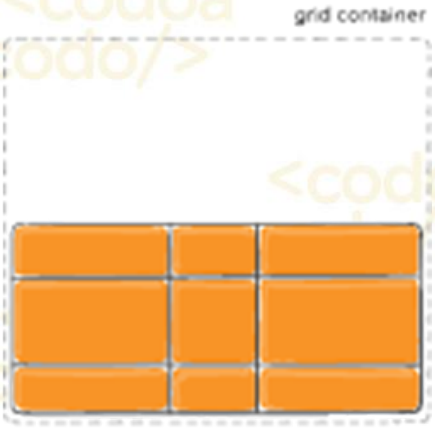


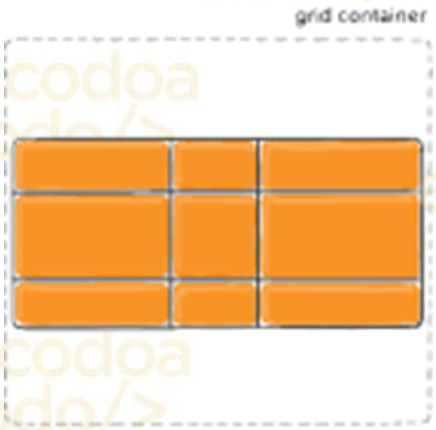


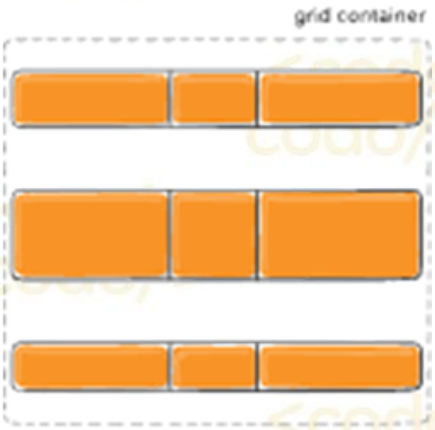
align-content - igual que el anterior pero cuando necesitamos centrar todo el contenido o distribuir las filas de forma uniforme.

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



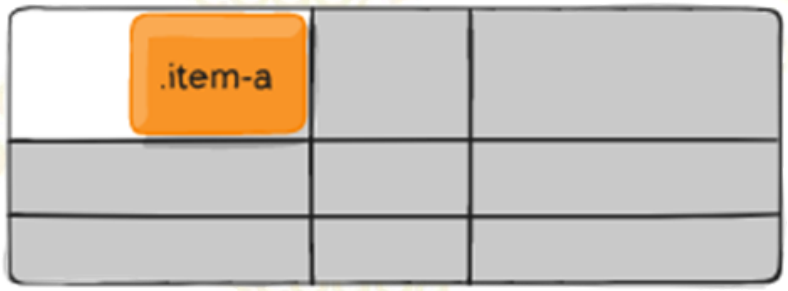






justify-self - nos permite definir la posición de un grid-item en el eje X.

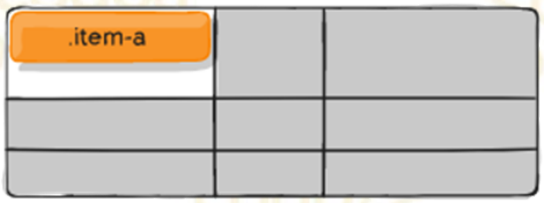
```
.item {
  justify-self: start | end | center | stretch;
}
```



Agencia de Aprendizaje a lo largo de la vida

align-self - nos permite definir la posición de un grid-item en el eje Y.

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



place-self - nos permite definir la posición de un grid-item en ambos ejes a la vez.

```
.item-a {
  place-self: center;
}
```