

# Flexbox (Parte I)

## Introducción

Cuando hablamos de construcción de layout lo primero que se nos vendrá a la cabeza será la propiedad floats. Esta propiedad, como hemos visto, funciona muy bien para la construcción de sitios web, sin embargo, cuando la creación de interfaces se hace más compleja esta propiedad podría no ser suficiente.

Para solventar este problema existe una tecnología que nos permitirá posicionar nuestros elementos de manera sencilla usando **flexbox**.

Durante toda esta unidad conoceremos los elementos que componen a estas cajas flexible, su comportamiento, además contrastaremos flexbox con la grilla de bootstrap y finalmente aplicaremos esta tecnología en un proyecto.

## Conociendo el proyecto

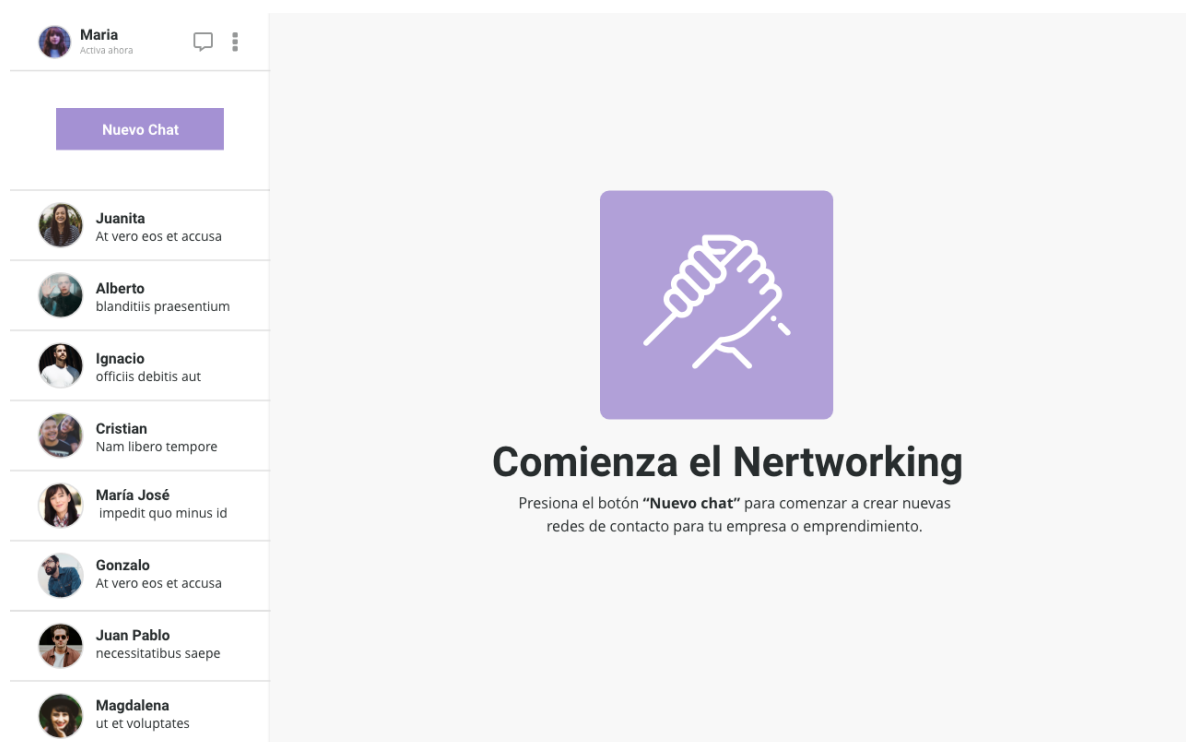


Imagen 1. Proyecto final.

El proyecto que realizaremos en esta ocasión será la vista principal de una aplicación web de mensajería la que tiene como objetivo conectar a usuarios a modo de generar redes de contacto entre empresas y emprendedores.

La interfaz de usuario entregada en el archivo comprimido contiene tres mockup que muestran el flujo que tendrá la vista en dispositivos móviles, tablets y de escritorio. Además tendremos a disposición las imágenes usadas en la *APP*, junto con una guía de estilos con tamaños, colores, tipos de fuentes y componentes usados por la aplicación.

En definitiva, esta sección nos dará las bases suficientes para construir el *layout* de nuestras maquetas utilizando técnicas actuales de posicionamiento y alineación con *CSS*.

## Conociendo *Flexbox*

---

Antes de comenzar directamente con el proyecto es importante detenernos un momento para conocer acerca de *flexbox*.

### ¿Qué es *flexbox*?

*Flexbox* es un sistema de elementos Flexibles de *CSS* que nos ayudará a crear, adaptar y automatizar la distribución de los elementos *HTML* de manera más eficiente, ya que con ella podremos distribuir el espacio y alinear el contenido de diferentes maneras dentro de un sitio o aplicación web. Esto significa que el contenido dentro de un contenedor con *flexbox* podrá ser posicionado, ordenado o alineado según el espacio disponible.

*Flexbox* es similar a las propiedades de bloque como los *floats*, pero a diferencia de ellos contiene algunas características que harán mas simple el alineamiento de contenidos dentro de interfaces complejas dentro de aplicaciones y sitios web.

### Conceptos básicos de *flexbox*

Algo importante y que diferencia a *flexbox* de otras opciones es que la dirección de los elementos es agnóstico a la dirección física en la que se encuentra el flujo normal de los elementos, es decir que los elementos que se encuentran al interior de este tendrán una disposición independiente a la que tiene el flujo normal del documento.

Para entender mejor este concepto conoceremos algunos términos relacionados al modelo de cajas flexible usado por esta herramienta.

## Modelo de cajas flexible

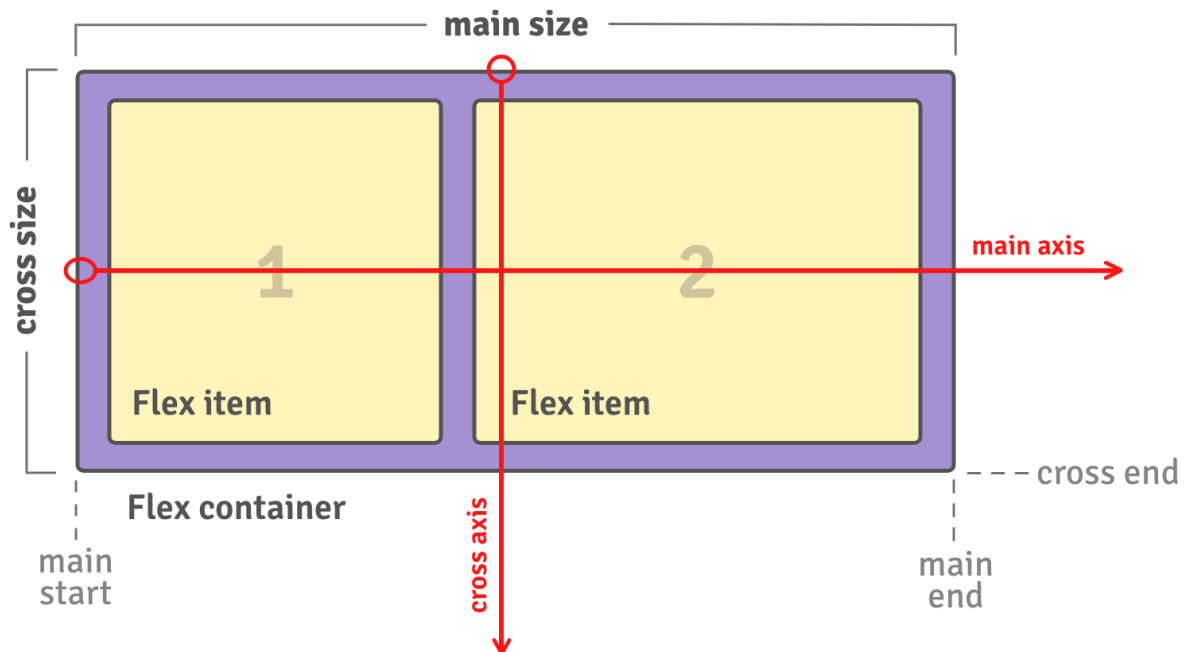


Imagen 2. Modelo de Cajas Flexible.

Primero, debemos saber que *flexbox* funciona con dos elementos principales: el primero es el contenedor de los elementos llamado **"Flex container"** y el segundo, son los ítems al interior del contenedor llamados **"Flex ítem"**.

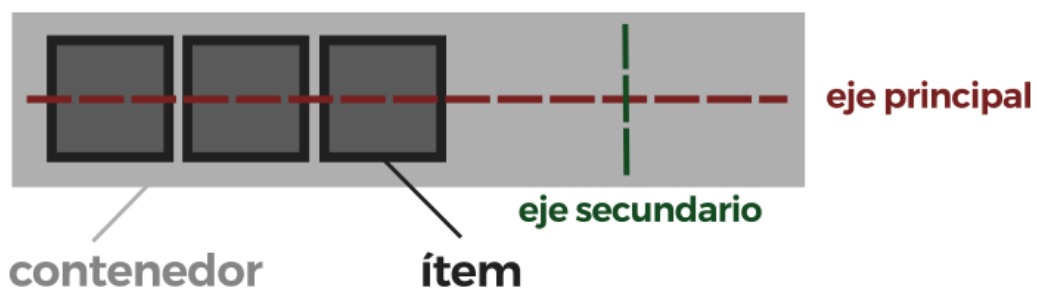


Imagen 3. Descripción gráfica de Flex container y Flex ítem.

Contenedor: Existe un elemento padre que es el contenedor que tendrá en su interior cada uno de los ítems flexibles y adaptables.

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

A diferencia de los elementos de bloques o inline, cuyo movimiento está basado en la orientación que tiene el elemento - sea este de bloque o de línea - *flexbox* basa su orientación a dos ejes principales los que se determinan con la propiedad **flex flow direction\***.

Para comenzar a trabajar en el modo flexbox hay que utilizar sobre el elemento contenedor la propiedad "display" que vimos en un capítulo previo, y especificar el valor flex o inline-flex dependiendo de como queramos que se comporte el contenedor: si como un elemento en línea, o como un elemento en bloque.

- inline-flex: Establece un contenedor de ítems flexible en línea, de forma equivalente a inline-block.
- Flex: Establece un contenedor de ítems flexible en línea, de forma equivalente a inline-block.

## Dirección de los ejes

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

Existen dos propiedades primordiales para la manipulación del comportamiento y dirección de los ítems a lo largo del eje principal del contenedor, que son los mencionados a continuación:

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

Imagen 4. Propiedades primordiales.

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

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

Esto nos permite el control sobre el orden de los elementos en una página, como se puede apreciar en el ejemplo:

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

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

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

Existe otra propiedad llamada flex-wrap donde se especifica el comportamiento del contenedor respecto a evitar el desborde (nowrap, por default) o permitir que lo haga, en este caso, sería un contenedor flexbox multilinea.

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

En base al ejercicio anterior podemos declarar flex-wrap.

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

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

En el caso de especificar nowrap en el contenedor, los 3 ítems se mostrarían en una misma línea del contenedor. Cada ítem tiene un 50% de ancho, en px serían 100px dado que el contenedor es de 200px. Si cada ítem equivale a 100px nos da un total de 300px, que no cubre el contenedor ya que este tiene un valor de 200px, por lo que flexbox reajusta los ítems flexibles para que quede en la misma línea de forma proporcionada.

Ahora modifica la propiedad flex-wrap para ver como se comporta.

## Alineación

Justify-content es una propiedad que sirve para colocar ítems de un contenedor mediante una disposición concreta a lo largo del eje principal:

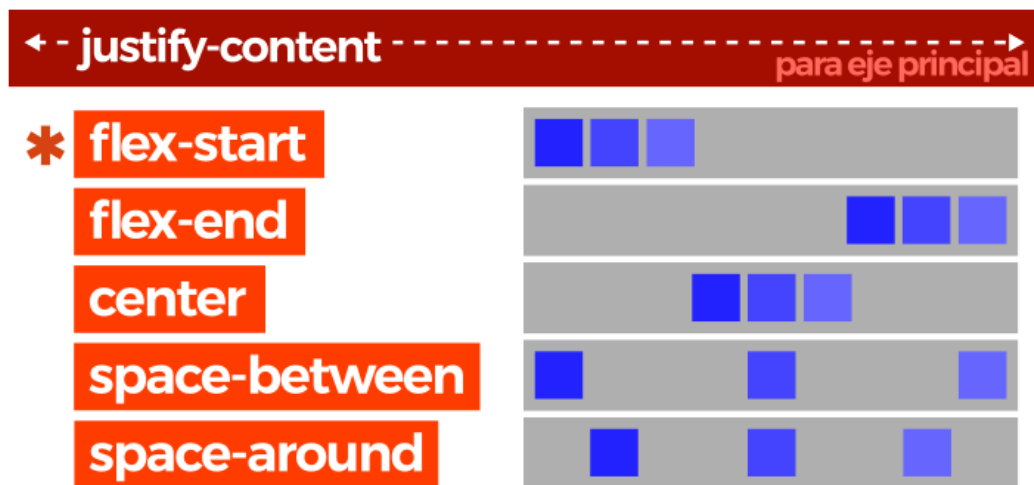


Imagen 5. Propiedad Justify-content.

Align-content nos permite alinear cada una de las líneas del contenedor. Los valores que puede tomar son los siguientes:

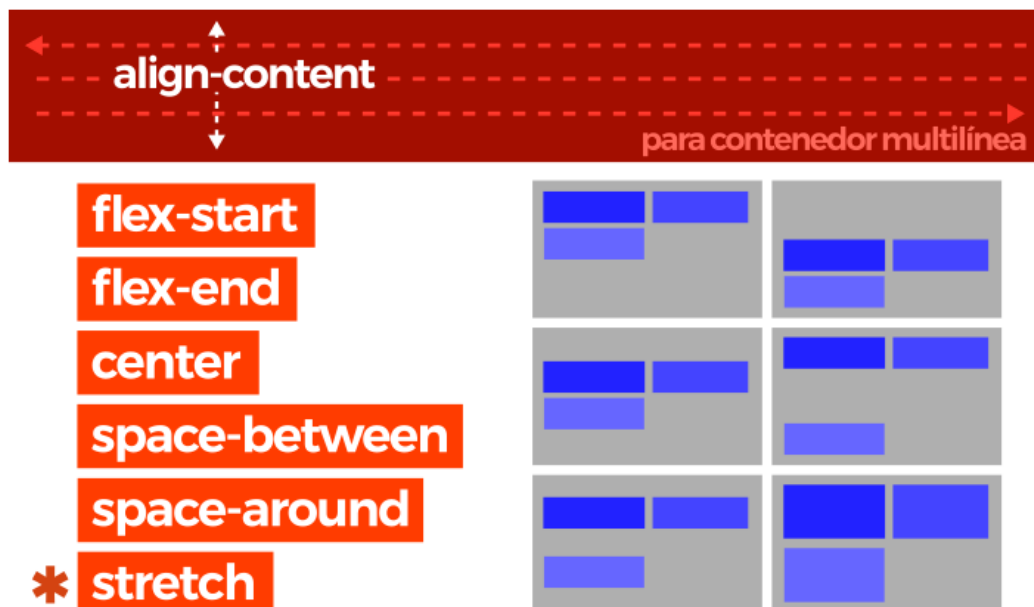


Imagen 6. Propiedad Align-content.

La propiedad align-items se encarga de alinear los ítems en el eje secundario del contenedor, lo hace sobre la línea actual. Hay que tener cuidado de no confundir align-content con align-items, su función es muy distinta.

Los valores que puede tomar son los siguientes:

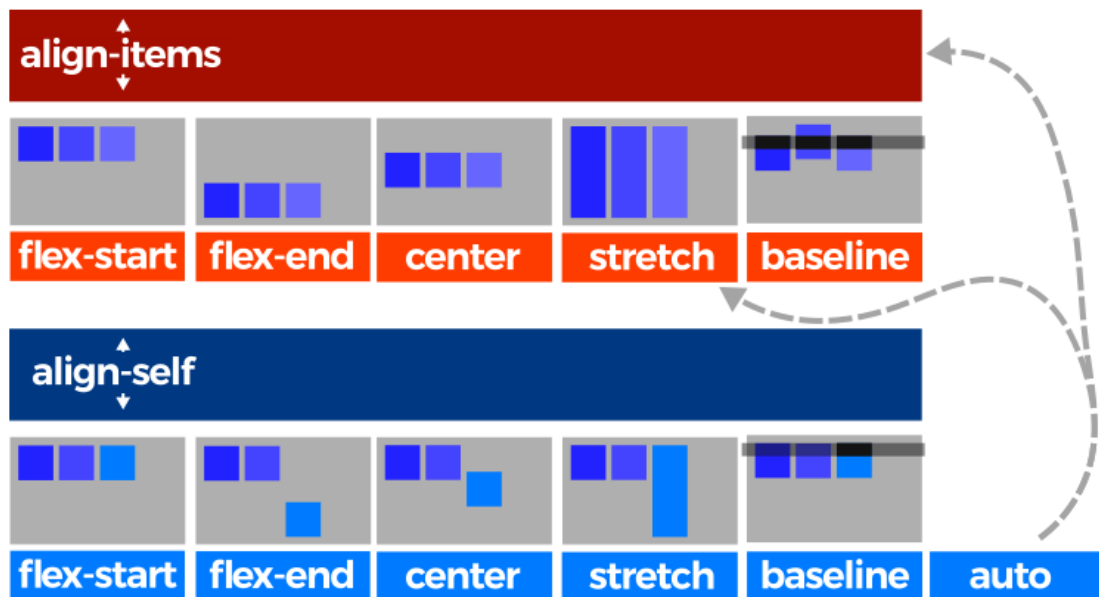


Imagen 7. Valores que toma align-items y align-self.

### ¿Qué es *flex flow direction*?

Este flujo se compone de dos elementos principales: el primero es el **main axis** o **main dimension** y el segundo es **cross axis** o **cross dimension**.

El primer eje es el que contendrá, por defecto, a los elementos de izquierda a derecha. Estos elementos comienzan por el "**main start**" y finalizan en el "**main end**".



Imagen 8. Flujo main axis.

El segundo eje está compuesto por un eje vertical que comienza arriba y termina abajo, comenzando por el "**cross start**" y finalizando en el "**Cross end**".

Es bueno acotar que el flujo normal de los elementos en *flexbox* sigue la dirección del **main axis**, o sea, que los elementos del contenedor se colocarán a lo largo de esta.



Imagen 9. Flujo cross axis.

## Ancho de los elementos

Por otra parte el tamaño que tendrá el contenedor o los elementos dependerán del flujo. Si el ancho y alto se basa en la dimensión principal el tamaño será el **main size** y si el ancho y alto se basa en la dimensión transversal se llamará **cross size**.

En resumen, todos los conceptos vistos nos ayudará a comprender de mejor manera el comportamiento que tienen las propiedades *flexbox* cuando se usan en un proyecto.



# Estructura de *flexbox*

Bien, ahora que conocemos los elementos que componen al modelo de cajas flexible es momento de crear nuestro proyecto de prueba para identificar de mejor manera los comportamientos que tiene *flexbox*.

Primero creemos un nuevo directorio llamado `test_flexbox` al interior del escritorio y luego pasaremos este hacia nuestro editor de texto.

Cuando estemos dentro, crearemos en la raíz un archivo `index.html`. Después crearemos un directorio `assets` que contendrá un carpeta `css`, en la que luego crearemos un `styles.css` para nuestros estilos.

Ahora ingresemos a `index.html` y después agreguemos la estructura *HTML* del documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Conociendo Flexbox</title>
</head>
<body>

</body>
</html>
```

Luego, integraremos la hoja de estilo agregando la ruta relativa a la etiqueta `<link>`.

```
<link rel="stylesheet" href="assets/css/styles.css">
```

## Contenedor e ítem

Bien, con lo anterior terminado comenzaremos a bajar algunos conceptos vistos en la unidad anterior.

Si recordamos el modelo de caja de *flexbox* se compone por dos elementos: el primero es el contenedor y el segundo son los ítems.

El contenedor es el padre y los ítems son todos los hijos directos del contenedor.

Pues bien, ahora agreguemos debajo de `<body>` una etiqueta `<div>` con clase `.container` y dentro de esta agregaremos tres `<div>` con clase `.item`.

```
<body>
  <div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
  </div>
</body>
```

Los que creamos anteriormente es la estructura base de cualquier elemento que quiera ser afectado por *flexbox*.

El resultado de lo anterior es el siguiente:



Imagen 10. Resultados ejemplo anterior.

## ***Flexbox: dirección y desborde***

Después de terminar de construir el contenedor e ítems para este caso práctico comenzaremos a conocer las propiedades que podremos usar al interior de este modelo de cajas iniciando con las propiedades especializadas en contenedores.

Si queremos transformar al padre y sus elementos hijos en flexibles debemos agregar en el contenedor la propiedad `display: flex;` en el contenedor. Esta propiedad define a un contenedor cómo flexible y a todos sus hijos directos.

Veamos cómo funciona esto ingresando al archivo `styles.css` y luego agregando algunos estilos a los elementos.

Comencemos agregando estilos en la raíz del documento usando la pseudo clase `:root`. En el agregaremos un `font-family: monospace`.

```
:root {  
  font-family: monospace;  
}
```

Debajo le daremos estilos al contenedor. Creemos la clase `.container` y dentro definiremos un `border: 2px solid black`, un `text-align: center` y finalmente un `font-size: 3rem`.

```
.container {  
  border: 2px solid black;  
}
```

Finalmente agregaremos los estilos a la clase `.item`. Definamos que los elementos hijos del contenedor tengan un ancho de `25%`, además de un borde solido negro de `1px`, un color de fondo `coral` y un color de fuente `white`.

```
.item {
  width: 25%;
  border: 1px solid black;
  background-color: coral;
  color: white;
}
```

Si abrimos el archivo `index.html` en el navegador veremos que los ítems del contenedor se encuentran apilados uno por sobre el otro siguiendo el comportamiento típico de un elemento de bloque como lo es el `<div>`. Si agregamos a la clase `.container` un `display: flex` veremos que los elementos cambiaron de posición de manera horizontal.

```
.container {
  display: flex;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```

Al agregar el `display: flex` definimos que todos los elementos al interior del contenedor tendrán un flujo definido por el modelo de cajas flexibles. Además si nos fijamos, los ítems del contenedor tienen un flujo que sigue al `main axis` que comienza a la izquierda desde el `main start` y finaliza a la derecha en el `main end`.

El proyecto hasta el momento debería verse de la siguiente forma:

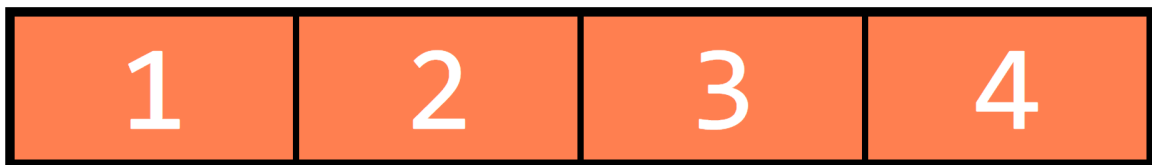


Imagen 11. Vista de avance de proyecto.

## Propiedades de contenedor: dirección

Esta dirección definida por defecto podremos cambiarla usando la propiedad llamada `flex-direction`, la cual nos permitirá definir si la dirección será definida por el `main axis` o por el `cross axis`.

Agreguemos la propiedad `flex-direction` con un valor `column` debajo de `display: flex`.

```
.container {
  display: flex;
  flex-direction: column;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```

Si recargamos veremos que los elementos que se encontraban de manera horizontal ahora están de manera vertical siguiendo el eje `cross axis` que comienza arriba en el `cross start` y finaliza abajo en el `cross end`. Si quisiéramos que los elementos sigan al `main axis` debemos cambiar el valor por `row`.

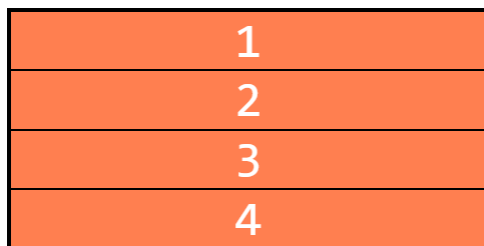


Imagen 12. Vista vertical de ejemplo.

También podremos hacer que los elementos tomen una dirección a la reversa de cómo fueron creados usando el valor `column-reverse` para el `cross axis` y `row-reverse` para `main axis`.

Cambiamos el valor `flex-direction` por `row-reverse` y veamos qué sucede.

```
.container {  
  display: flex;  
  flex-direction: row-reverse;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```

Como vemos la dirección de los ítems cambiaron y comienzan desde el `main end` y terminan en el `main start`.



Imagen 13. Uso de valor `flex-direction` por `row-reverse`.

## Propiedades de contenedor: desborde

Ahora bien, existe otra propiedad que nos ayudará a definir el ajuste de los elementos hijos contenidos por el padre.

En este caso nosotros tenemos dentro del contenedor 4 ítems que tienen un tamaño de ancho del `25%`. Si disminuimos el tamaño navegador los elementos tratarán de mantenerse en una línea, pero qué pasaría si quisiéramos cambiar ese ajuste por otro, como por ejemplo que cuando el espacio sea menor al definido en el elemento este se apile. Para lograr esto podremos usar la propiedad `flex-wrap`.

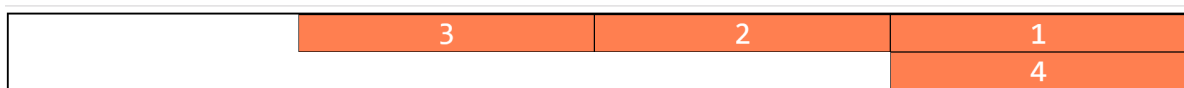
Esta propiedad nos permitirá cambiar el ajuste por defecto llamado `nowrap` los ítems por el que definamos nosotros. Este ajuste predefinido trata de mantener los elementos en una línea ajustándolos aún cuando los elementos tengan unidades absolutas en su interior.

Si queremos cambiar esto podremos usar un valor `wrap` los elementos bajarán dependiendo del tamaño que tengan los ítems.

Veamos un ejemplo de esto agregando debajo del `flex-direction` la propiedad `flex-wrap` y demos un valor de `wrap`.

```
.container {  
  display: flex;  
  flex-direction: row-reverse;  
  flex-wrap: wrap;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```

El resultado que veremos de esto será que el elemento con número 4 bajo aún cuando el tamaño de los 4 elementos suman un 100%.



	3	2	1
			4

Imagen 14. Descripción.

Esto se debe a que estos tienen un border de `2px` el cual hace que el tamaño sobrepase el 100%.

Para solucionar este problema podremos agregar a la clase `.item` un `box-sizing: border-box` para recalcular la caja y solucionar el problema.

```
.item {  
  box-sizing: border-box;  
  width: 25%;  
  border: 1px solid black;  
  background-color: coral;  
  color: white;  
}
```



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

Imagen 15. Resultado al agregarla clase `.item`.

Bien, cambiemos el tamaño de los ítems por `500px`.

```
.item {  
  width: 500px;  
  box-sizing: border-box;  
  border: 1px solid black;  
  background-color: coral;  
  color: white;  
}
```

Ahora los elementos en la página bajarán debido a que el viewport tiene un menor tamaño que los 4 ítems juntos, de modo que al no tener espacios estos bajan.

3	2	1
		4

Imagen 16. Resultado de que el viewport tenga un menos tamaño que los 4 ítems.

Finalmente, existe una propiedad con la que podremos definir el alineamiento y flujo de los elementos llamada `flex-flow`.

En nuestro caso, cómo tenemos definidas dos reglas podremos agregar `flex-flow` para contenerlas en una sola propiedad. Primero debemos agregar el valor de `flex-direction`, seguido por `flex-wrap`.

```
.container {  
  display: flex;  
  flex-flow: row-reverse wrap;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```

Si recargamos veremos que los ítems mantuvieron la misma dirección y flujo.

Como vimos definir la dirección y flujo de los elementos hijos de un contenedor es fácil y no requiere de mucho código para generar un gran cambio en nuestro *layout*.

## Flexbox: alineación

Luego de conocer cómo definir la dirección y flujo de los elementos dentro de un contenedor es momento de seguir conociendo las propiedades para elementos padres de *flexbox*.

Las propiedades que conoceremos a continuación son usadas para alinear los elementos siguiendo los ejes vistos en el modelo de cajas flexible, o sea, `main axis` y `cross axis`.

Comencemos conociendo la propiedad `justify-content`.

### Alineación siguiendo el *main axis*

Esta propiedad alinea los elementos flexibles a lo largo del `main axis` del contenedor. Esto significa que podremos distribuir de mejor manera el espacio que queda cuando los elementos tienen su tamaño definido y controlar los elementos cuando se desbordan.

Veamos en acción a esta propiedad agregando en el contenedor (`.container`) la propiedad `justify-content`. Dentro de esta propiedad podremos agregar diferentes valores como:

**flex-start:** Este es el valor que viene por defecto en los elementos flexibles y permite alinear los elementos a la izquierda en donde se encuentra el `main start`.

Para ver de buena forma los efectos de estas propiedades es mejor modificar el atributo width de los items.

```
.item{  
  width: 15%;  
}
```

Si agregamos este valor y luego, recargamos el navegador veremos que no hubo ningún cambio asociado en la alineación de los elementos.

```
.container {  
  display: flex;  
  flex-flow: row-reverse wrap;  
  justify-content: flex-start;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```



Imagen 17. Vista de la alineación de los elementos.

**flex-end:** El segundo valor alinea los elementos la derecha del `main axis`.

Cambiamos el valor por `flex-end` y recarguemos. Podremos ver que los elementos se alinearon a la derecha del eje principal.

```
.container {  
  display: flex;  
  flex-flow: row-reverse wrap;  
  justify-content: flex-end;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```



Imagen 18. Vista con el valor Flex-end.

**center:** Si deseamos centrar los elementos, podremos usar el valor `center`.

Si cambiamos el valor veremos que los elementos se centran.

```
.container {
  display: flex;
  flex-flow: row-reverse wrap;
  justify-content: center;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```



Imagen 19. Vista con el valor Center.

**space-between:** Ahora bien, si lo que queremos es posicionar de manera uniforme, debemos usar `space-between`, que ordenará los elementos a la izquierda al centro y a la derecha.

En este caso al tener definido un tamaño fijo no podremos ver bien como se comportan los elementos, así que quitaremos el ancho que agregamos y cambiaremos el valor de `justify-content` a `space-between`.

Si revisamos veremos que los elementos se separaron a la izquierda, dos al centro y uno a la derecha.

```
.container {
  display: flex;
  flex-flow: row-reverse wrap;
  justify-content: space-between;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```

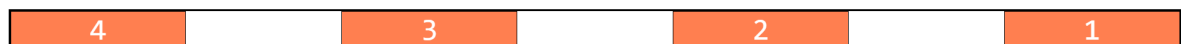


Imagen 20. Vista con el valor Space-between.

## Alineación siguiendo el cross axis

La alineación también se puede hacer siguiendo el eje transversal o `cross axis` usando la propiedad `align content`.

Esta propiedad es muy parecida a `justify-content`, ya que, nos ayudará a alinear los elementos siguiendo a un eje, pero esta vez seguirá al `cross axis`. Por defecto el valor de los elementos es `stretch`, que estira a los ítems respetando los anchos mínimos y máximos.

Para ver de mejor manera el comportamiento de los elementos usando este tipo de alineación, agregaremos un alto al contenedor de `1000px` y un ancho a los ítems de `500px`.

```
.container {
```



```

height: 1000px;
display: flex;
flex-flow: row wrap;
border: 2px solid black;
text-align: center;
font-size: 3rem;
}

.item {
width: 500px;
box-sizing: border-box;
border: 1px solid black;
background-color: coral;
color: white;
}

```

Si recargamos, veremos que los elementos tomaron todo el alto disponible.

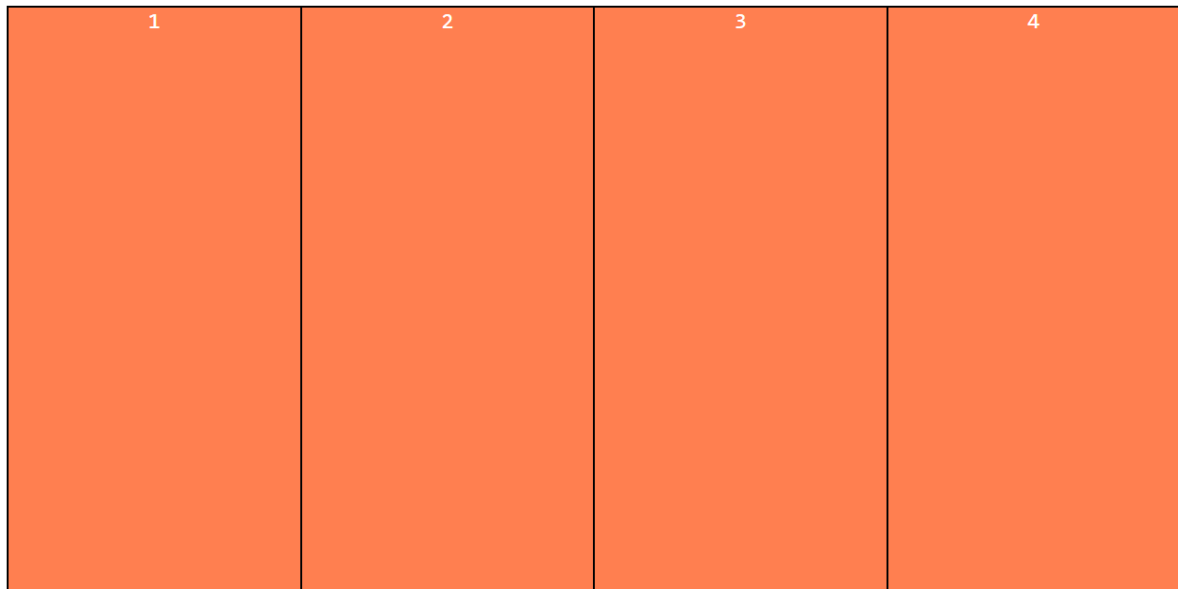


Imagen 21. Vista al utilizar valores de elementos Stretch.

Ahora, si cambiamos el valor a `flex-start` los elementos se alinearán al principio

```

.container {
height: 1000px;
display: flex;
flex-flow: row wrap;
align-items: flex-start;
border: 2px solid black;
text-align: center;
font-size: 3rem;
}

```

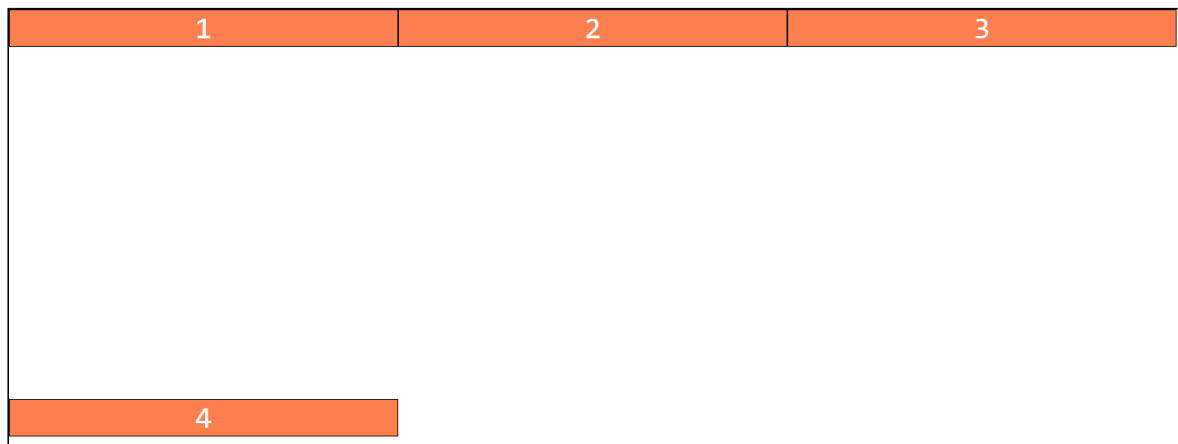


Imagen 22. Vista al utilizar el valor Flex-start.

Si cambiamos por `flex-end` estos se posicionarán al final.

```
.container {  
  height: 1000px;  
  display: flex;  
  flex-flow: row wrap;  
  align-items: flex-end;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```

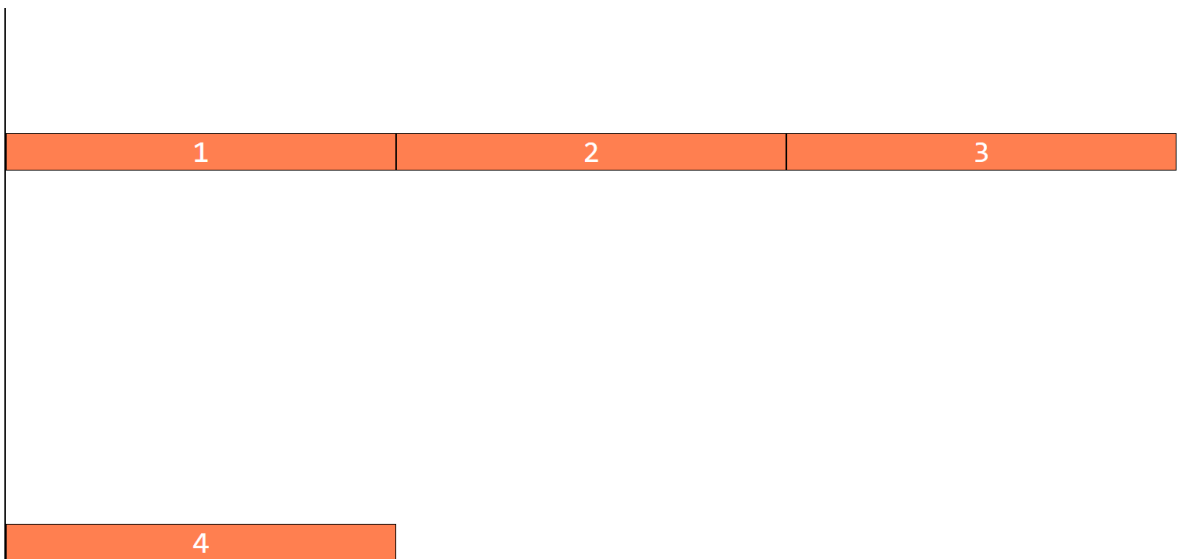


Imagen 23. Vista al utilizar el valor Flex-end.

También podremos centrar a los elementos usando `center`.

```
.container {  
  height: 1000px;  
  display: flex;  
  flex-flow: row wrap;  
  align-items: center;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```

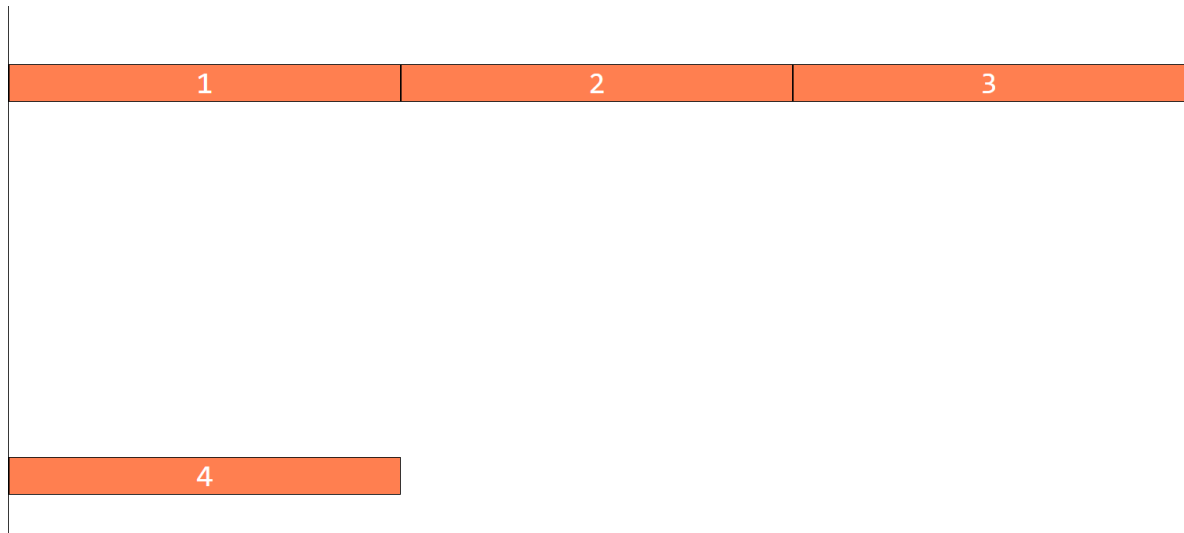


Imagen 24. Vista al utilizar el valor Flex-start y Center.

Existe un último valor que nos permitirá alinear los elementos usando como base la alineación del texto.

Si agregamos el valor `baseline` y luego, recargamos veremos que los elementos se movieron hacia la base del texto.

```
.container {  
  height: 1000px;  
  display: flex;  
  flex-flow: row wrap;  
  align-items: baseline;  
  border: 2px solid black;  
  text-align: center;  
  font-size: 3rem;  
}
```

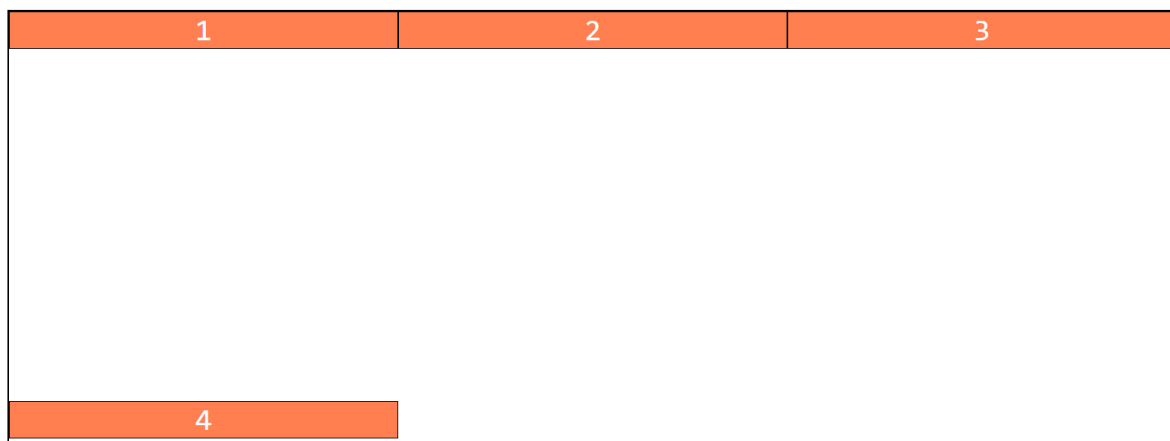


Imagen 25. Vista al utilizar el valor `baseline`.

## Alineación de contenido

Bien, ahora que conocemos como alinear los elementos de manera horizontal siguiendo el `main axis` o vertical siguiendo `cross axis`, revisaremos una última propiedad de contenedor que nos ayudará a alinear el contenido de nuestro contenedor usando la propiedad `align-content`.

Esta propiedad como mencionamos anteriormente nos ayudará a alinear el contenido siguiendo el `cross axis`. En esencia esta propiedad es similar a `align-items`, ya que alinea los elementos de manera vertical, pero a diferencia de la propiedad anterior toma a todos los elementos del contenedor, aún cuando los elementos se hayan desbordado con la propiedad `flex-wrap`.

Es importante tomar en cuenta que esta propiedad no funcionará si los elementos se encuentran en una sola línea.

Veamos esta propiedad en acción agregando debajo de `align-items` la propiedad `align-content`. Los valores que podremos usar son muy parecidos a `justify-content`.

Si agregamos `flex-start` veremos que los elementos se posicionarán arriba.

```
.container {
  height: 1000px;
  display: flex;
  flex-flow: row wrap;
  align-items: baseline;
  align-content: flex-start;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```

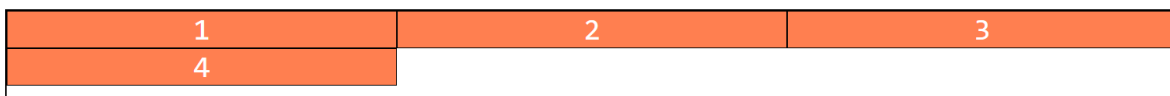


Imagen 26. Vista al utilizar el valor `Flex-start` y `baseline`.

Si agregamos `flex-end` los elementos se irán hacia el fondo.

```
.container {
  height: 1000px;
  display: flex;
  flex-flow: row wrap;
  align-items: baseline;
  align-content: flex-end;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```

1	2	3
4		

Imagen 27. Vista al utilizar el valor Flex-end y baseline.

Así mismo, si quisiéramos centrar los elementos podremos hacerlo con el valor `center`.

```
.container {
  height: 1000px;
  display: flex;
  flex-flow: row wrap;
  align-items: baseline;
  align-content: center;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```

1	2	3
4		

Imagen 28. Vista al utilizar el valor Center y baseline.

También, podremos posicionar el contenido de manera uniforme usando el valor `space-between`.

```
.container {
  height: 1000px;
  display: flex;
  flex-flow: row wrap;
  align-items: baseline;
  align-content: space-between;
  border: 2px solid black;
  text-align: center;
  font-size: 3rem;
}
```

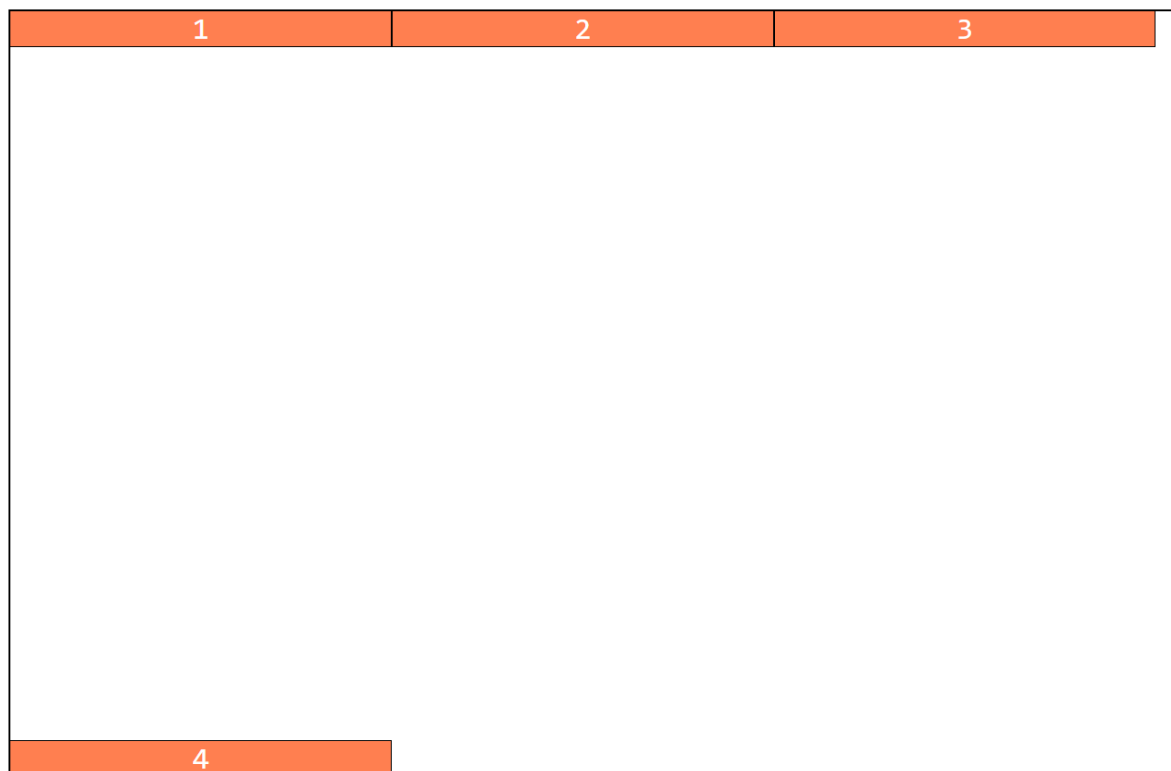


Imagen 29. Vista utilizando Space-between.

En conclusión usar estas propiedades para alinear contenido dentro de un contenedor nos será muy útil ya que con unas pocas propiedades podremos organizar y posicionar los elementos de manera fácil.

## ***Flexbox: orden y tamaño de los elementos***

En la unidad anterior vimos cómo alinear elementos de manera fácil dentro de un contenedor, ahora conoceremos otras propiedades relacionadas a los ítems.

Estas propiedades nos ayudarán a dar orden, tamaño y posición de manera independiente, sin afectar a todos los elementos.

### **Ordenar ítems del contenedor**

Para este caso práctico agregaremos algunas clases a los ítems a modo de afectar a uno en particular. Vamos a `index.html` y agreguemos en los elementos una clase `.item-` seguido por el número del elemento.

```
<div class="container">
  <div class="item item-1">1</div>
  <div class="item item-2">2</div>
  <div class="item item-3">3</div>
  <div class="item item-4">4</div>
</div>
```

Bien, ahora volveremos a `styles.css` y pondremos debajo estas cuatro clases.

```
.item-1 {...}

.item-2 {...}

.item-3 {...}

.item-4 {...}
```

Ahora en cada una agregaremos la propiedad `order` y como valor agregaremos un valor aleatorio del 1 al 4. El primero tendrá un `3`, el segundo `1`, el tercero un `4` y el último un `2`.

```
.item-1 {
  order: 3;
}

.item-2 {
  order: 1;
}

.item-3 {
  order: 4;
}

.item-4 {
  order: 2;
}
```

Sí guardamos y recargamos veremos que el orden de los elementos cambió gracias a la propiedad `order`.

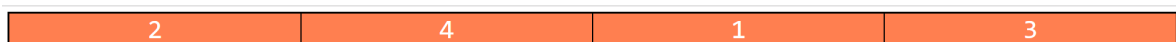


Imagen 30. Vista utilizando la propiedad `order`.

Esta propiedad es usada para escoger el orden los elementos dentro de un contenedor *flexbox*. Lo interesante de `order` que podemos decidir qué ítem irá primero y cuál último.

En el ejemplo vimos que el `item-2` paso a ser el primero, el `item-4` el segundo, el `item-1` como tercero y finalmente el `item-2` como cuarto. También podemos utilizar número negativos para ordenar dejar un elemento en el primer lugar.

## Aumentar tamaño de ítem

si quisiéramos aumentar el tamaño de un elemento de *flexbox*, podemos usar una propiedad llamada `flex-grow`, esta controla que tanto crecerá el flex-item para rellenar el espacio disponible. Su valor solo puede ser un número entero y no en negativo. Tal como se señala con anterioridad, esta propiedad se aplica si el espacio disponible es positivo, osea si el tamaño del contenedor es mayor a la suma de los flex-items y sus márgenes.

En `flex-grow` el valor definido indica cuantas unidades crecerá el ítem para calcular su tamaño. Por ejemplo, `flex-grow: 3` significa que el ítem crecerá 3 unidades.

La forma de dimensionar ese valor es con la siguiente formula:

***unit grow = espacio disponible / suma de flex-grow***

Veamoslo con un ejemplo:

```
.container { width: 500px; } .item-1 { flex-basis: 100px ; flex-grow: 1 } .item-2 { flex-basis: 100px ; flex-grow: 2 } .item-3 { flex-basis: 100px ; flex-grow: 1 } .item-4 { flex-basis: 100px ; flex-grow: 1 }
```

El espacio disponible es de 100px tamaño del contenedor menos la suma del tamaño de los items, y se distribuye de la siguiente forma:

$100px/5 = 20px \rightarrow$ ; Este es el valor de la unidad (unit grow).

El valor 5 es la suma de las unidades asignadas en el ejemplo anterior (1+2+1+1)

Luego esta unidad de crecimiento se multiplicará por el valor de flex-grow para determinar cuanto crecerá cada item.

item1- $\rightarrow$ ;  $20px * 1 = 20px$  item2- $\rightarrow$ ;  $20px * 2 = 40px$  item3- $\rightarrow$ ;  $20px * 1 = 20px$  item4- $\rightarrow$ ;  $20px * 1 = 20px$

Finalmente, el main size de cada item al imprimirse por el navegador será:

item1 - $\rightarrow$ ; 120px item2 - $\rightarrow$ ; 140px item3 - $\rightarrow$ ; 120px item4 - $\rightarrow$ ; 120px

Si los items tuvieran márgenes, el espacio disponible sería menor. Por ejemplo, si para el caso anterior todos los items tuvieran margin: 5px, el espacio disponible sería:

$500px - (100px+100px+100px+100px) - (5px+5px+5px+5px+5px+5px+5px+5px) = 60px$

## Reduciendo el tamaño de un ítem

Para reducir el tamaño de un elemento usaremos la propiedad `flex-shrink`. Esta propiedad define la capacidad de un elemento flexible de encogerse si es necesario. Para definir la proporción de los elementos deberemos hacer lo mismo que con `flex-grow` agregando un valor de `1` en cada ítem.

Si el espacio disponible es negativo (el tamaño del contenedor es menor a la suma de los tamaños de los items), de forma predeterminada los items se encogen en proporciones iguales para caber en una sola línea.

Para calcular el tamaño final de un item que se ha encogido, se usan los mismos principios que con flex-grow solo que en el sentido opuesto. Veamoslo con un ejemplo:

La operación para calcular el tamaño de un item se usa la misma logica que en flex-grow, solo que ahora es a la inversa, ese ejercicio lo puedes realizar a traves de la siguiente herramienta:

[Flexbox tester flex-shrink y flex-grow](#)

## Definiendo el tamaño de un ítem

En cuanto al tamaño de los elementos, *flexbox* tiene una propiedad que nos permitirá definir la longitud inicial de un elemento usando `flex-basis`. Esta propiedad para funcionar deberá ir acompañada por un valor en unidades relativas o absolutas el cual será el tamaño base. Probemos esto agregando esta propiedad al `.item-1`. El valor que tendrá es de `300px`.



```
.item-1 {
  order: 3;
  flex-grow: 1;
  flex-basis: 300px;
}
```

Si recargamos veremos que el tamaño del ítem 1 cambió de tamaño.



Imagen 31. Vista cambiando el tamaño de un ítem.

Este también podrá usar unidades relativas como porcentajes.

Cambiamos el valor por `20%`. Veremos que el elemento aún mantiene su tamaño, por lo que esta unidad también es funcional como tamaño inicial.

```
.item-1 {
  order: 3;
  flex-grow: 1;
  flex-basis: 20%;
}
```



Imagen 32. Vista utilizando el valor 20%.

También podremos hacerlo con la palabra clave `auto` la cual basará la longitud en base al espacio que tenga libre.

## Propiedad flex

Por último si queremos que ahorrar líneas de código podremos usar la versión reducida de estas tres propiedades llamada `flex`.

agreguemos en la clase `item-1` dónde se encuentran las propiedades, `flex-grow`, `flex-shrink` y `flex-basis` agregaremos la propiedad `flex`. dentro de ella definiremos en primer lugar el valor del `flex-grow` que es `1`, luego el de `flex-shrink`, de `1` y finalmente el `flex-basis` de `300px`.

Y finalmente, eliminemos las propiedades que unimos en la propiedad `flex`.

```
.item-1 {
  order: 3;
  flex: 1 1 300px;
}
```

Si recargamos veremos que los elementos matuvieron su tamaño aún cuando eliminamos las propiedades que daban el tamaño a los elementos.

Como vimos el definir el orden y el tamaño de un elemento es una gran técnica que nos ayudará controlar de manera independiente a este ítem flexible.

Para cerrar este tema te dejamos una guía completa de Flexbox, para que la analices y guardes en tus marcadores como guía.

[A Complete Guide to Flexbox](#)

## Resumen *Flexbox*

Recapitemos lo que llevamos conocido hasta el momento sobre *flexbox*:

- Sabemos que *flexbox* es una herramienta que nos permitirá construir elemento html de manera sencilla y estructurada.
- También conocemos la terminología que compone al modelo de cajas flexible.
- Así como también conocimos y aplicamos algunas propiedades propias de un contenedor, y otras especializadas en los ítems.

Ahora bien, todo este conocimiento aprendido hasta el momento lo aplicaremos en el proyecto, pero usando a un viejo amigo que en más de alguna vez nos ha sacado un problema. *Bootstrap* desde su versión 4 hacia adelante, comenzó a como base para su famosa grilla de 12 columnas.

Lo que podremos hacer con esta grilla no difiere en nada de lo que hemos visto hasta ahora, mas aún cuando conocemos y entendemos como se comportan los elementos de *flexbox*, ya que estos mismos comportamientos son usados por este *framework CSS*.

Sin ir más lejos ingresemos a la página principal de *Bootstrap* y luego escogamos la opción `documentation`. Aquí presionaremos la pestaña `Layout` y después en `Grid`.