Controlling The Alignment

Controlling The Alignment Of Flex Items In The Container

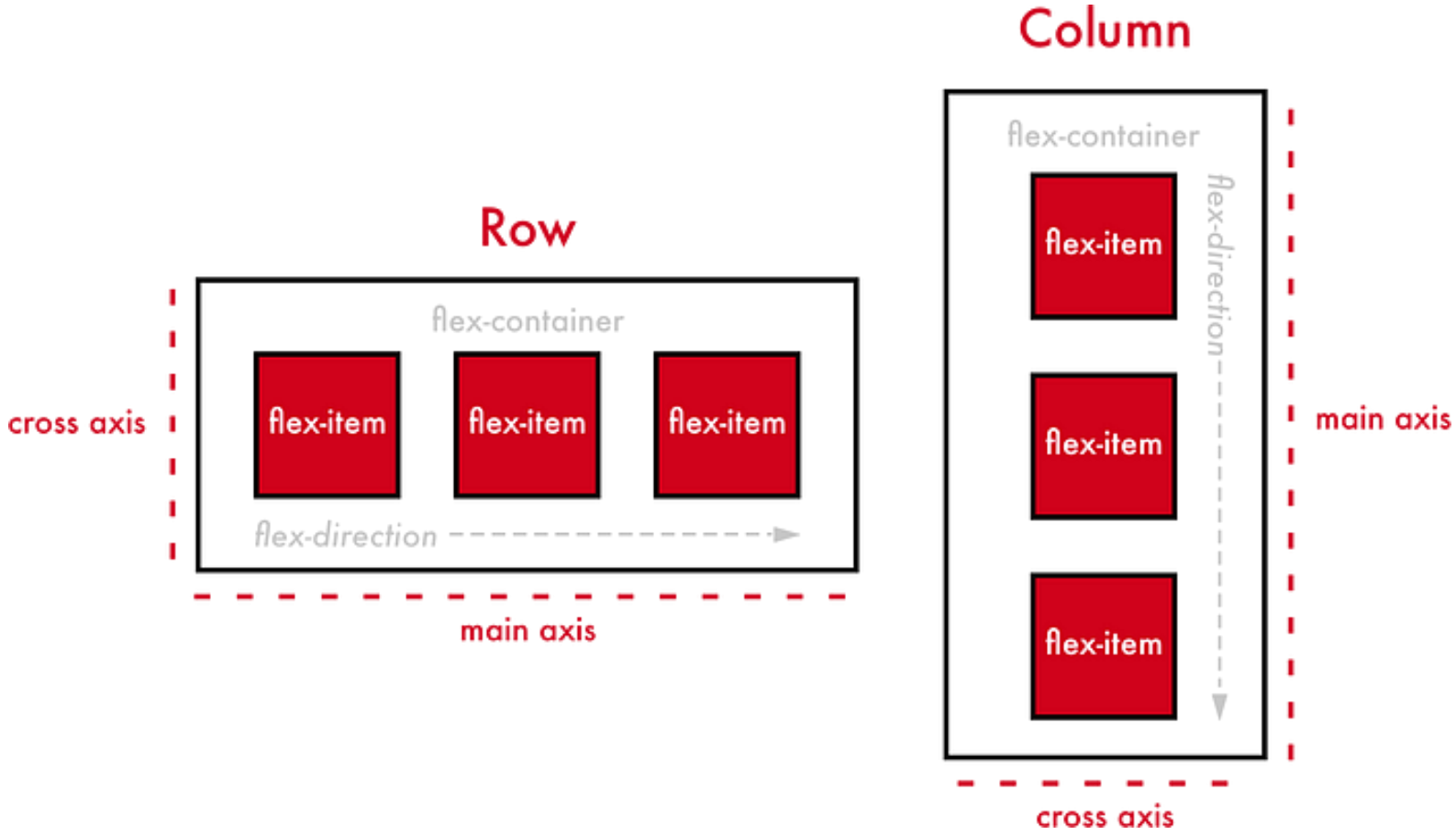
So far we've seen how to turn flexbox mode on, turning an element into a flex container and its children into flex items. We've also learned how to change the direction in which items flow, and allow them to wrap onto multiple lines.

The remaining set of container properties affects the alignment of items along the main axis (justify-content) and cross axis (align-items and align-content).

Aligning On The Main Axis

- The justify-content property defines how extra space should be distributed around or between items that are inflexible or have reached their maximum size.
- justify-content Values: flex-start | flex-end | center | space-between | space-around
- Default: flex-start
- Applies to: flex containers
- Inherits: no

Aligning On The Main Axis



Justify-content

Apply justify-content to the flex container element because it controls spacing within the container itself:

```
#container {
display: flex;
justify-content: flex-start;
 }
```

justify-content: flex-start; (default)

justify-content: flex-end;

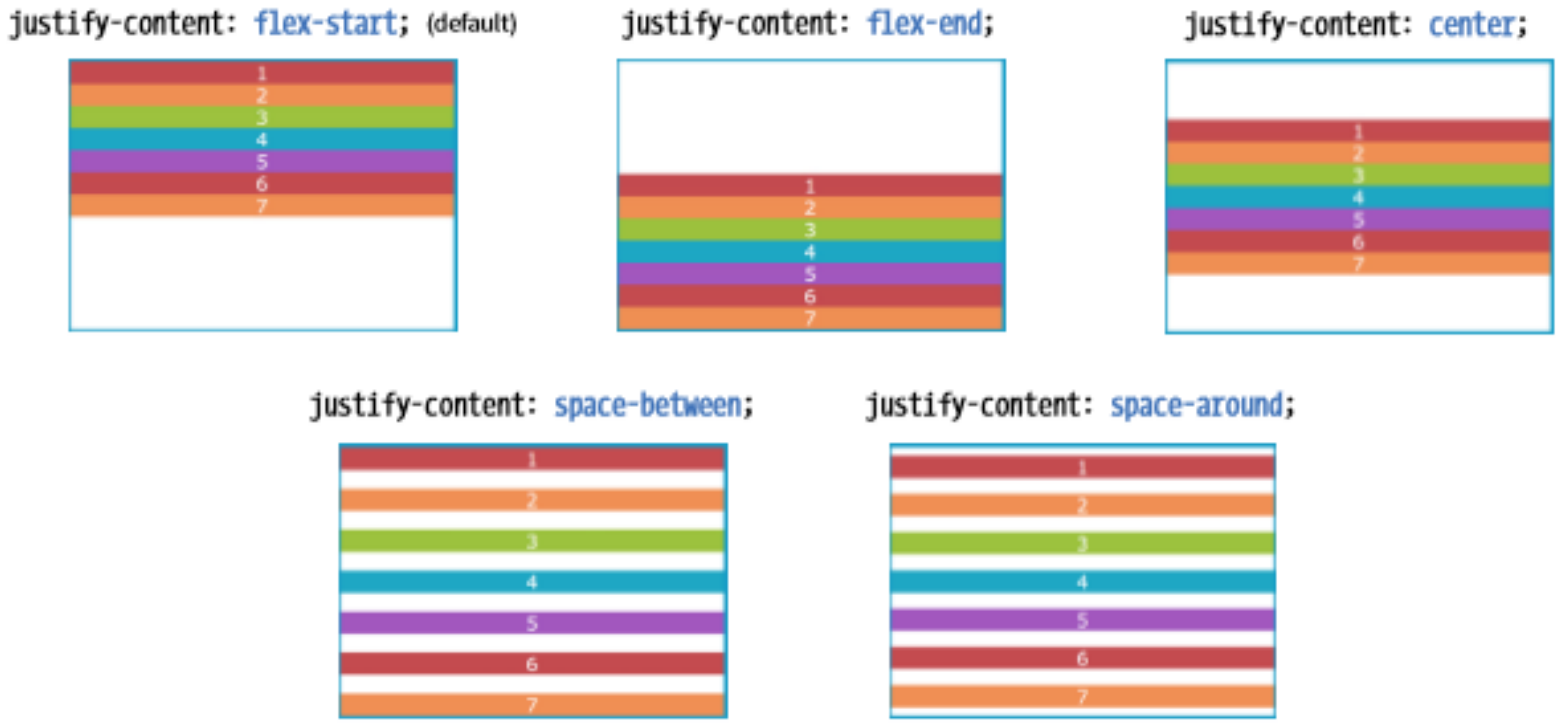
justify-content: center;

justify-content: space-between;

justify-content: space-around;

Justify-content- Column

- When the direction is set to a column with a vertical main axis, the keywords work the same way; however, there needs to be an explicit container height with space left over in order for you to see the effect.



Aligning On The Cross Axis

That takes care of arranging things on the main axis, but you may also want to play around with alignment on the cross axis (up and down when the direction is row, left and right if the direction is column). Cross-axis alignment and stretching is the job of the align-items property.

align-items

Values: flex-start | flex-end | center | baseline | stretch

Default: stretch

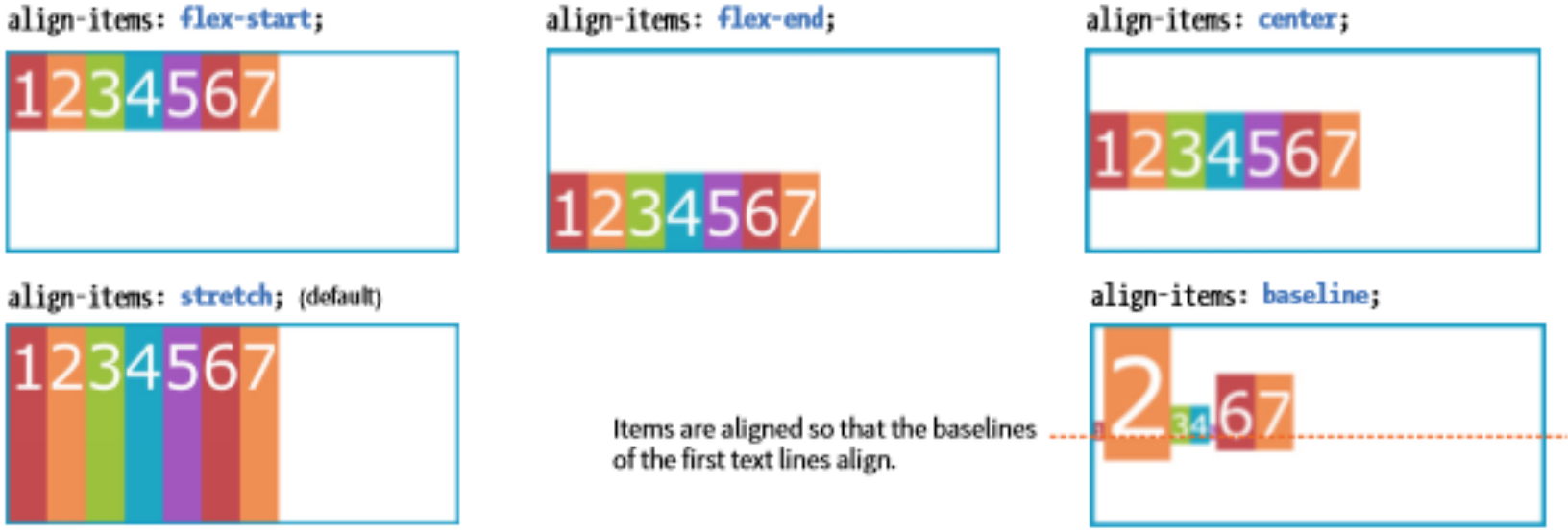
Applies to: flex containers

Inherits: no

Align-items

Like justify-content, the align-items property applies to the flex container (that can be a little confusing because "items" is in the name).

```
#container {
display: flex;
 flex-direction: row;
 height: 200px;
align-items: flex-start;
}
```



Align-self

If you'd like one or more items to override the cross-axis setting, use the align-self property on the individual item element(s). This is the first property we've seen that applies to an item, not the container itself. align-self uses the same values as align-items; it just works on one item at a time.

align-self

Values: flex-start | flex-end | center | baseline | stretch Default: stretch

Applies to: flex items

Inherits: no

```
.box4 {
 align-self: flex-end;
 }
```

Aligning Multiple Lines

The final alignment option, align-content, affects how multiple flex lines are spread out across the cross axis. This property applies only when flex-wrap is set to wrap or wrap-reverse and there are multiple lines to align. If the items are on a single line, it does nothing.

align-content

Values: flex-start | flex-end | center | space-around | space-between | stretch

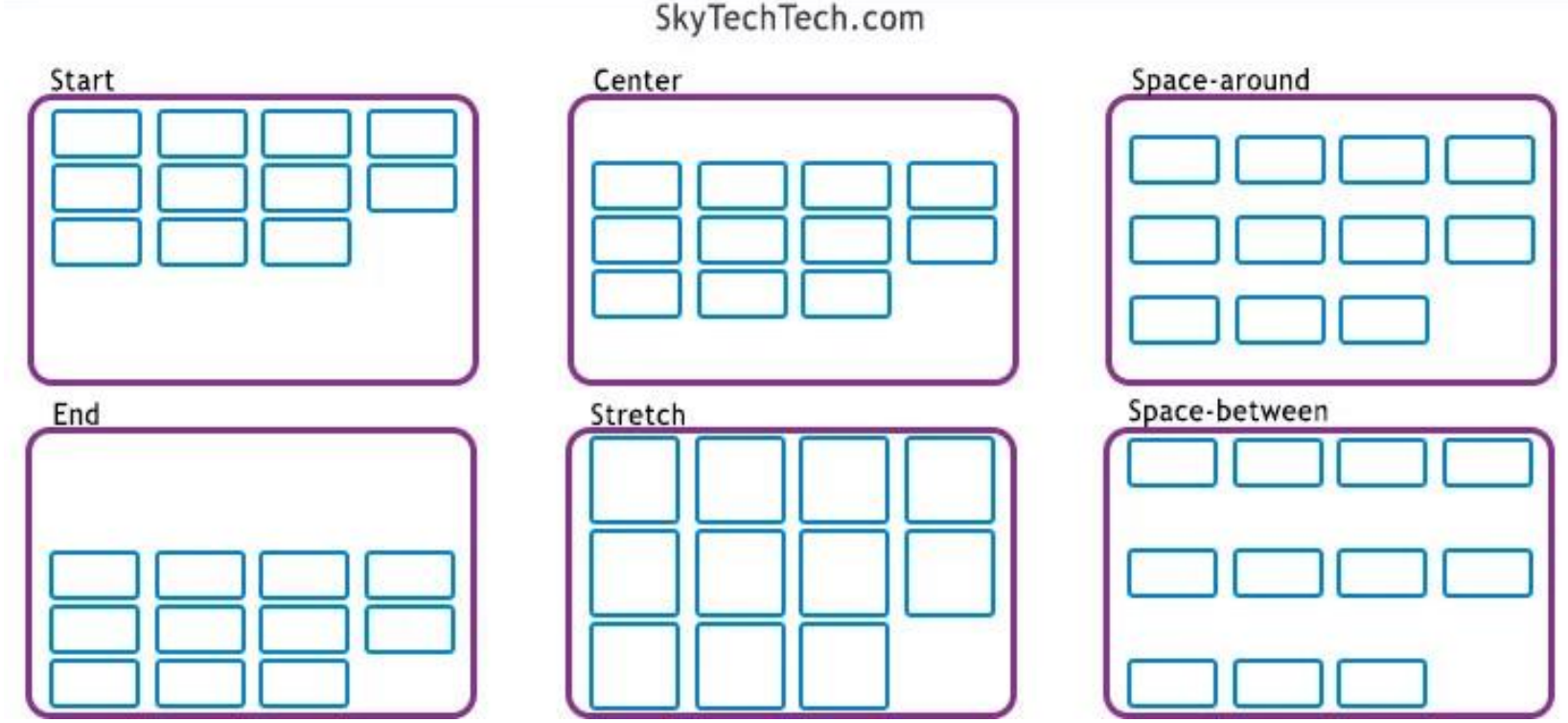
Default: stretch

Applies to: multi-line flex containers

Inherits: no

Example:

```css
.container {
  align-content: flex-start | flex-end | center | space-between
}
```



Example In Align-items

```
<ul>
<li class="logo"><img src="logo.png" alt="LoGoCo"></li>
  <li>About</li>
  <li>Blog</li>
<li>Shop</li>
<li>Contact</li>
 </ul>
ul {
display: flex;
align-items: center;
background-color: #00af8f;
list-style: none; /* removes bullets */
padding: .5em;
margin: 0; }
li {
margin: 0 1em; }
li.logo {
margin-right: auto;
}
```

Determining How Items “Flex” In The Container

One of the great marvels of the flexbox model is that items resize, or flex to use the formal term, to fit the available space. It’s this flexibility that makes Flexbox such a powerful tool for designing for the wide array of screen and browser window sizes we encounter as web designers.

flex

Values: none | 'flex-grow flex-shrink flex-basis’

Default: 0 1 auto

Applies to: flex items

Inherits: no

The value for the flex property is typically three flex properties listed in this order:

flex: flex-grow flex-shrink flex-basis;

Example

In this quick example, a list item starts at 200 pixels wide, is allowed to expand to fill extra space (1), but is not allowed to shrink (0) narrower than the original 200 pixels.

```
 li {
 flex: 1 0 200px;
}
```

Flex-basis

- By default, flex-basis is set to auto, which uses the specified width/height property values for the item size. If the item's main size property (width or height) is not set or is auto (its default), flex-basis uses the content width.
- In this example, the flex basis for the boxes is set to 100 pixels because the auto value uses the value set by width. Items are allowed to grow, taking up any extra space in the container, but they are not allowed to shrink
- box { width: 100px; flex: 1 0 auto; }
- flex: initial; This is the same as flex: 0 1 auto
- flex: auto; This is the same as flex: 1 1 auto
- flex: none; This is equivalent to flex: 0 0 auto
- flex-basis is set to any size other than zero (0), such as a particular width/height value or auto.

Expanding Items (Flex-grow)

The first value in the flex property specifies whether (and in what proportion) an item may stretch larger in other words, its flex-grow value.

flex-grow

Values: number

Default: 0

Applies to: flex items

Inherits: no

Flex-grow

THE MARKUP

```
<div id="container">
  <div class="box box1">1</div>
  <div class="box box2">2</div>
  <div class="box box3">3</div>
  <div class="box box4">4</div>
  <div class="box box5">5</div>
</div>
```

THE STYLES

```
.box {
  …
  flex: 1 1 auto;
}
```

flex: 0 1 auto; (prevents expansion)

flex: 1 1 auto; (allows expansion)

Squishing Items (Flex-shrink)

The second flex property value, flex-shrink, kicks in when the container is not wide enough to contain the items, resulting in a space deficit

flex-shrink

Values: number

Default: 1

Applies to: flex items

Inherits: no

Providing An Initial Size (Flex-basis)

The third flex value defines the starting size of the item before any wrapping, growing, or shrinking occurs (flex-basis).

flex-basis

Values: length | percentage | content | auto

Default: auto

Applies to: flex items

Inherits: no

```
box {
  flex: 0 1 100px;
}
```

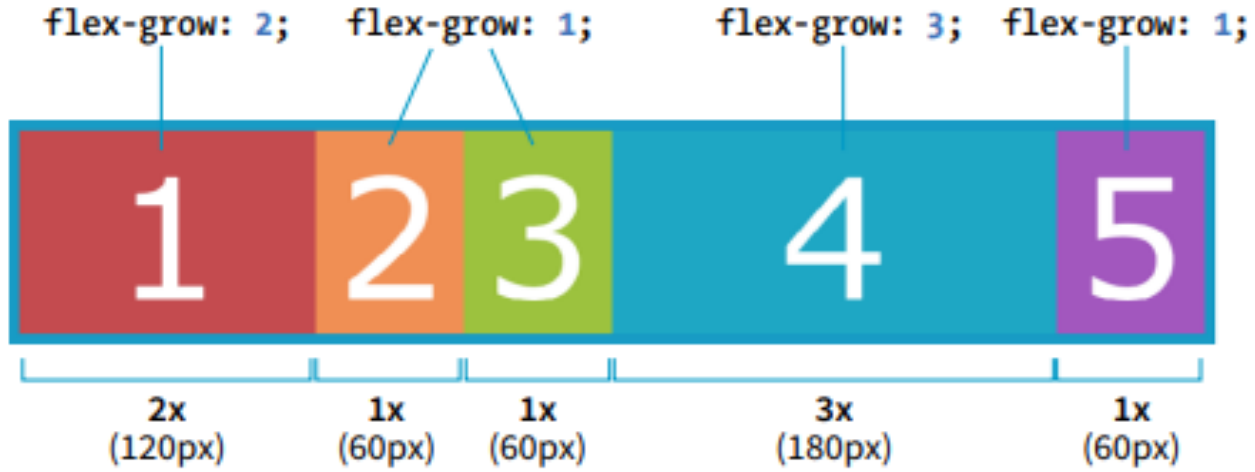
flex: 0 1 100px;

When the container is wide, the items will not grow wider than their flex-basis of 100 pixels because flex-grow is set to 0.

When the container is narrow, the items are allowed to shrink to fit (flex-shrink: 1).

Example in flex

```
.box {
  /* applied to all boxes */
  flex: 1 0 0%;
}
.box1 {
  flex: 2; /* shortcut value for flex: 2 1 0px */
}
.box4 {
  flex: 3; /* shortcut value for flex: 3 1 0px */
}
```



Changing The Order Of Flex Items

One of the killer features of Flexbox is the ability to display items in an order that differs from their order in the source.

To change the order of items, apply the order property to the particular item(s) you wish to move.

order

Values: integer

Default: 0

Applies to: flex items and absolutely positioned children of flex containers

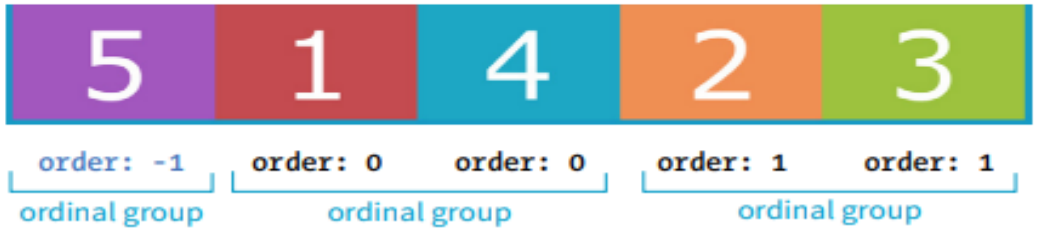
Inherits: no

Order

```
.box2, .box3 {
  order: 1
}
```


```
.box3 {
  order: 1;
}
```


```
.box5 {
  order: -1
}
```



A Simple Document

Here is a simple document with a header, a main section consisting of an article and two aside elements, and

a footer:

```
<header>…</header>
<main>
 <article><h2>Where It's At</h2></article>
 <aside id="news"><h2>News></h2></aside>
 <aside id="contact"><h2>Contact</h2><aside>
</main>
<footer>…<footer>
```

EXAMPLE

```
main {
  display: flex;
}
article {
  flex: 1 1 50%;
  order: 2;
}
#news {
  flex: 1 1 25%;
  order: 3;
}
#contact {
  flex: 1 1 25%;
  order: 1;
}
```