**Flexbox layout:**

The Flexbox Layout (Flexible Box) module (currently a W3C Last Call Working Draft) aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word "flex").

The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space, or shrinks them to prevent overflow.

Most importantly, the flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based). While those work well for pages, they lack flexibility (no pun intended) to support large or complex applications (especially when it comes to orientation changing, resizing, stretching, shrinking, etc.).

**Note:** Flexbox layout is most appropriate to the components of an application, and small-scale layouts, while the [Grid](http://css-tricks.com/snippets/css/complete-guide-grid/) layout is intended for larger scale layouts.

**Complete Guide to Flexbox**

### [Background](https://css-tricks.com/snippets/css/a-guide-to-flexbox/#flexbox-background)

The Flexbox Layout (Flexible Box) module (currently a W3C Last Call Working Draft) aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word "flex").

The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space, or shrinks them to prevent overflow.

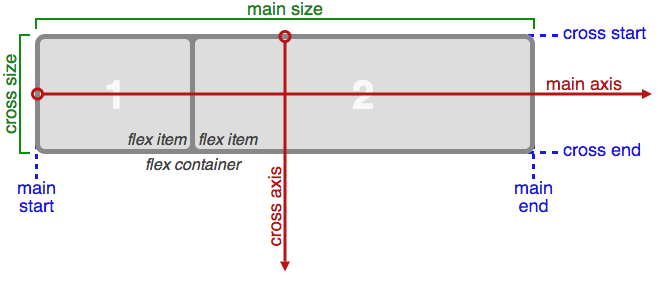
Most importantly, the flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based). While those work well for pages, they lack flexibility (no pun intended) to support large or complex applications (especially when it comes to orientation changing, resizing, stretching, shrinking, etc.).

**Note:** Flexbox layout is most appropriate to the components of an application, and small-scale layouts, while the [Grid](http://css-tricks.com/snippets/css/complete-guide-grid/) layout is intended for larger scale layouts.

### [Basics & Terminology](https://css-tricks.com/snippets/css/a-guide-to-flexbox/#flexbox-basics)

Since flexbox is a whole module and not a single property, it involves a lot of things including its whole set of properties. Some of them are meant to be set on the container (parent element, known as "flex container") whereas the others are meant to be set on the children (said "flex items").

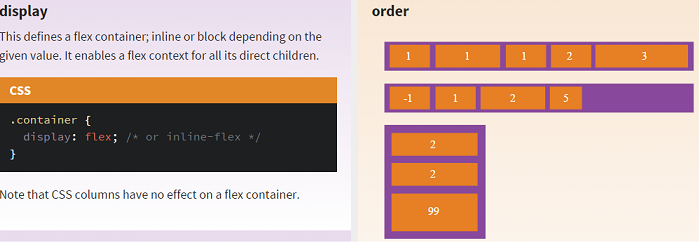
If regular layout is based on both block and inline flow directions, the flex layout is based on "flex-flow directions". Please have a look at this figure from the specification, explaining the main idea behind the flex layout.

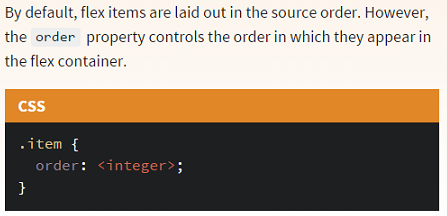


Basically, items will be laid out following either the main axis ( from main-start to main-end) or the cross axis (from cross-start tocross-end).

* **main axis** - The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the flex-direction property (see below).
* **main-start | main-end** - The flex items are placed within the container starting from main-start and going to main-end.
* **main size** - A flex item's width or height, whichever is in the main dimension, is the item's main size. The flex item's main size property is either the ‘width’ or ‘height’ property, whichever is in the main dimension.
* **cross axis** - The axis perpendicular to the main axis is called the cross axis. Its direction depends on the main axis direction.
* **cross-start | cross-end** - Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.
* **cross size** - The width or height of a flex item, whichever is in the cross dimension, is the item's cross size. The cross size property is whichever of ‘width’ or ‘height’ that is in the cross dimension.







When *flex* and *column* both came out into the mainstream many people thought they were the same thing column is used within a single element to create columns whereas flex is used across multiple elements

The **CSS multi-column layout** extends the block layout mode to allow the easy definition of multiple columns of text. People have trouble reading text if lines are too long; if it takes too long for the eyes to move from the end of the one line to the beginning of the next, they lose track of which line they were on. Therefore, to make maximum use of a large screen, authors should have limited-width columns of text placed side by side, just as newspapers do.

Unfortunately this is impossible to do with CSS and HTML without forcing column breaks at fixed positions, or severely restricting the markup allowed in the text, or using heroic scripting. This limitation is solved by adding new CSS properties to extend the traditional block layout mode.

## Using columns

### Column count and width

Two CSS properties control whether and how many columns will appear: [column-count](https://developer.mozilla.org/en-US/docs/Web/CSS/column-count) and[column-width](https://developer.mozilla.org/en-US/docs/Web/CSS/column-width).

The column-count property sets the number of columns to a particular number. E.g.,

<div style="-webkit-column-width:10em;">Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat

nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa

qui officia deserunt mollit anim id est laborum</div>

will display the content in two columns (if you're using a multi-column compliant browser):

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

The column-width property sets the minimum desired column width. If column-count is not also set, then the browser will automatically make as many columns as fit in the available width.

<div style="-webkit-column-width:10em;">Lorem ipsum dolor sit amet, consectetur adipisicing elit,

sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

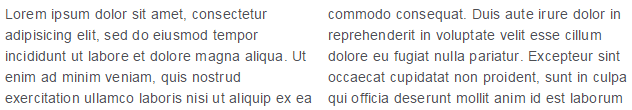
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

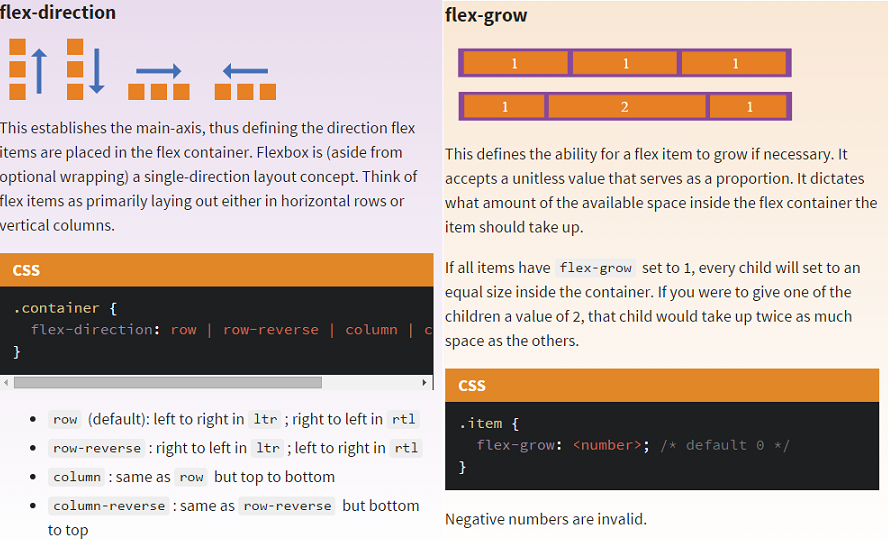
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat

nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa

qui officia deserunt mollit anim id est laborum</div>

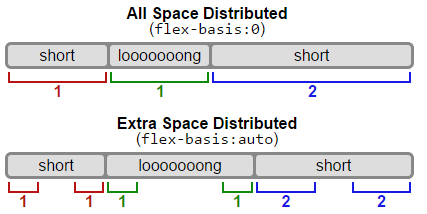
becomes:

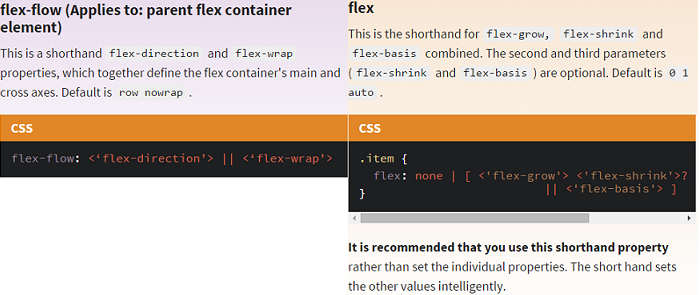




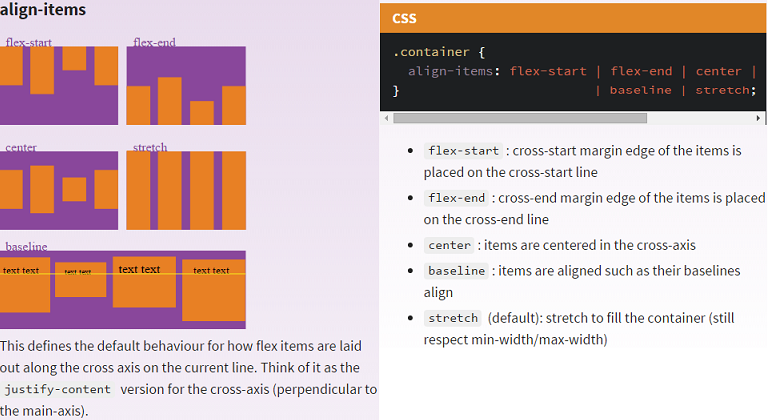


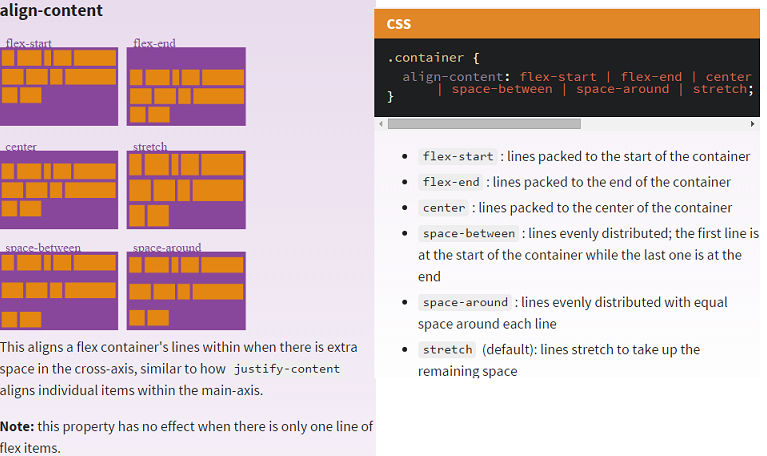










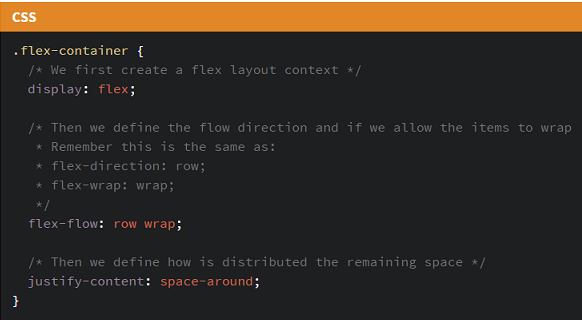


# Q) [What's the difference between align-content and align-items](http://stackoverflow.com/questions/27539262/flexbox-ask-whats-the-difference-between-align-content-and-align-items)?

A) Align-content aligns a flex container's lines within the flex container when there is extra space on the cross-axis. This property has no effect on single line flexible boxes. Align-items aligns flex items of the current flex line the same way as justify-content but in the perpendicular direction. This is the difference between the two.



Now let's use some more properties. Consider a list of 6 items, all with a fixed dimensions in a matter of aesthetics but they could be auto-sized. We want them to be evenly and nicely distributed on the horizontal axis so that when we resize the browser, everything is fine.



Done. Everything else is just some styling concern. Below is a pen featuring this example. Be sure to go to CodePen and try resizing your windows to see what happens.

*You can remove bullets by setting the list-style-type to none on the CSS for the <ul> element, for example*

ul

{

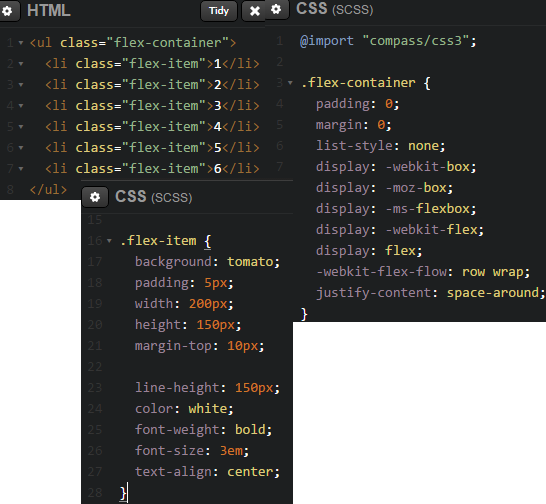
list-style-type: none;

}

*You might also want to add* padding:0; margin:0; *to that, if you want to remove indentation as well.*

*See*[*http://css.maxdesign.com.au/listutorial/index.htm*](http://css.maxdesign.com.au/listutorial/index.htm) *,for a great walkthrough of list formatting techniques.*

Example:



Output:



Let's try something else. Imagine we have a right-aligned navigation on the very top of our website, but we want it to be centered on medium-sized screens and single-columned on small devices. Easy enough.