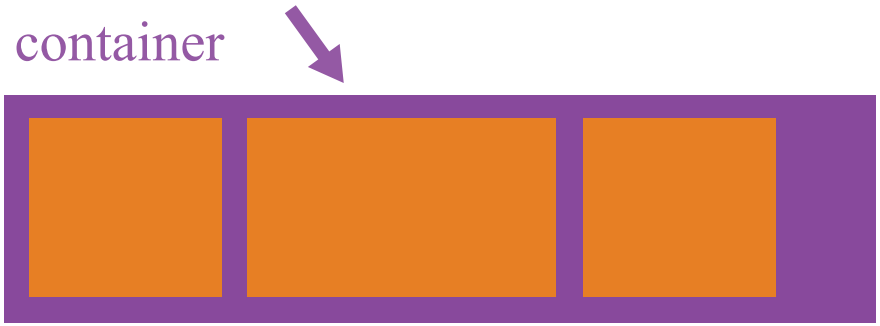
A Complete Guide to Flexbox

BY **CHRIS COYIER** LAST UPDATED ON NOVEMBER 12, 2017

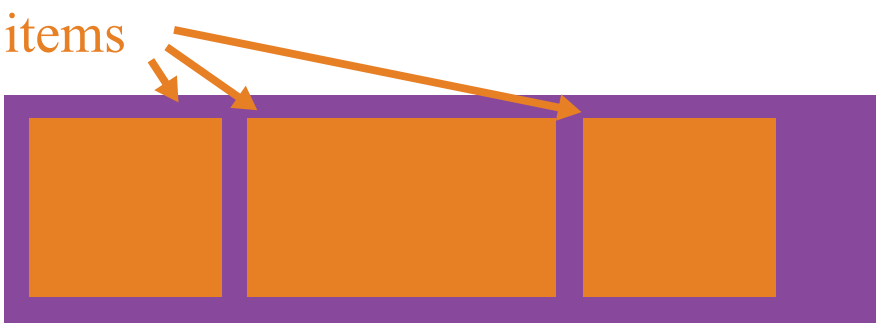
FLEXBOX, **LAYOUT**

- **Background**
- **Basics & Terminology**



Properties for the Parent

(flex container)



Properties for the Children

(flex items)

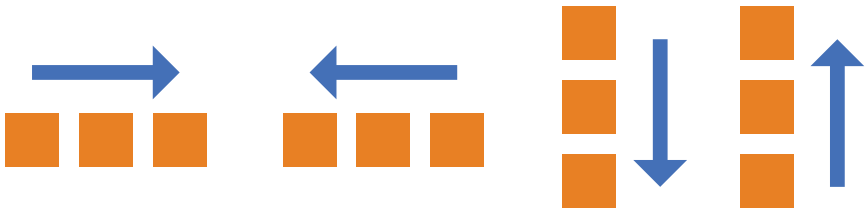
display

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

```css
.container {
  display: flex; /* or inline-flex */
}
```

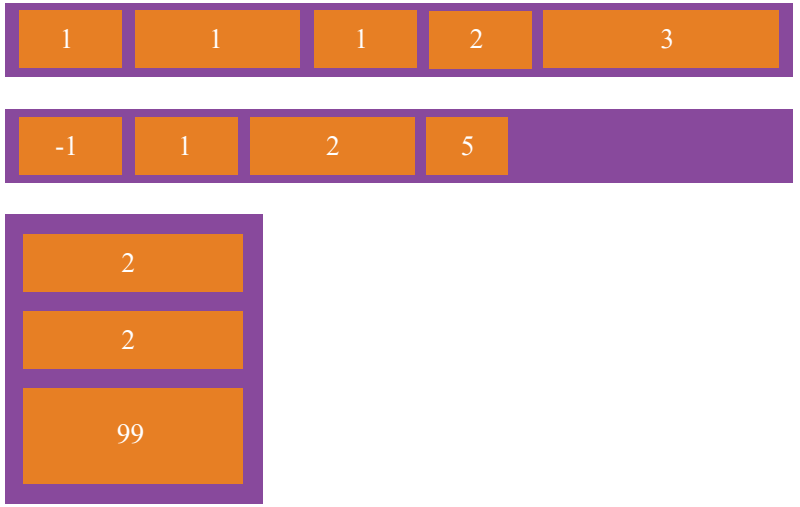
Note that CSS columns have no effect on a flex container.

flex-direction



This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

order



By default, flex items are laid out in the source order. However, the `order` property controls the order in which they appear in the flex container.

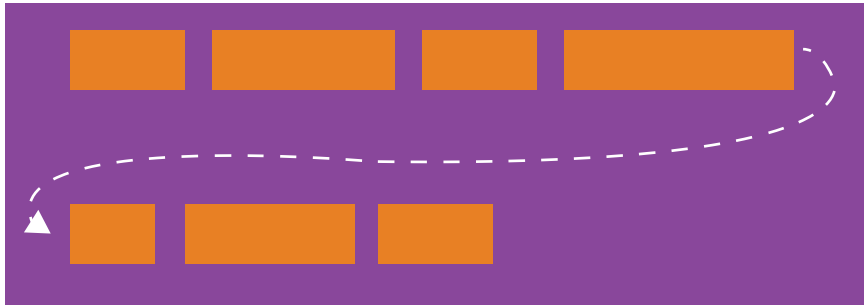
```css
.item {
  order: <integer>; /* default is 0 */
}
```

flex-grow

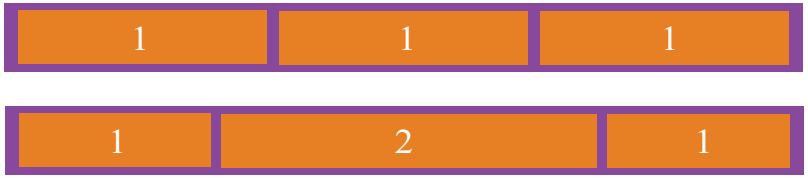
```css
.container {
  flex-direction: row | row-reverse | col
}
```

- `row` (default): left to right in `ltr` ; right to left in `rtl`
- `row-reverse` : right to left in `ltr` ; left to right in `rtl`
- `column` : same as `row` but top to bottom
- `column-reverse` : same as `row-reverse` but bottom to top

flex-wrap



By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property.



This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

If all items have `flex-grow` set to 1, the remaining space in the container will be distributed equally to all children. If one of the children has a value of 2, the remaining space would take up twice as much space as the others (or it will try to, at least).

```css
.item {
  flex-grow: <number>; /* default 0 */
}
```

Negative numbers are invalid.

flex-shrink

This defines the ability for a flex item to shrink if necessary.

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

- `nowrap` (default): all flex items will be on one line
- `wrap` : flex items will wrap onto multiple lines, from top to bottom.
- `wrap-reverse` : flex items will wrap onto multiple lines from bottom to top.

There are some visual demos of `flex-wrap` here.

flex-flow (Applies to: parent flex container element)

This is a shorthand `flex-direction` and `flex-wrap` properties, which together define the flex container's main and cross axes. Default is `row nowrap` .

```css
flex-flow: <‘flex-direction’> || <‘flex-w
```

justify-content

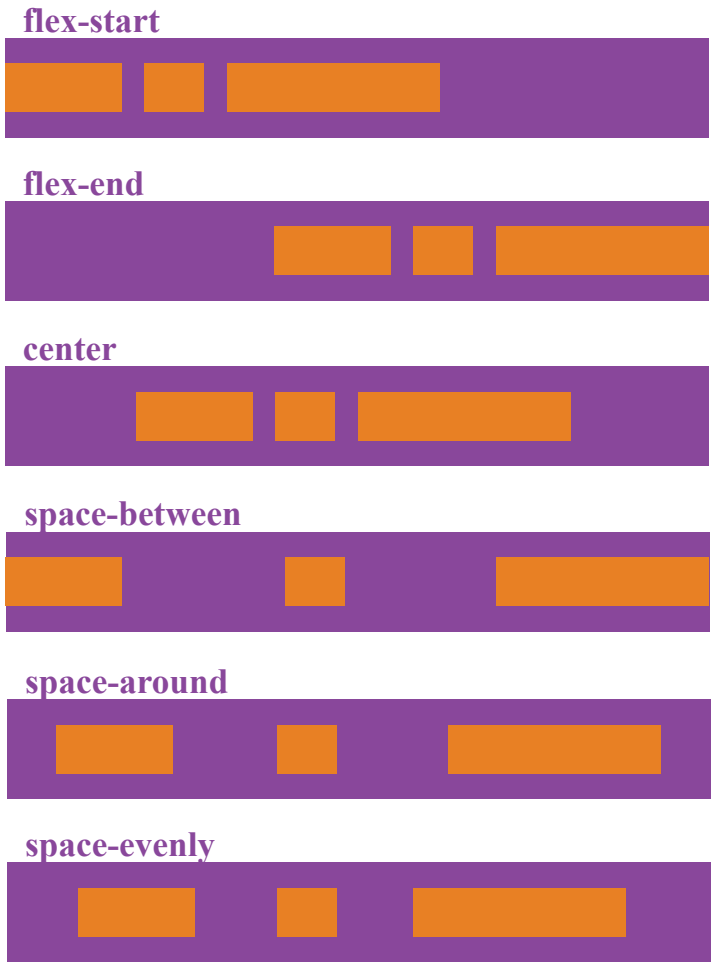
```css
.item {
  flex-shrink: <number>; /* default 1 */
}
```

Negative numbers are invalid.

flex-basis

This defines the default size of an element before the remaining space is distributed. It can be a length (e.g. 20%, 5rem, etc.) or a keyword. The `auto` keyword means "look at my width or height property" (which was temporarily done by the `main-size` keyword until deprecated). The `content` keyword means "size it based on the item's content" - this keyword isn't well supported yet, so it's hard to test and harder to know what its brethren `max-content` , `min-content` , and `fit-content` do.

```css
.item {
  flex-basis: <length> | auto; /* default
}
```



This defines the alignment along the main axis. It helps distribute extra free space left over when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

If set to `0`, the extra space around content isn't factored in. If set to `auto`, the extra space is distributed based on its `flex-grow` value. See this graphic.

flex

This is the shorthand for `flex-grow`, `flex-shrink` and `flex-basis` combined. The second and third parameters (`flex-shrink` and `flex-basis`) are optional. Default is `0 1 auto`.

```
css

.item {
  flex: none | [ <'flex-grow'> <'flex-shr
}
```

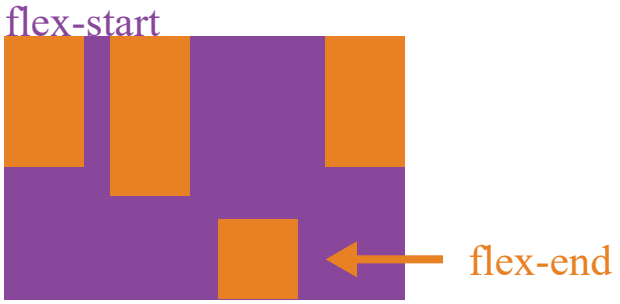
It is recommended that you use this shorthand property rather than set the individual properties. The short hand sets the other values intelligently.

align-self

```css
.container {
  justify-content: flex-start | flex-end
}
```

- `flex-start` (default): items are packed toward the start line
- `flex-end` : items are packed toward to end line
- `center` : items are centered along the line
- `space-between` : items are evenly distributed in the line; first item is on the start line, last item on the end line
- `space-around` : items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.
- `space-evenly` : items are distributed so that the spacing between any two items (and the space to the edges) is equal.

align-items

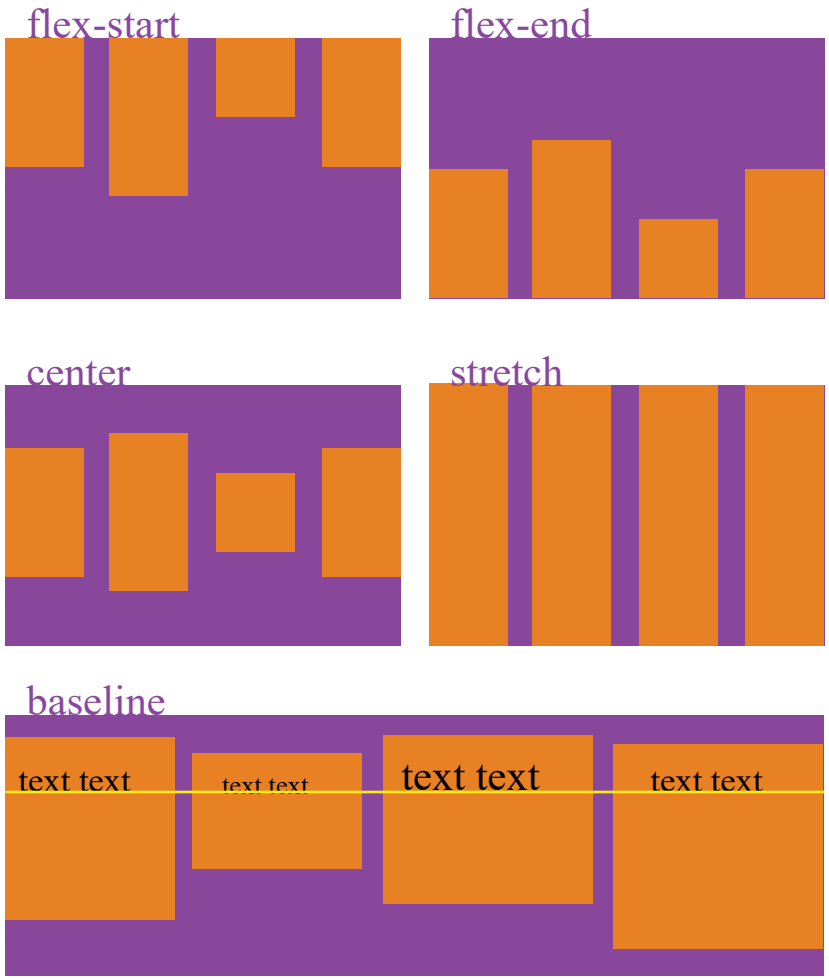


This allows the default alignment (or the one specified by `align-items`) to be overridden for individual flex items.

Please see the `align-items` explanation to understand the available values.

```css
.item {
  align-self: auto | flex-start | flex-en
}
```

Note that `float`, `clear` and `vertical-align` have no effect on a flex item.

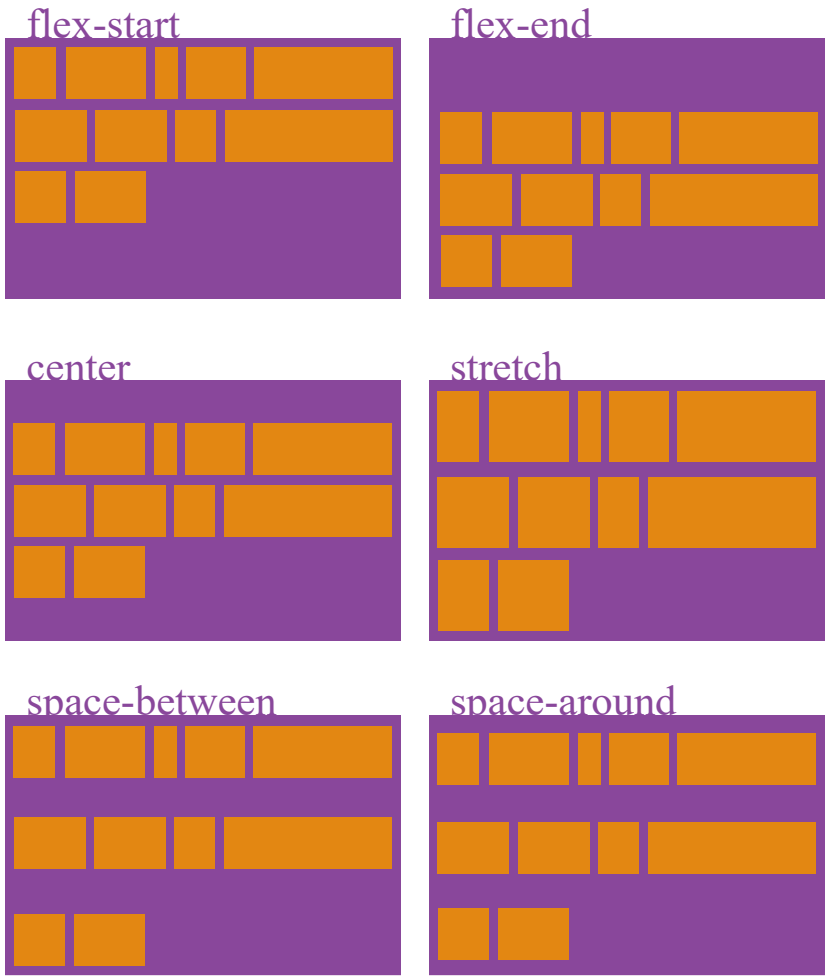


This defines the default behaviour for how flex items are laid out along the cross axis on the current line. Think of it as the `justify-content` version for the cross-axis (perpendicular to the main-axis).

```css
.container {
  align-items: flex-start | flex-end | ce
}
```

- `flex-start` : cross-start margin edge of the items is placed on the cross-start line
- `flex-end` : cross-end margin edge of the items is placed on the cross-end line
- `center` : items are centered in the cross-axis
- `baseline` : items are aligned such as their baselines align
- `stretch` (default): stretch to fill the container (still respect min-width/max-width)

align-content



This aligns a flex container's lines within when there is extra space in the cross-axis, similar to how `justify-content` aligns individual items within the main-axis.

Note: this property has no effect when there is only one line of flex items.

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

- `flex-start` : lines packed to the start of the container
- `flex-end` : lines packed to the end of the container
- `center` : lines packed to the center of the container
- `space-between` : lines evenly distributed; the first line is at the start of the container while the last one is at the end
- `space-around` : lines evenly distributed with equal space around each line
- `stretch` (default): lines stretch to take up the remaining space

Examples

Let's start with a very very simple example, solving an almost daily problem: perfect centering. It couldn't be any simpler if you use flexbox.

```css
.parent {
  display: flex;
  height: 300px; /* Or whatever */
```

```
}

.child {
  width: 100px;  /* Or whatever */
  height: 100px; /* Or whatever */
  margin: auto;  /* Magic! */
}
```

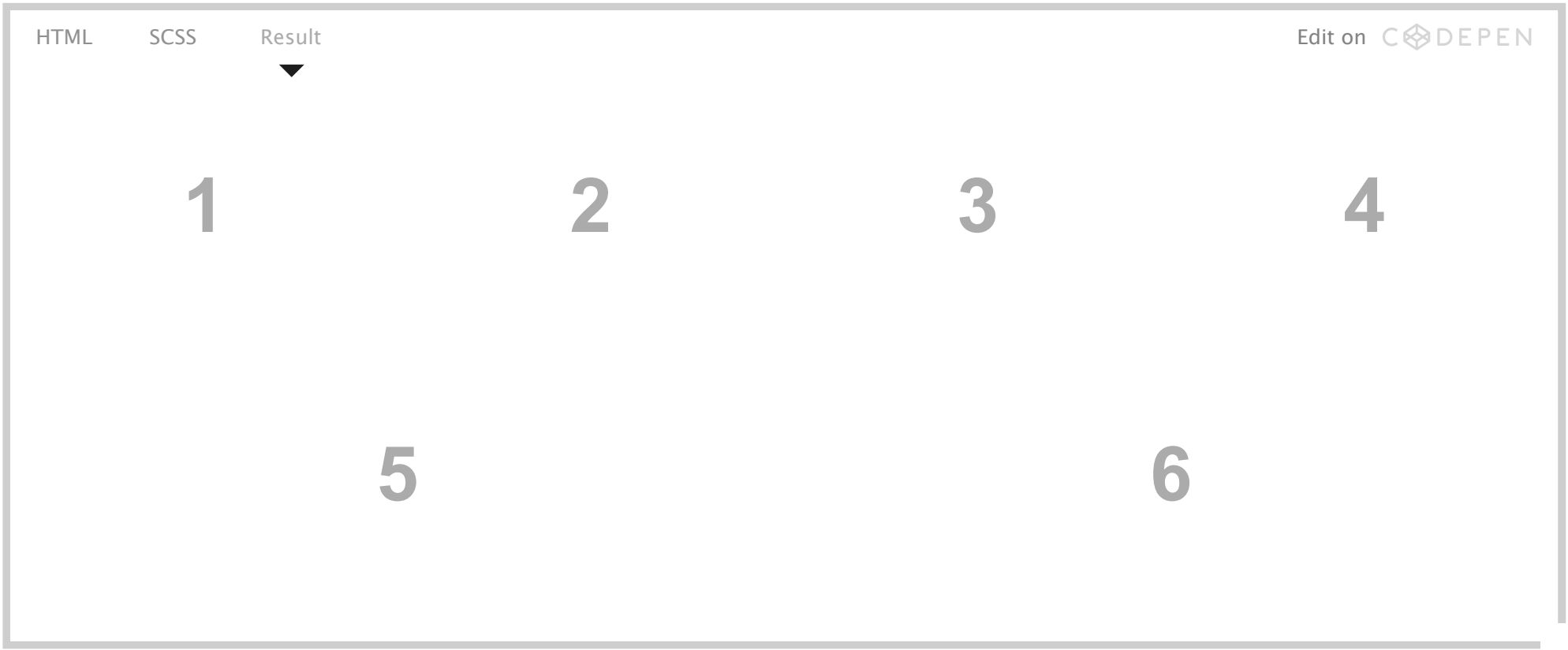
This relies on the fact a margin set to `auto` in a flex container absorb extra space. So setting a vertical margin of `auto` will make the item perfectly centered in both axis.

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 (without media queries!).

```
css

.flex-container {
  /* We first create a flex layout context */
  display: flex;

  /* Then we define the flow direction
     and if we allow the items to wrap
   * Remember this is the same as:
   * flex-direction: row;
   * flex-wrap: wrap;
   */
```

```
  flex-flow: row wrap;

  /* Then we define how is distributed the remaining space */
  justify-content: space-around;
}
```

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.



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.

```css
/* Large */
.navigation {
  display: flex;
  flex-flow: row wrap;
  /* This aligns items to the end line on main-axis */
  justify-content: flex-end;
}

/* Medium screens */
@media all and (max-width: 800px) {
  .navigation {
    /* When on medium sized screens, we center it by evenly distributing empty space around items
    justify-content: space-around;
  }
}

/* Small screens */
@media all and (max-width: 500px) {
  .navigation {
    /* On small screens, we are no longer using row direction but column */
    flex-direction: column;
```

```
    }
}
```

Let's try something even better by playing with flex items flexibility! What about a mobile-first 3-columns layout with full-width header and footer. And independent from source order.

css

```
.wrapper {
  display: flex;
  flex-flow: row wrap;
}

/* We tell all items to be 100% width, via flex-basis */
.wrapper > * {
```

```
  flex: 1 100%;
}

/* We rely on source order for mobile-first approach
 * in this case:
 * 1. header
 * 2. article
 * 3. aside 1
 * 4. aside 2
 * 5. footer
 */

/* Medium screens */
@media all and (min-width: 600px) {
  /* We tell both sidebars to share a row */
  .aside { flex: 1 auto; }
}

/* Large screens */
@media all and (min-width: 800px) {
  /* We invert order of first sidebar and main
   * And tell the main element to take twice as much width as the other two sidebars
   */
  .main { flex: 2 0px; }
  .aside-1 { order: 1; }
```

```
    .main    { order: 2; }
    .aside-2 { order: 3; }
    .footer  { order: 4; }
}
```

HTML CSS Result EDIT ON CODEPEN

Header

Aside 1

Aside 2

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Footer

▶ Prefixing Flexbox

▶ Related Properties

▶ Other Resources

▶ Bugs

Browser Support

Broken up by "version" of flexbox:

- (new) means the recent syntax from the specification (e.g. `display: flex;`)
- (tweener) means an odd unofficial syntax from 2011 (e.g. `display: flexbox;`)
- (old) means the old syntax from 2009 (e.g. `display: box;`)

Chrome: 20- (old) 21+ (new)	Safari: 3.1+ (old) 6.1+ (new)	Firefox: 2-21 (old) 22+ (new)	Opera: 12.1+ (new)	IE: 10 (tweener) 11+ (new)	Android: 2.1+ (old) 4.4+ (new)	iOS: 3.2+ (old) 7.1+ (new)

Blackberry browser 10+ supports the new syntax.

For more informations about how to mix syntaxes in order to get the best browser support, please refer to this article (CSS-Tricks) or this article (DevOpera).

Comments