

Flexbox

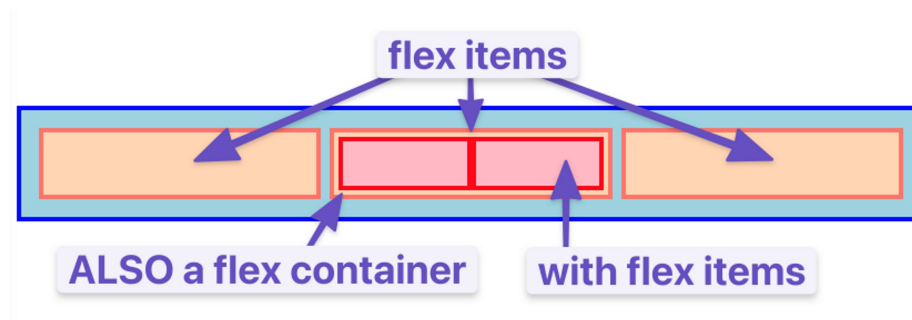
Flexbox is a way to arrange items into rows or columns. These items will flex (i.e. grow or shrink) based on some simple rules that you can define.

Flex Containers and Flex Items

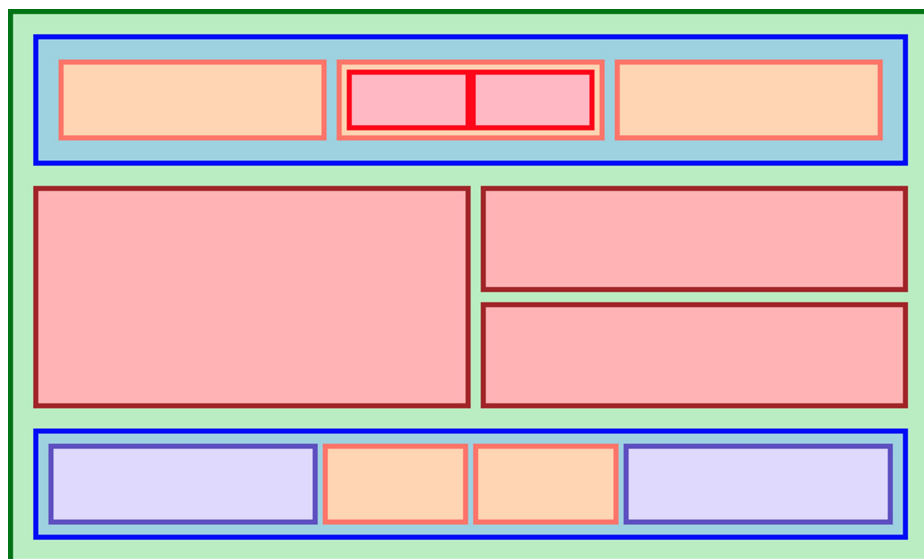
As you've seen, flexbox is not just a single CSS property but a whole toolbox of properties that you can use to put things where you need them. Some of these properties belong on the flex container, while some go on

the flex items. This is a simple yet important concept.

A flex container is any element that has `display: flex` on it. A flex item is any element that lives directly inside of a flex container.



Creating and nesting multiple flex containers and items is the primary way we will be building up complex layouts. The following image was achieved using only flexbox to arrange, size, and place the various elements. Flexbox is a very powerful tool.



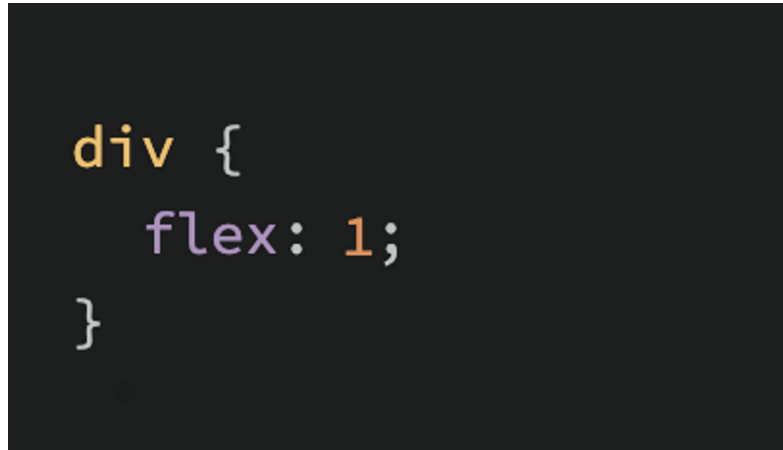
Growing And Shrinking

[The Flex Shorthand](#)

The flex declaration is actually a shorthand for 3 properties that you can set on a flex item. These properties affect how flex items size themselves within their container.

Shorthand properties are CSS properties that let you set the values of multiple other CSS properties simultaneously. Using a shorthand property, you can write more concise (and often more readable) stylesheets, saving time and energy.

In this case, flex is actually a shorthand for flex-grow, flex-shrink and flex-basis.



```
div {  
  flex: 1;  
}
```

In the above screenshot, flex: 1 equates to: flex-grow: 1, flex-shrink: 1, flex-basis: 0.

Flex-Grow

flex-grow expects a single number as its value, and that number is used as the flex-item's "growth factor".

When we applied flex: 1 to every div inside our container, we were telling every div to grow the same amount. The result of this is that every div ends up the exact same size. If we instead add flex: 2 to just one of the divs, then that div would grow to 2x the size of the others.

Flex-Shrink

flex-shrink is similar to flex-grow, but sets the "shrink factor" of a flex item. flex-shrink only ends up being applied if the size of all flex items is larger than their parent container. For example, if our 3 divs from above had a width declaration like: width: 100px, and .flex-container was smaller than 300px, our divs would have to shrink to fit.

The default shrink factor is flex-shrink: 1, which means all items will shrink evenly. If you do *not* want an item to shrink then you can specify flex-shrink: 0;. You can also specify higher numbers to make certain items shrink at a higher rate than normal.

An important implication to notice here is that when you specify flex-grow or flex-shrink, flex items do not necessarily respect your given values for width. In the above example, all 3 divs are given a width of 250px, but when their parent is big enough, they grow to fill it. Likewise, when the parent is too small, the default behavior is for them to shrink to fit. This is not a bug, but it could be confusing behavior if you aren't expecting it.

Flex-Basis

flex-basis simply sets the initial size of a flex item, so any sort of flex-growing or flex-shrinking starts from that baseline size. The shorthand value defaults to flex-basis: 0%. The reason we had to change it to auto in the flex-shrink example is that with the basis set to 0, those items would ignore the item's width, and everything would shrink evenly. Using auto as a flex-basis tells the item to check for a width declaration

Important Note About Flex-Basis:

There is a difference between the default value of flex-basis and the way the flex shorthand defines it if no flex-basis is given. The actual default value for flex-basis is auto, but when you specify flex: 1 on an element, it interprets that as flex: 1 1 0. If you want to only adjust an item's flex-grow you can simply do so directly, without the

shorthand. Or you can be more verbose and use the full 3 value shorthand `flex: 1 1 auto`, which is also equivalent to using `flex: auto`.

In practice you will likely not be using complex values for `flex-grow`, `flex-shrink` or `flex-basis`. Generally, you're most likely to use declarations like `flex: 1`; to make divs grow evenly and `flex-shrink: 0` to keep certain divs from shrinking.

The list below summarizes the effects of the four [flex](#) values that represent most commonly-desired effects:

[flex: initial](#)

Equivalent to [flex: 0 1 auto](#). (This is the initial value.) Sizes the item based on the [width/height](#) properties

[flex: auto](#)

Equivalent to [flex: 1 1 auto](#). Sizes the item based on the [width/height](#) properties, but makes them fully flexible, so that they absorb any free space along the [main axis](#). If all items are either [flex: auto](#), [flex: initial](#), or [flex: none](#), any positive free space after the items have been sized will be distributed evenly to the items with [flex: auto](#).

[flex: none](#)

Equivalent to [flex: 0 0 auto](#). This value sizes the item according to the [width/height](#) properties, but makes the flex item [fully inflexible](#). This is similar to [initial](#), except that flex items are not allowed to shrink, even in overflow situations.

[flex: <positive-number>](#)

Equivalent to [flex: <positive-number> 1 0](#). Makes the flex item flexible and sets the [flex basis](#) to zero, resulting in an item that receives the specified proportion of the free space in the flex container. If all items in the flex container use this pattern, their sizes will be proportional to the specified flex factor. By default, flex items won't shrink below their minimum content size (the length of the longest word or fixed-size element). To change this, set the [min-width](#) or [min-height](#) property. (See [§4.5 Automatic Minimum Size of Flex Items](#).)

Axes

The most confusing thing about flexbox is that it can work either horizontally or vertically, and the way some rules work changes a bit depending on which direction you are working with.

The default direction for a flex container is horizontal, or `row`, but you can change the direction to vertical, or `column`.

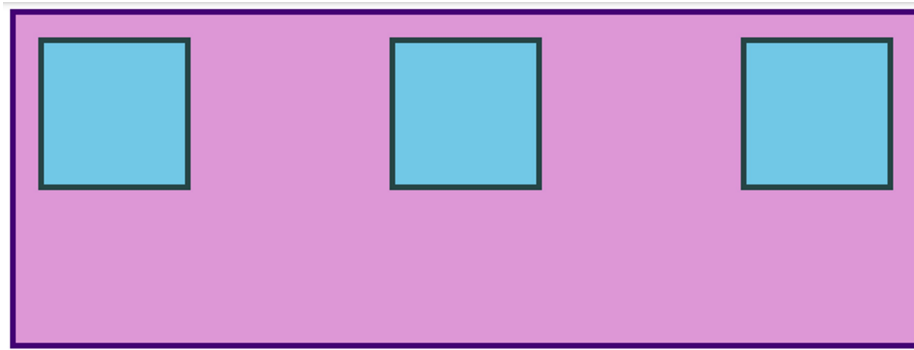
The direction can be specified in CSS like so:

```
.flex-container {  
  flex-direction: column;  
}
```

There are situations where the behavior of `flex-direction` could change if you are using a language that is written top-to-bottom or right-to-left, but you should save worrying about that until you are ready to start making a website in Arabic or Hebrew.

[Alignment](#)

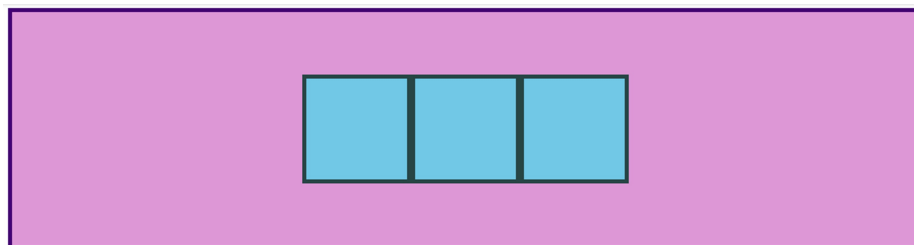
Remove `flex: 1` from `.item` and add `justify-content: space-between` to `.container`. Doing so should give you something like this:



justify-content aligns items across the main axis.

To change the placement of items along the cross-axis use align-items

Try getting the boxes to the center of the container by adding align-items: center to .container. The desired result looks like this:



Because justify-content and align-items are based on the main and cross axis of your container, their behavior changes when you change the flex-direction of a flex-container. For example, when you change flex-direction to column, justify-content aligns vertically and align-items aligns horizontally. The most common behavior, however, is the default, i.e. justify-content aligns items horizontally (because the main axis defaults to horizontal), and align-items aligns them vertically.

From <<https://www.theodinproject.com/lessons/foundations-alignment>>

Gap

One more very useful feature of flex is the gap property. Setting gap on a flex container simply adds a specified space between flex items, very similar to adding a margin to the items themselves. gap is a **new** property so it doesn't show up in many resources yet, but it works reliably in all modern browsers, so it is safe to use and very handy! Adding gap: 8px to the centered example above produces the result below.

Unlike justify-content and align-items, align-self is applied to the child element, not the container. It allows us to change the alignment of a specific child along the cross axis:

