Flexbox Basics

Flexbox is a simple way of building a layout in CSS and is widely supported.

Browser Support

You can check browser version support for various HTML elements and CSS properties on the following site:

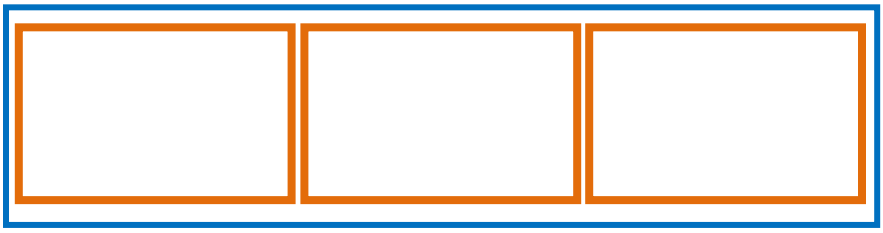
http://caniuse.com

The above site may not always have every feature's browser support but it does contain a lot of them, which can be helpful when you are coding a webpage.

In order to understand flexbox, you need to understand the terminology. There are two types of boxes for flexbox:

1. Flex container
2. Flex item

A **flex container** is the **wrapper** container. It holds the **flex items**. There are specific properties which should only be set on a flex container and some which should only be set on a flex item. This means that **you cannot use flex item properties on a flex container**. Doing this may result in unexpected behaviour.



For flexbox, most of the properties go on the flex container. These properties allow you to **set how the children (flex items) will display and behave**. The properties for the flex items are mostly for just changing the order of item display and sizing.

Flex container properties

display: flex

This is what makes an element a flex container.

flex-direction: row | column | row-reverse | column-reverse

This sets the direction in which flex items within the container will flow. By default, this will be set to

row. Flexbox is one-directional, so a single flex container can only display in the row direction or column direction.

flex-wrap: wrap | nowrap

This defines whether or not flex items can break (wrap) to a new line if an element can't fit in a row (for example).

flex-flow: <flex-direction> <flex-wrap>

This is the shorthand property which allows you to set both flex-direction and flex-wrap.

justify-content

This property adjusts alignment along the main axis (for a row, the main axis is a horizontal line; for a column, the main axis is a vertical line).

Possible values are:

- **flex-start**
  All flex items will bunch at the start of the container (unless you used flex-grow on a flex item).
- **flex-end**
  All flex items will bunch at the end of the container (unless you used flex-grow on a flex item).
- **center**
  All flex items will bunch at the center (unless flex-grow used on a flex item).
- **space-between**
  This will evenly spread out flex items so that they touch the ends of the flex container.
- **space-evenly**
  This will break up the excess whitespace around flex items.
- **space-around**
  This is similar to **space-evenly** except the space at before and after the first and last flex items are half the space between the middle flex items.

align-items

This property adjusts alignment along the cross axis (axis perpendicular to the main axis). For a row, the cross axis is a vertical line. For a column, the cross axis is a vertical line.

There are a few possible values:

- **flex-start**
  For a row, align flex items to the top edge of the flex container. For a column, align flex items to the left edge of the flex container.
- **flex-end**
  For a row, align flex items to the bottom edge of the flex container. For a column, align flex items to the right edge of the flex container.
- **center**
  For a row, vertically center flex items against each other. For a column, horizontally center flex items against each other.
- **stretch**
  For a row, make flex items stretch to vertically fill the flex container. For a column, make flex items stretch to *horizontally* fill the flex container.

- **baseline**
  Align flex items to the bottom lines of their content.

gap, row-gap, column-gap

gap sets the space between all flex items within the flex container. If flex items break to a new line (flex-direction: row), the gap will appear between columns and rows of flex items within the same flex container.

row-gap sets the space between rows of flex items within the same flex container.

column-gap sets the space between columns of flex items within the same flex container.

Flex item properties

order

By default this is set to 0. When everything is set to the same order number, the flex items will be displayed in the order they appear in the semantic code. If you just want to push one flex item to the end, you only need to target the one flex item to have **order: 1** (because 1 is greater than 0, the default).

Avoid setting an order number on everything because this ends up making it more difficult to tweak the order later.

flex-grow

If this is set to a number other than 0, it will allow the flex item to grow to fill any extra whitespace. If flex items all have a flex-grow of 1, the flex items will grow evenly to fill the flex container.

flex-shrink

Setting this to something to be allowed to shrink if the flex container gets too small.

flex-basis: 0 | auto | <length>

Flex-basis is used to set the default size of a flex item (before any growing/shrinking).

Setting flex-basis to a length (e.g. 30%) will allow you to set a width. Note that when using flexbox, I recommend sticking with flex-basis if you want to set a specific width as it behaves more predictably than using width (using width with flex-grow, for example, can lead to unexpected behaviour where the width is ignored).

When using flex-grow to auto-calculate width, it's useful to set flex-basis to 0. For example, for a row of flex items, if all flex items have a flex-grow of 1, using flex-basis: 0 will evenly set the widths of the flex items (by splitting the space in the flex container by the number of flex items minus any gaps). flex-basis:0 means to consider the entire space of the flex-container when calculating flex item sizes.

flex-basis: auto only considers extra whitespace (outside of the flex item content) when calculating flex item sizes.

Try the following:

```
<div class="flex-container">
  <div class="flex-item">Test</div>
  <div class="flex-item">More content</div>
  <div class="flex-item">Even more content</div>
</div>
```

Now let's try out some CSS:

```
.flex-container {
  display: flex;
  gap: 10px;
  border: 1px solid #000; /* black border for flex container */
  padding: 15px;
}
.flex-item {
  border: 1px solid red;
  /* try out each line separately, commenting out the ones that you
aren't testing */
  flex-grow: 1;
  flex-basis: 0;
  flex-basis: auto;
  flex-basis: 300px;
}
```

flex: <flex-grow> <flex-shrink> <flex-basis>

This is the shorthand property to set flex-grow, flex-shrink and flex-basis all in one go. For older browsers, this shorthand property sometimes has more consistent behaviour than when setting each property separately.

An example of a simple 3-column layout:

```
<header id="header">Header</header>
<div class="flex-container">
  <main id="main">Main stuff</main>
  <aside id="left-sidebar" class="sidebar">Left sidebar</aside>
  <aside id="right-sidebar" class="sidebar">Right sidebar</aside>
</div>
<footer id="footer">Footer</footer>
```

To make the middle box twice as wide as the sidebars, you can use the following CSS which uses flex-grow (for the above HTML):

```
.flex-container {
  display: flex;
  flex-flow: row wrap;
```

```
}
.sidebar {
  flex: 1 0 0; /* if flex-shrink and flex-basis are 0, the width will be
calculated using the flex-grow to determine the proportions of how the size
will be split up */
}
#main {
  flex: 2 0 0;
}
```

Important!

If you are using flexbox to build your responsive/"stretchy" container layout, I recommend that you **avoid using the width property** to set the widths of those same containers. If using flex-grow with width, it may not behave in ways you'd expect if you're new to flexbox.

If you put a border around each item, you will notice that there is no space in between items. To add spacing, you can simply add a gap. You can add gaps between flex items by **setting on the flex container**:

```
gap: 1em; /* set a 1em gap between rows and a 1em gap between columns */
```

This is a shortcut of:

```
row-gap: 1em;
column-gap: 1em;
```

You can also set an order for flex items using the `order` property. This allows you to reorder items if need be.

To move the first sidebar left, simply use the following CSS:

```
#left-sidebar {
  order: -1;
}
```

As the other containers don't have an explicit order set, setting the order of the left sidebar to -1 will set it as the first flex item in the flex container.

You can add a wrapper to restrict the overall page width. Typically, 1200px wide for desktop is a good choice for ease of reading (for a common 1366px wide screen). For very large screens, it's not always comfortable to read if the content goes from edge to edge though for 1080p resolution desktop screens you can go wider but I would make sure to have multiple columns for sure at that point.

You can wrap the layout containers with a wrapper div:

```
<div class="page">
  <header id="header">
  ...
```

```
  </footer>
</div>
```

Add the following CSS rule to make the page div **at most** 1200px across. Notice that we set the **`max-width`** to `1200px` and not the `width`. This is so that the width can shrink for smaller screen sizes but will not grow beyond 1200px wide.

```
.page {
  max-width: 1200px;
}
```

Now that the page width is set, make sure to center the page content to the browser window. Add the following declaration to your `.page` rule (i.e. the declaration block):

```
margin: 0 auto;
```

The above line will horizontally center within a wider container any **block element** with a width narrower than 100%. It does this by setting auto margins to the left and right (i.e. auto-calculate left and right margins).

You can also add padding (space within the border edge) to each container:

```
#header, #main, #footer, .sidebar {
  padding: 1em;
}
```

Box-sizing

At times you may notice as you style that the width changes when adding borders and/or padding (if using the width property). This is because the standard box-model does not include border width or padding width in the width and height calculations. This can be fixed using the following CSS (put it near the top of the file):

```
html {
  box-sizing: border-box;
}
*, *:before, *:after {
  box-sizing: inherit;
}
```

```
box-sizing: [border-box/content-box];
```

To use the non-standard box model, set the `box-sizing` property to **border-box**. This is also a very useful mode when implementing responsive design because it is easier to know what the final width of the element is since it includes padding and borders. Under the standard box model, only the content width is counted so any padding or borders expand the overall width (which is why in the example, the right sidebar went on the next line).

Use the value **content-box** to use the standard box model.

Older versions of Safari (under 5.1), Chrome (under 10), and Firefox (under 29) will require you to use

vendor prefixes:

```
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
```

Newer browser versions support the `box-sizing` property so there is less need to use the -webkit and -moz prefixes now as opposed to a few years ago. (These prefixes are hacks for each browser vendor.)

Note that when you use vendor prefixes, you should always put the vendor-prefixed values first and the unprefixed value last.

Read more:

https://css-tricks.com/box-sizing/

You can also add spacing after the header and before the footer to space out the containers a little by adding a top and bottom margin to the .flex-container.

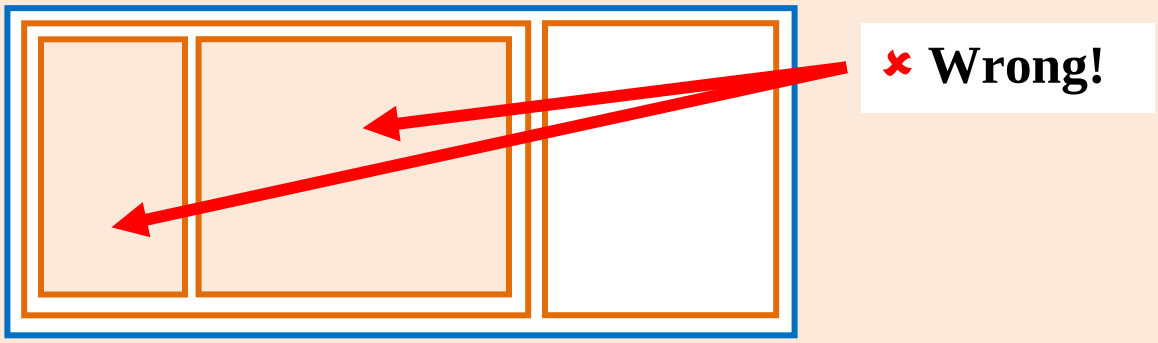
```
.flex-container {
  display: flex;
  flex-flow: row wrap;
  margin:1em 0;
}
```

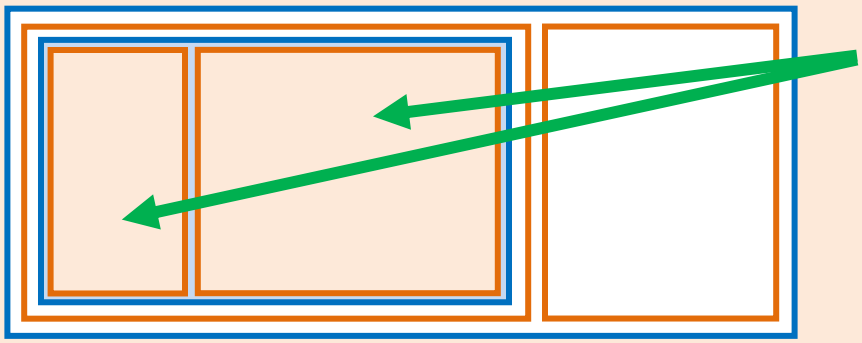
To read more about flexbox, see: https://css-tricks.com/snippets/css/a-guide-to-flexbox/. I recommend a look at this guide as there are some useful alignment properties.

Can you nest flex items within another flex item?

Not exactly. The reason for this is that a flex item must always be a child of a flex container. If you then want to set the parent flex item to display:flex this will break the layout because now you have both properties for a flex container *and* a flex item on the same element.

The way around this is to nest a flex container (blue) within the flex item (orange) you want to add more flex items to.





CSS3 Custom Properties

Custom properties have been added to CSS3 but it's relatively "new" so the support may vary across older browsers. It depends on how far back you need to support. Edge did not fully support this until version 16 (released October 2017). Other desktop browsers had support around 2016.

CSS custom properties can be considered as primitive variables in that you can use them to name, for example, a commonly used property (e.g. a primary colour). These custom property names are denoted using two dashes, like so:

```
--primary-color: #efefef;
```

It is common to declare these in the :root pseudo-class for global variables. For example:

```
:root {
  --primary-color: #efefef;
}
```

The above line of CSS will represent `#efefef` whenever `var(--primary-color)` is used.

For example:

```
#header {
  background-color: var(--primary-color);
}
```

If you are interested to learn more about CSS variables, you can take a look at the following resources:

- https://css-tricks.com/making-custom-properties-css-variables-dynamic/
- https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties