Flexbox

Learning Objectives

- Explain the use cases for Flexbox.
- Define main axis and cross axis and how we orient them.
- Explain the difference between `justify-content` and `align-items`.

Framing

HTML was created as a document-oriented language. CSS emerged as a way to use language to precisely define stylistic features in a way that wouldn't clutter the semantic content or worse destroy the semantic value all together. CSS pursued the related goal of normalizing styling across browsers. In many ways it achieves this goal well; yet it remains one of the most frustrating parts of web development.

It's difficult to establish new CSS standards. The CSS Working Group works to further the language of CSS, but with competing views of the members and the problem of cross-browser consistency, this can take considerable time.

Alignment has traditionally been one of the key contributors to this aggravation. In the last decade, the importance of alignment has sky rocketed.

Why might this be?

Today we'll be learning about two modern CSS tools to help with making an easier-to-write and more versatile website layout.

Flexbox, a layout mode introduced with CSS3, is at this point widely implemented across different browsers. CSS Grid, an even newer layout mode which borrows from the best parts of CSS bootstrap, is also becoming popular. It is not as widely implemented as Flexbox but it does work on most modern browsers.

We can check https://caniuse.com/ to see what browsers support what we want to implement.

Problem 1: Vertical Alignment

Let's start out by talking about a problem that anybody who has written CSS has likely dealt with:

I have a `div`. I would like to center it vertically and horizontally on my page.

You Tell Me: What Should I Try?

```
<html>

  <body>

    <div> Div 1 </div>

  </body>

</html>

body {

  min-height: 100vh;

  margin: 0 auto;

}



div {

  width: 100px;

  height: 100px;

  background: #990012;

}
```

These might work... > **Padding**: The simplest approach would be to set equal padding on the top and bottom of the container (body) element. We would need to know the exact height of the element and container in order to get this exactly right. This can also get tedious when there is more than one element in a container. > > **Margin**: Similarly, we could add some margin to the element we are

trying to center. The same issues remain. > > **Absolute Positioning**: You could use properties like `top` and `left` to position an element in the center. This, however, removes it from the document flow.

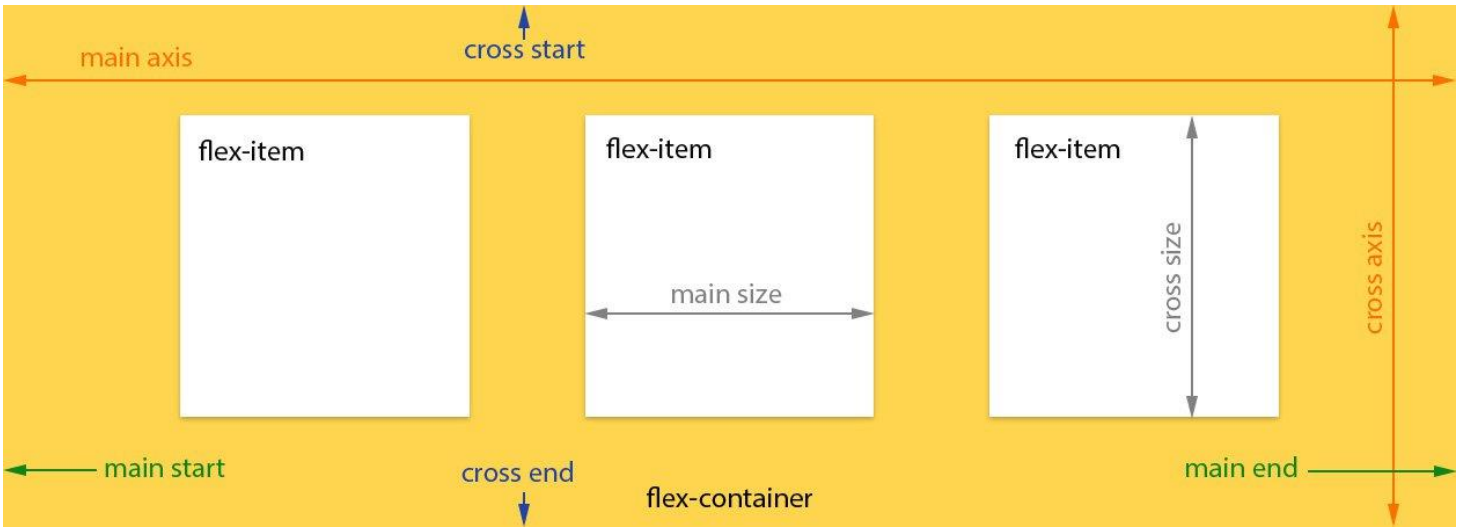
These could work in other scenarios... > **`line-height`**: When vertically centering a single line of text, you can set the line-height to that of the whole container. > > **`vertical-align`**: Used to align words within a line of text (e.g., superscript, subscript).

The tough part is that how to vertically center a element depends on its context meaning that an element has to look to its parent and then align itself; siblings start to make this very difficult. Depending on your situation, one or more of the above techniques could work. Here's an enlightening post on the matter.

Flexbox to the Rescue

```
body {

  min-height: 100vh;

  margin: 0 auto;

  display: flex;

  justify-content: center;

  align-items: center;

}
```

How It Works



flexbox diagram

When you declare `display: flex;` in a CSS rule, whatever is targeted by that rule becomes a **flex container**.

The flexbox approach differs from the methods described in the CodePen above in that the arrangement of elements is managed by the **parent** container. The child of a **flex container** is called a **flex item**. We can change the way flex items display by setting item-specific properties that will come later in the lesson.

After the `display` property, the most important flexbox property to understand is `flex-direction`. It is very important to remember that the `flex-direction` orients **flex container's main-axis**. The main axis can set to run vertically or horizontally depending on the value of `flex-direction`. All other flex-related properties are defined in terms of the main axis.

First, use `flex-direction` to indicate whether you want the flex items in the container to "read" left-to-right (`row`), right-to-left (`row-reverse`), top-to-bottom (`column`), **or** bottom-to-top (`column-reverse`).

flex-direction	main-axis start
row (default)	left
column	top
row-reverse	right
column-reverse	bottom

The `justify-content` property aligns content relative to the **main axis**. Possible values are: `flex-start` (default), `flex-end`, `center`, `space-between`, and `space-around`.

What do you think each does; does the flex-direction affect this?

The `align-items` property is similar to `justify-content` but aligns relative to the **cross-axis**. There are similar options here: `flex-start`, `flex-end`, `center`, `stretch` (default), and `baseline` (items are aligned by their baselines / where the text is).

By default, a **flex container** will arrange its children in a single row or column. The `flex-wrap` property can modify this with the values `nowrap` (default), `wrap`, and `wrap-reverse`.

When text is wrapping, `align-content` controls how the rows or columns are arranged relative to the cross-axis: `flex-start`, `flex-end`, `stretch` (default), `center`, `space-between`, and `space-around`.

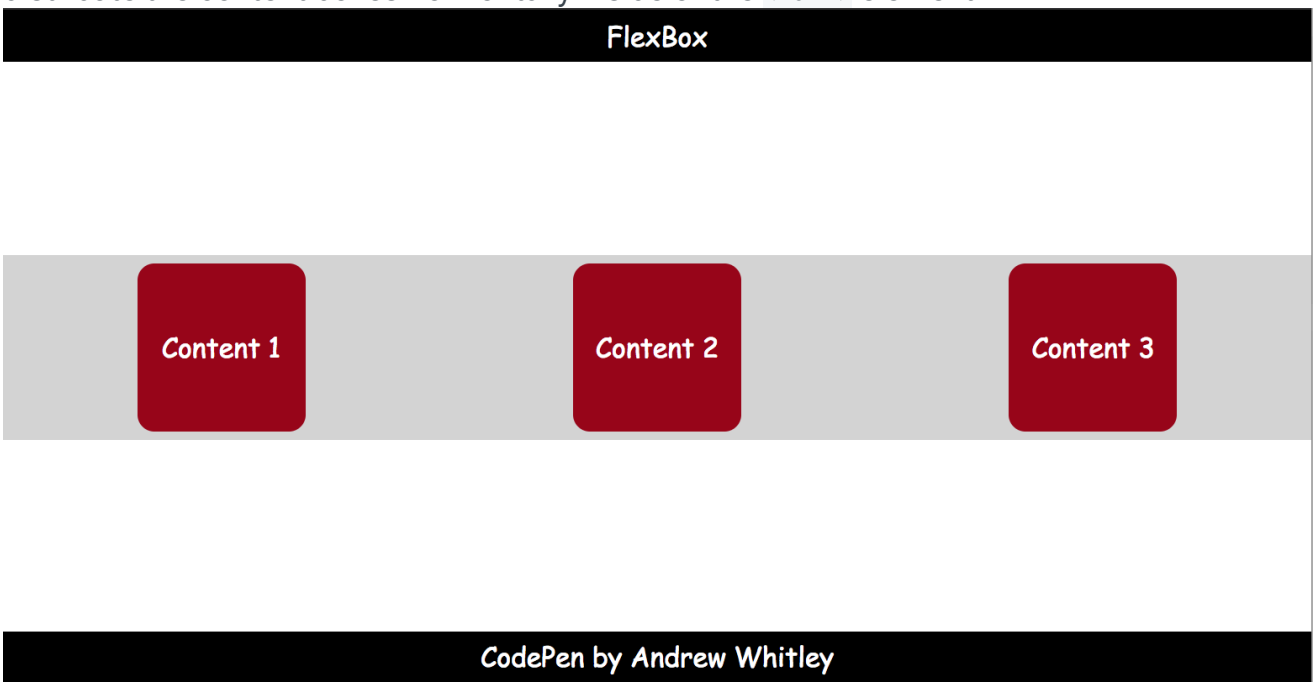
In Summary...

Property	What's It Do?	Examples
display		`flex`
flex-direction	Sets the directional flow of flex items	`row`, `column`
justify-content	Align along main axis	`center`, `space-between`
align-items	Align along cross-axis	`flex-start`, `center`

That's a lot of CSS properties! Don't worry, you're not expected to memorize all of them. Being a developer is less about knowing everything off the top of your head and more about knowing best practices and where to find more info Here's a great resource.

Problem 2: Make the Footer Stick

I want my footer to lie along the bottom of my page. Once I've accomplished that, I want to evenly distribute the content boxes horizontally inside of the `<main>` element.



flexbox layout

You Tell Me: What Should I Try?

```
<html>

  <header>

    FlexBox

  </header>

  <main>

    <section>Content 1</section>

    <section>Content 2</section>

    <section>Content 3</section>

  </main>

  <footer>

    This is the Footer

  </footer>

</html>

body {

  min-height: 100vh;

  margin: 0 auto;

  font: 12pt Comic Sans MS;

}
```

```
header, footer {
  width: 100%;
  height: 30px;
  background: #000000;
  color: #FFFFFF;
  text-align: center;
  line-height: 30px;
}

main {
  background: #D3D3D3;
}

section {
  width: 100px;
  background: #990012;
  color: #FFFFFF;
  border-radius: 10px;
```

```
  margin: 5px;

  text-align: center;

  line-height: 100px;

}
```

Making the footer lie against the bottom of the screen is pretty easy: just use absolute or fixed positioning. However, using absolute or fixed positioning means everything else on the page ignores my footer. The text of `<main>` could easily run under the footer. We want the text to "push" the footer to the end of the page.

Flexbox to the Rescue

```
body {

  min-height: 100vh;

  margin: 0 auto;

  font: 12pt Comic Sans MS;

  display: flex;

  flex-direction: column;

  justify-content: space-between;

}
```

How is main axis of the `body` oriented here? What about the cross-axis? > Main: vertically, Cross: horizontally

Now let's horizontally distribute the `<section>` elements containing the page's content inside of the `<main>`. What element should we style?

```
main {
```

```
  background: #D3D3D3;

  display: flex;

  justify-content: space-around;

}
```

Some Helpful Resources

- flexbox.io
- The Ultimate Flexbox Cheatsheet
- CSS Tricks' Guide to Flexbox
- A Visual Guide to CSS Flexbox Properties
- Solved by Flexbox
- Flexplorer
- Holy Grail Layout - Solved By Flexbox