Display (incl. Flexbox)

There are several values you can add for the display property.

📚 **Resource:** Mozilla Documentation

- `block` – The element takes up the full width of its parent container and forces a new line after it (e.g., `<div>`, `<p>`). This is the default for many elements.
- `inline` – The element only takes up as much width as necessary and does not force a new line (e.g., `<span>`, `<a>`).
- `inline-block` – Like `inline`, but allows setting width and height. Also, you can adjust top and bottom padding (whereas for inline, you can only adjust horizontal padding).
  - You could use this for navigation. I tend to use flex, but it's up to you and your project.
- `flex` – Turns an element into a flex container, allowing flexible child positioning.
- `grid` – Turns an element into a grid container.

Additionally:

- `none` – Completely removes the element from the document flow (it won't take up space at all).
- `table`, `table-row`, `table-cell` – Mimics table behavior.
- `inherit` – Takes the `display` value of its parent.

Why set display to `none`?

- `display: none` – The element is removed from the document layout; it won't take up space.
  - Example: Hiding a dropdown menu until the user interacts with a button.
  - Example: Hiding navigation links on smaller screens in favor of a hamburger menu.
  - Example: Toggling different sections of a page based on user preferences.

You can also hide an element by changing its visibility property and its opacity property, but these mean something different:

- `visibility: hidden` – The element remains in the layout (it takes up space) but remains invisible.
  - Example: Keeping layout consistency while hiding content
- `opacity: 0` – The element remains fully interactive and in the layout but is completely transparent.
  - Example: Often for animations, hover effects, or gradual fade-ins

The Basics of Flexbox

📚 **Resource: Mozilla Documentation**

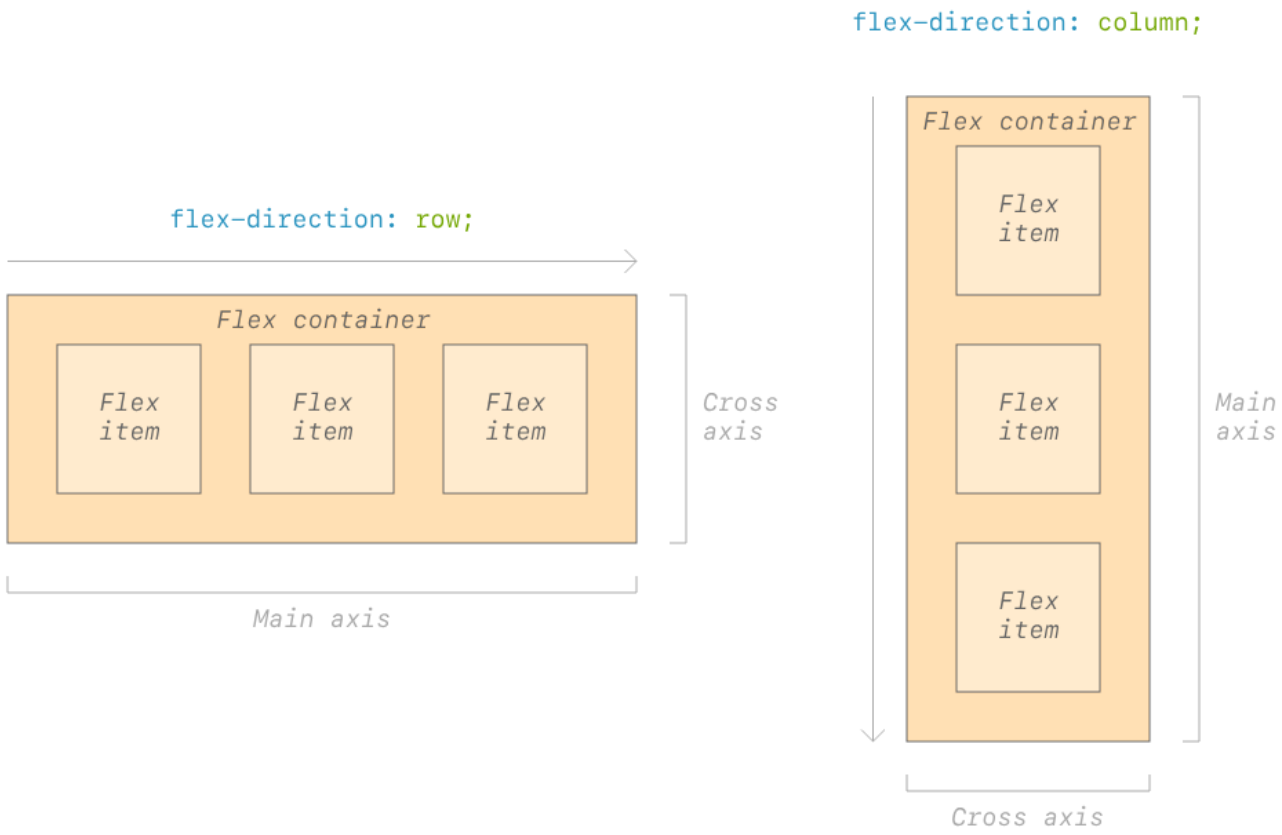
Flexbox, short for *flexible boxes*—which folks will often just shorten all the way to *flex*—is a later (mid-2010s, depending on how you count) addition to CSS.

- *Flex* was created to facilitate and allow CSS layouts that the box model either made difficult, brittle, or even impossible. It is a display property.
- It is frequently used for spacing and aligning items in a responsive manner.

Flexbox is a *one-dimensional* layout system—meaning it is (usually) focused on arranging items either horizontally in rows, or vertically in columns.

`flex` & `flex-direction`

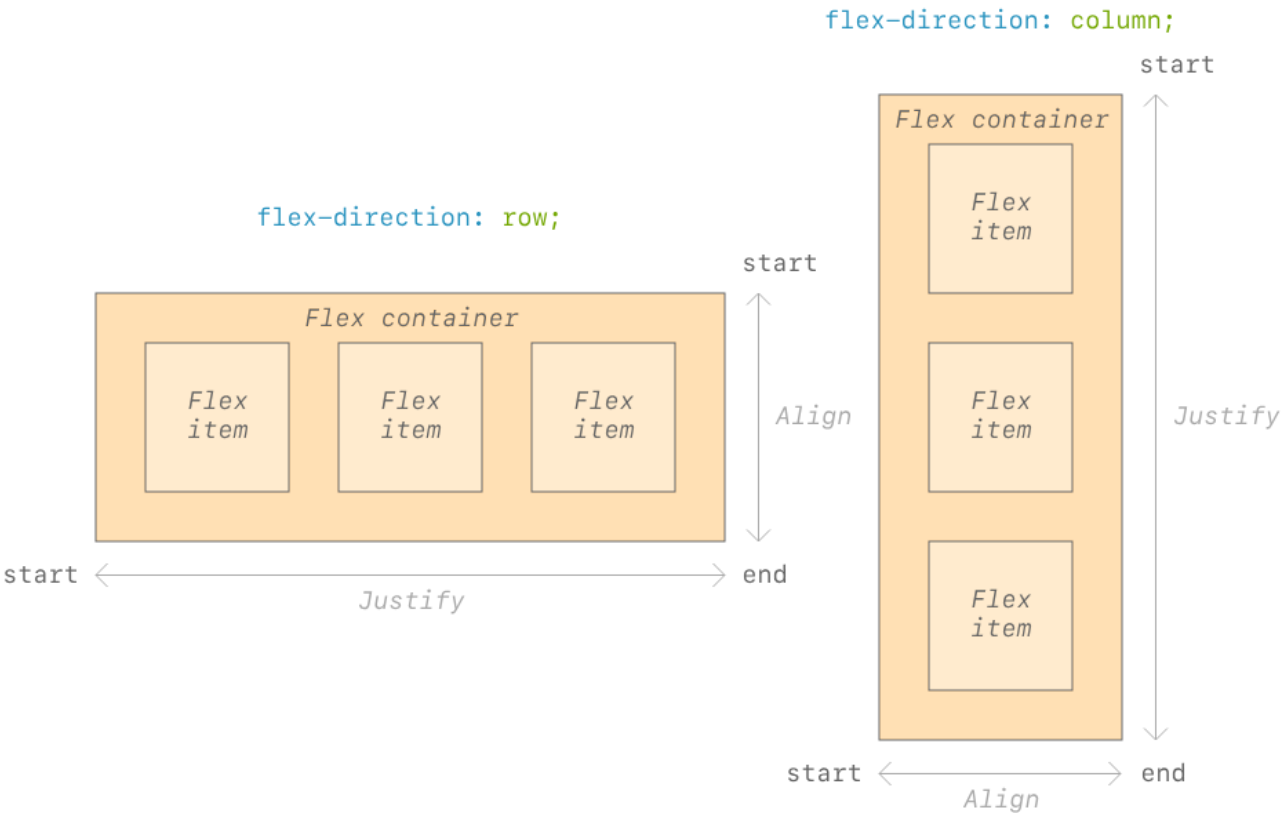
`flex-direction`: By default flex containers behave as `flex-direction: row;`, so you'll only be adding this property when you want something going vertical—with `flex-direction: column;` :



credit: michael fehrenbach

`justify-content` & `align-items`

- **`justify-content`** controls spacing along the **main axis** (horizontal by default, unless `flex-direction` is changed).
- **`align-items`** controls spacing along the **cross axis** (perpendicular to the main axis).



credit: michael fehrenbach

Possible values include:

- flex-start
- flex-end
- center
- space-between
- space-around
- space-evenly

`align-self`

This is an individual override for an `align-items` property set on the parent—which adjusts (with the same keywords/values) the alignment of the *specific* child element it is applied to.

The flex property value is applied to the parent container

Flex is applied to a parent element, but it controls the layout of its children. When you set display: flex; on a container, you're not changing the container itself much—you're defining how its children will behave inside it.

- This is different from most CSS properties, which typically affect the element they're applied to directly.

There is also `display: inline-flex;` which behaves the same, but the parent behaves as an inline element while its children are flexing.

Let's play a game

📚 **Resource:** This website is my holy grail. It might yours too. If I answer your Flexbox questions, I'm probably looking at this website.