

CSS Display + Positioning

CSS display and Positioning Demo code.zip 3.9KB

Goals

- Compare different values for the *display* property
- Compare different values for the *position* property
- Provide an overview of responsive design and CSS transitions
- Highlight features of production-level CSS

Display

"How should this be displayed?"

- *inline* (e.g. <i>, <image> ,)
 - Only takes as much space as needed
 - ignores width and height properties
 - Next inline item is side-by-side
- block (e.g. <form> , <div> , <section>)
 - o 100% of parent width *unless* set via *width* or *max-width*
 - o can modify width and height with CSS
 - Next item is on a separate line

Goals

Display

Flexbox

Common

Flexbox

Properties

Flexbox

Resources

Position

Responsive Design

> Media Queries

Transitions

Professional CSS

Class Naming Conventi...

"Meta-CSS"

Systems

• inline-block

- Side-by-side, like inline
- Respects width and height, like block

Feature	block	inline	inline-block
Side-by-side layout	×	✓	✓
Respects width / height properties	✓	×	✓

none

- Don't show, don't take up any space
- flex
 - Enables Flexbox (see below).

Plus lots of specific ones related to lists, tables, and so on

Display Docs

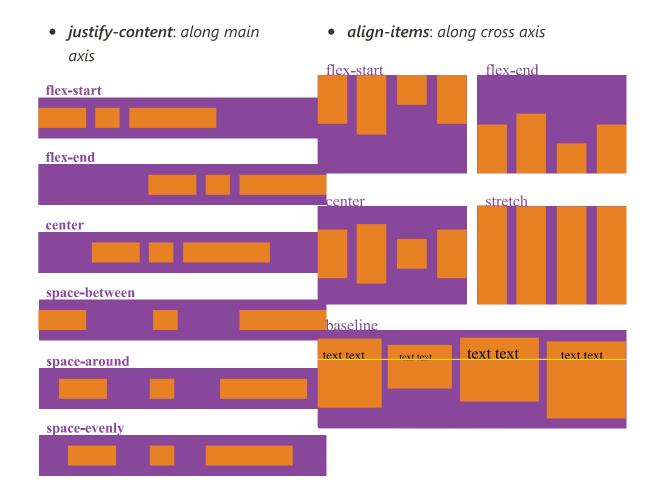
Flexbox

Flexbox is a large topic, beyond the scope of a single lecture.

TL;DR: flexbox makes common layouts easier to create

Common Flexbox Properties

On an element with display set to flex:



Flexbox Resources

Why Flexbox

A Guide to Flexbox

Flexbox Froggy

Flexbox Defense

Position

"How should this be positioned?"

- static
 - Default until you say otherwise
 - Appears at normal place in "flow"

"Positioned"

• All non-static are considered "positioned"

relative

- relative to normal position (via top, right, bottom, left)
- takes same space as if it were static & in normal flow

absolute

- absolute position from closest positioned ancestor
 - other relative settings, like width, come from that now, too
- removed from flow, takes up no space

fixed

- absolutely positioned from viewport (stays there on scroll)
- removed from flow, takes up no space

P (

Other Position Values

There are many other possibilities here, but many are uncommon to work with.

Two additional useful ones are:

sticky

A combination of *absolute* and *fixed*; this is useful for allowing something to scroll on the screen, but then fix to the viewport before it scrolls out of sight.

Responsive Design

- Bad old days: separate sites for desktop & mobile
- Semi-bad semi-old days: every site should have a native app
- Responsive design: same site with adjustments in CSS

Media Queries

```
/* default to smaller devices ("mobile-first") */ img.headshot
{ display: block; width: 70%; } @media (min-width: 800px) {
img.headshot { display: inline-block; width: 10em; } /* other
"desktop-size" stuff here...*/ }
```

Mostly, we'll use Bootstrap for this

Transitions

transition: property duration function delay [, ...]

- What property changes? (all for all)
- Over what time (eg, *3s*)
- What function determines values? (ease, linear)
- What delay before this? (eg, *0.5s*)
- Can list several properties

Transition Intro

Professional CSS

Industry practice varies here!

Common ideas:

- avoid !important where reasonable
- generally, prefer classes over IDs (IDs for JS is fine!)
- "why" comments in CSS can be even more important
- adopt & use a naming scheme

Class Naming Conventions

• CSS about a "component" is named (via class) after component:

```
.Tweet { /* ...declarations go here ... */ }
```

• Variations/small subcomponents get "dashed" name:

```
.Tweet-likes { /* stuff about the photo ... */ }
```

• Used-lots-of-places "utilities" get lowercase classes:

```
.muted { font-size: 80%; color: gray; }
```

• This is a light version of style called BEM

"Meta-CSS" Systems

There are meta-systems for CSS, doing things like:

• Letting you associate colors with names (en "our logo color")