

0. Choosing between a web and a mobile app

(A carryover from last lecture)

- Android (Java or Kotlin) or iOS (Objective C or Swift)
- React Native (Javascript)
- Flutter (Dart)
- Progressive Web App, or PWA (Javascript)
- Responsive Web App (Javascript)

1. Introduction

Lecture overview

- You will design the UI in Figma, and you'll need to implement those designs
- Knowing how to do this using modern tools can save lots of time and craft
- Doing this right is very helpful to those with accessibility concerns

- You know:
 - That HTML is for describing the content of a web document and represents a tree of elements, each with optional named attributes
 - That CSS is for describing the style of a web document
 - How to interpret CSS selectors, e.g. `#wrapper`, `input[type=text]`, or `.box`

- We're dealing with web layout only
- Some concepts apply very directly to other platforms
- E.g. React Native only uses Flexbox for layout

- Display types (e.g. block, inline, float)
- Positioning elements (e.g. relative, absolute, fixed)
- Flexbox layout
- Grid layout
- Responsive design (for mobile-friendly pages)

2. Display types

Normal flow

- Browsers have a default location for each element
- Depends on window dimensions
- Called *normal flow*
- Two types: block and inline

- *Block elements* stack vertically.
- Examples: paragraphs, headers, lists
- CSS: `display: block;`

- *Inline elements* flow left-to-right and wrap to new lines.
- Examples: words in a paragraph, italic, bold, code, links
- CSS: `display: inline;`

`https://comp523.cs.unc.edu/demo.2019-09-11.1.
normal-flow.html`

2. Display types

Floating elements

Floating elements

- Some elements should not be block or inline
- Example: an image in a magazine
- Text or other inline elements flow around it
- You can float an element to the left or right
- CSS: `float: left;` or `float: right;`
- Floats don't contribute directly to size of parents

Dependence on window dimensions

- Different window sizes can be surprising
- Or different amounts of text
- You can set an element to “clear” the float
- CSS: `clear: left;`
- Can clear left, right, or both

`https:
//comp523.cs.unc.edu/demo.2019-09-11.2.floats.html`

2. Display types

Table layout

- Anything can be displayed as a table
- CSS properties:
 - `display: table;`
 - `display: table-row;`
 - `display: table-cell;`
 - `display: table-caption;`
 - `caption-side: bottom;`

Example (from MDN)

`https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Introduction#Table_layout`

3. Flexbox layout

Basic concepts

Flex container and flex items

- Flex layout is opt-in
- CSS: `display: flex;`
- Element with flex display property is *flex container*
- Direct children are *flex items*
- Enables flexible dimensions *in one primary dimension*

- Top-level property on flex container: flex direction
- CSS: `flex-direction: row`
- Valid values:
 - `row`
 - `row-reverse`
 - `column`
 - `column-reverse`
- `row` is default for web (for English); `column` is default for React Native

Main axis and cross axis

- Flex direction determines the *main axis*
- The perpendicular direction is the *cross axis*
- CSS properties affect either the main or cross axis, so are dependent on the flex direction

`https://comp523.cs.unc.edu/demo.2019-09-11.3.
flex-direction.html`

3. Flexbox layout

Aligning and justifying items

- The *flex start* depends on the writing direction
- Default: left and top for English, but also depends on flex direction
- The *flex end* is the opposite side

Aligning on the cross axis

- Use `align-items` CSS property to align items on the cross axis
- Common values:
 - `flex-start` (default)
 - `flex-end`
 - `center`
 - `stretch`
 - `baseline`

- Use `justify-content` CSS property to justify items on the main axis
- Common values:
 - `flex-start` (default)
 - `flex-end`
 - `center`
 - `space-between`
 - `space-around`
 - `space-evenly`

`https://comp523.cs.unc.edu/demo.2019-09-11.4.
flex-alignment.html`

3. Flexbox layout

Automatic sizing

- *Flex basis* is how much space to give a flex item along the main axis
- The item's final size is also affected by growing and shrinking properties
- CSS property: `flex-basis`
- Valid values:
 - `auto` (default)
 - a size, like `100px` or `2rem`

- Items may grow to fill the available space
- Disable this behavior with `flex-grow: 0`
- Enable it with a positive number for `flex-grow`
- What does the number mean?

- First, determine initial size of all items from `flex-basis` values
- Then, there may be some space left over in the flex container
- Divide this up fractionally based on `flex-grow` values of items
- Example: values of 2, 1, 1 means that the first item gets twice the space of the other two

- Similarly, `flex-shrink` will shrink items if the main axis overflows
- Set `flex-shrink: 0;` to disable shrinking
- Shrinking takes into account the minimum size of items so may seem broken or inconsistent

- Combination of `flex-grow`, `flex-shrink` and `flex-basis` properties
- CSS: `flex: 1 1 auto`
- Special values:
 - `initial` is equivalent to `0 1 auto`
 - `auto` is equivalent to `1 1 auto`
 - `none` is equivalent to `0 0 auto`
 - a number, e.g. `1` is equivalent to `1 1 0`

`https://comp523.cs.unc.edu/demo.2019-09-11.5.
flex-sizing.html`

3. Flexbox layout

Interlude

`http://flexboxfroggy.com/`

4. Positioning elements

Static, relative, absolute, fixed, and sticky

- This is the default
- Just where the browser would place things normally
- Statically positioned elements are considered unpositioned

- *Relative positioning* offsets an element from its static position
- CSS: `position: relative;`
- Use `top` and `left` CSS properties to specify the offset
- (Can also/instead specify `bottom` and `right`)
- These values can be negative
- Space is still reserved for the element's static position
- Demo: <https://comp523.cs.unc.edu/demo.2019-09-11.6.relative-positioning.html>

Absolute positioning

- *Absolute positioning* positions an element relative to an ancestor
- Reference frame is nearest positioned (i.e. non-static) ancestor
- CSS: `position: absolute;` and `top` and `left` properties
- Space is not reserved for element's hypothetical static position
- Demo: <https://comp523.cs.unc.edu/demo.2019-09-11.7.absolute-positioning.html>

Fixed positioning

- *Fixed position* is when an element stays in a single spot on a screen
- Demo: <https://www.davidwalsh.name/demo/css-fixed-position.php>

Sticky positioning

- *Sticky positioning* is a combination of static and fixed positioning
- Easiest way to describe this is by showing it
- Demo: <http://html5-demos.appspot.com/static/css/sticky.html>

5. Responsive web design

A quick overview

- Goal: let your site look good on any sized screen
- An incantation:
 - `<meta name="viewport" content="width=device-width, initial-scale=1" />`
- Design for mobile first (because really basic phones don't support media queries)
- Then add media queries to use more horizontal space on bigger screens
- Turn on responsive design mode in your browser dev tools
- Example: <https://applab.unc.edu>

6. Grid layout

A few resources

- Like flexbox, but for flexible layout in *two* dimensions
- Sophisticated and flexible, but not difficult!
- Supported in all major browsers:
<https://caniuse.com/#feat=css-grid>
- MDN overview:
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout
- Tutorial game: <https://cssgridgarden.com/>