CSS

Positioning

Today's Agenda

- Blocks vs Inline vs Inline-Block (no really)
- CSS Positioning
- Positioning Exercise
- External Style Sheets
- External Style Sheet Exercise
- Mock Ups
- To Spec Exercise

Display Property

The *display* property is the most important CSS property for controlling layout.

\- W3C

- it specifies if/how something is displayed (duh)

display: block

- Most elements are displayed as blocks by default
- block elements are always on a new line
- they always take up the full width of their parent
  - div, p, section, article, h1 - h6, header, footer, ul, li (just to name a few block elements)

display: inline

- the 2nd most common default display type
- inline elements do NOT start on a new line
- they only take up as much space as needed
  - a, span (just to name couple)

display: inline-block

- the element is treated as an inline-level block, meaning…
- It’s content is treated like it’s in a block, while the element itself is treated as inline
- Doesn’t have a newline, and content is only as big as needed, but padding & margin are respected
  - img, button, select

    (This is why images don’t play nice with centering!)

Centering a block

- margin: 0 auto

(psst: to center an image horizontally, make its display block)

Play more with Display ;)

Try using various display properties on your portfolio page, such as centering your image, or your nav menu.

Positioning with Position

The CSS positioning properties allow you to position an element.

It can also place an element behind another, and specify what should happen when an element's content is too big.

http://www.w3schools.com/css/css_positioning.asp

Position

Elements can be positioned more specifically using the top, bottom, left, and right properties.

These properties will not work unless the position property is set first. They also work differently depending on the which position value you set.

There are four different position options.

position: static

- the default, normal way of positioning by the natural “flow” of the browser
- elements are not affected by top, right, bottom, and left properties when they're statically positioned

This is the default for EVERYTHING, and is what you set on something to return it to “normal”.

position: fixed

- the element is set in relation to the browser window and will not move, even if the page is scrolled
- positioning is by the top, right, bottom, left properties, in relation to the browser window

position: absolute

- similar to fixed, but instead of the browser window, the element is positioned in relation to the *first parent element that has a position other than static*
- That's important – the **FIRST** parent that isn't static positioned.
- It’s can be tempting to just set position:absolute or fixed on everything, but this makes sites extremely unresponsive to various screen sizes and/or devices

position: relative

- a relative positioned element is positioned relative to its normal position in the flow
- the space it would normally take up is still preserved, even though the element is not there
- Relative elements are often used to contain Absolute elements

Relative is my favorite CSS position. Is that TMI? That might be TMI.

Playing with Positions

- Use position to attach your footer to the bottom of your web page (not the browser window)
  - make a footer if you don't have one.. ;)
- Use position to make something else go something it wouldn't normally
  - get creative ;D

Wrapper Classes

- wrappers are elements, commonly divs, articles, or sections, that enclose one or more elements in your Markup

Wrappers – Why?

- to group elements semantically
  - to separate page heading from body text from sidebar from footer
- to group elements cosmetically
  - to add a surrounding border or a common background image or color
- to group elements in layout
  - to keep them all in the same column when columns are floated next to one another
- to enable special positioning
  - a wrapper is given relative positioning in order to contain child elements with absolute positioning

Wrappers – The Ultimate Why

to make it more convenient to specify elements in CSS and JavaScript by referring to their parent, **without having to id or class each child**

Check the Wrapper

Look at your HTML and see if you're using wrappers effeciently.

Would adding a wrapper help reduce the complexity of your CSS?

- Yes?

    Go for it!

External Stylesheets

External Stylesheets are .css files that you load into your web page. But why?

- All your CSS is in one, easy to find place
- You can easily use the same CSS across multiple web pages
- You site will load quicker once the browser caches the CSS
- You can easily change the look and feel of your site with 1 file change

When should you do what?

Really? You should use styles…

- inline: only for debugging
- in the style tag: for big debugging, or when you're first building a page
- external: ALWAYS! (unless it's a very tiny site. Very tiny. Tinny. Tiny.)

External Style Sheets

- It's time for our CSS to move out
- Move your CSS (everything but the <style> tags) to a CSS file (.css extension)
- Add a link tag to load your stylesheet into your web page

```
<link rel="stylesheet" type="text/css" href="mystyle.css">
```

Remember when I said I'd talk about selectors?

- element, element
  - div,p,a will apply the style to all divs, ps, and as
- element element
  - div p – Selects all <p> elements inside elements
- element>element
  - div>p – Selects all <p> elements that are direct children of a element
- element+element
  - div+p – Selects all <p> elements that are placed immediately after elements
- element~element
  - p~ul –Selects every <ul> element that are preceded by a <p> element

Multiple Class Names

You can put as many class names as you want in the class attribute, separated by a space.

<p class=”body_text quote flashy_text”>

The browser will apply all the styles it can from all of the listed classes, overriding any duplicates in the order you give.

Review on Styling

- id's:
  - should be used as a last resort, and only for very important elements
- class names:
  - are better, and should be used on wrapper elements rather than specific elements
- inline styles:
  - should never be used in production

Review on Styling

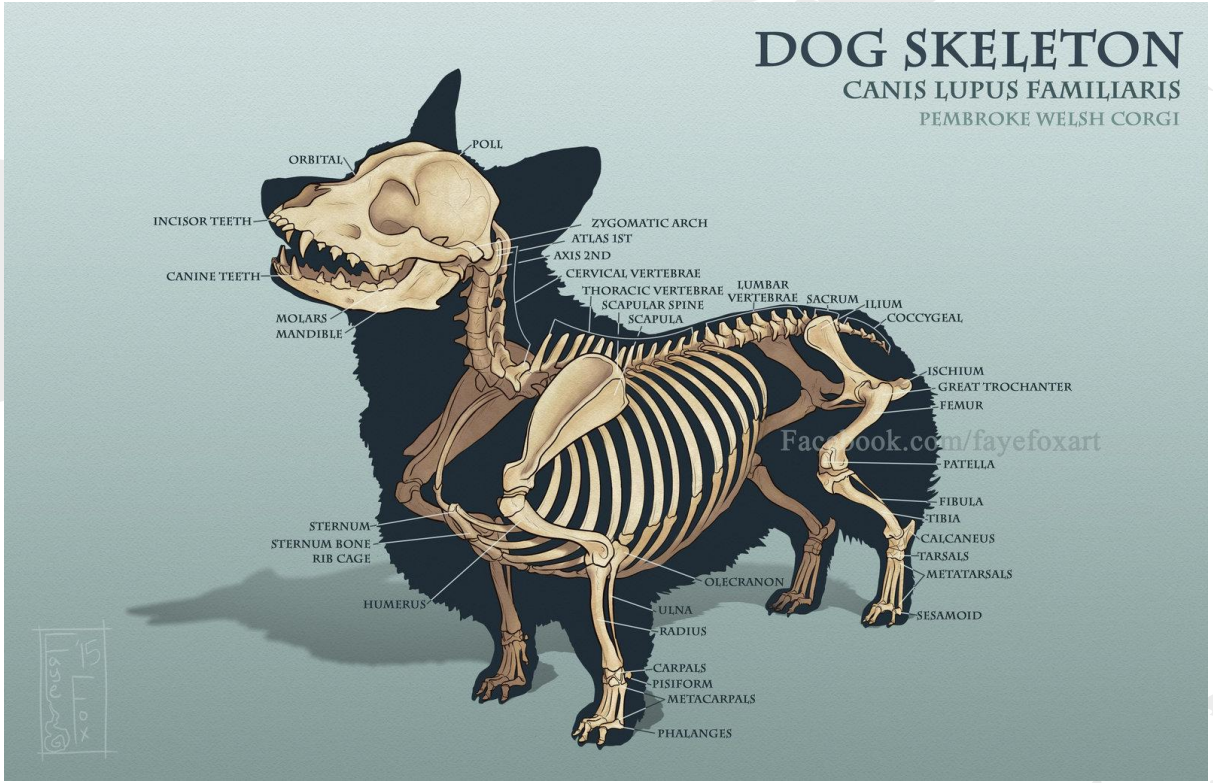
- Styles that come later in the file override previously defined styles of lower or similar specificity
- Style selectors which are more specific override other selectors, no matter where in the file they are

Review on Styling

- Because of this, it's better to format your CSS sheets going from General to Specific
  - the order of your CSS should closely resemble the order of your HTML page
  - styles that affect your whole page should be near the top
- You should still group like or similar selectors together, with comments defining sections

Remember Corgi Skeletons?

If HTML is the Skeleton...



… then CSS defines its appearance.

Different CSS, means different appearances.

Sometimes, very different!

Corgi Stylesheets

Can you now understand most of these jokes?

CSS is Very Powerful

- We have only barely scratched the surface of CSS
- It’d be impossible for me to teach you how to do everything – only practice and more practice will teach you all the tricks
- If you want to learn to do something, find an example of it and inspect how it’s done there

CSSZenGarden.com

The same HTML, different CSS – Completely different web pages.

This is a bit more practical than looking at a Corgi’s code ;)

ARIA

Accessible Rich Internet Applications

- ARIA allows us to define our websites in such a way that accessible users can more readily enjoy the Wild Wild Web (www)
- We already use some ARIA defined tags (article, section, nav, etc)

(www doesn't actually stand for Wild Wild Web, but I think it might as well.)

ARIA in HTML5

ARIA is integrated into HTML5, but not completely. There are still things missing and the role attribute is needed.

The role attribute is added to HTML elements to tell accessible devices what ARIA-defined elements are on a page.

ARIA in HTML5

If you can use a native HTML element [HTML5] or attribute with the semantics and behaviour you require already built in, instead of re-purposing an element and adding an ARIA role, state or property to make it accessible, then do so.

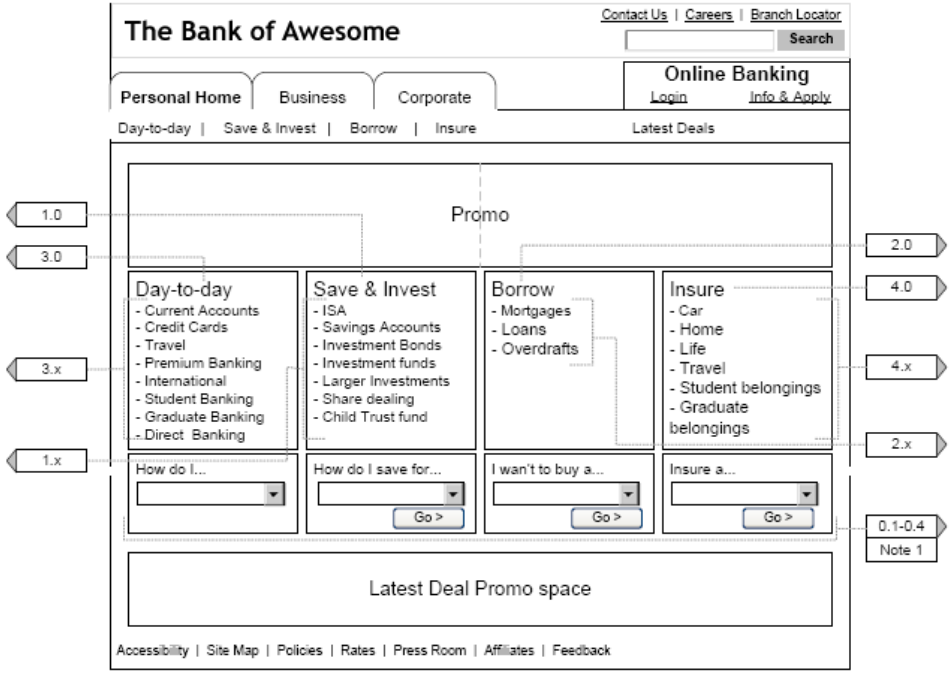
Otherwise, use a role! What's not fully implemented?

http://www.paciellogroup.com/blog/2014/10/aria-in-html-there-goes-the-neighborhood/#html5na

Developing to Spec – Wireframes

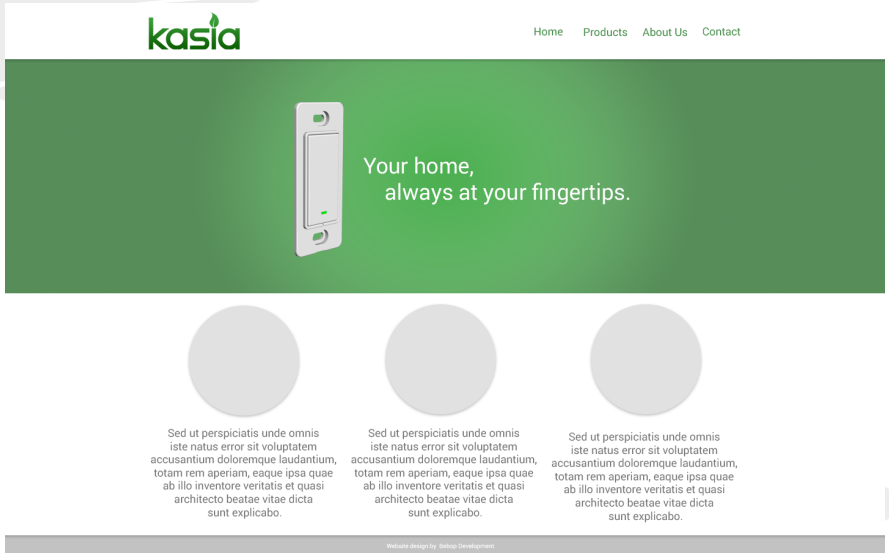
Wireframes are the blueprints of a design – they outline the components, general placement, and interaction for an interface

- They're like interface UMLs



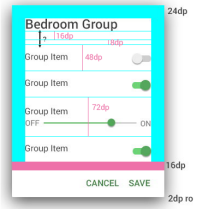
Designing to Spec – Mockups

Mockups are realistic representations of a design – with all the colors, positioning, and elements flushed out. Mostly. Sometimes. It depends.

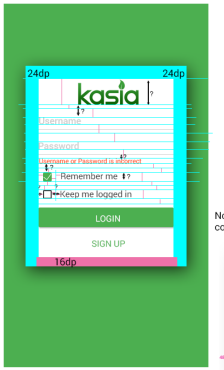


Designing to Spec – Specs

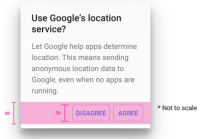
- Specs are essentially mockups with specifications and rules for the design outlined.
- Redlines specify key things like padding and margin
- I <3 coding to Redlines, but I hate making them in Photoshop

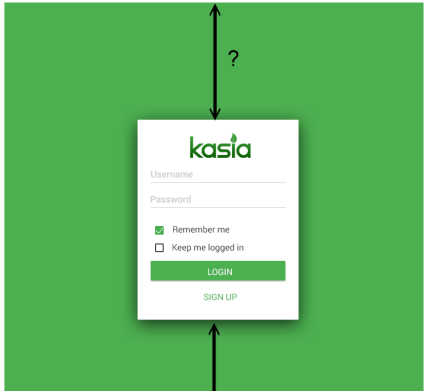


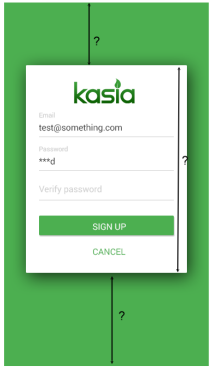
Titles have 16dp of padding below them.
The list has 8dp of padding on top.



Note that buttons are 48dp in height, but only have colored area of 36dp, as per Google Specs.







Helpful Tools

Chrome Ruler – Helpful for when there's no specs

Colorzilla – Color picker widget

To Spec Practice

Use the HTML provided to create a CSS stylesheet that causes the web page to look exactly like the Mock-up.

- Your CSS file must be name “zen_style.css” or it WILL NOT WORK
- You are NOT allowed to modify the HTML doc