HTML & CSS Level 1: Week 5

June 2, 2014

Instructor: Devon Persing

Last week

- CSS abbreviations
- Using `class` and id for `styles`
- Pseudo-classes
- Fancy backgrounds

{ } Abbreviated hex colors

```
color: #333333;

/* becomes */

color: #333;


color: #aa0099;

/* becomes */

color: #a09;
```

{ } Abbreviated font styles

```
font-style: italic;
font-variant: small-caps;
font-weight: bold;
font-size: 1em;
line-height: 1.5em;
font-family: Helvetica, sans-serif;

/* becomes */

font: italic small-caps bold 1em/1.5em
Helvetica, sans-serif;

/* font-size & font-family are required! */
```

{ } border abbreviations

```
border-top-width: 4px;
border-right-width: 3px;
border-bottom-width: 4px;
border-left-width: 3px;
border-style: solid;
border-color: #a00;

/* becomes */

border: 4px 3px solid #a00;
```

{ } margin and padding abbr.

```
margin-top: 20px;
margin-right: 30px;
margin-bottom: 40px;
margin-left: 50px;

/* margin works just like padding! */
margin: 20px 30px 40px 50px;

margin: 20px 40px;

margin: 20px;
```

<> `class` attributes in HTML

- Classes can be shared by multiple elements on a page

```
<h1 class="kittens">...</h1>

<span class="kittens">...</span>
```

- Elements can have multiple classes

```
<div class="kittens puppies">...</div>

<div class="kittens puppies birds">...</div>
```

{ } `class` selectors in CSS

- Start with a **period** (`.`)
- Can style any element with the class

```
.kittens { color: #000000; }
```

- Or can be used to style only a specific **type** of element with the class

```
h3.kittens { color: #000000; }
```

- More specific than an HTML type selector

<> `id` attributes

- IDs *cannot* be shared by multiple elements on a single page
- Elements *cannot* have multiple IDs

```
<div id="kittens">...</div>

<div id="puppies">...</div>

<div id="birds">...</div>
```

{ } `id` selectors in CSS

- Start with a **hash/pound sign** (**#**)
- Can style the single element with the ID

```
#kittens { color: #000000; }
```

- More specific than a class selector

Mixing `class` and `id` attributes

- Elements can have **`id`** and **`class`** attributes at the same time

```
<div id="kittens">...</div>

<div id="puppies" class="small floppy">...</div>

<div id="birds" class="small feathery">...</div>
```

- ID selector styles can be used to override class selector styles

Be thoughtful in your selectors

- Recommended order of attack:
  a. Type selectors
  b. Class selectors
  c. Descendent selectors
  d. ID selectors
- If you overuse IDs in your styles, **you're going to have a bad time**

Award designed by Luc Poupard from the Noun Project

{ } Pseudo-classes are conditional

- **Pseudo-classes** are added to a selector to add conditional styles to an element
- Most commonly used to style **states** of **<a>** elements and form elements

```
a:link { /* the default state of a link */ }

a:visited { /* a link that's been clicked */ }

a:hover { /* a link that has a mouse hover */ }

a:focus { /* a link that has keyboard focus */ }

a:active { /* a link that is being clicked */ }
```

{ } `:hover` versus `:focus`

- **`:hover`** is for a link or other interactive element that has a **mouse hover**
- **`:focus`** is for a link or other interactive element that has **keyboard focus**
- Browsers have their own default **`:focus`** styles for **accessibility**

```
a:hover, a:focus {
/* it's good practice to style them together! */
}
```

{ } :hover for other elements

- **:hover** can be used to style hover states for some non-interactive elements to create a more dynamic experience

```
div { /* a div with a background... */
    background: #99ff66;
}

div:hover { /* ...could have another on hover */
    background: #ff6600;
}
```

{ } `:before` and `:after`

- **`:before`** is a pseudo-element before an element
- **`:after`** is a pseudo-element after an element
- We used these in our **`border-box`** reset
- These can be manipulated to simplify border box handling, layouts, add transparent background images to containers, and more

{ } Transparent background-color

```
.block {
   /* text is black and centered */
   color: #000000;
   text-align: center;
   background-color: #bc7384; /* for IE8 */
   background-color: rgba(188,115,132,0.5);
}
```

I'm partially covering up a kitten. :|

{ } Styling a background image

- The property is **`background-image`**
- The value is a **URL where an image lives**

```
.kittens {
  background-image: url("img/kittens.jpg");
}
```

{ } Repeating a background

```
/* repeat the background horizontally */
background-repeat: repeat-x;

/* repeat the background vertically */
background-repeat: repeat-y;

/* don't repeat the background */
background-repeat: no-repeat;
```

{ } Positioning a background

- **`background-position`** values include both the x-axis and y-axis
- x-axis first, y-axis second
- Can be **`left`**/**`right`** **`top`**/**`bottom`** *or* any measurement (pixels, %, ems, etc.)

```
/* position a background in the left top corner */
background-position: left top;
```

{ } More background

- You can also add almost all of your other **background-** styles to **background**:

```
/* a div with a light gray background, and a background
image that doesn't repeat and is positioned in the
bottom right */

div {
   background: #eee url("img/kitten.jpg")
no-repeat bottom right;
}
```

{ } Using the axes

```
/* left to right */
.gradient {
    background-color: black; /* for old browsers */
    background-image: linear-gradient(to right, black,
    white);
}
/* toward the top right corner */
.gradient {
    background-color: black; /* for old browsers */
    background-image: linear-gradient(to top right, black,
    white);
}
```

{ } Background attachment

```
/* have the background scroll (the default) */
background-attachment: scroll;

/* have the background stick regardless of scrolling */
background-attachment: fixed;
```

...and some others

{ } The magical image background

```
/* make a full-sized, fixed image background that covers
the whole container */
.puppies {
  background-image: url("img.png");
  background-repeat: no-repeat;
  background-position: center center;
  background-attachment: fixed;
  background-size: cover;
}
```

This week

- Browser style resets
- Introduction to layouts
- iframes and media
- Course evaluation

Resetting browser default styles

Browser defaults can be a pain

- Every browser has slightly different styles
- Different types of elements get different font sizes, line-height, padding, margins, etc.
- Tweaking styles for individual types of elements is time consuming

A blank slate

- Reset styles strip out browser default and let us make our own defaults
- We'll use the canonical reset stylesheet: **http://meyerweb.com/eric/tools/css/reset/**

Two ways to add reset styles

- **Method one:** Put reset styles into **their own .css file** and load it before your existing stylesheet

```
<link href="css/reset.css" rel="stylesheet">
<link href="css/styles.css" rel="stylesheet">
```

- **Method two:** Put reset styles **into the top** of your existing stylesheet

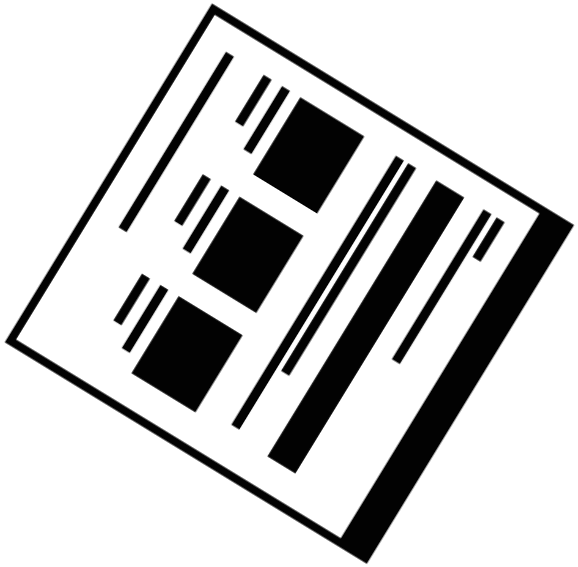
Method one: Separate styles

1. Copy reset styles
2. Paste into a new blank document in your text editor
3. Save your styles as a different .css file (e.g., **`reset.css`**) in your css folder
4. Add a link to the reset stylesheet in **`<head>`**, *before* your existing styles

Method two: One stylesheet

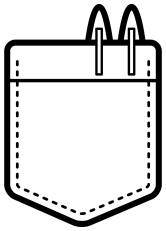
1. Copy reset styles
2. Paste at the very top of your existing stylesheet (e.g., **styles.css**) so they load first
3. Save your stylesheet

(ノ^_^)ノ



Intro to layouts

Layout designed by Yamini Chandra from the Noun Project



A brief history of web layouts

- Before CSS, we used **`<table>`** elements to make layouts :(
- With CSS we can use a variety of properties to arrange elements on the screen by adjusting the flow of the page
  - **Pros:** Any content can be displayed anywhere!
  - **Cons:** Any content can be displayed anywhere!

{ } Setup: Centering our page

- Most modern websites sit in the middle!
- To do this, give your **<body>** (or another container that wraps the whole page):
  - a **width** value
  - a left *and* right **margin** value of **auto**

```
body {
   width: 960px;
   margin: 0px auto;
}
```

{ } Layout properties

- **`display`**: for dictating how elements behave within the box model
- **`position`**: for moving elements in and out of the page flow altogether
- **`float`**: for moving elements around within the page flow

{ } The `display` property

- Remember block, inline, and inline-block elements?
- You can roll your own with the **`display`** property
- The most common ones are:
  - **`display: block;`**
  - **`display: inline;`**
  - **`display: inline-block;`**

{ } Why use `display`?

- Make a link look like a button
- Add padding and margins to a "naturally" inline element like a **`<span>`**
- Make a list of navigation links horizontal
- Any use cases to keep style and content separate

{ } `inline-block` layout

- **`inline-block`** creates elements that take up space but line up in a row, like **`<img>`** tags
- We can use it to make a horizontal navigation menu on our pages by applying **`display: inline-block`** to our menu's **`li`** elements

{ } `inline-block` example

- Make a list of navigation links horizontal and look button-y

```
.nav li { /* for positioning */
   display: inline-block;
   vertical-align: top;
}

.nav li a { /* for button-y-ness */
   display: block;
   padding: 20px;
   background: #5fc09a;
}
```

{ } `inline-block` layout fix

- **`inline-block`** was designed for text
  - adds a bit of space after each element for readability
  - defaults to sitting at the bottom of the line
- When using it for layouts, you can adjust:

```
.section {
    display: inline-block;
    margin-right: -4px;
    vertical-align: top;
}
```

{ } Using the `position` property

- The **`position`** property lets us arrange elements:
  - In relation to the flow (**`relative`**)
  - In a very specific place outside of the flow or within a **`relative`** element (**`absolute`**)
  - In relation to the browser window (**`fixed`**)
- How **position** is applied depends on to where the element is in the flow by default

{ } Tweaking the `position`

- We can dictate where elements go down to the pixel *within* the page, too, with **`absolute`**
- **`absolute`**ly positioned elements need a parent that is **`relative`**ly positioned

```
.child {
    position: absolute;
    right: -10px;
    top: 30px;
}
```

```
.parent {
        position: relative;
    }
```

{ } Using `position: fixed;`

- **`position: fixed`** is a way to make content "stick" to the browser window, regardless of where the user scrolls
- Commonly used to make headers, navigation, or footers that follow the page as it scrolls

{ } Floating down the river

- Our page is a flowing river of HTML elements
- Elements can be floated on the river to allow other elements to flow around them
- Most common way to make a multi-column layout

{ } `float` property

- Easiest way to offset content like images, pullquotes, or other elements within the flow of a document
- Has three values: **`left`**, **`right`**, **`none`**
- Makes elements block elements automatically

```
.titanic {
    float: none;
}
```

{ } The simplest column layout

1. Have a parent element (like **`<body>`**) and give it a width value
2. Give elements you want to be columns **`width`** values that add up to the parent's **`width`** (child A + child B = parent)
3. Make your columns **`float: left`**
4. Voila!

{ } `clear` property

- HTML river showing up in weird places?
- The **`clear`** property fixes **`float`** and also has three values:

```
clear: left;
clear: right;
clear: both;
```

{ } self-clearing floats

- One of the tricky things about floats is when to stop floating
- You can use a pseudo-element on the parent of the floated elements to create a "self-clearing" float

```
.parent:after {
  content: "";
  display: table; /* we'll talk about this shortly! */
  clear: both;
}
```

{ } `float` layout review

- The **`float`** property lets us take elements out of the main flow of the page and put them to one side (**`left`**) or the other (**`right`**)
- Floated elements get **`display: block`** applied to them by default
- Floats can be "cleared" with the **`clear`** property

{ } `display: table`

- Want a grid-like layout?
- **`display: table`** works like a table container to flexibly fill its container
- **`display: table-row`** works like a table row
- **`display: table-cell`** works like a **`<td>`** or **`<th>`**

Responsive layouts

Responsive Design designed by Nithin Viswanathan from the Noun Project

{ } Responsive/adaptive blocks

- The width of a block will adjust based on
  - how wide its **parent** is
  - how wide the **browser window** or **viewport** is
- Uses **% widths** instead of pixel widths

```
.wrapper { width: 100%; }
.main { width: 80%; }
.sidebar { width: 20% }
```

{ } Preserving readability

- What if you have a 100% wide container on a really wide screen?
- To prevent blocks from getting too wide or too narrow, you can give them a pixel **`min-width`** or a **`max-width`**

```
body {
    width: 100%; /* will be 100% of its parent... */
    max-width: 1024px; /* until it hits this! */
}
```

{ } `@media` queries

- Designed to apply different styles based on the way content is being presented
- Commonly used to style web pages for print or other alternative presentations
- Can be used to call different styles based on the size of a user's device or browser window, along **breakpoints**

{ } @media example

```
@media only screen and (max-width: 520px) {
    /* any styles for screens/browsers up to 520px wide */
    .main { width: 100%; }
    h1 { color: #fff; }
}
```

{ } another @media example

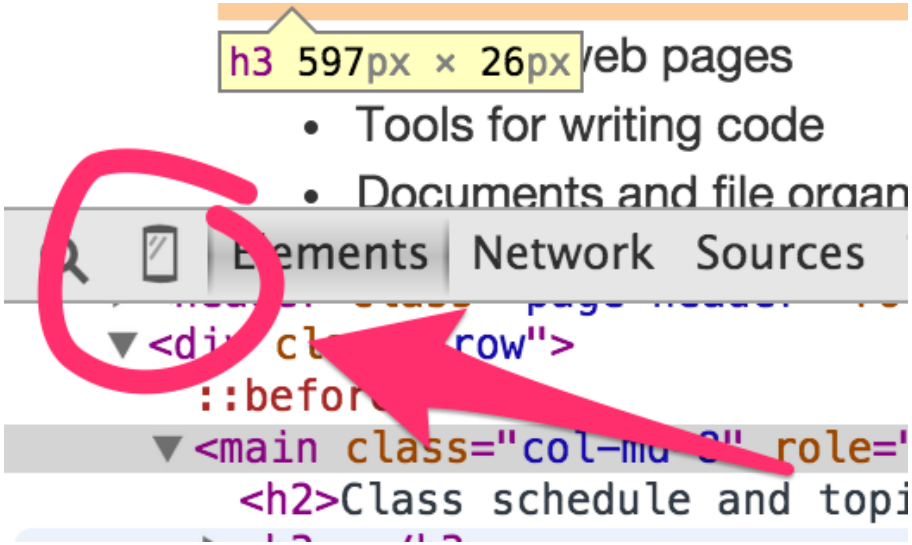
```
@media only screen and (min-width: 521px) and (max-width: 768px) {

    /* any styles for screens/browsers between 521px and 768px */

    .main { width: 80%; }

    h1 { color: #000; }

}
```

{ } Best practices for responsive

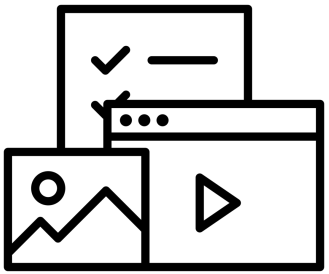
- Floated layouts are usually the easiest to make "fully" responsive
- Absolute and fixed position layouts can break down on smaller screens
- **There are no perfect breakpoints**
- Change your layout when it starts to break or look broken!

{ } Testing devices

- The best way to test across devices is to test across actual devices
- Chrome Web Inspector has a way to easily cheat though:



Embeddable content



Media designed by Garrett Knoll from the Noun Project

<> Embedded content and media

- Embedded content is what it sounds like: content, usually media, that is embedded in our HTML page
- We already know one embeddable element: the **`<img>`** tag
- Probably the next most common type of embedded content is the **`<iframe>`**

<> `<iframe>` implementation

- Used to load content from **another HTML document** into an HTML page
- iframes have a **`src`** attribute
- Commonly used to:
  - Embed media (like YouTube videos)
  - Add **social widgets** (like the Facebook Like button)
  - Load 3rd party ads on a page

<> Good practices for `iframe`

- **Include fallback HTML** in case the iframe fails to load
- **Specify the iframe's dimensions** with CSS or HTML attributes

```
<iframe src="page.html" width="200" height="400">

    If you can see this, your browser doesn't support
      iframes. <a href="example.html">Here's a direct link
      to the content.</a>

</iframe>
```

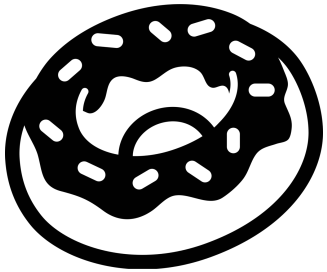
<> An example YouTube iframe

- There's very little reason to make your own iframes to include in your own pages since…you can just make your own content
- Let's drop a YouTube iframe into a page and look under the hood…

<> `<video>` and `<audio>`

- HTML5 introduced **`<video>`** and **`<audio>`** embeddable elements (and others)
- Adds **default playback controls** that can be managed with Javascript
- Can fall back to Flash media
- The current trend for "background" videos? Those are HTML5 videos!

Extra goodies

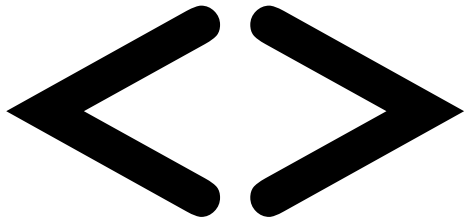


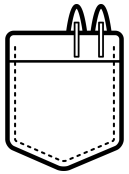
Donut designed by Jacob Halton from the Noun Project

Topics related to HTML and CSS

- New HTML5 containers
- Javascript
- More CSS
- Libraries and frameworks
- Mobile-first thinking
- Accessibility
- Version control

New HTML5 containers





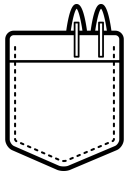
Pre-HTML5 structure

- Previously, HTML only had semantic tags for content (ex.: **`<h1>`**, **`<p>`** and **`<td>`**)
- We faked it for containers by using **`<div>`** elements with semantic-sounding **`id`** or **`class`** attributes (ex.: **`id="header"`**)
- Designed to "futureproof" and create semantic structure for chunks of content

Pocket designed by Michela Tannoia from the Noun Project

<> Major HTML5 containers

- **`<header>`**: header of a container
- **`<nav>`**: navigation links
- **`<main>`**: primary content
- **`<section>`**: a group of related content
- **`<article>`**: what is says on the tin
- **`<aside>`**: supportive, non-primary stuff
- **`<footer>`**: footer of a container



Support for older browsers

- IE9 and other older browsers have no native support for shiny new HTML5 tags
- These browsers can be ~~tricked~~ *gently coaxed* into displaying and styling new HTML5 elements via Javascript
- The most popular method is the **HTML5 shim**: https://code.google.com/p/html5shim/

Installing the HTML5 shim

1. Download the shim zip file and unzip it
2. Find the **`html5shiv.js`** file, and move it to a **`js`** directory in your files
3. Inside the **`<head>`** element of all your pages, add:

```
<!--[if lt IE 9]>
   <script src="js/html5.shiv.js"></script>
<![endif]-->
```

Toolset designed by Calvin Goodman from the Noun Project

Notes on using HTML5 containers

- HTML5 is an experiment in process and documentation!
- Elements will come and go
- When in doubt, use an online resource like **http://html5doctor.com/** (these will be more up to date than books)
- If you're not sure, use a **`<div>`**!

Javascript

- The third pillar of the web along with HTML and CSS
- Embedded into an HTML document with the **`<script>`** tag
- Allows for additional interactivity and data manipulation that isn't possible with HTML and CSS alone

Javascript use examples

- Hiding, showing, moving, etc., content based on user actions
- Displaying controls for HTML5 media
- Drawing content on the screen based on data (ex.: Chart.js)
- Collecting data about the type of browser, device, and internet connection a user has

Javascript libraries

- A set of **pre-made scripts**
- A platform for **common user interface patterns**
- Designed to work out of the box
- Designed to work with plugins and other libraries to provide extra functionality
- Probably the most common is **jQuery**

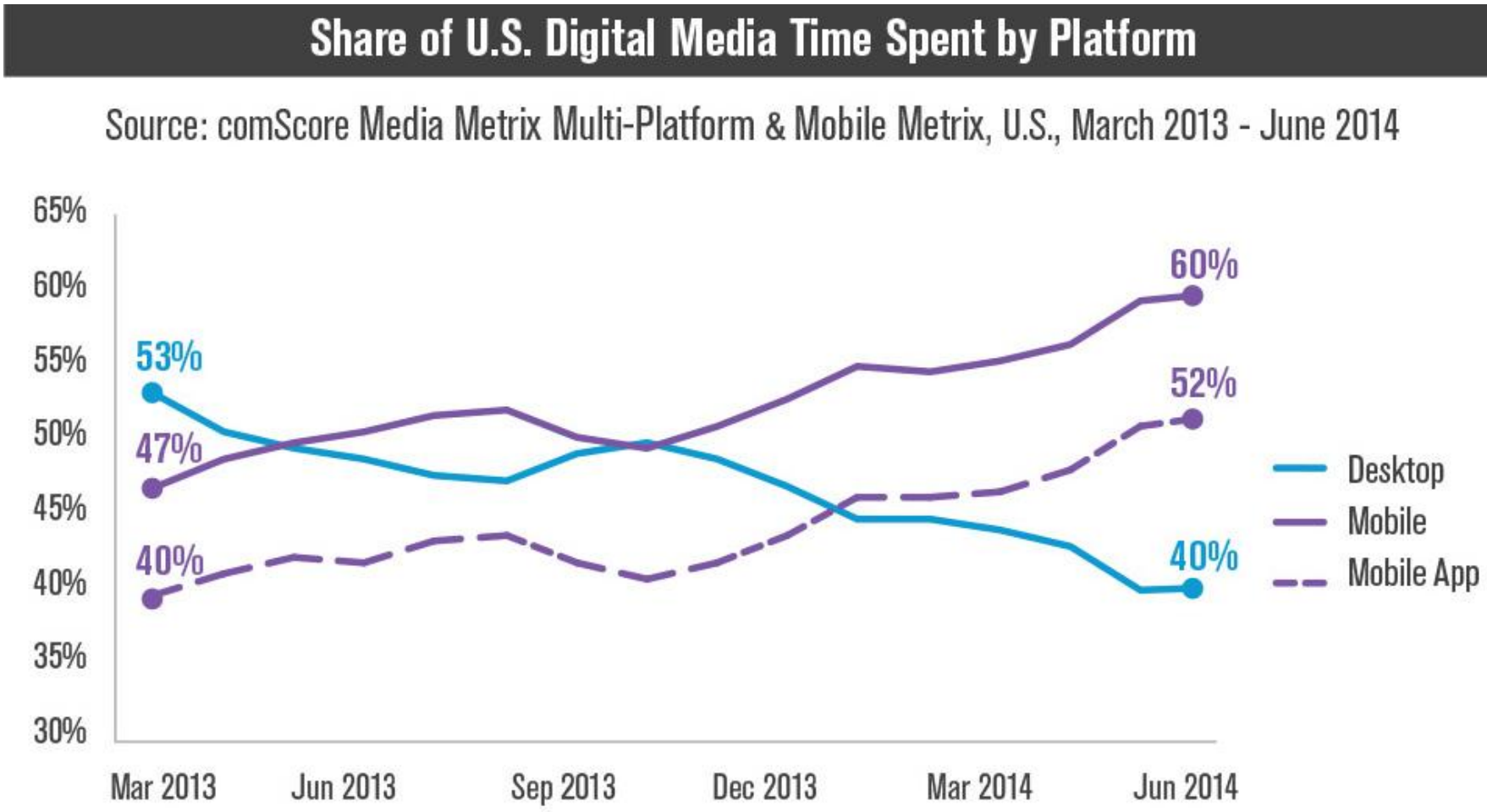
Javascript and CSS frameworks

- A set of **pre-made scripts and styles** for quickly prototyping or iterating on projects
- **Heavily tested** and **prevents having to roll your own** Javascript and styles to complete a common task
- One of the most common is **Bootstrap**

Mobile and tablet-first

- Means thinking about scaling up using **progressive enhancement**
- **Defining the base experience that can work on a smartphone** and add enhancements to tablets, then laptops and desktops
- Only add bells and whistles **when a system can more easily support them**

Why think mobile-first?



"Majority Of Digital Media Consumption Now Takes Place In Mobile Apps," Tech Crunch, August 21, 2014

Web accessibility (a11y)

- **Web accessibility** is about providing support for people with:
  - Blindness and low vision or color-blindness
  - Deafness
  - Issues with motor skills
  - Issues with learning, remembering, and paying attention
- ~20% of people have some kind of disability that affects their daily life

Developing for a11y

- **Logical content order** and **semantic HTML**
- Media **alternatives** (ex.: `alt` attributes)
- **Keyboard** focus (`:focus`) and interactions
- Sufficient **color contrast**
- W3C's **WCAG 2.0 guidelines**

CSS3 and CSS4

- **CSS3+4** techniques add extra **refinements**, **depth**, **transitions**, **animations**, **rotations**, and **typography**
- Frequently **combined with Javascript**
- Range from simple (rounded corners) to **full-blown interactive experiences** previously only possible with Flash or Javascript alone

Fancy CSS examples

CSS-only animated gradient text

CSS-only bounce animation

CSS-only typing animation

CSS-only animated pixel bunny

CSS, SVG, Javascript roller coaster

Hosting your code

- **Wordpress:** Wordpress.com or WP Engine
- **Squarespace**: Templated portfolio-type sites
- **Shopify**: Templated online stores
- **Github.io**: Hosted, version-controlled pages

Version control for code

- **Version control** is a method of storing versions of files in a **repository**
- Helps **prevent** accidental deletions, additions, mistakes, and errors in live code for you
- **Tracks and manages conflicts** between files
- Common systems are **git** and **svn**

Version control integration

- Version control lets us (more) safely share code between developers and collaborate on projects
- Can be integrated into systems for deploying code onto live sites
- For example, the code for our class site is stored online in GitHub, and the site is served with GitHub Pages

Before we go...

1. Visit www.svcseattle.com/evaluation/
2. Choose **"HTML and CSS - Level 1 (May Session) (Spring 15)"** from the dropdown
3. Fill out the evaluation
4. Please be honest and constructively critical!

Thank you!

It was really fun.