HTML & CSS: Week 5

March 10, 2014

Instructor: Devon Persing

This week

- Data tables
- Layout reviewed (and improved!)
- iframes and media
- Hosting your code
- Fun stuff to learn next and questions
- Course evaluation

Data tables

<> What's a `table` good for?

- Presenting data in a tabular format
  - Listings of people, addresses, etc.
  - Financial data
  - Product feature comparisons
- HTML emails :(

<> Basic table elements

- **`<table>`** wraps all **table** elements
- **`<tr>`** creates a **row** of table cells
- **`<th>`** creates a **table header** cell for a column *or* a row
- **`<td>`** creates a regular **table data** cell within a row

<> A basic table

```
<table>
    <tr>
        <th>Column 1 Header</th>
        <th>Column 2 Header</th>
    </tr>
    <tr>
        <td>Column 1 Data Cell</td>
        <td>Column 2 Data Cell</td>
    </tr>
</table>
```

<> `<th>` attributes

- For accessibility, it's good practice to use a scope attribute for table header cells if there are row headers:
  - **`scope="col"`** for table headers that describe a column
  - **`scope="row"`** for table headers that describe a row
- Creates an explicit connection for data cells that have multiple headers

<> A table with scope attributes

```
<table>
    <tr>
        <th scope="col">Column 1 Header</th>
        <th scope="col">Column 2 Header</th>
        <th scope="col">Column 3 Header</th>
    </tr>
    <tr>
        <th scope="row">Row 2 Header</th>
        <td>Row 2, Column 2 Data Cell</td>
        <td>Row 2, Column 3 Data Cell</td>
    </tr>
</table>
```

{ } Styling table elements

- All of our box model styles can be applied!
- Some additional styles for cells:
  - **`border-spacing`** puts space between cells

    ```
    table { border-spacing: 4px; }
    ```

  - **`border-collapse`** makes cell borders overlap or sit up against each other each other

    ```
    table { border-collapse: collapse; }
    ```

Layouts reviewed and improved

Responsive Design designed by Nithin Viswanathan from the Noun Project

{ } `inline-block` layout

- We used it to make a horizontal menu on our pages by applying **`display: inline-block;`** to our menu's **`li`** elements
- **`inline-block`** makes elements take up space but line up in a row
- You can use **`inline-block`** for whole containers to make column layouts too

{ } `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;
}
```

{ } `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

{ } self-clearing floats

- You can use a **pseudo-element** on the **parent of the floated elements** to create a "self-clearing" float
- It's like a friendly ghost that helps your layout!

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

{ } Using the `position` property

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

{ } Using `position: fixed;`

- **`position: fixed;`** makes content "stick" to the browser window, regardless of where the user scrolls
- Commonly used to make headers, navigation, or footers that stick

```
.nav {
position: fixed;
top: 0;
left: 0;
}
```

{ } 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;
    }
```

{ } `display: table;`

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

{ } 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

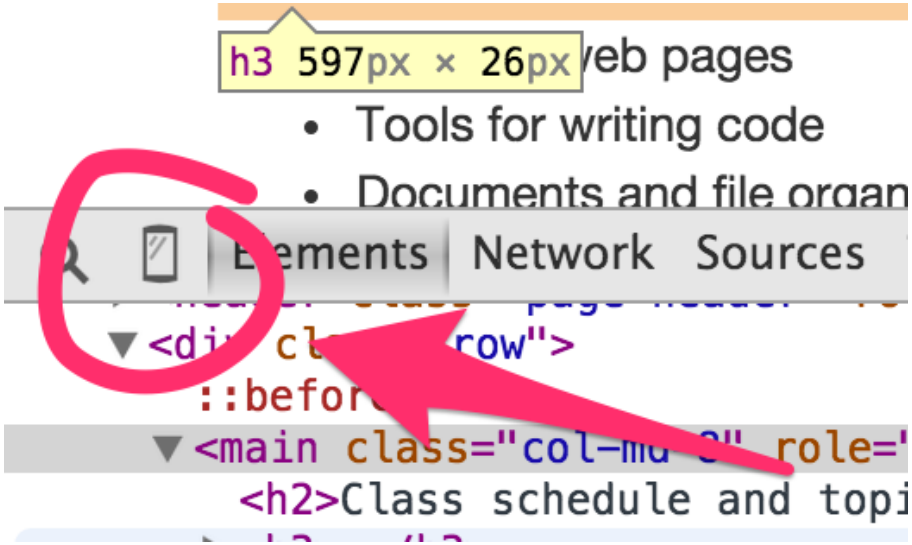
```
@media only screen and (min-width: 521px) and (max-width: 768px) {

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

    .main { width: 80%; }

    h1 { color: #000; }

}
```

{ } Best practices for responsive

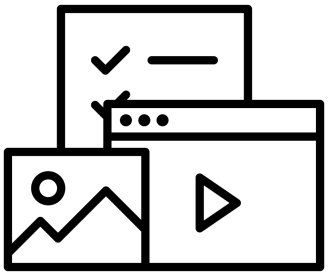
- Floated layouts are usually the easiest to make "fully" responsive
- Absolute and fixed position layouts can break down on smaller screens
- **There are no perfect breakpoints**
- Change your layout when it starts to break or look broken!

{ } Testing devices

- The best way to test across devices is to test across 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…

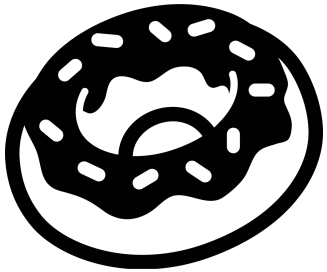
<> `<video>` and `<audio>`

- 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!

Hosting your code

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

Extra goodies

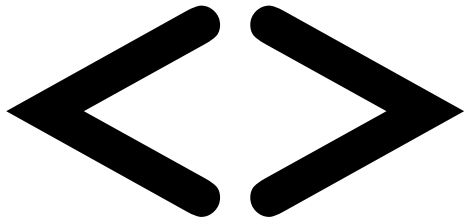


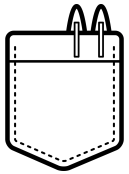
Donut designed by Jacob Halton from the Noun Project

Topics related to HTML and CSS

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

New HTML5 containers





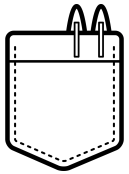
Pre-HTML5 structure

- 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**

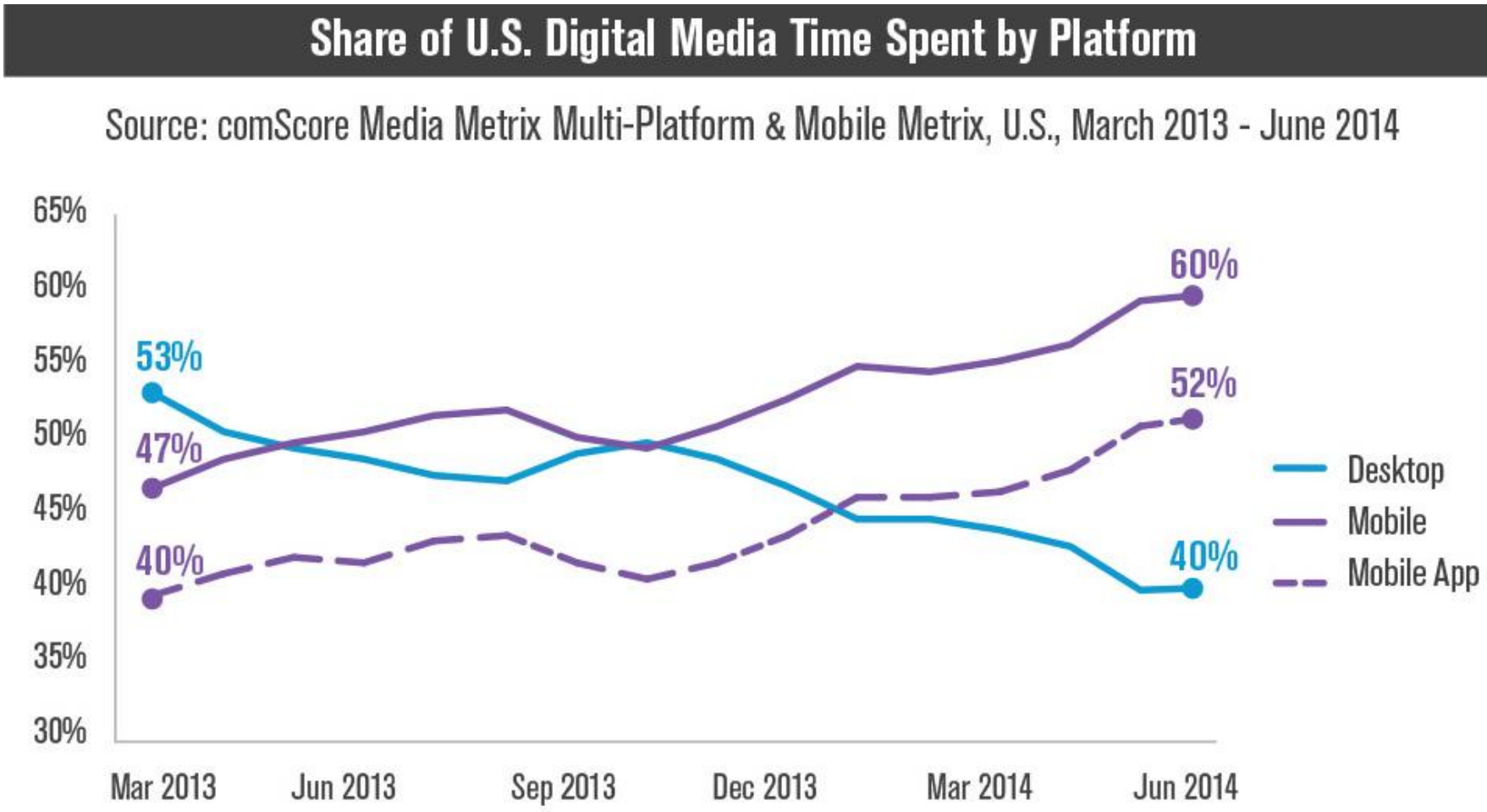
Javascript and CSS frameworks

- 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

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 (February Session) (Winter 15)"** from the dropdown
3. Fill out the evaluation
4. Please be honest and constructively critical!

Thank you!

It was really fun.