Weeks 2-3: Responsive Web Design (RWD)

RWD overview

Powerpoint slides to introduce RWD problems, 3 components

Configuring RWD web design

We’ll be converting a web page from a standard fixed-width layout to RWD over 2 lectures.

Step 1: Initial HTML configuration

Responsive meta tag: http://css-tricks.com/snippets/html/responsive-meta-tag/

normalize.css: Normalizes margins/padding and other browser defaults for all browsers. Does not eliminate them like a reset stylesheet might.
http://necolas.github.io/normalize.css/

Box model: Can use traditional box model or hack it with border box
http://www.paulirish.com/2012/box-sizing-border-box-ftw/

```
html {
  box-sizing: border-box;
}
*, *:before, *:after {
  box-sizing: inherit;
}
```

Browser support issues for box-sizing may also be a thing to deal with, depending on what browsers you must support — see http://caniuse.com/#search=box-sizing

In border-box, the CSS width includes border and padding, but not margin. The standard box model includes padding, margin, and border.

For example, if you have a box with 20px margin, 10px padding, and 2px border around it on all 4 sides, with a width: 200px; set, the total box width under the traditional content-box model is 264px (20px margin left and right, plus 10px padding left and right, plus 2px border left and right, plus 200px).

Under the border-box model, the content width is not 200px but 176px. That’s 200px minus 10px padding left and right, minus the 2px border left and right. The overall width of the box here is 240px, which includes 200px (content, border, padding) plus the margin (20px left and right).

Step 2: convert sizes from pixels to ems, convert to a fluid layout

Units in RWD: usually em’s, but can use other units (typically % or px): http://pxtoem.com/
Convert almost all pixel dimensions to % or em. In general, use em where the value will be impacted by different font sizes. Use % for big page dimensions or other major dimensions on the page. (Generally I use for widths of containers.)

Images for div img and main img are enhanced by fluid dimensions:
width: 100%;
max-width: 100%;

This allows the image to display at 100% of its width and no wider. This prevents the image from leaving its container, overlapping with other images, etc. This scales the HTML dimensions of the image — it does NOT reduce file size of the image for smaller dimensions. This is NOT "responsive images", which we cover later. This is a little hack that will make images behave for the moment while we style around them.

```
body { font-size: 100%; }

wrapper {
width: 97%;
max-width: 1200px;
}

header {
width: 100%;
}

header h1 {
padding: 0.5em;
}

nav ul a {
padding: 1em;
}

main {
padding-right: 3em;
}

main h2 {
font-size: 3em;
padding-top: 1em;
}

main p {
font-size: 1.5em;
}
```

```
ADD: div img {
width: 100%;
max-width: 225px;
}

main img {
margin-right: 1.875em;
width: 100%;
max-width: 400px;
}

div {
margin-left: 2.5em;
}

footer {
padding: 2em 0;
margin: 3em 0;
}
```

Step 3: Configuring a fixed grid system

For the moment, switch from index.html to grid.html. Grid.html will serve as a prototype for some new HTML and CSS in this step. We'll then fit this code back to index.html.

We're stepping back to a simpler HTML design to think about the grid system. I've chosen the most complex row in the design to prototype here, along with a few potential variations I want to plan for.

We are building a 4 column grid system. Just because you can build 12 columns doesn't mean you should! If you want to build 12 columns, take the work for 4 columns and expand. (Yeah, more math.) You don't have to build 12 columns unless you think you'll need them. That's the advantage of a fully custom design!

HTML has changed as follows:

```
<section class="row">

   <div class="col-1”>…</div>
   <div class="col-1”>…</div>
   <div class="col-1”>…</div>
   <div class="col-1”>…</div>

</section>
```

Rows have been defined, each holding up to 4 columns. Rows contain the clear to balance the floats used in each column.

Each row should contain 4 columns. That can be 4 col-1's, 2 col-2's, a col-1 and a col-3, a col-1, col-2, and col-1, or a col-4.

Why <section>? No reason in particular, except it makes the code easier to read while we work on this mini-example. This way we avoid many layers of nested div's, which gets hard to read. Nope, not very semantic, but very practical when learning. In "real life", I'd recommend using div's for your rows unless you have a compelling reason to use a different tag.

In the CSS, we add a clear for the rows http://css-tricks.com/snippets/css/clear-fix/

Then style each column. Based on settings for the single column, we can calculate the additional widths for the other columns. For example, for a 2 column width, add 20.8333333% plus 20.8333333%, plus 3.333333% which was in between the two columns.

Background colors were added to make it visually clear in the browser which class is activated on which row.

Note the grid is NOT currently responsive — there are no media queries.

CSS added:

```
/* grid system */
/* row class will clear floats from previous row */
.row:after {
    content:"";
    display: table;
    clear:both;
}
[class*='col-'] {
      float: left;
      margin-left: 3.33333%;
      min-height: 1px;
}
.col-1 {
      width: 20.83333%;
      background-color: #ccf;
}
.col-2 {
      width: 44.99999%;
      background-color: #cff;
}
.col-3 {
      width: 69.16666%;
      background-color: #ffc;
}
.col-4 {
      width: 93.33333%;
      background-color: #fcf;
      margin-right: 3.33333%;
      }
```

Phase 4: Media Queries

Now that we have a grid system in place, we're ready to make it responsive.

Media query background, including syntax and how they can be used (it's more than pixel breakpoints!): https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

Great tutorial: http://css-tricks.com/css-media-queries/

The usual question now is where should breakpoints be placed? A breakpoint is the point where CSS rules either start to apply or stop being applied to the page.

Here's some typical media queries used with a 960px grid system: http://www.sitepoint.com/build-a-responsive-design-using-960-grid/

```
.img {width:100%;}
```

```
/* Everything before the media queries gets
applied to screen sizes greater than 960 pixels */

/****************/
/* MEDIA QUERIES */
/****************/

/* Tablet Screen Sizes */
@media only screen and (min-width: 768px) and (max-width: 959px)
{ … }

/* Mobile Landscape Screen Sizes */
@media only screen and (min-width: 480px) and (max-width: 767px)
{ … }

/* Mobile Portrait Screen Sizes */
@media only screen and (max-width: 479px) { … }
```

Note that this design is not “mobile first” — it builds the desktop layout, then makes the layout progressively smaller.

Bootstrap uses the following media queries (which are mobile first):

```
/* Extra small devices (phones, less than 768px) */
/* No media query since this is the default in Bootstrap */

/* Small devices (tablets, 768px and up) */
@media (min-width: 768px) { ... }

/* Medium devices (desktops, 992px and up) */
@media (min-width: 992px) { ... }

/* Large devices (large desktops, 1200px and up) */
```

```
@media (min-width: 1200px) { ... }
```

What are the right media queries for you? It depends on your design and how things look. Also, mobile first or “desktop first” may also be the right approach for you.

(Why did we use “desktop first”? We’re retrofitting a site that was already created with new data to make it work on a desktop. If you want to build with “mobile first”, the procedure is the same as described in this tutorial, but your default styles are the mobile settings. After that, add media queries for when the page is wider.)

We start by putting in these media queries and styles:

```
/* media queries */

/* Mobile Landscape Screen Sizes */
@media only screen and (min-width: 480px) and (max-width: 767px)  {
[class*='col-'] {
      float: left;
      margin-left: 3.33333%;
      min-height: 1px;
}
 .col-1 {
      width: 44.99999%;
      background-color: #ccf;
}
.col-2 {
      width: 44.99999%;
      background-color: #cff;
}
.col-3 {
      width: 93.33333%;
      background-color: #ffc;
      margin-right: 3.33333%;
      }
.col-4 {
      width: 93.33333%;
      background-color: #fcf;
      margin-right: 3.33333%;
      }
}

/* Mobile Portrait Screen Sizes */
@media only screen and (max-width: 479px) {
      [class*='col-'] {
            width: 100%;
            margin-left: 0;
            min-height: 1px;
      }
      .col-1 {
            background-color: #ccf;
      }
      .col-2 {
            background-color: #cff;
      }
      .col-3 {
            background-color: #ffc;
      }
      .col-4 {
            background-color: #fcf;
      }
}
```

Note that the widths and layouts will change with the structure of the page.

However, the above layouts don’t work well with a 1-3 layout. In this case, we may want to differentiate .col-1 into two types of situations. .col-1-2 will go from a single column at large sizes to spanning 2 columns at smaller sizes. .col-1-4 goes from occupying one column at large sizes to occupying all 4 columns at small sizes. This gives us more flexibility in layout for smaller screens and results in the following changes to CSS (maintain the other styles for columns 2-4):

```
/* media queries */


/* Mobile Landscape Screen Sizes */
@media only screen and (min-width: 480px) and (max-width: 767px)  {
 .col-1-2 {
      width: 44.99999%;
      background-color: #ccf;
}
 .col-1-4 {
      width: 93.33333%;
      background-color: #ccf;
      margin-right: 3.33333%;
}
}
```

```
/* Mobile Portrait Screen Sizes */
@media only screen and (max-width: 479px) {
  .col-1-2,
  .col-1-4 {
      background-color: #ccf;
}
}
```

Phase 5: Apply grid and media queries to the original page design. Tweak CSS as required.

Now that we have prototyped some grid settings and media queries that work with our most complex row in our design, let's try applying these settings to the rest of the page.

First, add the "default" grid settings. These should be applied around line 101 (at the end of the current stylesheet) and include style for the row and the way the page will look for large desktop layouts.

Remove the old div class that was originally in the CSS, or the page look will be a disaster!

```
/* row class will clear floats from previous row */
.row:after {
    content:"";
    display: table;
    clear:both;
}
[class*='col-'] {
      float: left;
      margin-left: 3.33333%;
      min-height: 1px;
}
.col-1-2,
.col-1-4 {
      width: 20.83333%;
}
.col-2 {
      width: 44.99999%;
}
```

```
.col-3 {
    width: 69.16666%;
}
.col-4 {
    width: 93.33333%;
    margin-right: 3.33333%;
}
```

Next, add our two media queries:

```
/* Mobile Landscape Screen Sizes */
@media only screen and (min-width: 480px) and (max-width: 767px)  {
    [class*='col-'] {
        float: left;
        margin-left: 3.33333%;
        min-height: 1px;
    }
     .col-1-2 {
        width: 44.99999%;
    }
     .col-1-4 {
        width: 93.33333%;
    }
    .col-2 {
        width: 44.99999%;
    }
    .col-3 {
        width: 93.33333%;
        margin-right: 3.33333%;
    }
    .col-4 {
        width: 93.33333%;
        margin-right: 3.33333%;
    }

}

/* Mobile Portrait Screen Sizes */
@media only screen and (max-width: 479px) {
    [class*='col-'] {
        width: 100%;
        margin: 0;
        min-height: 1px;
    }
}
```

Check the design. Are these default breakpoints still OK, or do they need tweaking?

The “Discover Arlington” box doesn’t look too awesome… maybe some additional tweaking is needed? We can reduce the font size as the screen get smaller, for example. (We’ll deal with resizing this image the right way in phase 7.)

```
/* adjust "discover" box for smaller width */
@media only screen and (min-width: 702px) and (max-width: 866px)  {
main h2 {
	font-size: 2.5em;
	padding-top: 0em;
}
main p {
	font-size: 1.5em;
	font-family: 'Cabin Condensed', sans-serif;
}
}
```

OK, better… maybe time to add another tweak for smaller screens? And how about that nav bar? You could add some JavaScript to make that appear/disappear (maybe behind a Hamburger Button!), or you could shorten it using the styles below.

```
/* adjust "discover" box again and change nav layout */
@media only screen and (max-width: 701px)  {
main {
	padding: 0 3.333333%; /* aligns with boxes underneath */
}
main h2 {
	font-size: 2em;
	padding-top: 0em;
}
main p {
	font-size: 1.2em;
	font-family: 'Cabin Condensed', sans-serif;
}
main img {
	float: none;
	margin: 0;
	width: 100%;
	max-width: 400px;
}
nav ul a {
	display: block;
	float: none;
	padding-left:3.333333%;
}
/* splits nav into 2 columns
http://stackoverflow.com/questions/13104818/simple-2-column-
navigation-with-css-and-a-single-list */
nav li:nth-child(even) {
  width: 50%;
  float: right;
}
nav li:nth-child(odd) {
  width: 50%;
  float: left;
}
}
```

Phase 6: Using media queries to load the right background image, without loading all options

It's probably best to go read this article first.
http://timkadlec.com/2012/04/media-query-asset-downloading-results/

This describes a series of tests to find out the best way of handling multiple images within a web page with media queries. The goal is to have only the image relevant for the current dimensions to download. Remember with most CSS that you write, you'll arrange it so only one image shows — that's not hard. But making sure only one image downloads is tricky!

In addition to the following styles in this media query, add these:

```
@media only screen and (max-width: 701px)  {
header {
	background: url(../img/boston-710.jpg) no-repeat;
	width: 100%;
	height: 149px;
}
header h1 {
	padding: 100px 0 0 10px;
	margin: 0;
	font-size: 1.6em;
}
header h1 a {
	color: #fff;
	text-decoration: none;
}
}
```

For the larger version of the header, this is currently specified in the portion of the stylesheet without media queries (around lines 29-43). We need to remove these styles from this part of the stylesheet and put them in an appropriate media query. Otherwise, the boston-1200.jpg image may always load, even if it's then overridden by the media query for a smaller image.

Add this media query:

```
@media only screen and (min-width: 702px)  {
header {
	background: url(../img/boston-1200.jpg) no-repeat;
	width: 100%;
	height: 218px;
	font-family: 'Cabin Condensed', sans-serif;
	color: #fff;
}
header h1 {
	padding: 0.5em;
	margin: 0;
}
header h1 a {
	color: #fff;
	text-decoration: none;
}
```

Phase 7: Using responsive images with Picturefill

Choosing a Responsive Image Solution
http://www.smashingmagazine.com/2013/07/08/choosing-a-responsive-image-solution/

Which Responsive Image Solution Should You Use
http://css-tricks.com/which-responsive-images-solution-should-you-use/

PictureFill: A responsive image polyfill
http://scottjehl.github.io/picturefill/

PictureFill makes use of the <picture> element, a new tag not supported with many browsers (yet). PictureFill uses a polyfill to make the browser respond to this tag. We are working with PolyFill 2.1, which offers native <picture> support — a great way to make your page future-forward.

1. Download PictureFill from above link and place in a js folder.

2. Add the following code to the HTML page just before </head>

```
	<script>
		// Picture element HTML5 shiv
		document.createElement( "picture" );
	</script>
```

```
      <script src="js/picturefill.min.js" async></script>
```

3. Change the HTML in the document for <main>, starting around line 33:

```
<main class="col-4">
  <!-- Substituting image using Picturefill
  <img src="img/townhall-400.jpg" alt="Welcome to Arlington!"> -->

  <picture>
      <!--[if IE 9]><video style="display: none;"><![endif]-->
      <source srcset="img/townhall-400.jpg" media="(min-width:
1014px)">
      <source srcset="img/townhall-300.jpg" media="(min-width: 702px)
and (max-width: 1013px) ">
      <!--[if IE 9]></video><![endif]-->
      <img srcset="img/townhall-188.jpg" alt="Welcome to Arlington!">
  </picture>

  <h2>Discover the world of Arlington!</h2>
  <p>Beautiful parks, great schools, and a strong sense of history
makes Arlington one of the great places to live in eastern
Massachusetts.</p>
  <p><a href="http://www.arlingtonma.gov" target="_blank">Read more
&gt;&gt;</a></p>
</main>
```

4. Our CSS will cause this JS to break! The problem is here:

```
main img {
      float: left;
      margin-right: 1.875em;
      width: 100%;
      max-width: 400px;
}
```

This is causing the images to take up more width than you'd expect. Comment out this style (or delete it) and change it to this:

```
main img {
      width: auto;
      max-width: auto;
      float: left;
      margin-right: 1.875em;
}
```

Why the auto settings for width and max-width? Remember the div img style applies to the main image as well, since main is inside a div!

5. The media query for the <main> section of the page isn't working for me with the newly subbed images.

Change

```
@media only screen and (min-width: 702px) and (max-width: 866px)  {
```

to

```
@media only screen and (min-width: 1014px) {
```

Change

```
@media only screen and (max-width: 701px)  {
main {
	padding: 0 3.333333%; /* aligns with boxes underneath */
}
```

to

```
@media only screen and (max-width: 1013px)  {
```

(get rid of the main padding — it doesn't work anymore!)

Also within this media query, get rid of main img styling - this conflicts with PictureFill (same as described above)

So the new media query reads like this:

```
/* adjust "discover" box again  */
@media only screen and (max-width: 1013px)  {
main h2 {
	font-size: 2em;
	padding-top: 0em;
}
main p {
	font-size: 1.2em;
	font-family: 'Cabin Condensed', sans-serif;
}
/* main img {
	float: none;
	margin: 0;
	width: 100%;
	max-width: 400px;
} */
}
```

Maintain the rest of the old media query, which should look like this:

```
/* adjust  nav layout */
@media only screen and (max-width: 701px)  {
nav ul a {
	display: block;
	float: none;
	padding-left:3.333333%;
}
/* splits nav into 2 columns
http://stackoverflow.com/questions/13104818/simple-2-column-navigation-with-css-and-a-single-list */
nav li:nth-child(even) {
  width: 50%;
  float: right;
}
nav li:nth-child(odd) {
  width: 50%;
  float: left;
}
header {
	background: url(../img/boston-710.jpg) no-repeat;
	width: 100%;
	height: 149px;
}
header h1 {
	padding: 100px 0 0 10px;
	margin: 0;
	font-size: 1.6em;
}
header h1 a {
	color: #fff;
	text-decoration: none;
}
}
```

Finally, I'd like to modify the styling for the smallest mobile version with the new picture. To that mobile media query (max width 479px), add the following:

```
main {
	padding-right: 1em;
}
main h2 {
	font-size: 1.5em;
	padding-top: 0em;
}
main p {
	font-size: 1em;
	font-family: 'Cabin Condensed', sans-serif;
}
main img {
	padding-right: 0em;
}
```

Phase 8: Bonus! Fix those columns that just won't go to the same height

I know those columns with the equal-height problem have been driving you CRAZY. I wanted to be sure to get through all of the most important material first, so if there's time, I'll show you how you can make your columns equal height and get around the wrapping problem the grid is introducing.

1. Download grids.min.js from here:
https://github.com/Sam152/Javascript-Equal-Height-Responsive-Rows

Save in your js folder.

2. You will need jQuery to make this code work. Also, grids.min.js must be in the head of the document, not near </body>. Therefore, we'll load jQuery in the head of the document as well.

Link to jQuery and to grids.min.js BEFORE the PictureFill code, around lines 10-11.

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>"
<script src="js/grids.min.js"></script>
```

3. Select the elements that need to be equal height. In this case, each of our elements has a class of col-1-2, so let's leverage that to bind the grids JS.

```
		jQuery(function($) {
			$('.col-1-2').responsiveEqualHeightGrid();
		});
```

This can be combined with the <script> tag that already exists for PictureFill:

```
	<script>
		jQuery(function($) {
			$('.col-1-2').responsiveEqualHeightGrid();
		});
		// Picture element HTML5 shiv
```

```
        document.createElement( "picture" );
    </script>
```

Grid reordering overview

Look at inside.html. This is the design for an inside page to match the home page we've been working with.

I have listed the styles specific to this page as an additional stylesheet called inside.css. It's more efficient to combine these styles with those from the home page, but you can leave them as separate files if you wish.

This has a left aside bar with some facts about Arlington. The main content is on the right side of the page. The layout is a 1 col/3 col split.

When the browser width is reduced, the left column stacks on top of the right – exactly what you'd expect to do. But is this good user experience?

Probably not for a mobile device. Ideally, you'd see the main content first (the article tag), and after that would be the left column information (the aside tag).

We can rearrange the HTML such that the article comes first and the aside comes second. Now our "Arlington by the Numbers" feature is on the right side of the screen at desktop resolutions, but it's stacking in the right order at mobile resolutions.

But Marketing demands a left column on the desktop, but the left column should go to the bottom on mobile. What's a developer to do?

1: Reorder the HTML

In the HTML ordering, place the article first, and the aside second. This ensures that the stacking is correct on mobile devices. The source order is also correct for a search engine, which will read the article before the aside.

```
<article class="col-3">
    <h2>About Samuel Whittemore</h2>
     <img src="img/whittemore-300.jpg" alt="Monument to Samuel
Whittemore." class="">
     <p>Samuel Whittemore served in the British Army for many years,
but on April 19, 1775, at the age of 80, he was on the side of the
American Revolution.</p>
…
</article>
<aside class="col-1-4">
    <h3>Arlington by the Numbers</h3>
    <ul>
```

```
	<li>Arlington population, <a
href="http://quickfacts.census.gov/qfd/states/25/2501640.html"
target="_blank">2010 census</a>: 42,844</li>


…
</aside>
```

2: Set the columns to a position of relative

Relative positioning: positions elements from a position relative to their containing element. Element retains the shape it would have had if it were not positioned. The space the element normally occupies is preserved.

By default, our content doesn't move at all with a position of relative. We have not set the properties of left, top, bottom, or right, so the content stays in place. However, we now have the option to move it.

```
[class*='col-'] {
	position: relative;
}
```

3: Create new push and pull classes

Frequently these types of classes that reorder our grid are referred to as "push" and "pull". Imagine standing on the left side of your web page, looking at the row containing the aside and the article. To reorder these at the desktop dimension, what you want to do is push the article to the right by one column, then pull the aside over to the left by 3 columns. That would change the look of the page, even though the HTML would be in the opposite order inside.

The HTML in the article and aside tags changes to this:

```
<article class="col-3 col-push-1">
<aside class="col-1-4 col-pull-3">
```

The corresponding CSS looks like this:

```
.col-push-1 {
	left: 24.16666%;
/* push over 1 column, 20.83333% plus one margin of 3.3333% */
}
.col-pull-3 {
	left: -72.49999%;
/* pull over 3 columns, 69.1666% plus one margin of 3.3333% */
}
```

These classes are default desktop classes, so they should NOT be in a media query.

Look at your web page – looks great on a desktop!!! Shrink the browser down – how does it look on smaller devices?...

OK, so we need to fix that.

If you look at the media queries we have, for the classes .col-3 and .col-1-4, they have two display options – a desktop option (where these are next to each other) and a tablet AND mobile option (where they stack vertically).

Therefore, we can write a media query to address both tablet and mobile for these classes, or we can slot these styles into the existing queries:

```
/* Mobile Landscape Screen Sizes */
@media (max-width: 767px) {
  .col-push-1 {
    left: 0;
  }
  .col-pull-3 {
     left: 0;
  }
}
```

These styles reset the upper left corner of these classes to the left side of the mobile screen, so they are not displaced from the side of the page.