HTML5 and CSS

Responsive Design

Introduction

It's a multi-device world. People view Web content on smartphones, tablets, laptops, and more. Modern websites have to do more than "work" in different-sized devices; they have to be inviting and accessible in those devices. Today, we'll implement responsive design —in other words, Web pages that change their looks based on a device's viewing size. In responsive design, Web pages "respond" to a user's viewport, which is the size of the browser window on the device he or she is using. It's possible for responsive designs to be simple—using different-sized images for mobile phones and full-sized laptops, for example. Or it can mean a radically different design for different devices . . . specifically, creating truly inviting, accessible mobile versions of Web pages.

The Elements of Responsive Design

Let's survey key elements in an inviting, accessible mobile site:

A high-contrast color scheme: This is readable in bright, outdoor light on a device that has weaker backlighting than you'd find on a laptop or desktop monitor.

Faster-loading images: These allow pages to open more quickly in slower 3G or 4G mobile connections. (You can reduce image downloading time by using smaller images or fewer colors.)

No hovering: Responsive sites avoid horizontal menu bars that require a visitor to hover over a menu item to see a submenu.

Larger type: On a full-sized monitor screen, 12-pixel type displays at something like a sixth of an inch in height, but that same 12-pixel type on a higher-resolution mobile device screen might well reduce the size of the type to a quarter of that size.

Larger links: These are easier for users to identify and click. The best way to make links accessible in mobile devices is to use large, clickable buttons.

Few columns: Three-column layouts are typical for pages targeted to full-sized screens, but many times, the best design approach for a smartphone is to put everything in a single column. Responsive page designs avoid too many columns.

This list isn't comprehensive, but it gives you a sense of how different mobile design is from creating pages for laptops and desktops.

What Does Responsive Design Look Like?

The website of Time magazine (www.time.com) is a good example of responsive design. Visitors using full-sized browsers see a page layout built around three columns.

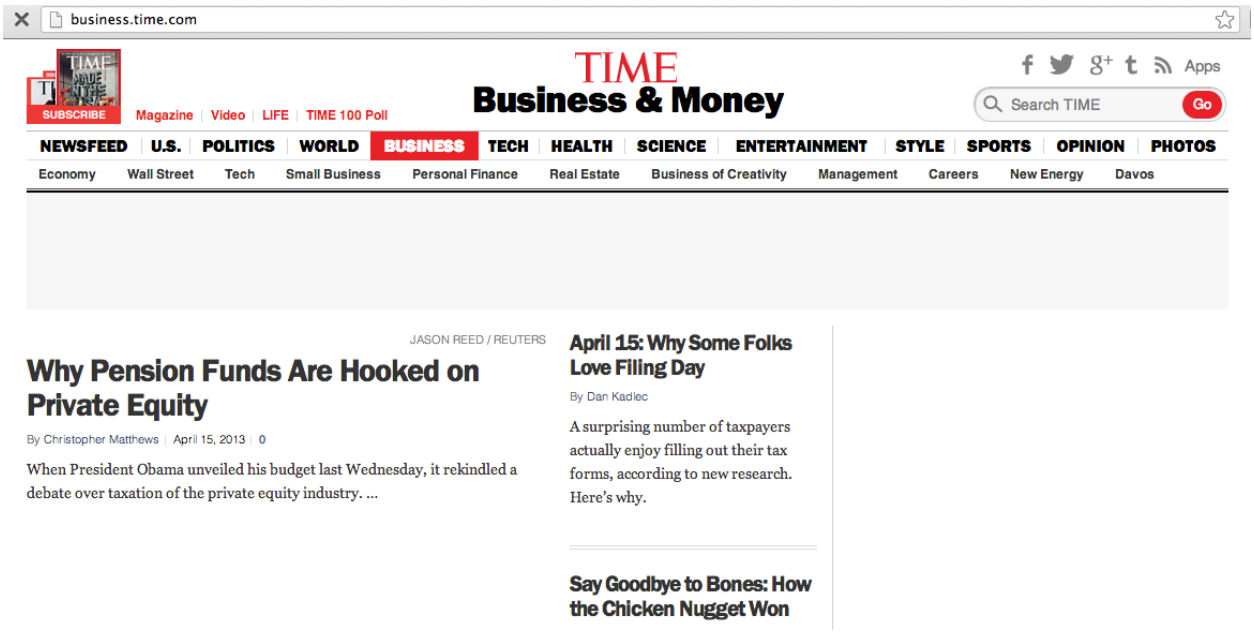

Figure 1 The full-size site of time magazine

Visitors using tablet browsers see a page layout built around two columns.

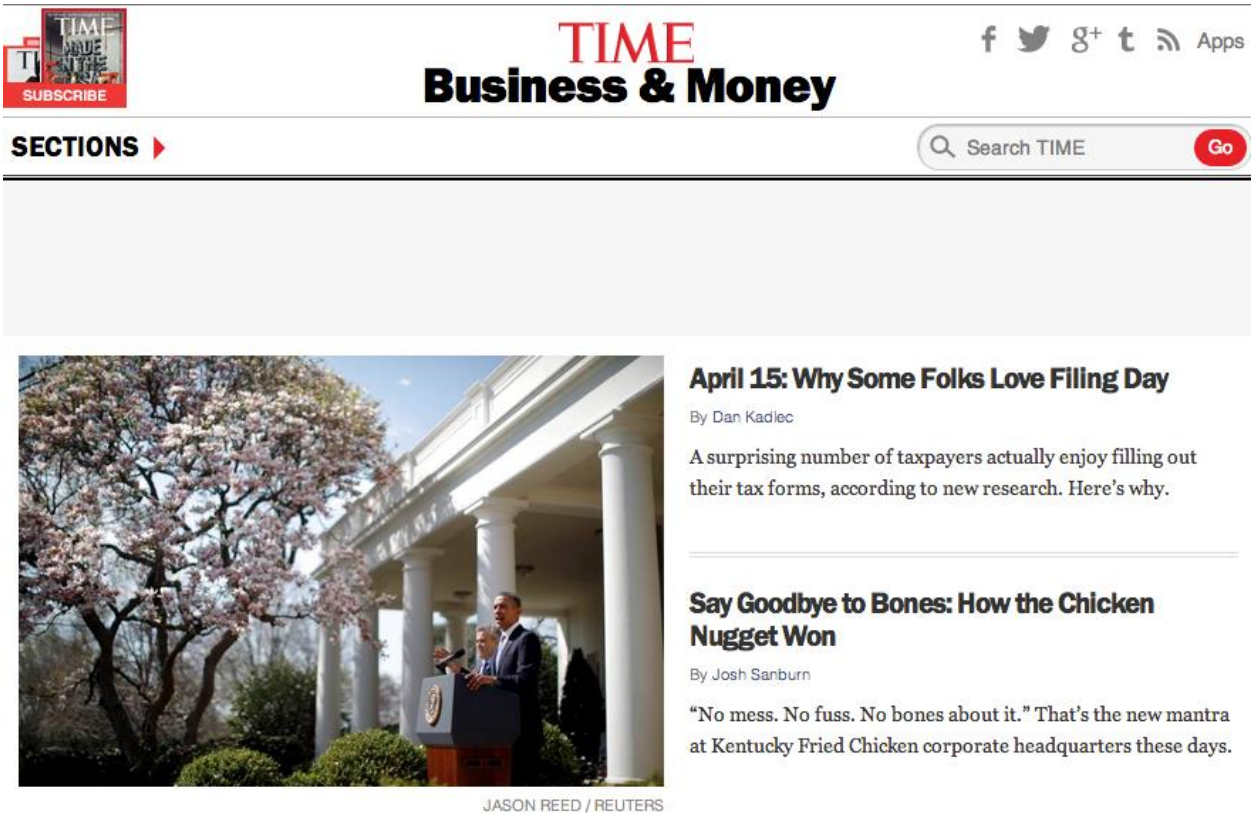

Figure 2 A tablet sized site of magazine

And visitors to the Time site who are using smartphones see a page layout built with a single column.

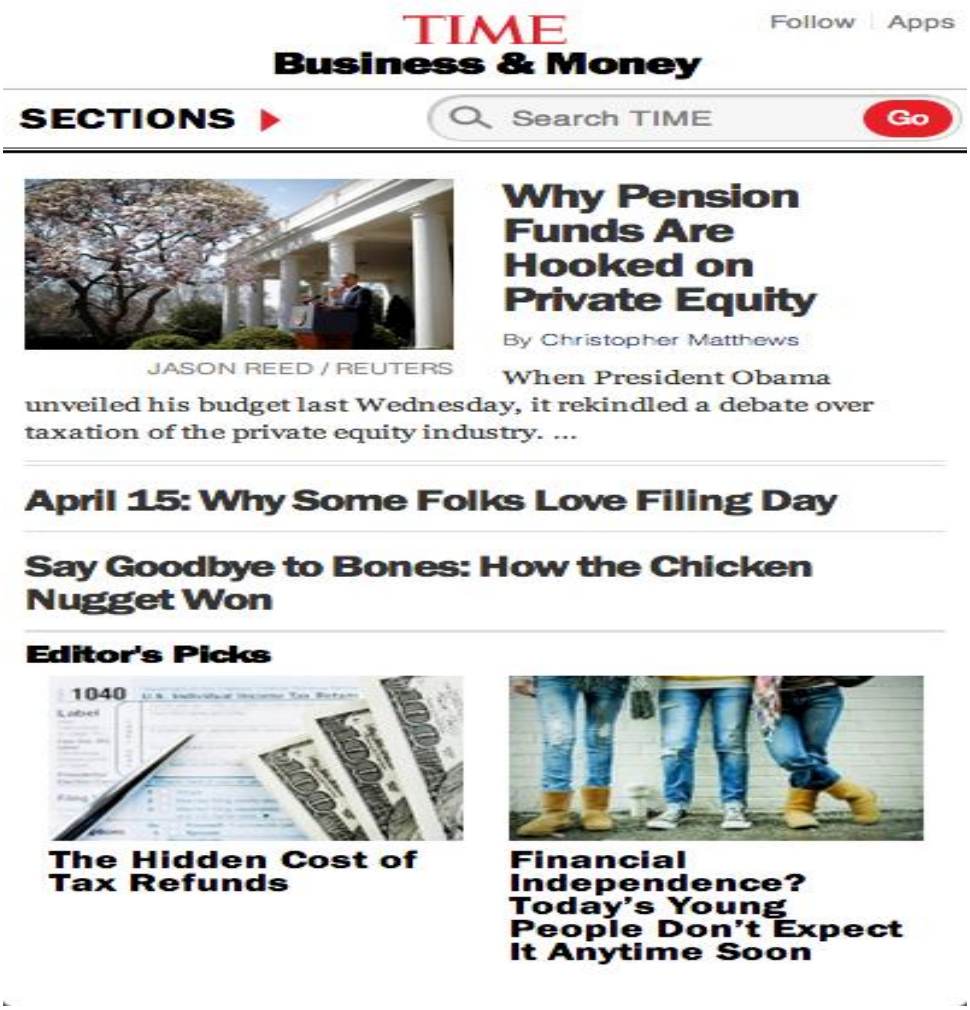

Figure 3 Smart Phone version of Time

Today, we'll explore three approaches that you can apply individually or combine to create responsive design.

Three Approaches to Mobile Design

You can choose from three approaches to creating mobile sites:

Fluid design: Designing page elements in percentages instead of fixed values (like pixels)

Media queries: Creating separate style sheets for different-sized browsing environments

jQuery Mobile apps: Generating animated, interactive Web apps that run in mobile browsers

Today, we will look into all of these approaches, starting with fluid design.

How Fluid Design Has Changed

Fluid design evolved in the era of desktops and laptops. Designers wanted to provide attractive designs that adapted to a user's browser window width. Take, for example, a Web page designed to display 1,200 pixels wide. Let's say that page had a 1,200-pixel-wide table divided into several columns. (I know! We don't design pages that are 1,200 pixels wide anymore, and we don't use tables for page design. But

I'm invoking these examples to shed light on how fluid design evolved and how it fits into modern design techniques). If a user viewed this 1,200-pixel-wide page on a desktop computer with the browser window resized to 800 pixels, he or she would cut off a third of the page content. And some laptops don't have screens wide enough to display a 1,200-pixel-wide page, even if the user maximizes the browser window.

Enter fluid design. Creative and cutting-edge designers (for their time) began sizing page design elements (like tables or div tags) in percentages. That way, if a user resized his or her browser width to 600 pixels, he or she would still see the entire page, but everything would be half-size.

Essentially, fluid design uses relative values (like 50%) instead of absolute values (like 480 pixels) to define the size of elements. You can see an example of fluid design at wxyzjewelry.com. As you resize your browser window, the images change size. And if the browser window is too narrow, the number of columns decreases. Feel free to visit the site and try it for yourself. (I didn't design this site—I just chose it because it's a good example.)

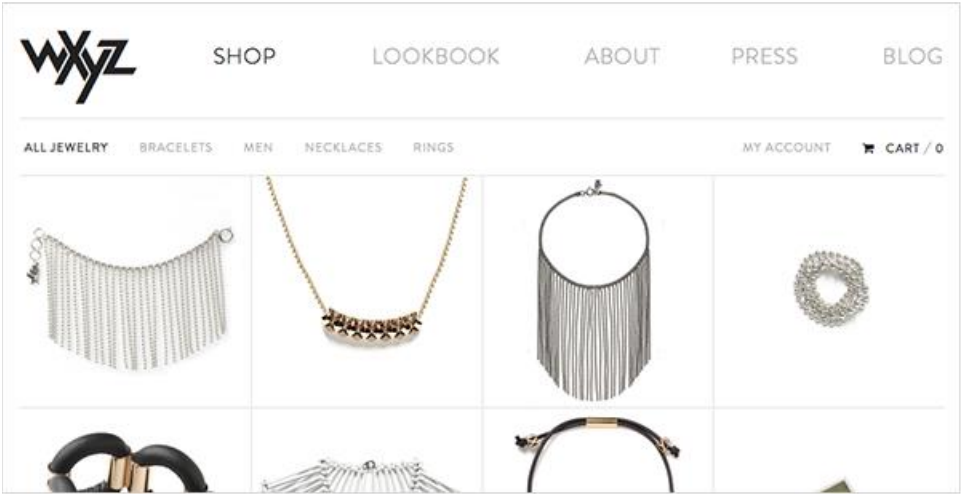

Figure 4 fluid design at wxyz jewelary

Today's design challenges for different viewports (monitor sizes) are different and greater. The difference between designing for a smartphone and a laptop involves factors beyond size—like designing for different lighting conditions, different download speeds, lack of a mouse, and so on. In that light, fluid design isn't sufficient in and of itself to provide a fully optimized mobile experience. But it remains a part of responsive design. Over the rest of this lecture, you'll see how fluid design works and how it fits into the bigger picture of responsive design.

Applying Fluid Design

Let's take a CSS file—the one we've been using, for instance—and apply fluid design to it. We'll start by creating a simplified version of the template we've been using for class. To do that, create a new HTML file in your code editor. Copy this code into your new, blank, open document:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
```

```
<title>Responsive Template</title>
<link href="full.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="wrapper">
<div id="banner">
<h1>Fluid Layout Website for [your class nickname]</h1> </div>
<div id="left-column">
<h3>Links...</h3>
<p><a class="button" href="index.html">Home</a></p>
<p><a class="button" href="video.html">Video</a></p>
<p><a class="button" href="feedback.html">Feedback</a></p>
</div>
<div id="right-column">
<h2>Right Column Heading Here </h2>
<p>Right column content here </p>
<div class="box"><p>Box content</p></div>
<div class="box"><p>Box content</p></div>
<div class="box"><p>Box content</p></div>
<div class="box"><p>Box content</p></div>
<div class="box"><p>Box content</p></div>
<div class="box"><p>Box content</p></div>
</div>
<!--The clear class style clears (removes) float-->
<div class="clear"></div>
<div id="footer">
<h5>Site by [your class nickname]</h5>
</div>
</div>
</body>
</html>
```

Save the file as responsive.html.

There's one thing in the HTML code that might seem strange. I put links inside a .button class style. That will allow us to define mobile-friendly styling for links easily. You'll see how it works before this lecture is over. Next create a simple CSS file with this code:

```
@charset "UTF-8";
body
{
    background-color: black;
    font-family: Verdana, Geneva, Arial, sans-serif;
    padding:0px;
    margin:0px;
}
h1,h2,h3,h4,h5,h6,p,li
```

```
{
    margin-left:15px;
}
#wrapper
{
    width: 960px;
    height: 800px;
    margin-left: auto;
    margin-right: auto;
    background-color:LightSlateGrey;
}
#right-column
{
    float: right;
    width: 600px;
    height: 600px;
    background-color:LightGray;
}
#left-column
{
    float: left;
    width: 360px;
    background-color:silver;
    height:600px;
}
#footer
{
    height:80px;
}
#banner
{
    height:120px;
    background-color:SlateGrey;
}
.box
{
    height: 150px;
    width: 150px;
    float: left;
    background-color:tan;
    margin: 15px;
    opacity:.5;
}
.clear
{
```

```
    clear: both;
}
```

Save the file as full.css. With both files saved, open responsive.html in a browser. Try resizing your browser window. You'll see that the elements (like the wrapper, columns, and boxes) remain the same size no matter how narrow you make it. In fact, if we make the browser width too narrow, some of the content gets cut off.

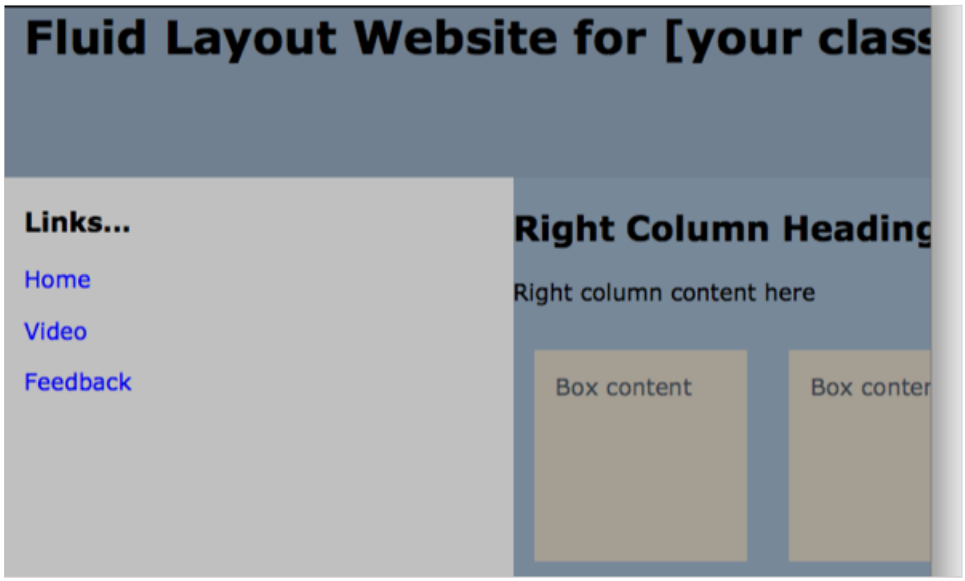

Figure 5 Examining page layout in a narrow browser width

Let's connect this same HTML page with a CSS style sheet that defines a fluid grid. We'll do that by replacing our width values with percentages.

Let's break this down with two examples:

1. If we define a box width of 50%, and that box is simply sitting on an HTML page, enclosed only by <body> and </body> tags, then the box will display at 50% of the width of the entire Web page. And that width will depend on the size of a user's computer and how he or she has sized the browser window.
2. If the same box is inside a div tag (like a wrapper ID div) that's 800 pixels wide, then the box will behalf that width, or 400 pixels wide.

Let's see how this works by applying fluid design to our responsive.html file.

Start by saving the open CSS file as fluid.css. Then change the link in the responsive.html page to this new style sheet. The new link should be: **<link href="fluid.css" rel="stylesheet" type="text/css">**

Now experiment by making these changes to fluid.css:

1. Change the #wrapper selector width to 66%. This forces the width of the wrapper to be 66% of the width of a user's browser window.
2. Change the #left-column selector width to 33%. This forces the width of the left column to 33% of the available space inside the wrapper.

3. Change the #right-column selector width to 66%. This forces the width of the right column to 66% of the available space inside the wrapper.
4. Change the .box selector width to 33%. This forces the width of each box to 33% of the available space inside the right column.

To test the new style sheet, change the style sheet link in the responsive.html file to link to fluid.css. I gave the new link code for this earlier, but again, it is:

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

Try resizing your browser window. This time, the elements adapt to the viewport width.

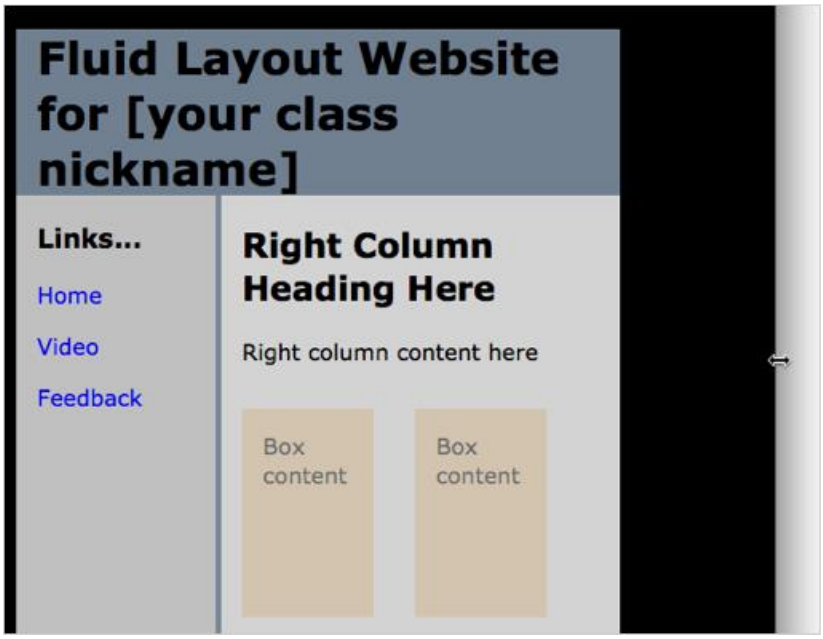

Figure 6 Examining responsive page layout in a narrow browser width

Beyond Fluid Design

Fluid page layout makes pages accessible when users resize their browser windows. However, this approach leaves two problems unsolved:

- For reasons we've been discussing today, mobile pages should be more distinct from full-sized pages than is easily accomplished with fluid designs. Fluid designs do not, for example, solve the problem of creating pages that download more quickly with smaller image files. Nor do they solve other accessibility and aesthetic issues involved in mobile design, such as using high-contrast color schemes that make screens easier to read in sunlight.
- Fluid design makes it difficult to fix the width of a page in desktops and laptops (for example, at 960 pixels wide) when that's what you want to do. For instance, if you rely on fluid design to resize the width of a page to 50% of a user's browser width, the width of the page will constantly shift as a user tweaks his/her their browser window. That can make pages too responsive, jumping in size every time a user adjusts the window width.

You can solve those challenges with responsive design—specifically, with media queries.

Media Queries: A Quick Overview

HTML and CSS have long provided media queries, which are tools that let you create separate style sheets for different media. For example, you can use media queries to provide separate styling for screens(monitors) and printers. Media queries go inside the <head> element. The basic syntax for a media query is:

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

For example, a media query that attached a style sheet of print.css for printing and a style sheet of screen.css for monitors could look like this:

```
<link href="screen.css" rel="stylesheet" type="text/css" media="screen">
<link href="print.css" rel="stylesheet" type="text/css" media="print">
```

In the era of tablets and smartphones, media queries allow you to define separate style sheets for different viewports.

```
<link href="filename.css" rel="stylesheet" type="text/css" media="(max-width/min-
width: dimension)">
```

You can use a max-width property to define a maximum width for a style sheet or a min-width property to define a minimum width. And you can combine max-width and min-width properties. I'll show you how that works. But first let's survey the widths of typical devices.

There's a wide range of viewport sizes! But the three basic ones are:

- Smartphones (usually defined as 480 pixels wide or narrower)
- Full-sized (laptops and desktops, usually defined as 960 pixels wide or wider)
- Tablets (often defined as everything between 481 and 960 pixels wide)

This commented code defines a media query that links three separate style sheets, one for each type of device:

```
<!-- full-size monitor style sheet-->
<link href="full.css" rel="stylesheet" type="text/css" media="only screen and
(min-width:960px)">
<!-- tablet style sheet-->
<link href="tablet.css" rel="stylesheet" type="text/css" media="only screen and
(min-width:481px) and (max-width:960px)">
<!-- phone style sheet-->
<link href="phone.css" rel="stylesheet" type="text/css" media="only screen and
(max-width:480px)">
```

These media queries combine "only screen" properties that exclude printers or other devices from the media query. And that the media query for the tablet style combines a min-width value and a max-width value.

Forcing Devices to Report Actual Width

Manufacturers of mobile devices have their own approaches to making pages look good in their devices. Those approaches are complex and technical, but they boil down to resizing page elements in ways that designers (people like us!) don't want and didn't factor into page design. The solution is to add a line of

code to the <head> element that forces mobile devices to report their actual viewport dimensions to a browser instead of distorting their viewport size. Here's that code:

<meta name="viewport" content="width=device-width">

Adding this line of code makes viewport-based media queries more accurate and more reliable in smartphones and tablets.

Adding Media Queries

Let's review the code you'll need to apply media queries to our responsive.html file. The entire <head> element for the responsive.html file should look something like this:

```
<head>
    <meta charset="UTF-8">
    <title>Responsive Template</title>
    <meta name="viewport" content="width=device-width">
    <!-- full-size monitor style sheet-->
    <link href="full.css" rel="stylesheet" type="text/css" media="only screen and
(min-width:960px)">
    <!-- tablet style sheet-->
    <link href="tablet.css" rel="stylesheet" type="text/css" media="only screen
and (min-width:481px) and (max-width:960px)">
    <!-- phone style sheet-->
    <link href="phone.css" rel="stylesheet" type="text/css" media="only screen
and (max-width:480px)">
</head>
```

Creating a CSS File for a Full-Sized Browser

We created full.css, it will serve as the style sheet for visitors who are using desktops, laptops, or tablets that have a viewport of 960 pixels or wider. If you test responsive.html in a browser, it'll look fine—as long as your browser width is 960 pixels or wider. If you try to view the page in a phone, on a tablet, or in a desktop or laptop browser window that's narrower than 960 pixels, it will look naked—without any styling. That's because the media query we defined is looking for either a tablet.css or phone.css file, and we haven't created that yet.

Creating a CSS File for Tablets

Now let's build an alternate style sheet for tablets. First, we'll resize elements to fit a tablet, integrating some of the fluid design techniques we learned. And we'll apply a brighter, higher-contrast color scheme that'll be easier to read in bright sunlight. Ready to give that a try? I'm going to provide you with code for the tablet.css file first, and then I'll have you follow a series of steps to deconstruct what goes into our tablet-friendly style sheet. Save this new CSS file as tablet.css. Note that the new CSS file customizes many style properties for tablets, but both style sheets have the same set of style selectors. Here's the code for tablet.css:

```
@charset "UTF-8";
body {
color: white;
font-family: Verdana, Geneva, Arial, sans-serif;
```

```
padding:0px;
margin:0px;
}
h1,h2,h3,h4,h5,h6,p,li {
margin-left:15px;
margin-top:0px;
}
#wrapper {width: 100%;
height: 800px;
margin-left: auto;
margin-right: auto;
background-color:DarkGrey;
}
#right-column {
float: none;
height: 600px;
background-color: DarkOrange;
}
#left-column {
float: none;
height:120px;
}
#footer {
height:80px;
background-color:black;
}
#banner{
height:60px;
background-color:black;
}
.box {
width: 45%;
min-height:120px;
float: left;
background-color:black;
margin: 15px;
opacity:1;}
.button {
font-size:x-large;
text-decoration:none;
background-color:LightBlue;
display:block;
width:25%;float:left;
padding:8px;
margin:5px;
```

```
border-radius:15px;
box-shadow:5px 5px 3px gray;
}
.button:hover {
box-shadow:5px 5px 3px black;
}
.clear{
clear: both;
}
```

Let's deconstruct this CSS file and see what changes led to a tablet-friendly look and feel.

1. The new CSS file relies on the width of the viewport to define the width of most content, so I changed the width property for the #wrapper, #left-column, and #right-column selectors. And I turned off the float property from the #left-column and #right-column selectors.
2. Getting rid of the float property for the left and right columns reduced the number of columns. The "left" and "right" columns now fill the entire width of the viewport. But the page still displays three columns of boxes. Let's reduce the number of columns that the boxes use from three to two. Do that by changing the width of the .box style to 45%. Then define a min-height property as well: Set it to 120 pixels (px).
3. We're stacking our left-column and right-column div tags vertically (one on top of the other) instead of horizontally in columns because vertical columns are easier to use in a narrow tablet. In order to keep text elements (like h1, h2, and so on) from forcing vertical spacing between these elements, add margin-top:0px to the definition for the line of code that defines style rules for the six headings, paragraph, and list elements. You can add that parameter (:margin-top:0px;) at the end of the style definition for these elements: h1,h2,h3,h4,h5,h6,p,li
4. Now the tablet has a high-contrast color scheme:
    a. The opacity of the .box class style is 1 (for full opacity, which gives a brighter color).
    b. The #wrapper background-color is Dark Grey.
    c. The #right-column background-color is Dark Orange.
    d. The #banner, #footer, and .box background-color are now black.
5. Remember in Week 1, when I mentioned that you don't want to make your user hover their mouse over a link? The .button and .button:hover styles provide much more mobile-friendly links. Test the tablet.css style by opening responsive.html in a browser window on a laptop or desktop computer and resizing the browser window width until the tablet.css style becomes active.

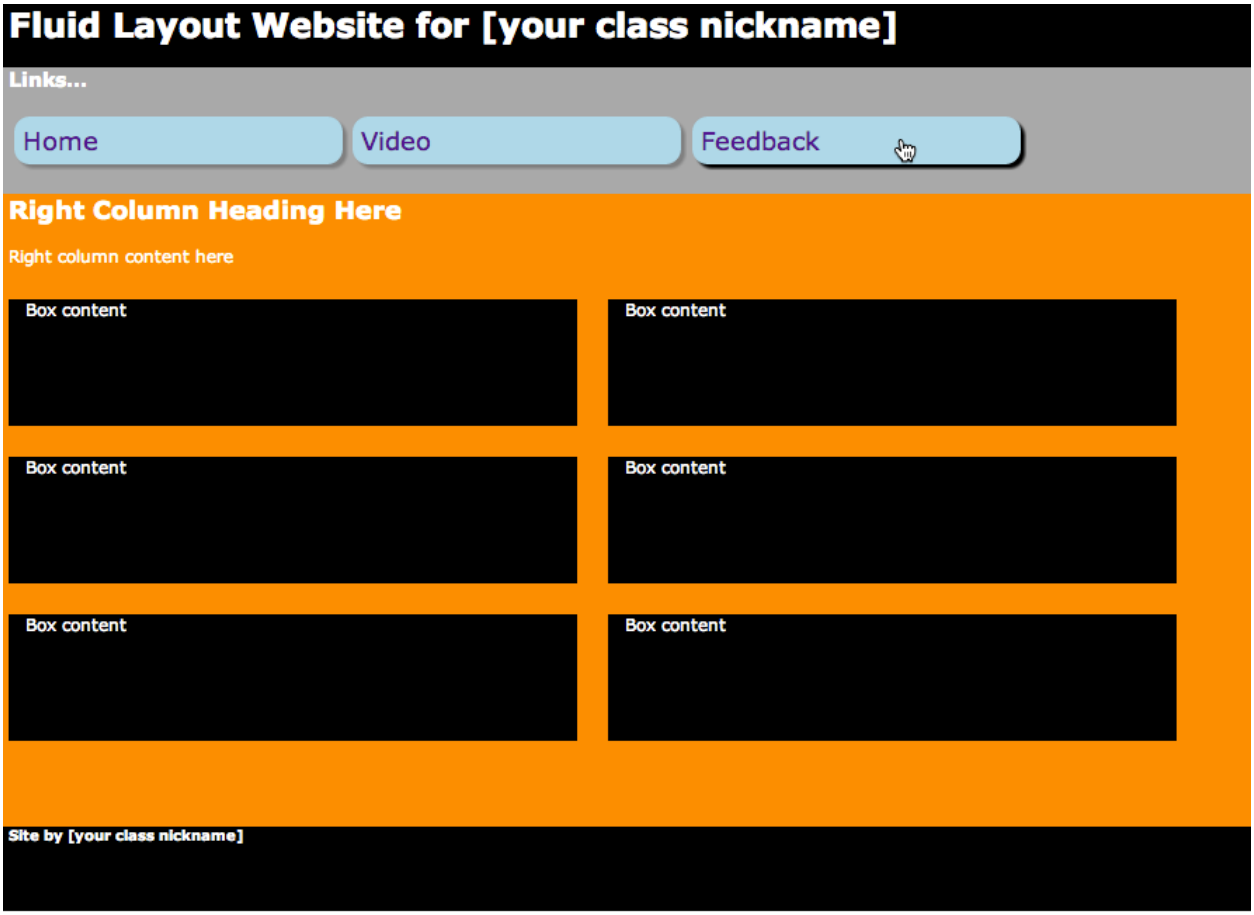

Figure 7 Tablet view in browser window

We've established a basic approach to alternative, mobile-friendly style sheets that feature high-contrast color, reduced columns, and mobile-friendly links. What about a phone.css style sheet for a mobile phone? I'll let you do that one yourself for assignment-01.

Responsive Design in a Single CSS File

We created a second, distinct style sheet for tablets. And we used a media query in the HTML page to "tell" browsers that if their viewport was less than 960 pixels wide, they should go get that stylesheet and apply it. With that technique, we were able to create a tablet version of our site that looks quite different from the full-screen version. It had higher color contrast, fewer columns, and easier-to-tap links. It's possible to organize all your media queries in a single CSS file. The process is similar to what we just went through.

Defining Media Queries With @media

When you define media queries in a single style sheet, you first define the media query and then the style. The combined syntax looks like this:

```
@media only screen and (min-width : 481px)
{
selector { property: value;
```

```
property: value;
property: value;
}
selector 2 {
property: value;
property: value;
property: value;
}
}
```

Note the open "{" and close "}" symbols that define the media query and each selector. You can also define media queries with more than one condition. For example, here's how to define a media query for devices larger than a smartphone but smaller than a standard tablet:

@media screen and (min- width: 481px) and (max-device-width: 768px) {

And here's an example that defines a media query for viewports with a maximum viewport of 480 pixels. This one also includes a few typical style changes you might make for a mobile phone:

```
@media only screen
and (max-width : 480px) {
body {
color: white;
font-family: Verdana, Geneva, Arial, sans-serif;
padding:0px;
margin:0px;
}
h1,h2,h3,h4,h5,h6,p,li {
margin-left:15px;
margin-top:0px;
}
#wrapper {width: 100%;
}
#right-column {
width:100%;
float: none;
height: 1200px;
background-color: DarkOrange;
}
#left-column {
float: none;
height:120px;
width:100%;
}
#footer {
height:80px;
background-color:black;
```

```
}
#banner{
height:100px;
background-color:black;
}
.box {
width: 90%;
min-height:120px;
background-color:dark-red;
margin: 15px;
opacity:1;}
.button {
font-size:x-large;
text-decoration:none;
background-color:LightBlue;
display:block;
width:25%;float:left;
padding:5px;
margin:5px;
border-radius:5px;
box-shadow:5px 5px 3px gray;
}
.button:hover {
box-shadow:5px 5px 3px black;
}
}
```

If you deconstruct the code for this media query, you'll see it applies styling techniques that closely resemble the ones we used earlier in creating a tablet-friendly style. The main difference, aside from some tweaking of dimension values to make the style work well in smartphones, is that this media query has to be embedded within a single CSS file. That means you don't define media queries in the HTML file.

As we've seen, you can define media queries with multiple CSS files or within a single CSS file. Designers disagree about which approach is better. The advantage to using a single CSS file is that you don't need to build media queries into your HTML pages. Just create a style sheet, link to it, and let the CSS file handle sorting out different viewport widths. On the other hand, breaking CSS stylesheets into separate files for each viewport makes editing those style sheets more manageable. You decide which technique to use.

Images and Responsive Design

How does responsive design handle images? It often sizes them with percentages. For example, a tablet-style sheet might include a selector for the <img> tag that defines height and width at 66%. And a stylesheet for mobile phones might define images at 50%. Or you might define a width at a set value for images (like 480 pixels for a mobile phone style sheet) and set the height at auto. A more powerful way to make images fit with mobile devices is to create completely different content for desktop and mobile users. We will look into this approach next week.