Introduction to HTML5

(HTML5.2 version 1.1.1)

Copyright Information

The Author

Nat Dunn

Nat Dunn founded Webucator in 2003 to combine his passion for Web development with his business expertise and to help companies benefit from both. Nat began programming games in Basic on a TRS-80 at age 14. He has been developing Web sites and providing Web development training since 1998.

Accompanying Class Files

This manual comes with accompanying class files, which your instructor or sales representative will point out to you. Most code samples and exercise and solution files found in the manual can also be found in the class files at the locations indicated at the top of the code listings.

Due to space limitations, the code listings sometimes have line wrapping, where no line wrapping occurs in the actual code sample. This is indicated in the manual using three greater than signs: >>> at the beginning of each wrapped line.

In other cases, the space limiitations are such that we have inserted a forced line break in the middle of a word. When this occurs, we append the following symbol at the end of the line before the actual break: »»

Table of Contents

1. Laying out a Page with HTML5

In this lesson, you will learn...

1. How to lay out a page with HTML 4 (the "old" way).
2. How to lay out a page with HTML5.
3. The differences between the HTML5 and HTML 4 structures.
4. About the new HTML5 Doctype.
5. About the simpler script and style tags used in HTML5.

This lesson begins with a quick review of a basic HTML 4 page and then dives right in to HTML5 code. We're not going to spend time reviewing history or discussing the hows and whys here, but we'll come back to that later (see page 17). First, we want to get you looking at some code.

1.1 Page Structure

Laying out a Page with HTML 4 - the "old" way

HTML 4 includes semantic tags that describe the content they hold. For example, the `<h1>` tag holds a top-level heading. Web developers have (or at least should have) long since stopped using headings for formatting purposes alone. Likewise, using tables for laying out pages has been long frowned upon. Articles like Throwing tables out the window (http://stopdesign.com/archive/2004/07/27/throwing-tables.html) were written back in 2004. However, the alternative to tables, namely divs, is only one step in the right direction. It's a move from misusing a semantically meaningful element (`table`) to using a semantically meaningless element (`div`).

HTML5 takes the obvious next step. But before we look at it, let's take a quick look at a page laid out with HTML 4 using `div`s.

Code Sample

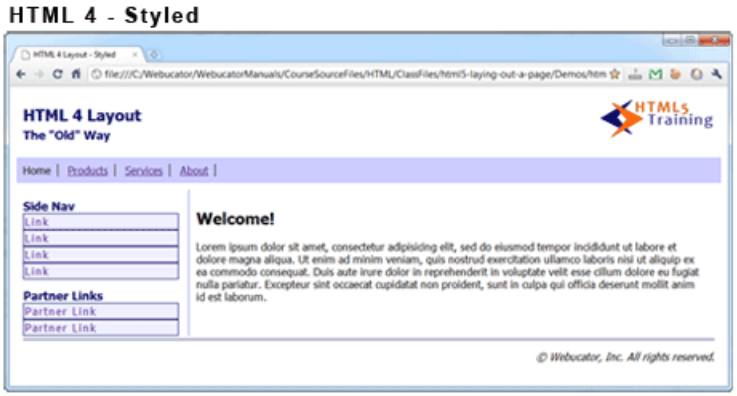
html5-laying-out-a-page/Demos/html4-layout.html

```
      -------Lines 1 through 6 Omitted-------
7.    <body>
8.    <div id="header">
9.     <div id="mainheadings">
10.     <h1>HTML 4 Layout</h1>
11.     <h2>The "Old" Way</h2>
12.    </div>
13.    <img id="logo" src="Images/logo.png" alt="Logo"/>
14.   </div>
15.   <ul id="mainnav">
16.    <li>Home</li>
17.    <li><a href="">Products</a></li>
18.    <li><a href="">Services</a></li>
19.    <li><a href="">About</a></li>
20.   </ul>
21.   <div id="container">
22.    <div id="sidebar">
23.     <h2>Side Nav</h2>
24.     <ul id="sidenav">
25.      <li><a href="">Link</a></li>
26.      <li><a href="">Link</a></li>
27.      <li><a href="">Link</a></li>
28.      <li><a href="">Link</a></li>
29.     </ul>
30.     <h2>Partner Links</h2>
31.     <ul id="partnerlinks">
32.      <li><a href="">Partner Link</a></li>
33.      <li><a href="">Partner Link</a></li>
34.     </ul>
35.    </div>
36.    <div id="content">
37.     <h2>Welcome!</h2>
38.     <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
          >>> eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
          >>>  enim ad minim veniam, quis nostrud exercitation ullamco laboris
          >>>  nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
          >>>  in reprehenderit in voluptate velit esse cillum dolore eu fu »»
          >>> giat nulla pariatur. Excepteur sint occaecat cupidatat non
          >>> proident, sunt in culpa qui officia deserunt mollit anim id
          >>> est laborum.</p>
```

```
39.    </div>
40.   </div>
41.   <div id="footer">
42.    <p>&copy; Webucator, Inc.  All rights reserved.</p>
43.   </div>
44.   </body>
45.   </html>
```

Code Explanation

The above code will render the following (with and without styling):



An interesting thing about this page is the use of `id`s to provide meaning (and hooks for CSS and JavaScript) to the `<div>` tags. We'll come back to this after looking at how we would structure the same page with HTML5.

Laying out a Page with HTML5

The HTML5 code below will render the same as the HTML 4 version above. Open html5-laying-out-a-page/Demos/html5-layout.html and html5-laying-out-a-page/Demos/html5-layout-styled.html in your browser to see the pages.

Code Sample

html5-laying-out-a-page/Demos/html5-layout.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Layout</title>
6.   </head>
7.   <body>
8.   <header>
9.    <hgroup>
10.    <h1>HTML5 Layout</h1>
11.    <h2>The HTML5 Way</h2>
12.   </hgroup>
13.   <img id="logo" src="Images/logo.png" alt="Logo">
14.  </header>
15.  <nav id="mainnav">
16.   <ul>
17.    <li>Home</li>
18.    <li><a href="">Products</a></li>
19.    <li><a href="">Services</a></li>
20.    <li><a href="">About</a></li>
21.   </ul>
22.  </nav>
23.  <div id="container">
24.   <aside id="sidebar">
25.    <h2>Side Nav</h2>
26.    <nav id="sidenav">
27.     <ul>
28.      <li><a href="">Link</a></li>
29.      <li><a href="">Link</a></li>
30.      <li><a href="">Link</a></li>
31.      <li><a href="">Link</a></li>
32.     </ul>
33.    </nav>
34.    <h2>Partner Links</h2>
35.    <ul id="partnerlinks">
36.     <li><a href="">Partner Link</a></li>
37.     <li><a href="">Partner Link</a></li>
38.    </ul>
39.   </aside>
```

```
40.   <div id="content">
41.    <h2>Welcome!</h2>
42.    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
         >>> eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
         >>>  enim ad minim veniam, quis nostrud exercitation ullamco laboris
         >>>  nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
         >>>  in reprehenderit in voluptate velit esse cillum dolore eu fu »»
         >>> giat nulla pariatur. Excepteur sint occaecat cupidatat non
         >>> proident, sunt in culpa qui officia deserunt mollit anim id
         >>> est laborum.</p>
43.   </div>
44.  </div>
45.  <footer>
46.   <p>&copy; Webucator, Inc.  All rights reserved.</p>
47.  </footer>
48.  </body>
49.  </html>
```

In the next sections, we will review the differences between the HTML 4 and HTML5 files shown above.

1.2 New HTML5 Structural Tags

We mentioned that we used *meaningful* `id`s for *meaningless* `div`s in the HTML 4 layout. However, these `id`s were only meaningful to us. The browser doesn't have any awareness of their meaning. HTML5 fixes this. Many of the structural tags coincide (and not coincidentally) with the most common values existing web pages use for the `id` and `class` attributes. Opera researched the most common `class`

values[1] from more than 2 million URLs. The table below shows the 200 most popular (with some gaps).

Rank	Value	Frequency	%
1	footer	179,528	3.6%
2	menu	146,673	2.9%
3	style1	138,308	2.8%
4	msonormal	123,374	2.5%
5	text	122,911	2.5%
6	content	113,951	2.3%
7	title	91,957	1.8%
8	style2	89,851	1.8%
9	header	89,274	1.8%
10	copyright	86,979	1.7%
11	button	81,503	1.6%
12	main	69,620	1.4%
13	style3	69,349	1.4%
14	small	68,995	1.4%
15	nav	68,634	1.4%
16	clea	68,571	1.4%
88	mainleve nav	16,472	0.3%
89	section	16,297	0.3%
90	descriptio	15,071	0.3%
185	row1	8,652	0.2%
186	article	8,649	0.2%
187	h1	8,625	0.2%

Source: http://devfiles.myopera.com/articles/572/classlist-url.htm

The highlighted rows show class names that have corresponding structural elements in HTML5:

1. **header** - holds the header content of the document or a section in the document.
2. **footer** - holds the footer content of the document or a section in the document.
3. **menu** - deprecated in HTML 4. Brought back to life in HTML5 to hold form controls (think of the **File** menu on a desktop application).
4. **nav** - holds navigational links.

1. See Source: http://devfiles.myopera.com/articles/572/classlist-url.htm.

5. **section** - holds a section of the document (see page 25).
6. **article** - holds an article (see page 25).

The `header`, `footer`, and `nav` elements are shown in the demo above. We'll cover `section` and `article` later in the course.

Notably missing from the popular class names is **aside**. However, both **left** and **right** were in the top 25. "Aside" is a better name for an element as it catches the semantic meaning without implying page position.

A couple of notes on other popular class names in the list:

1. **content** (number 6) and **main** (number 12). When designing pages, we often break them up into header, footer, one or two side columns and a **main content area**. It is very helpful to be able to section off this main content area for CSS styling and for accessibility purposes (e.g., Skip Navigation links). However, for some reason there is no HTML5 element corresponding to this area.
2. **small** (number 14). The `<small>` tag is **not** new to HTML5, but it carries new significance. It is used to mark up the "small print" or side comments. The `<big>` tag, on the other hand, is deprecated in HTML5.

1.3 Page Simplification

HTML5 has made some simplifications to the page:

1. The new doctype is simply: `<!DOCTYPE HTML>`.
2. The character set is simply declared with: `<meta charset="UTF-8">`.
3. The `type` attribute is no longer required on `<style>` and `<script>` tags. The default types are CSS and ECMAScript, respectively.

Exercise 1 Converting an HTML 4 Page to an HTML5 Page

15 to 25 minutes

In this exercise, you will convert a basic HTML 4 page to an HTML5 page. The CSS documents have already been created for you, such that, when you're finished, your HTML5 page should render exactly like the HTML 4 page.

1. Open html5-laying-out-a-page/Exercises/html4-layout.html.
2. Save the file as html5-layout.html.
3. Turn the page from an HTML 4 page into an HTML5 page. Make sure to change the stylesheet reference to point to style-html5.css.

***Challenge**

See if you can get your HTML5 page to look like the screen shot below without modifying the HTML at all:

Exercise Solution

html5-laying-out-a-page/Solutions/html5-layout.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Layout - Exercise</title>
6.   <link href="style-html5.css" rel="stylesheet">
7.   </head>
8.   <body>
9.   <header>
10.   <hgroup>
11.    <h1>HTML5 Solutions</h1>
12.    <h2>HTML5 Products and Services</h2>
13.   </hgroup>
14.   <img id="logo" src="Images/logo.png" alt="Logo">
15.  </header>
16.  <nav id="mainnav">
17.   <ul>
18.    <li>Home</li>
19.    <li><a href="">Products</a></li>
20.    <li><a href="">Services</a></li>
21.    <li><a href="">About</a></li>
22.   </ul>
23.  </nav>
24.  <div id="content">
     -------Lines 25 through 50 Omitted-------
51.  </div>
52.  <aside id="partners">
53.   <div>
54.    <h3>Partner 1</h3>
     -------Lines 55 through 63 Omitted-------
64.   </div>
65.  </aside>
66.  <footer>
67.   <p>&copy; Webucator, Inc.  All rights reserved.</p>
68.  </footer>
69.  </body>
70.  </html>
```

Code Explanation

Note that we will further improve this page with `section` and `article` elements later in the course.

Challenge Solution

html5-laying-out-a-page/Solutions/style-html5-challenge.css

```
1.   body {
2.    font-family:tahoma;
3.   }
4.
5.   header {
6.    display:block;
7.    padding:0px 0px 5px 0px;
8.    margin:0px 20px;
9.    height:80px;
10.   width:1030px;
11.   float:left;
12.  }
13.
14.  header hgroup {
15.   display:block;
16.   float:right;
17.  }
     -------Lines 18 through 30 Omitted-------
31.  header #logo {
32.   float:left;
33.  }
34.
35.  nav#mainnav {
36.   display:block;
37.   clear:both;
38.   width:1030px;
39.   height:14px;
40.   margin:10px 0px 0px 20px;
41.   border-left:3px solid #ccf;
42.   border-bottom:3px solid #ccf;
43.   border-right:3px solid #ccf;
44.   padding:5px;
45.   background-color:#eee;
46.   font-size:small;
47.   letter-spacing: 4px;
48.   word-spacing:20px;
49.   text-align:center;
50.  }
51.
```



```
52.  nav#mainnav ul {
53.   margin:0px;
54.   padding:0px;
55.  }
56.
57.  nav#mainnav li {
58.   display:inline;
59.  }
60.
61.  #content {
62.   margin:0px 20px;
63.   float:left;
64.  }
     -------Lines 65 through 74 Omitted-------
75.  #content div h2  {
76.   clear:both;
77.   float:right;
78.   font-size:xx-large;
79.   color:#006;
80.  }
81.
82.  #content>div {
83.   clear:both;
84.  }
85.
86.  #content div div {
87.   padding:10px;
88.   margin-bottom:10px;
89.   width: 700px;
90.   border:1px solid #006;
91.   font-size:small;
92.  }
93.
94.  aside#partners {
95.   width: 275px;
96.   margin:40px 20px;
97.   display:block;
98.   float:left;
99.   background-color:#eee;
100.  border:3px solid #ccf;
101. }
102.
```

```
103. aside#partners div {
104.  padding:5px;
105.  font-size:x-small;
106. }
107.
108. aside#partners h2 {
109.  margin-bottom:0px;
110.  font-size:large;
111.  color:#006;
112. }
113.
114. footer {
115.  display:block;
116.  clear:both;
117.  margin:10px 20px;
118.  border-top:1px solid #006;
119.  font-style:italic;
120.  width:1040px;
121.  text-align:right;
122. }
123.
124. a:hover {
125.  color:#f60;
126. }
```


1.4 Conclusion

In this lesson, you have learned about the new HTML5 structural tags and how to layout a basic HTML5 page.

2. HTML5 - How We Got Here

In this lesson, you will learn...

1. What problems HTML 4 addresses.
2. What problems XHTML addresses.
3. How HTML5 addresses these problems.
4. About the new features of HTML5.
5. What is in the HTML5 Spec.
6. About browser support for HTML5.

In this lesson, we will discuss the differences between HTML 4 and HTML5 (aside from the space and the number).

2.1 The Problems HTML 4 Addresses

HTML 4 was introduced in 1997 (yeah, a long time ago). The biggest gain in HTML 4 was the separation of formatting into CSS. Formatting tags like `<font>`, `<s>` and `<u>` and attributes like `bgcolor`, `height` and `width` were deprecated in favor of corresponding CSS properties. This allowed for big gains in accessibility and much easier, more semantic mark up (e.g., tableless layout).

There were also additional accessibility improvements, including requiring the `alt` attribute on `<img>` tags and allowing for the `title` attribute on almost all tags.

It took browsers awhile to conform, but eventually they managed and now web designers can safely take advantage of most HTML 4 / CSS features.

2.2 The Problems XHTML Addresses

HTML is an SGML-based language and SGML-based languages are not easy to extend or consume generically. The major issue is that such languages are too flexible. The two major problems are:

1. Not all tags are closed.
2. Boolean attributes take no values.

As such, the tool consuming these languages (e.g., a browser or an editor), must be aware of every aspect of the language.

On the other hand, XML-based languages are stricter:

1. All tags must be closed.
2. All attributes must have values.

XML also enforces case sensitivity and use of quotation marks to enclose attribute values, but it is the two issues above that make XML so easily extensible.

A nice side effect was that it provided freedom **from** choice. And with this freedom comes the knowledge that you can look at any valid XHTML document and know what to expect. No need for a style guide to tell authors when to and not to put quotation marks around their attributes or to write in lowercase or uppercase letters.

As it turns out, XHTML wasn't adopted as whole-heartedly as had been anticipated and, as such, didn't add as much value as we all had hoped it would. The W3C stopped work on XHTML 2 in 2009.

2.3 The New More Flexible Approach of HTML5 - Paving the Cowpaths

HTML5 takes a much more flexible approach than XHTML did. HTML5 is designed with the idea that authors have been writing HTML in many different ways over the years and there are zillions of web pages out there that don't adhere to the strict XHTML standards. Rather than render those page useless, let's just relax the standard a bit (well, a whole lot). They call this "paving the cowpaths."

As an example of this flexibility, all of the following are permitted in HTML5:

```
1. <link type="text/css" href="style.css"/>
2. <LINK TYPE="text/css" HREF="style.css"/>
3. <link type=text/css href=style.css>
4. <LINK TYPE=text/css HREF=style.css>
5. <LiNk TyPe=text/css hReF="style.css">
```

As the above shows:

1. HTML5 is case-insensitive.
2. HTML5 allows for unclosed tags, but you can use the XML-style shortcut close tag if you want.
3. HTML5 does not require quotes around attribute values (unless the values have spaces in them).

This new flexibility could lead to a bit of chaos on your development team. Different HTML authors will take different approaches. Our recommendation is that you choose one approach and stick to it. In this course, for example, we use the following guidelines:

1. Write tags and attributes in all lowercase letters (even event handlers like `onclick`).
2. Do not use short-cut close tags for void/empty elements.
3. Put all attribute values in quotes. (Why? Because attribute values that have spaces in them have to be in quotes. And I do not like the idea of having some attributes in quotes and some not.)
4. Minimize attributes when you can.

Again, it doesn't matter so much which guidelines you choose, but it'll make your life easier if you specify some.

2.4 New Features of HTML5

The table below shows the new elements that HTML5 has introduced. We will cover most of these in this course.

New HTML5 Elements

#	Tag	Description
1	`<article>`	For a standalone article on a page. Articles can be nested within other articles; for example, a blog post would be contained in `<article>` tags and each comment would be contained within a nested `<article>` tag.
2	`<aside>`	For content contained in an aside. Often used for navigation elements or for a list of articles or categories (e.g., in a blog).
3	`<audio>`	For embedding audio files.
4	`<canvas>`	For creating drawings natively in the browser.
5	`<command>`	For command buttons similar to what you might see in the Microsoft Office 2010 ribbon. `<command>` must be nested in `<menu>`.
6	`<datalist>`	For a dropdown list providing built-in functionality similar to a JavaScript autocomplete boxes.
7	`<details>`	For expandable (usually initially hidden) content to provide more information about an element.
8	`<embed>`	For backwards compatibility with the non-standard (but well supported)`<embed>` tag in HTML 4.
9	`<figcaption>`	For captions on images. (In HTML 4, there was no way to semantically associate a caption with an image.
10	`<figure>`	For wrapping embedded content (e.g., an image) with a `<figcaption>`.
11	`<footer>`	For the footer of a page or a section.
12	`<header>`	For the header of a page or a section.
13	`<hgroup>`	For grouping `<h1>`...`<h6>` tags on a page. For example, the title and subtitle of a page could be an `<h1>` and `<h2>` grouped in an `<hgroup>` tag.
14	`<keygen>`	For a generated key in a form
15	`<mark>`	For showing marked (or highlighted) text. Unlike `<strong>` or `<em>`, `<mark>` doesn't give the text any special meaning. The best example is marking a word or phrase that a user has searched on within the search results.
16	`<meter>`	For a measurement within a set range.
17	`<nav>`	For holding a group of navigation links.
18	`<output>`	For holding output (e.g., produced by a script).
19	`<progress>`	For a progress indicator (e.g., for a loading).
20	`<rp>`	Used within `<ruby>` tags to tell browsers that cannot render the East Asia characters properly what extra characters (usually parentheses) to display.

#	Tag	Description
21	`<rt>`	Used within `<ruby>` tags to show how to pronounce East Asia characters.
22	`<ruby>`	For ruby annotations. (See http://www.w3.org/TR/ruby for examples.)
23	`<section>`	For creating a `<section>` on the page. This helps the browser (user agent) determine the page outline.
24	`<source>`	For indicating media sources within `<video>` and `<audio>`.
25	`<summary>`	For the header of a `<detail>` element. The `<summary>` would show by default.
26	`<time>`	For holding a machine-readable date and/or time while displaying a friendly date and/or time.
27	`<video>`	For embedding video files.
28	`<wbr>`	An opportunity to insert a line break within a word. (e.g., super`<wbr>`califragilistic`<wbr>`expialidocious)

HTML5 and JavaScript

The HTML5 specifications show an API for each HTML5 element, which gives instructions on how to access an elements methods and properties via JavaScript.

JavaScript Cowpaths

Some JavaScript practices long supported by browsers but not officially in the HTML 4 specification have been specified in HTML5:

1. `innerHTML`
2. `XMLHttpRequest`
3. `JSON`
4. `element.getElementsByClassName()`

Additional Changes

1. Native `audio` and `video` - covered in HTML5 Audio and Video (see page 57).
2. Huge advances with forms - covered in HTML5 Forms (see page 77).
3. New ways to store data in the client - covered in HTML5 Web Storage (see page 109).

4. Canvas for creating drawings natively in the browser - covered in HTML5 Canvas (see page 129).
5. HTML5 introduces the new `contenteditable` attribute, which makes the content of a tag editable in the browser:
   A. Open html5-how-we-got-here/Demos/html5-layout.html in your browser.
   B. Click on the content in the HTML5 Training section and start editing.

Modernizr

Modernizr (http://www.modernizr.com/) is a relatively small JavaScript file that checks the user's browser for HTML5 feature support. The name is a bit of a misnomer as it doesn't actually modernize the browser. It doesn't add any missing features, it just gives developers an easy way of figuring out if the browser supports a given feature, so they can write conditional code like this:

```
if (Modernizr.canvas) {
 //use canvas to create awesome drawing application
} else {
 alert("Go get yourself a browser that supports canvas.");
}
```

We use Modernizr in this course (see html5-common/modernizr.min.js in your class files. For the latest version, check http://www.modernizr.com/ (http://www.modernizr.com).

> If your curious what features your browser supports, check out http://www.html5test.com.

2.5 The HTML5 Spec(s)

There are two official HTML5 specifications:

1. WHATWG (http://www.whatwg.org/specs/web-apps/current-work/multipage/)
2. W3C (http://www.w3.org/TR/html5/)

The history is a bit complex - a lot of fighting and bickering. But the end result is that we have two specifications, both with the same editor: Ian Hickson. WHATWG seems to be sort of a playground, which the editor and his contributors use to

innovate. We expect that the W3C specification will ultimately be considered authoritative.

The HTML5 specs are incredibly long, but most of it is describing how user agents should deal with HTML5. Don't be afraid to use it as a reference. It can be intimidating at first, but can be very useful once you get used to it.

2.6 Current State of Browser Support

Browser support is coming along surprisingly quickly considering Ian Hickson has said that HTML5 is unlikely to reach Candidate Recommendation until 2012 and full Recommendation before 2022. However, the reality is that browsers that don't support HTML5 very well make up a significant share of the market. For example, as of this writing, IE6 and IE7, which have virtually no support for HTML5, still have more than 20% of the market share. See the chart below[2]:

That said, we can start playing with HTML5 now and even developing full-fledged applications. We just need to be aware of browser limitations and attempt to degrade gracefully.

2.7 Conclusion

In this lesson, you have learned how we got to HTML5 and the major changes it introduces.

2. See http://marketshare.hitslink.com/browser-market-share.aspx?qprid=2.

3. Sections and Articles

In this lesson, you will learn...

1. How to use `<section>` and `<article>` tags to eliminate inherent HTML 4 structure problems.
2. What outlining is and how it is determined.
3. How heading tags (`<h1>`, `<h2>`, etc.) affect a document's structure/outline.

In HTML 4, we use the `<div>` tag to separate HTML pages into parts. Sometimes those parts were structurally meaningful. For example, a page describing a course might include an overview, goals, prerequisites, and an outline. Each of those parts might be enclosed in a `<div>` tag with meaningful ids to provide meaningful structure to the page. However, `<div>` tags are also used to separate parts of a page for styling purposes, for example to create a column layout. In this case the areas encompassed in `<div>` tags might not be structurally different. Browsers cannot distinguish between structurally meaningful and meaningless divs, so they do not attribute any special significance to either kind.

This lesson explains the purpose of and difference between `<section>` and `<article>` tags, how the differ from `<div>` tags and how they affect a page's "outline."

3.1 The section Tag

HTML5 introduces the `<section>` tag to break up a page in a meaningful way, leaving `<div>` tags to be used only for structurally meaningless page sectioning. First we will look at the HTML 4 way and then we'll see how to "fix" things with HTML5.

The HTML 4 Way

This following two demos show how pages can be "sectioned" in HTML 4.

Code Sample

html5-sections/Demos/html4-course-outline.html

```
      -------Lines 1 through 8 Omitted-------
9.    <h1>CSS Training</h1>
10.   <div id="overview">
11.    <h2>Class Overview</h2>
12.    <p>This CSS training class teaches students to use Cascading Style
         >>>  Sheets to format HTML pages.</p>
13.   </div>
14.   <div id="goals">
15.    <h2>Class Goals</h2>
16.    <ul>
17.     <li>Learn the benefits of CSS.</li>
18.     <li>Learn to avoid using deprecated tags and attributes.</li>
19.     <li>Learn CSS syntax.</li>
20.     <li>Learn to use &lt;div&gt; and &lt;span&gt; tags appropriately.</li>
         >>>
21.     <li>Learn most of the common properties and their values.</li>
22.     <li>Learn to create custom CSS cursors.</li>
23.     <li>Learn to style links with CSS to create "CSS Buttons".</li>
24.     <li>Learn to work with borders, margin, and padding (the box mod »»
         >>>  el).</li>
25.     <li>Learn to style tables with CSS.</li>
26.    </ul>
27.   </div>
28.   <div id="outline">
29.    <h2>Class Outline</h2>
30.    <ol>
31.     <li>Crash Course in CSS</li>
32.     <li>CSS Fonts</li>
33.     <li>CSS Text</li>
34.     <li>Colors and Backgrounds</li>
35.     <li>Custom Cursors</li>
36.     <li>CSS and Links</li>
37.     <li>Borders, Margins and Padding</li>
38.     <li>Styling Tables with CSS</li>
39.    </ol>
40.   </div>
41.   <div id="tech-reqs">
42.    <h2>Technical Requirements</h2>
43.    <ol>
44.     <li>HTML or Text Editor</li>
```



```
45.     <li>Web Browser</li>
46.    </ol>
47.   </div>
48.   </body>
49.   </html>
```

Code Explanation

We have used some CSS to make the page render as follows:

CSS Training

Class Overview

This CSS training class teaches students to use Cascading Style Sheets to format HTML pages.

Class Goals

- Learn the benefits of CSS.
- Learn to avoid using deprecated tags and attributes.
- Learn CSS syntax.
- Learn to use and tags appropriately.
- Learn most of the common properties and their values.
- Learn to create custom CSS cursors.
- Learn to style links with CSS to create "CSS Buttons".
- Learn to work with borders, margin, and padding (the box model).
- Learn to style tables with CSS.

Class Outline

1. Crash Course in CSS
2. CSS Fonts
3. CSS Text
4. Colors and Backgrounds
5. Custom Cursors
6. CSS and Links
7. Borders, Margins and Padding
8. Styling Tables with CSS

Technical Requirements

1. HTML or Text Editor
2. Web Browser

Note that each "section" is separated visually, but without the CSS they would not be. Also note that the browser doesn't know that these divisions have structural meaning. To illustrate this point, we'll add a couple meaningless `<div>` tags.

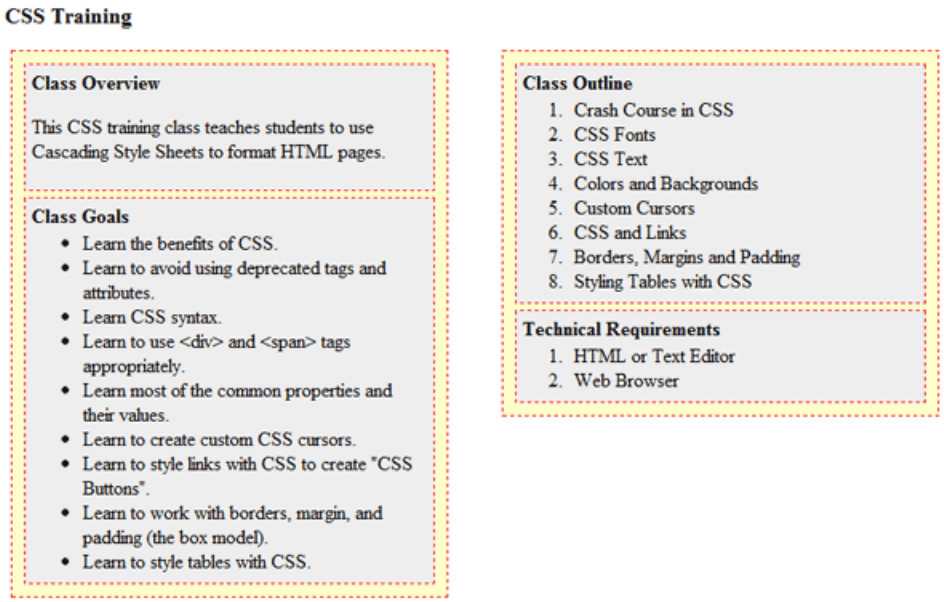
Code Sample

html5-sections/Demos/html4-course-outline2.html

```
      -------Lines 1 through 8 Omitted-------
9.    <h1>CSS Training</h1>
10.   <div id="left">
11.    <div id="overview">
      -------Lines 12 through 13 Omitted-------
14.    </div>
15.    <div id="goals">
      -------Lines 16 through 27 Omitted-------
28.    </div>
29.   </div>
30.   <div id="right">
31.    <div id="outline">
      -------Lines 32 through 42 Omitted-------
43.    </div>
44.    <div id="tech-reqs">
      -------Lines 45 through 49 Omitted-------
50.    </div>
51.   </div>
52.   </body>
53.   </html>
```


Code Explanation

With new "left" and "right" divs, we can now style the page with CSS to render as follows:



But these two new `<div>` tags have not provided any new context to the page. They are purely there for formatting purposes.

The HTML5 Way

Now let's look at the HTML5 way.

Code Sample

html5-sections/Demos/html5-course-outline.html

```
1.   <!DOCTYPE HTML>
     -------Lines 2 through 8 Omitted-------
9.   <h1>CSS Training</h1>
10.  <div id="left">
11.   <section id="overview">
12.    <h2>Class Overview</h2>
     -------Lines 13 through 13 Omitted-------
14.   </section>
15.   <section id="goals">
16.    <h2>Class Goals</h2>
     -------Lines 17 through 27 Omitted-------
28.   </section>
29.  </div>
30.  <div id="right">
31.   <section id="outline">
32.    <h2>Class Outline</h2>
     -------Lines 33 through 42 Omitted-------
43.   </section>
44.   <section id="tech-reqs">
45.    <h2>Technical Requirements</h2>
     -------Lines 46 through 49 Omitted-------
50.   </section>
51.  </div>
52.  </body>
53.  </html>
```

Code Explanation

The above code will render the following:

CSS Training

Class Overview

This CSS training class teaches students to use Cascading Style Sheets to format HTML pages.

Class Goals

- Learn the benefits of CSS.
- Learn to avoid using deprecated tags and attributes.
- Learn CSS syntax.
- Learn to use and tags appropriately.
- Learn most of the common properties and their values.
- Learn to create custom CSS cursors.
- Learn to style links with CSS to create "CSS Buttons".
- Learn to work with borders, margin, and padding (the box model).
- Learn to style tables with CSS.

Class Outline

1. Crash Course in CSS
2. CSS Fonts
3. CSS Text
4. Colors and Backgrounds
5. Custom Cursors
6. CSS and Links
7. Borders, Margins and Padding
8. Styling Tables with CSS
   s

Technical Requirements

1. HTML or Text Editor
2. Web Browser

Notice how we used `<section>` tags for the structurally meaningful parts:

- Overview
- Goals
- Outline
- Technical Requirements

And `<div>` tags for the structurally meaningless "left" and "right" divisions.

Display of HTML5 Structural Elements

By default, browsers treat the `<div>` tag as a block-level element as per instructions provided by the W3C[3]. However, the W3C has not specified anything for the new HTML5 structural tags (e.g., `header`, `footer`, `aside`, `section`, `article`, `nav`). As such, the different browser makers have had to decide individually whether to display these elements as blocks or inlines.

Consider the following code sample:

3. See http://www.w3.org/TR/CSS2/sample.html.

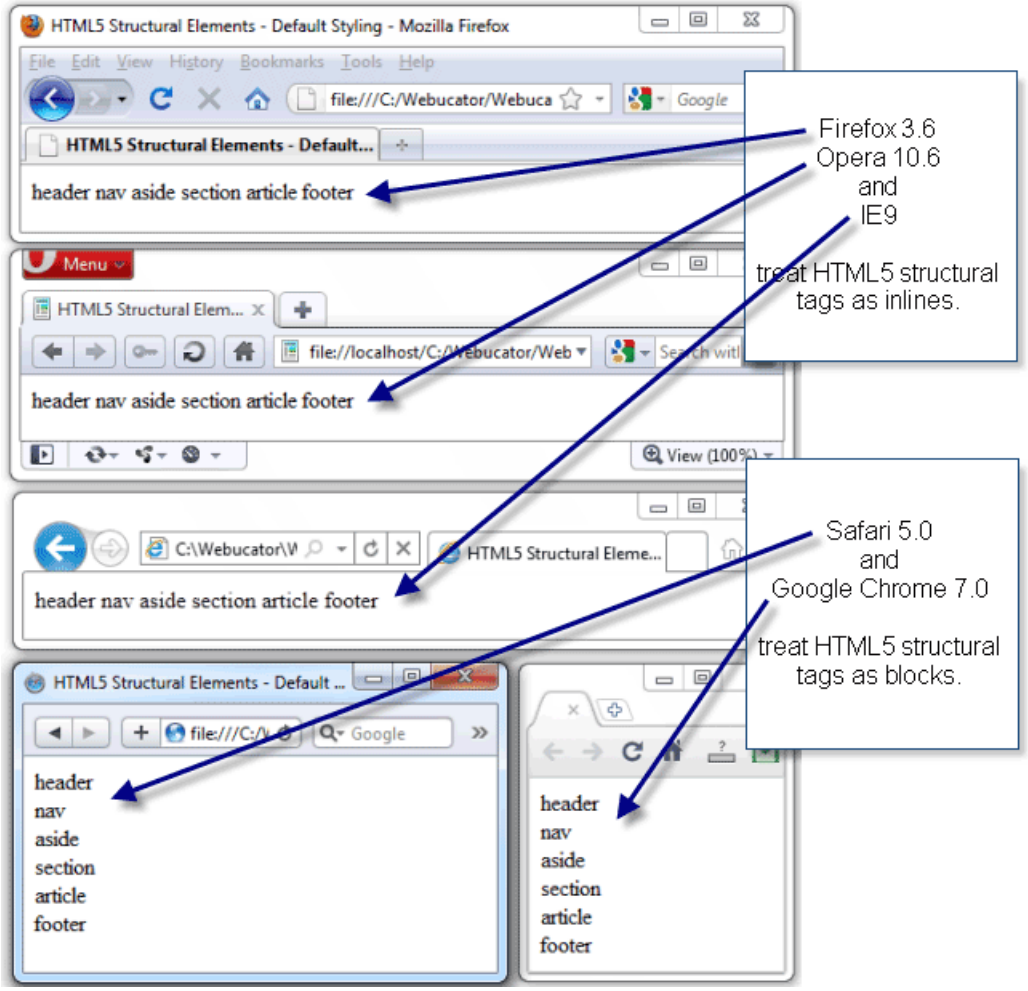
Code Sample

html5-sections/Demos/html5-structural-tags.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Structural Elements - Default Styling</title>
6.   </head>
7.   <body>
8.    <header>header</header>
9.    <nav>nav</nav>
10.   <aside>aside</aside>
11.   <section>section</section>
12.   <article>article</article>
13.   <footer>footer</footer>
14.  </body>
15.  </html>
```


Code Explanation

Here is how different browsers display this page:



As the image above shows, the following browsers treat HTML5 structural tags as inlines:

- Firefox 3.6
- Opera 10.6
- IE9 Beta

And the following browsers treat them as blocks:

- Safari 5.0
- Google Chrome 7.0

To equalize the browsers, you may want to include the following code in your CSS:

```
header, footer, aside, section, article, nav, hgroup {
 display: block;
}
```

3.2 The article Tag

The W3C specification described the `article` element as follows[4]:

> The article element represents a self-contained composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content.

The major difference between the `article` element and the `section` element is that an `article` element encapsulates content that could stand alone and might be of interest outside the context of the page. The most obvious example is a blog entry; however, we could also apply this to the whole of our course description as you can imagine wanting to syndicate all of our outlines to a website that aggregates course information from different training companies.

To do this, we would just wrap everything up in an article tag:

4. See http://dev.w3.org/html5/spec/Overview.html#the-article-element.

Code Sample

html5-sections/Demos/html5-course-outline-article.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.    <article id="css-course">
10.    <h1>CSS Training</h1>
11.    <div id="left">
12.     <section id="overview">
      -------Lines 13 through 14 Omitted-------
15.     </section>
16.     <section id="goals">
      -------Lines 17 through 28 Omitted-------
29.     </section>
30.    </div>
31.    <div id="right">
32.     <section id="outline">
      -------Lines 33 through 43 Omitted-------
44.     </section>
45.     <section id="tech-reqs">
      -------Lines 46 through 50 Omitted-------
51.     </section>
52.    </div>
53.   </article>
54.   </body>
55.   </html>
```

Code Explanation

This would have no visual impact on the page.

Note that you can nest articles. The common example used is a comment in a blog article. However, we might want to change our **overview** from a `section` element to an `article` element recognizing that the course aggregation site might want to have a page aggregating the overviews:

Code Sample

html5-sections/Demos/html5-course-outline-article-nested.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.    <article id="css-course">
10.    <h1>CSS Training</h1>
11.    <div id="left">
12.     <article id="overview" title="Webucator CSS Class Overview">
13.      <h2>Class Overview</h2>
14.      <p>This CSS training class teaches students to use Cascading Style
          >>>  Sheets to format HTML pages.</p>
15.     </article>
16.     <section id="goals">
      -------Lines 17 through 28 Omitted-------
29.     </section>
30.    </div>
31.    <div id="right">
32.     <section id="outline">
      -------Lines 33 through 43 Omitted-------
44.     </section>
45.     <section id="tech-reqs">
      -------Lines 46 through 50 Omitted-------
51.     </section>
52.    </div>
53.   </article>
54.   </body>
55.   </html>
```

In addition to nesting articles within articles, you can:

- Nest sections within sections.
- Nest sections within articles.
- Nest articles within sections.

Exercise 2 Using section and article Elements

15 to 25 minutes

In this exercise, you will modify an HTML page we worked on earlier in the course to replace *meaningless* `div` elements with *meaningful* `section` and `article` elements.

1. Open html5-sections/Exercises/html5-layout.html.
2. Replace *meaningless* `div` elements with *meaningful* `section` and `article` elements. Note that there is room for interpretation here, so there is no one correct solution.
3. To keep the page looking as it did before, you will also need to modify html5-sections/Exercises/style-html5.css.

Exercise Solution

html5-sections/Solutions/html5-layout.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 23 Omitted-------
24.   <div id="content">
25.    <article id="services" title="HTML5 Solutions Services">
26.     <h1>Services</h1>
27.     <section id="training">
      -------Lines 28 through 30 Omitted-------
31.     </section>
32.     <section id="consulting">
      -------Lines 33 through 35 Omitted-------
36.     </section>
37.    </article>
38.    <article id="products" title="HTML5 Solutions Products">
39.     <h1>Products</h1>
40.     <section id="editor">
      -------Lines 41 through 43 Omitted-------
44.     </section>
45.     <section id="templates">
      -------Lines 46 through 48 Omitted-------
49.     </section>
50.    </article>
51.   </div>
      -------Lines 52 through 70 Omitted-------
```

Code Explanation

An argument could be made for turning the "content" `div` into a `section`; however, the main reason for wrapping that content in a div tag is to be able to position the main content on the page, so we left it as a `div`.

Likewise, you could argue for making each partner `div` a `section` or even an `article`, but we wrapped them in `div` tags mostly so that we could float them left.

Finally, turning the "services" and "products" `div` elements into `article` elements is a bit arbitrary too. You could just as easily make them `section`s.

Exercise Solution

html5-sections/Solutions/style-html5.css

```
1.    body {
2.     font-family:tahoma;
3.    }
4.
5.    header, footer, aside, section, article, nav, hgroup {
6.     display: block;
7.    }
      -------Lines 8 through 70 Omitted-------
71.   #content section h2  {
72.    float:right;
73.    font-size:xx-large;
74.    color:#006;
75.   }
76.
77.   #content>article {
78.    clear:both;
79.   }
80.
81.   #content article section {
82.    float:left;
83.    padding:10px;
84.    width: 500px;
85.    border:1px solid #006;
86.    font-size:small;
87.   }
      -------Lines 88 through 119 Omitted-------
```

3.3 Outlining

The HTML5 specification describes an algorithm to determine the outline of a web page. This allows:

1. User agents (e.g., browsers and screenreaders) the ability to present data to the consumer in a logical way.
2. Syndicators to syndicate portions of a page's content in a structurally meaningful way.
3. Authors to create pages with any number of heading levels.

Sectioning

To understand how the outlining algorithm works, you need to know about **sectioning roots** and **sectioning content**.

Sectioning Roots

Sectioning roots are elements that have their own outline and are not included in the ancestral sectioning roots and sectioning content. The following elements are sectioning roots:

1. `body`
2. `blockquote`
3. `td`
4. `details`
5. `fieldset`
6. `figure`

So each of the above elements has its own outline.

Sectioning Content

Sectioning content is defined by elements that can (but don't necessarily) have their own headers and footers. The following elements define sectioning content:

1. `section`
2. `article`

3. `aside`
4. `nav`

While each sectioning content element has its own outline, it also contributes to the outline of ancestral sectioning roots and sectioning content.

So, in considering an outline, first start with the sectioning root and then drill down from there.

The Outline's "List Items"

It helps to think of an outline as a numbered list with the nesting of list items determined by the sectioning content elements and the content of the list items determined by heading elements.

- `h1`
- `h2`
- `h3`
- `h4`
- `h5`
- `h6`
- `hgroup`

Notice the `hgroup` element is included in the list of heading elements. Only the highest-level `h#` element nested within the `hgroup` element is considered in the outline.

Two factors play into the outline level:

1. The nested level of the sectioning content element.
2. The *relative* level of heading within a sectioning content element.

Consider the following demo:

Code Sample

html5-sections/Demos/html5-outlining.html

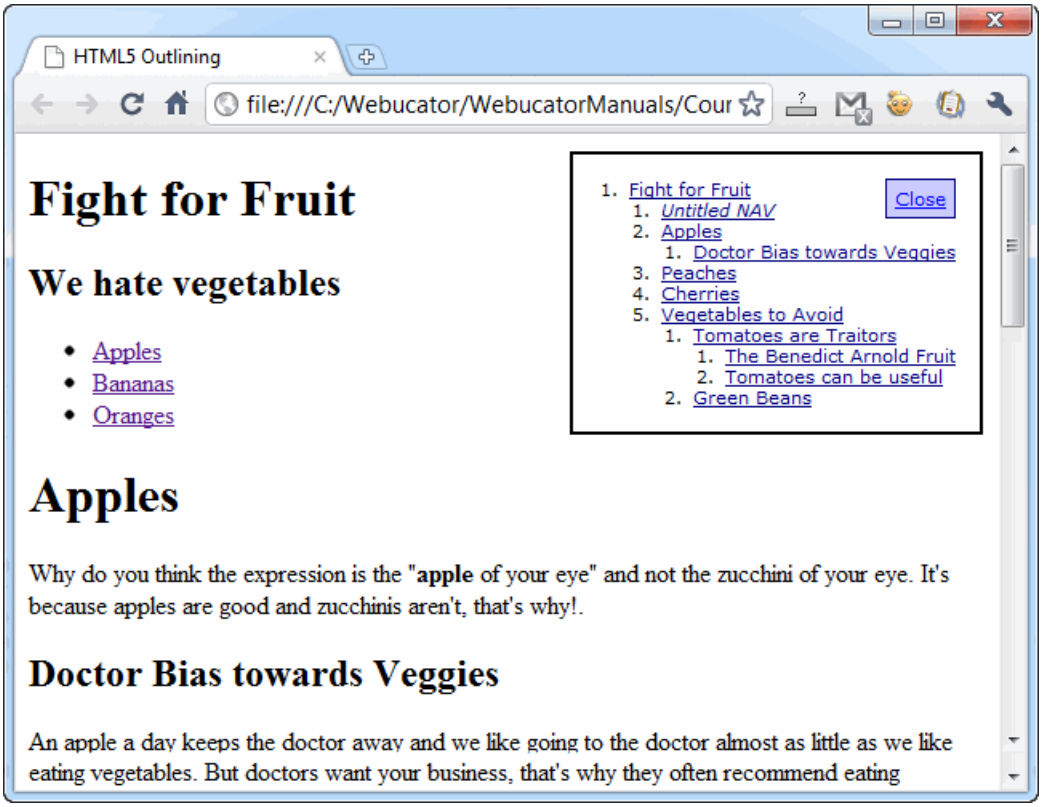
```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Outlining</title>
6.   </head>
7.   <body>
8.   <header>
9.    <hgroup>
10.    <h1>Fight for Fruit</h1>
11.    <h2>We hate vegetables</h2>
12.   </hgroup>
13.  </header>
14.  <nav id="mainnav">
15.   <ul>
16.    <li><a href="">Apples</a></li>
17.    <li><a href="">Bananas</a></li>
18.    <li><a href="">Oranges</a></li>
19.   </ul>
20.  </nav>
21.  <section>
22.   <h1>Apples</h1>
23.   <p>Why do you think the expression is the "<strong>apple</strong> of
        >>> your eye" and not the zucchini of your eye.  It's because apples
        >>>  are good and zucchinis aren't, that's why!.</p>
24.   <h2>Doctor Bias towards Veggies</h2>
25.   <p>An apple a day keeps the doctor away and we like going to the doctor
        >>>  almost as little as we like eating vegetables. But doctors
        >>> want your business, that's why they often recommend eating
        >>> vegetables...</p>
26.  </section>
27.  <section>
28.   <h1>Peaches</h1>
29.   <p>If someone tells you that you're a peach, that's a good thing right.
        >>>   But who wants to be told that they're a squash. Nobody.</p>
        >>>
30.  </section>
31.  <section>
32.   <h1>Cherries</h1>
33.   <p>Remember the line from the musical Oklahoma in the song "I can't
        >>> say No:"</p>
```



```
34.   <blockquote>
35.    <p>S'posin' 'at he says 'at yer lips're "like cherries"</p>
36.   </blockquote>
37.   <p>Having lips like cherries is flattering, right? Would you like lips
        >>>  like eggplant? Of course, not. Cheeries = good.  Eggplant =
        >>> bad.</p>
38.  </section>
39.  <aside>
40.   <h1>Vegetables to Avoid</h1>
41.   <p>Really you should avoid all veggies as they are likely to make your
        >>>  taste buds squirm, but there are a couple you should avoid at
        >>>  all costs.</p>
42.   <article>
43.    <h1>Tomatoes are Traitors</h1>
44.    <p>Tomatoes are the Benedict of all "veggies" and should never be
        >>> eaten.</p>
45.    <h2>The Benedict Arnold Fruit</h2>
46.    <p>Nobody likes at traitor...</p>
47.    <h2>Tomatoes can be useful</h2>
48.    <p>We're not saying never to buy tomatoes. Just don't eat them.  They
        >>>  can be useful for other things, such as throwing at bad actors.
        >>>
49.   </article>
50.   <article>
51.    <h1>Green Beans</h1>
52.    <p>If it's green, it ain't ripe yet.  Nuff said.</p>
53.   </article>
54.  </aside>
55.  <footer>
56.   <p>&copy; Fruit Lobby.  All rights reserved.</p>
57.  </footer>
58.  </body>
59.  </html>
```

Code Explanation

Using the HTML5 outliner bookmarklet discussed earlier, we can see the resulting outline:



You'll notice that the "Doctor Bias towards Veggies" heading, which is an `h2` element, is at the same level as the "Tomatoes are Traitors" heading, which is an `h1` element. This is because the "Tomatoes are Traitors" `h1` is nested three levels deep in sectioning content elements (body -> aside -> article), while the "Doctor Bias towards Veggies" `h2` is nested only two levels deep ((body -> section).

The screen shot in the example above shows the interpreted outline in the upper-right course. As modern browsers don't yet support the outlining algorithm, we've used the h5o HTML5 Outliner bookmarklet[5] to display the outline. Follow these instructions to get the bookmarklet:

1. Go to http://code.google.com/p/h5o/.
2. Click on the Bookmarklet link.
3. Download the Bookmarklet HTML (.html extension) file (Make sure to get the IE version if you're using Internet Explorer).
4. You may need to change the extension from .txt to .html.

5. See http://code.google.com/p/h5o/.

5. Open the file in your HTML5-compliant browser.
6. Drag the link on the page into your "Favorites" or "Bookmark" tool bar.

Exercise 3 Determining the Outline

10 to 20 minutes

In this exercise, you will try to determine the outline of an HTML page.

1. Review the code below.
2. Create a list either on paper or in a text editor or word processor that shows the HTML outline as specified by the HTML5 specification.

Exercise Code

html5-sections/Exercises/html5-outlining.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Outlining</title>
6.   <link href="style-html5-outlining.css" rel="stylesheet">
7.   </head>
8.   <body>
9.   <header>
10.   <hgroup>
11.    <h1>HTML5 Solutions</h1>
12.    <h2>HTML5 Products and Services</h2>
13.   </hgroup>
14.   <img id="logo" src="Images/logo.png" alt="Logo">
15.  </header>
16.  <nav id="mainnav">
17.   <ul>
18.    <li>Home</li>
19.    <li><a href="">Products</a></li>
20.    <li><a href="">Services</a></li>
21.    <li><a href="">About</a></li>
22.   </ul>
23.  </nav>
24.  <div id="content">
25.   <article id="services" title="HTML5 Solutions Services">
26.    <h1>Services</h1>
27.    <section id="training">
28.     <h2>HTML5 Training</h2>
29.     <img src="Images/training.png" alt="Training">
30.     <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
         >>>  eiusmod tempor incididunt ut labore et dolore magna aliqua.
         >>> Ut enim ad minim veniam, quis nostrud exercitation ullamco la »»
         >>> boris nisi ut aliquip ex ea commodo consequat. Duis aute irure
         >>>  dolor in reprehenderit in voluptate velit esse cillum dolore
         >>>  eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
         >>> non proident, sunt in culpa qui officia deserunt mollit anim
         >>> id est laborum.</p>
31.    </section>
32.    <section id="consulting">
33.     <h2>HTML5 Consulting</h2>
```

```
34.      <img src="Images/consulting.png" alt="Consulting">
35.      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
          >>>  eiusmod tempor incididunt ut labore et dolore magna aliqua.
          >>> Ut enim ad minim veniam, quis nostrud exercitation ullamco la »»
          >>> boris nisi ut aliquip ex ea commodo consequat. Duis aute irure
          >>>  dolor in reprehenderit in voluptate velit esse cillum dolore
          >>>  eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
          >>> non proident, sunt in culpa qui officia deserunt mollit anim
          >>> id est laborum.</p>
36.     </section>
37.    </article>
38.    <article id="products" title="HTML5 Solutions Products">
39.     <h1>Products</h1>
40.     <section id="editor">
41.      <h2>HTML5 Editor</h2>
42.      <img src="Images/editor.png" alt="Editor">
43.      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
          >>>  eiusmod tempor incididunt ut labore et dolore magna aliqua.
          >>> Ut enim ad minim veniam, quis nostrud exercitation ullamco la »»
          >>> boris nisi ut aliquip ex ea commodo consequat. Duis aute irure
          >>>  dolor in reprehenderit in voluptate velit esse cillum dolore
          >>>  eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
          >>> non proident, sunt in culpa qui officia deserunt mollit anim
          >>> id est laborum.</p>
44.     </section>
45.     <section id="templates">
46.      <h2>HTML5 Templates</h2>
47.      <img src="Images/templates.png" alt="Templates">
48.      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
          >>>  eiusmod tempor incididunt ut labore et dolore magna aliqua.
          >>> Ut enim ad minim veniam, quis nostrud exercitation ullamco la »»
          >>> boris nisi ut aliquip ex ea commodo consequat. Duis aute irure
          >>>  dolor in reprehenderit in voluptate velit esse cillum dolore
          >>>  eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
          >>> non proident, sunt in culpa qui officia deserunt mollit anim
          >>> id est laborum.</p>
49.     </section>
50.    </article>
51.   </div>
52.   <aside id="partners">
53.    <div>
54.     <h3>Partner 1</h3>
```



```
55.     <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
          >>> eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
          >>>  enim ad minim veniam, quis nostrud exercitation ullamco laboris
          >>>  nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
          >>>  in reprehenderit in voluptate velit esse cillum dolore eu fu »»
          >>> giat nulla pariatur. Excepteur sint occaecat cupidatat non
          >>> proident, sunt in culpa qui officia deserunt mollit anim id
          >>> est laborum.</p>
56.    </div>
57.    <div>
58.     <h3>Partner 2</h3>
59.     <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
          >>> eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
          >>>  enim ad minim veniam, quis nostrud exercitation ullamco laboris
          >>>  nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
          >>>  in reprehenderit in voluptate velit esse cillum dolore eu fu »»
          >>> giat nulla pariatur. Excepteur sint occaecat cupidatat non
          >>> proident, sunt in culpa qui officia deserunt mollit anim id
          >>> est laborum.</p>
60.    </div>
61.    <div>
62.     <h3>Partner 3</h3>
63.     <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
          >>> eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
          >>>  enim ad minim veniam, quis nostrud exercitation ullamco laboris
          >>>  nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
          >>>  in reprehenderit in voluptate velit esse cillum dolore eu fu »»
          >>> giat nulla pariatur. Excepteur sint occaecat cupidatat non
          >>> proident, sunt in culpa qui officia deserunt mollit anim id
          >>> est laborum.</p>
64.    </div>
65.   </aside>
66.   <footer>
67.    <p>&copy; Webucator, Inc.  All rights reserved.</p>
68.   </footer>
69.   </body>
70.   </html>
```

***Challenge**

1. Open html5-sections/Demos/html5-outlining.html in your editor.
2. Add an embedded style sheet so that only the headings are shown and that they are sized and tabbed according to their outline level as shown in the screen shot below:

If you use the HTML5 outliner bookmarklet, don't worry about the "Untitled Nav" that it shows. That's just a warning to indicate that there is a sectioning content element without a heading, but you don't have to have a heading in every sectioning content element.

Exercise Solution

The solution is shown below:

1. HTML5 Solutions
    1. Services
        1. HTML5 Training
        2. HTML5 Consulting
    2. Products
        1. HTML5 Editor
        2. HTML5 Templates
    3. Partner 1
    4. Partner 2
    5. Partner 3

Challenge Solution

html5-sections/Solutions/fight-for-fruit.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 5 Omitted-------
6.    <style>
7.     h1, h2, h3 {
8.      background-color:#eee;
9.     }
10.
11.    h1 {
12.     font-size:x-large;
13.    }
14.
15.    h2, aside>article>h1 {
16.     font-size:large;
17.     margin-left:50px;
18.    }
19.
20.    aside>article>h2 {
21.     font-size:medium;
22.     margin-left:100px;
23.    }
24.    ul, p {
25.     display:none;
26.    }
27.   </style>
      -------Lines 28 through 81 Omitted-------
```


Accessibility

Again, it is the **relative** heading level within a sectioning content element that determines the outline level. So, from an outlining point of view, the following two code samples are the same:

```
<section>
 <h1>Heading</h1>
 <p>Content</p>
 <h2>Heading</h2>
 <p>Content>
</section>
```

```
<section>
 <h2>Heading</h2>
 <p>Content</p>
 <h3>Heading</h3>
 <p>Content>
</section>
```

In a couple of ways, the first sample is nicer. It allows us to think about each sectioning content element individually, starting each with an `h1` element. In this way, we also get an unlimited number of heading levels as we can create lower-level headings simply by nesting sectioning content elements.

However, there are two big downsides to the first sample:

1. **It creates a CSS nightmare.** You have to write rules for every possible nesting level of the sectioning content elements:

   ```
   section>section>h1 {}
   section>section>h2 {}
   section>article>h1 {}
   section>article>h2 {}
   article>section>h1 {}
   article>section>h2 {}
   section>aside>h1 {}
   section>aside>h2 {}
   aside>section>h1 {}
   aside>section>h2 {}
   ...
   ```

   Yikes!
2. **Modern browsers do not currently support the HTML5 outlining algorithm**, which means that accessibility tools such as JAWS Screen Reader[6] have to rely solely on the `h#` elements to determine the heading levels. Because

6. See http://www.freedomscientific.com/products/fs/jaws-product-page.asp.

these tools use heading levels as a way of providing simple page navigation (e.g., jumping from `h1` to `h1`), people using screen readers will not be able to experience your page correctly if you restart the heading level numbers with each new sectioning content element.

3.4 Conclusion

In this lesson, you have learned

- To work with the `<section>` and `<article>` tags.
- How the HTML5 outlining algorithm works.

4. HTML5 Audio and Video

In this lesson, you will learn...

1. How to use the `<audio>` tag.
2. How to use the `<video>` tag.
3. How to detect audio and video failure.
4. How to code for browsers that do not support the `<audio>` and `<video>` tags.

In this lesson, you will learn how to use the new HTML5 audio and video elements. As different browsers currently support different types of media, you will learn how to provide the necessary options to make your media work across browsers. You will also learn how to gracefully degrade your audio and video code.

4.1 Supported Media Types

There has been a lot of discussion about what file formats should be recommended in the HTML5 specification, but it is currently file format neutral. At one point the specification recommended OGG, but it removed that recommendation, in part due to Apple's refusal to implement OGG.[7]

From a practical point of view, to be sure that your media will play in your user's HTML5-compliant browser:

- For audio, you should provide both MP3 and OGG (.ogg) files.
- For video, you should provide both MP4 and OGG (.ogv) files. Also, in May 2010, Google introduced a new file format, WebM[8], which is meant to address the need for an open, royalty-free, media file format.

> Note that you must configure your web server to deliver the audio and video mime types that you use.
>
> In Apache, you can use `AddType video/ogg .ogv .ogg`. For more information, see https://developer.mozilla.org/en/Properly_Configuring_Server_MIME_Types and scroll down to the section labeled "How to set up your server to send the correct MIME types."
>
> In IIS, you can add Mime types through Computer Management. For more information, see http://technet.microsoft.com/en-us/library/cc725608(WS.10).aspx.

7. See http://en.wikipedia.org/wiki/Use_of_Ogg_formats_in_HTML5#Opposition.
8. See http://www.webmproject.org.

Mime Types

Your web server must be configured to serve audio and video files with the proper mime type.

- For information on adding mime types to IIS, see http://technet.microsoft.com/en-us/library/cc725608(WS.10).aspx.
- For information on adding mime types to Apache HTTP Server, see https://developer.mozilla.org/en/Properly_Configuring_Server_MIME_Types and scroll down to the section labeled "How to set up your server to send the correct MIME types."

This will not be an issue for this class if you are opening the files locally.

4.2 The audio Element

Adding audio to an HTML5 page couldn't be more simple. At its most basic, the tag looks like this:

```
<audio src="audio-file.mp3"></audio>
```

However, the above code would not *do* anything or even show up on the page.[9]

To give the user the ability to play and pause the audio, you need to add the `controls` attribute, which can be minimized as shown in the following demo. Open the demo in Chrome, IE9 or Safari to see it work.

9. Open html5-audio-and-video/Demos/audio-useless.html in your editor and then in your browser for proof.

Code Sample

html5-audio-and-video/Demos/audio-controls.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Audio - controls</title>
6.   <link href="style.css" rel="stylesheet" type="text/css">
7.   </head>
8.   <body>
9.   <h1>HTML5 Audio - controls</h1>
10.  <article>
11.   <hgroup>
12.    <h2>Casey at the Bat</h2>
13.    <h3>By Ernest Lawrence Thayer</h3>
14.    <h4>from the San Francisco Examiner - June 3, 1888</h4>
15.   </hgroup>
16.   <audio src="../Media/casey-at-the-bat.mp3" controls></audio>
17.
18.   <p>The Outlook wasn't brilliant for the Mudville nine that day:<br>
19.   The score stood four to two, with but one inning more to play.<br>
20.   And then when Cooney died at first, and Barrows did the same,<br>
21.   A sickly silence fell upon the patrons of the game.</p>
     -------Lines 22 through 84 Omitted-------
```

Code Explanation

The controller looks different in different browsers and there is no way to customize the controller using CSS. Opera's controller is shown below:

Audio Formats

If you open the last demo (html5-audio-and-video/Demos/audio-controls.html in Firefox 3.6, the controller will look like this:

That's because Firefox 3.6 does not support the MP3 format. Opera 10.6 also does not support MP3, but it shows its regular controller; it just won't play the file.

Browsers that support MP3

1. IE (from version 9 beta)
2. Chrome (from version 3.0)
3. Safari 5.0

Browsers that support OGG

1. Firefox 3.5
2. Opera 10.5+
3. Chrome 3.0+

Multiple Sources

Instead of using the `<audio>` tag's `src` attribute, you can nest one or more `<source>` tags within the `<audio>` tag, each with its own `src` attribute and a `type` attribute to let the browser know if the file is of a mime type that it supports (and therefore should bother loading). Browsers will use the first file they support. The code sample below shows how to make our audio tag work in all HTML5-compliant browsers.

Code Sample

html5-audio-and-video/Demos/audio-multiple-sources.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 15 Omitted-------
16.    <audio controls>
17.     <source src="../Media/casey-at-the-bat.ogg" type="audio/ogg;
         >>>  codecs=vorbis">
18.     <source src="../Media/casey-at-the-bat.mp3" type="audio/mpeg">
19.    </audio>
      -------Lines 20 through 86 Omitted-------
```

audio Tag Attributes

Attribute	Description
`src`	Points to the audio file. Only used when there are no nested `source` elements.
`controls`	Boolean. If present, indicates that controller will be displayed.
`preload`	Possible values: • `auto`: Browser should choose whether to preload the file. • `metadata`: Browser should only preload the metadata as the user is likely not to need the file. This is the default. • `none`: Browser should not preload anything as the user is likely not to need the file.
`autoplay`	Boolean. If present, audio will begin to play as soon as it has loaded.
`loop`	Boolean. If present, audio repeats indefinitely.

We have already discussed the `src` and `controls` attributes.

The `preload` attribute is relatively self-explanatory. You should set it to "auto" if you know the browser will need to download the file. Otherwise, you can most likely leave it out or set it to "metadata" (the default).

autoplay

You can get the audio file to play in the background by removing the `controls` attribute and adding the `autoplay` attribute as shown in the following demo.

Code Sample

html5-audio-and-video/Demos/audio-autoplay.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 15 Omitted-------
16.    <audio controls autoplay>
17.     <source src="../Media/casey-at-the-bat.ogg" type="audio/ogg;
         >>>  codecs=vorbis">
18.     <source src="../Media/casey-at-the-bat.mp3" type="audio/mpeg">
19.    </audio>
      -------Lines 20 through 87 Omitted-------
```

Code Explanation

Open the file to hear the poem.

loop

And the `loop` attribute simply makes the audio file restart again when it reaches the end.

Code Sample

html5-audio-and-video/Demos/audio-loop.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 43 Omitted-------
44.    <div>
45.     <p>
46.      <strong>Add Cheering</strong>: <em><a href="http://creativecom »»
         >>>  mons.org/licenses/sampling+/1.0/">Creative Commons Li »»
         >>>  cense</a></em><br>
47.      <small>- downloaded from <a href="http://www.freesound.org/sam »»
         >>>  plesViewSingle.php?id=22952">The Freesound Project</a></small>
         >>>
48.     </p>
49.     <audio controls loop>
50.      <source src="../Media/cheer.ogg" type="audio/ogg; codecs=vorbis">
51.      <source src="../Media/cheer.mp3" type="audio/mpeg">
52.     </audio>
53.    </div>
      -------Lines 54 through 95 Omitted-------
```

Code Explanation

You can now add cheering when Casey comes to bat.

Getting and Creating Audio Files

1. The Freesound Project (http://www.freesound.org/) - a website for downloading free sounds under the Creative Commons (http://creativecommons.org/) license. Requires registration.
2. Audacity (http://code.google.com/p/audacity/) - free, open source software for recording and editing sounds. Available for Windows, Mac, and Linux.
3. oggdropXPd (http://www.rarewares.org/ogg-oggdropxpd.php) - Windows-based drag and drop tool for quickly converting MP3 files to OGG.
4. Switch Audio File Converter Software (http://www.nch.com.au/switch/) - tool for quickly converting between audio file types. Available for Mac and Windows. Free and commercial licenses available.

4.3 The video Element

The `video` element is very similar to the `audio` element. Like with audio, it can be as simple as:

```
<video src="video-file.mp4"></video>
```

But it's always a good idea to provide multiple `source` options as different browsers support different video types. As with audio, browsers will use the first file they find that they support.

video Tag Attributes

Attribute	Description
`src`	Points to the video file. Only used when there are no nested `source` elements.
`controls`	Boolean. If present, indicates that controller will be displayed.
`preload`	Possible values: • `auto`: Browser should choose whether to preload the file. • `metadata`: Browser should only preload the metadata as the user is likely not to need the file. This is the default. • `none`: Browser should not preload anything as the user is likely not to need the file.
`autoplay`	Boolean. If present, video will begin to play as soon as it has loaded.
`loop`	Boolean. If present, video repeats indefinitely.
`height`	Height in pixels. You should use the actual height of the video.
`width`	Width in pixels or percentage. You should use the actual width of the video.

Exercise 4 Video - Multiple Sources

10 to 15 minutes

In this exercise, you will create an HTML5 file from scratch that plays video files.

1. Create a new HTML5 file called video-multiple-sources.html in the html5-audio-and-video/Exercises directory.
2. Write the code to include the justin.mp4 (mime type is video/mp4) and justin.ogv (mime type is video/ogg) files as source options. Both files are located in the html5-audio-and-video/Media directory.
3. Play around with video attributes such as: `controls`, `autoplay`, and `loop`.

Exercise Solution

html5-audio-and-video/Solutions/video-multiple-sources.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>HTML5 Video - controls</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<h1>HTML5 Video - controls</h1>
<video controls autoplay height="480" width="360">
 <source src="../Media/justin.mp4" type="video/mp4">
 <source src="../Media/justin.ogv" type="video/ogg">
</video>
</body>
</html>
```


Creating and Converting Video Files

1. Miro Video Converter (http://www.mirovideoconverter.com/) - free, open source tool for quickly converting between video file types. Available for Mac and Windows.
2. Windows Live Movie Maker (http://explore.live.com/windows-live-movie-maker) - free tool for making movies (WMV files) on Windows 7.

4.4 Accessibility

The HTML5 specification includes a `<track>` element, which would be used to provide subtitles, captions, descriptions, chapter information, and metadata for media delivered using the `audio` and `video` elements. This is still in a bit of flux and browsers don't yet support it, but if you wish to learn more, check out http://dev.w3.org/html5/spec/video.html#the-track-element.

4.5 Scripting Media Elements

The `audio` and `video` elements are accessible and controllable via the DOM. Here are some of the important methods and properties[10]:

Method/Property	Description
`currentTime`	Read/Write. The current play location in seconds.
`duration`	Read only. The length of the media file in seconds.
`played`	Boolean. True if media file has been started.
`ended`	Boolean. True if media file has been played in full.
`autoplay`	Boolean. Read/Write.
`loop`	Boolean. Read/Write.
`play()`	plays media from current location (as per `currentTime`)
`pause()`	pauses media

The following demo shows how to use the media API to create rudimentary custom controls and to report the current time of a media file.

10. See http://www.w3.org/TR/html5/video.html#media-elements.

Code Sample

html5-audio-and-video/Demos/audio-javascript.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 6 Omitted-------
7.    <script>
8.     var playTimer;
9.
10.    window.addEventListener("load",function() {
11.     document.getElementById("cmd-play").addEventListen »»
         >>>  er("click",play,false);
12.     document.getElementById("cmd-pause").addEventListen »»
         >>>  er("click",pause,false);
13.     document.getElementById("cmd-restart").addEventListen »»
         >>>  er("click",restart,false);
14.    },false);
15.
16.    function play() {
17.     document.getElementById("my-audio").play();
18.     reportTime();
19.    }
20.
21.    function reportTime() {
22.     var curTime = document.getElementById("my-audio").currentTime;
23.     document.getElementById("play-time").innerHTML=Math.round(curTime);
24.     playTimer = setTimeout(reportTime,500);
25.    }
26.
27.    function pause() {
28.     document.getElementById("my-audio").pause();
29.     if (typeof playTimer != "undefined") {
30.      clearTimeout(playTimer);
31.     }
32.    }
33.
34.    function restart() {
35.     document.getElementById("my-audio").currentTime=0;
36.     play();
37.    }
38.   </script>
39.   </head>
40.   <body>
41.   <h1>HTML5 Audio - JavaScript</h1>
```



```
42.  <article id="poem">
43.   <hgroup>
44.    <h2>Casey at the Bat</h2>
45.    <h3>By Ernest Lawrence Thayer</h3>
46.    <h4>from the San Francisco Examiner - June 3, 1888</h4>
47.   </hgroup>
48.   <audio id="my-audio">
49.    <source src="../Media/casey-at-the-bat.ogg" type="audio/ogg;
         >>>  codecs=vorbis">
50.    <source src="../Media/casey-at-the-bat.mp3" type="audio/mpeg">
51.   </audio>
52.   <menu type="toolbar">
53.    <button id="cmd-play" title="Play">Play</button>
54.    <button id="cmd-pause" title="Pause">Pause</button>
55.    <button id="cmd-restart" title="Restart">Restart</button>
56.   </menu>
57.   <div id="time-display">
58.    Time: <output id="play-time">0</output> seconds.
59.   </div>
     -------Lines 60 through 127 Omitted-------
```

Code Explanation

Notice:

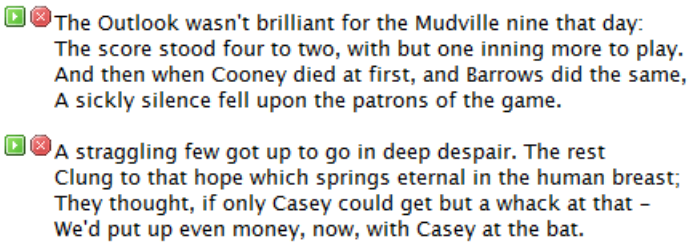
1. The `menu` with the three buttons.
2. The `output` element to show the current time.
3. The JavaScript:
   A. When the window loads, we add event listeners for each of the three menu buttons to call the `play()`, `pause()`, and `restart()` functions.
   B. `play()` - calls the audio element's `play()` method and then calls the `reportTime()` function.
   C. `reportTime()` - populates the `output` element with the audio element's `currentTime` value, rounded to the nearest second. Iteratively calls itself every half second to recheck the `currentTime` value.
   D. `pause()` - calls the audio element's `pause()` method and clears the timer.
   E. `restart()` - sets the audio element's `currentTime` value to 0 and calls `play()`.

Exercise 5 Media API

20 to 30 minutes

In this exercise, you will add a feature to the preceding demo that allows the user to jump to the beginning to a stanza.

1. Open html5-audio-and-video/Exercises/audio-javascript.html in your editor.
2. Notice that each `<p>` tag now has an `id` of the format `pos-` and a number. This number represents the time (in seconds) at which this stanza begins.
3. You will add JavaScript code to insert working **play** and **pause** images (found in the Images folder) at the beginning of each stanza, like this:



   - When the **play** image is clicked, the audio should jump to that stanza and play.
   - When the **pause** image is clicked, the audio should pause.

***Challenge**

1. Add code to the `reportTime()` function so that the stanza currently being played is given the "highlight" class (already defined in style.css.
2. Make sure to remove the class when the stanza is no longer being played.

Exercise Solution

html5-audio-and-video/Solutions/audio-javascript.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 6 Omitted-------
7.    <script>
8.     var playTimer;
9.
10.    window.addEventListener("load",function() {
11.     document.getElementById("cmd-play").addEventListen »»
         >>>  er("click",play,false);
12.     document.getElementById("cmd-pause").addEventListen »»
         >>>  er("click",pause,false);
13.     document.getElementById("cmd-restart").addEventListen »»
         >>>  er("click",restart,false);
14.     var stanzas = document.getElementById("poem").getElementsByTag »»
         >>>  Name("p");
15.     for (var i=0; i<stanzas.length; ++i) {
16.      insertPlayButton(stanzas[i]);
17.     }
18.    },false);
19.
20.    function play() {
21.     document.getElementById("my-audio").play();
22.     reportTime();
23.    }
24.
25.    function reportTime() {
26.     var curTime = document.getElementById("my-audio").currentTime;
27.     document.getElementById("play-time").innerHTML=Math.round(curTime);
28.     playTimer = setTimeout(reportTime,500);
29.    }
30.
31.    function pause() {
32.     document.getElementById("my-audio").pause();
33.     if (typeof playTimer != "undefined") {
34.      clearTimeout(playTimer);
35.     }
36.    }
37.
38.    function restart() {
39.     document.getElementById("my-audio").currentTime=0;
40.     play();
```



```
41.    }
42.
43.    function insertPlayButton(stanza) {
44.     var pos=stanza.id.split("-")[1];
45.     var startHTML = stanza.innerHTML;
46.     var buttonHTML = "<img src='Images/play.gif' title='Play'
         >>>  onclick='jumpTo(" + pos + ")'><img src='Images/pause.gif' ti »»
         >>>  tle='Play' onclick='pause()'>";
47.     stanza.innerHTML=buttonHTML+startHTML;
48.    }
49.
50.    function jumpTo(pos) {
51.     document.getElementById("my-audio").currentTime=pos;
52.     play();
53.    }
54.   </script>
      -------Lines 55 through 143 Omitted-------
```

Challenge Solution

html5-audio-and-video/Solutions/audio-javascript-challenge.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 23 Omitted-------
24.
25.    function reportTime() {
26.     var curTime = document.getElementById("my-audio").currentTime;
27.     document.getElementById("play-time").innerHTML=Math.round(curTime);
28.     playTimer = setTimeout(reportTime,500);
29.     var stanzas = document.getElementById("poem").getElementsByTag »»
         >>>  Name("p");
30.     var pos;
31.     var stanzaFound=false;
32.     for (var i=stanzas.length-1; i>=0; --i) {
33.      pos=stanzas[i].id.split("-")[1];
34.      if (pos <= curTime && !stanzaFound) {
35.       stanzaFound=true;
36.       stanzas[i].className="highlight";
37.      } else {
38.       stanzas[i].className="";
39.      }
40.     }
41.    }
      -------Lines 42 through 155 Omitted-------
```

4.6 Dealing with Non-Supporting Browsers

The `audio` and `video` elements have been implemented to hide and ignore any unknown children, which means that we can throw code inside of them to be consumed/displayed by non-HTML5 browsers.

Code Sample

html5-audio-and-video/Demos/video-non-supporting-browsers.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Video - controls</title>
6.   <link href="style.css" rel="stylesheet" type="text/css">
7.   </head>
8.   <body>
9.   <h1>HTML5 Video - controls</h1>
10.  <video controls autoplay height="480" width="640">
11.   <source src="../Media/justin.mp4" type="video/mp4">
12.   <source src="../Media/justin.ogv" type="video/ogg">
13.   <p class="warning">Your browser doesn't support the video tag.</p>
14.  </video>
15.  </body>
16.  </html>
```

Code Explanation

If you don't have any non-HTML5-compliant browsers installed, you may wish to install IETester (http://www.debugbar.com/download.php), which is a free tool for seeing how older version of Internet Explorer will display your pages.

Graceful Degradation

Because the code encapsulated within the HTML5 `<video>` and `<audio>` tags is not displayed, we can use that to gracefully degrade to Flash and then from there to an image.

Kroc Camen wrote a "chunk of HTML code" called Video for Everybody (http://camendesign.com/code/video_for_everybody), which is freely reusable. The code is shown below (with the WebM format added in):


```
<!-- first try HTML5 playback: if serving as XML, expand 'controls'
 to 'controls="controls"' and autoplay likewise -->
<!-- warning: playback does not work on iPad/iPhone if you include
 the poster attribute! fixed in iOS4.0 -->
<video width="640" height="360" controls>
 <!-- MP4 must be first for iPad! -->
 <source src="__VIDEO__.MP4" type="video/mp4" /><!-- WebKit video
-->
 <source src=""__VIDEO__.webm" type="video/webm" /><!-- WebM Format
 -->
 <source src="__VIDEO__.OGV" type="video/ogg" /><!-- Firefox / Opera
 -->
 <!-- fallback to Flash: -->
 <object width="640" height="360" type="application/x-shockwave-
flash" data="__FLASH__.SWF">
  <!-- Firefox uses the 'data' attribute above, IE/Safari uses the
 param below -->
  <param name="movie" value="__FLASH__.SWF" />
  <param name="flashvars" value="controlbar=over&im »»
age=__POSTER__.JPG&file=__VIDEO__.MP4" />
  <!-- fallback image. note the title field below, put the title
of the video there -->
  <img src="__VIDEO__.JPG" width="640" height="360" alt="__TITLE__"

   title="No video playback capabilities, please download the video
 below" />
 </object>
</video>
<!-- you *must* offer a download link as they may be able to play
the file locally. customise this bit all you want -->
<p>
 <strong>Download Video:</strong>
 Closed Format: <a href="__VIDEO__.MP4">"MP4"</a>
 WebM Format: <a href="__VIDEO__.webm">"WebM"</a>
 Open Format: <a href="__VIDEO__.OGV">"Ogg"</a>
</p>
```

4.7 Conclusion

In this lesson, you have learned how to add HTML5 audio and video to your pages.

For additional details on audio and video types, see http://wiki.whatwg.org/wiki/Video_type_parameters.

5. HTML5 Forms

In this lesson, you will learn...

1. About Modernizr, the JavaScript library for testing for HTML5 support.
2. About HTML5's new form fields and attributes.
3. About new types of inputs in HTML5.
4. About built-in HTML5 form validation.
5. About the new HTML5 output, progress and meter elements.

The promise of HTML5 forms is great - richer, more meaningful, and backward-compatible forms that are consistent across browsers and include built-in client-side validation (read, no need for JavaScript for form validation).

The current reality is far from great - only Opera does a half-way decent job of implementing most of the new form fields. That said there are some things you can do now to take advantage of Opera's implementation and be ready for other browsers without causing any harm in the non-supporting browsers.

So let's dig in and learn how to use the new HTML5 form features.

> In HTML5, form elements like `input`, `select`, and `textarea` no longer have to be nested within a `<form>` tag.

5.1 Modernizr

Before we get into HTML5 forms, we will re-introduce you to a great little JavaScript library called Modernizr (http://www.modernizr.com/). Modernizr is used to check the browser for feature support so you can write code like this:

Syntax

```
if (Modernizr.xyzFeature) {
 //do this;
} else {
 //do something different
}
```

We have included it in the html5-common folder in the root of the class files. Here's an example of how you can check for the new `email` input type:

Code Sample

html5-forms/Demos/modernizr.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 5 Omitted-------
6.    <script src="../../html5-common/modernizr.min.js"
         >>>  type="text/javascript"></script>
7.    <script>
8.     if (Modernizr.inputtypes.email) {
9.      alert("woohoo!");
10.    } else {
11.     alert("boohoo!");
12.    }
13.   </script>
      -------Lines 14 through 17 Omitted-------
```

To see which HTML5 features it supports, open html5-forms/Demos/modernizr-full-check.html in your browser.

We use **modernizr** in some of the demos in this lesson.

5.2 New Input Types

HTML5 introduces thirteen new input types:

1. search
2. tel
3. url
4. email
5. datetime
6. date
7. month
8. week
9. time
10. datetime-local
11. number

12. range
13. color

Unfortunately, the browser implementation of these new input types is still spotty at best. The table below shows the current state of browser support for these input types according to http://www.findmebyip.com/litmus:

Platform:	**Macintosh**				**Windows**									
Browser:														
Version	**11**	**3.6**	**5**	**7**	**11**	**3.6**	**4.03**	**5**	**6**	**7**	**8**	**9**	**7**	**8**
search	**Y**	N	**Y**	**Y**	**Y**	N	**Y**	**Y**	N	N	N	N	**Y**	**Y**
tel	**Y**	N	**Y**	**Y**	**Y**	N	**Y**	**Y**	N	N	N	N	**Y**	**Y**
url	**Y**	N	**Y**	**Y**	**Y**	N	N	**Y**	N	N	N	N	**Y**	**Y**
email	**Y**	N	**Y**	**Y**	**Y**	N	N	**Y**	N	N	N	N	**Y**	**Y**
datetime	**Y**	N	N	N[11]	**Y**	N	N	N	N	N	N	N	N	N
date	**Y**	N	N	N	**Y**	N	N	N	N	N	N	N	N	N
month	**Y**	N	N	N	**Y**	N	N	N	N	N	N	N	N	N
week	**Y**	N	N	N	**Y**	N	N	N	N	N	N	N	N	N
time	**Y**	N	N	N	**Y**	N	N	N	N	N	N	N	N	N
datetime-local	**Y**	N	N	N	**Y**	N	N	N	N	N	N	N	N	N
number	**Y**	N	N	N	**Y**	N	N	N	N	N	N	N	N	**Y**
range	**Y**	N	**Y**	**Y**	**Y**	N	N	**Y**	N	N	N	N	**Y**	**Y**
color	**Y**	N	N	N	**Y**	N	N	N	N	N	N	N	N	N

The good news is that these browsers all fall back to `type="text"` when they don't recognize an input type.

search

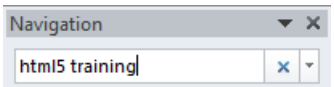
The new HTML5 `search` input type is only supported by:

1. Safari
2. Chrome

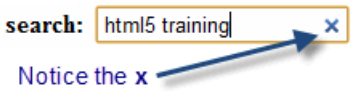
11. See note on (see page 82)

3. Firefox 4
4. Opera

Most input fields are meant to be filled out only one time and then submitted for processing. But a search box is a bit different. For example, consider Microsoft Word 2010's search box:



Notice the **x** used for clearing the box. If you look at search boxes in other applications, you'll notice many of them provide a simple way to clear the text. HTML5 browsers that support the `search` input type are taking the same approach. For example, here's Chrome's search box:



Note that the **x** doesn't show up until you have entered some text into the field.

tel

The new HTML5 `tel` input type is only supported by:

1. Safari
2. Chrome
3. Firefox 4
4. Opera

Even in your computer-based supporting browsers, you don't really gain anything by using the `tel` input type. As telephone numbers can come in all different formats, there are no constraints on what can be entered here. You could, however, add your

own custom validation to all telephone inputs and use the `type="tel"` as a means of finding them.

Also, user agents are free to provide a different/better means for filling out input fields based on their type. For example, the iPhone provides a more appropriate interface for filling out fields of the `tel` type:

url and email

The new HTML5 `url` and `email` input types are only supported by:

1. Safari
2. Chrome
3. Opera

Supporting browsers can provide validation for `type="url"` fields to make sure the user enters a valid URL.

Also, like with `type="tel"`, user agents are free to provide a different/better means for entering URLs and email addresses.

For example, for `url` types the iPhone provides keys for ".", "/" and ".com" and does not provide a "space" key as spaces are not allowed in URLs.

For emails, the iPhone provides an "@" and "." keys. For some reason, it does not provide a ".com" key.

> Interestingly enough, the iPhone also provides a "space" key for emails. This is because `email` input types can include a `multiple`[12] attribute, which, when

12. The email input type's multiple attribute is only supported by Opera 11.

included, allows users to enter multiple emails delimited by spaces. If the iPhone were a little smarter, it would only include the "space" key when the "multiple" attribute was present.

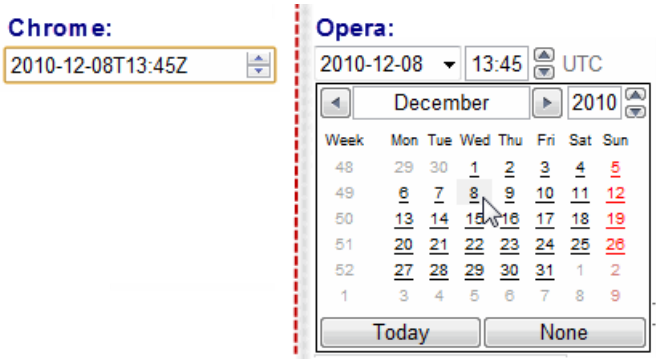
date/time input types

The new HTML5 `date`, `datetime`, `datetime-local`, `month`, `week`, and `time` input types are only supported by:

1. Opera
2. Chrome

The table above, which is based on Modernizr's reporting, indicates that Google Chrome doesn't support these date/time types. In fact, Chrome does support them and, according to chromium.org[13], it *has* since version 5. However, according to GitHub[14], Chrome's support is not yet fully implemented and allows badly formatted dates to pass validation.

We did not notice this in our own testing, but we do feel that Chrome 8's current implementation of the date fields is pretty lame. It does not provide nice widgets for entering dates and times, but rather just lets the user either type in a valid entry or scroll through dates and times using up down arrows. The values shown by Chrome are not at all user friendly:



What user is going to know to enter "2010-12-08T13:45Z" for a date and time?

number

The new HTML5 `number` input type is only supported by:

13. See http://www.chromium.org/developers/web-platform-status/forms.
14. See https://github.com/Modernizr/Modernizr/issues/closed#issue/154.

1. Opera
2. Chrome

Note that although Safari isn't listed, it does recognize valid numbers to the same extent that Chrome and Opera do. It just doesn't provide a nice interface for entering numbers on the computer. On the iPhone, it does give you a number keypad, which is nice.

Both Opera and Chrome use up and down buttons (spinboxes) to scroll through numbers and they also allow you to use the up and down arrows on the keyboard. Opera's spinbox is shown below:

All three implementations (Opera, Chrome and Safari) are subpar in that they don't accept decimals in their standard format (a bug has been filed[15]). To see this:

1. Open html5-forms/Demos/input-validity.html in Chrome, Opera, or Safari.
2. In the number field, enter 3.3 or some other decimal.
3. Press the **Check Validity** button. You'll receive an alert reading "3.3 is NOT a valid number." Oops.

range

The new HTML5 `range` input type is only supported by:

1. Opera
2. Chrome
3. Safari

All supporting browsers currently represent `range` input types as sliders. This is nice, but the problem is that they don't indicate a value, so the user doesn't know what they've selected. Consider the following code:

15. See https://bugs.webkit.org/show_bug.cgi?id=42484.

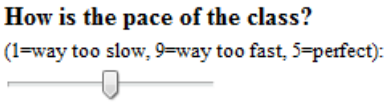
Code Sample

html5-forms/Demos/input-range.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.     <label for="pace">
10.     <strong>How is the pace of the class?</strong><br>
11.     <small>(1=way too slow, 9=way too fast, 5=perfect):</small>
12.    </label><br>
13.    <input type="range" name="pace" id="pace" min="1" max="9" value="5">
      -------Lines 14 through 16 Omitted-------
```

Code Explanation

The above code will render as follows (in Google Chrome):



The problem is that there is no indication of the current value. We can try fixing this with this simple little trick:

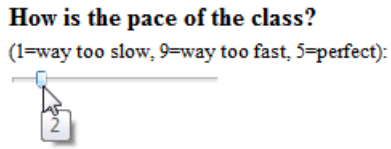
Code Sample

html5-forms/Demos/input-range-title.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.     <label for="pace">
10.     <strong>How is the pace of the class?</strong><br>
11.     <small>(1=way too slow, 9=way too fast, 5=perfect):</small>
12.    </label><br>
13.    <input type="range" name="pace" id="pace" min="1" max="9" value="5"
14.     title="5" onchange="this.title=this.value">
      -------Lines 15 through 17 Omitted-------
```


Code Explanation

The above code will render as follows (in Safari):



This trick works pretty well in Safari, which displays the `title` value immediately after the mouse hovers over the field. However, with Chrome and Opera there is a slight delay, which makes this solution less than ideal. Also, the value is only visible when the cursor is over the field.

The output Element

There is a new HTML5 `output` element used to show output generated by a script on a page. For example, you could use it to show an error message or the result of a calculation based on values entered in form fields.

We can use the `output` element to create a better solution for making our `range` slider more user friendly:

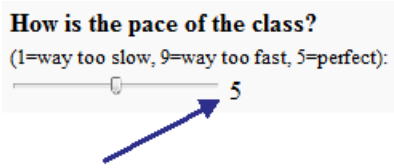
Code Sample

html5-forms/Demos/input-range-output.html

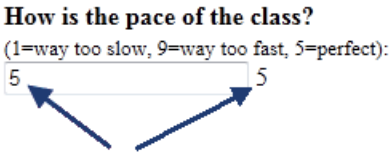
```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.     <label for="pace">
10.     <strong>How is the pace of the class?</strong><br>
11.     <small>(1=way too slow, 9=way too fast, 5=perfect):</small>
12.    </label><br>
13.    <input type="range" name="pace" id="pace" min="1" max="9" value="5"
14.    onchange="document.getElementById('paceoutput').innerHTML=this.value;">
          >>>
15.    <output for="pace" id="paceoutput">5</output>
      -------Lines 16 through 18 Omitted-------
```

Code Explanation

The above code will render as follows (in Safari):



Notice the number 5 immediately after the slider telling us the current value of the slide. This number will update when the slider value changes. This solution works in browsers that support `range`. However, it creates a problem with non-supporting browsers. Here's how Internet Explorer 9 displays this page:



Notice that 5 is repeated twice: once within the text field and once after it.

To fix this, we need to populate the `output` element dynamically if and only if the `range` input type is supported. Here's a possible solution:

Code Sample

html5-forms/Demos/input-range-output-dynamic.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>HTML5 Form - Input Range Type</title>
6.   <script src="../../html5-common/modernizr.min.js"
        >>>  type="text/javascript"></script>
7.   <script>
8.    window.addEventListener("load", function() {
9.     if (Modernizr.inputtypes.range) {
10.     document.getElementById('paceoutput').innerHTML=document.getElement »»
         >>>  ById('pace').value;
11.     document.getElementById("pace").addEventListener("change", function()
         >>>   {
12.       document.getElementById('paceoutput').innerHTML=this.value;
13.     }, false);
14.    }
15.   }, false);
16.  </script>
17.  </head>
18.  <body>
19.  <form method="post" action="example.xyz">
20.   <label for="pace">
21.    <strong>How is the pace of the class?</strong><br>
22.    <small>(1=way too slow, 9=way too fast, 5=perfect):</small>
23.   </label><br>
24.   <input type="range" name="pace" id="pace" min="1" max="9" value="5">
25.   <output for="pace" id="paceoutput"></output>
26.  </form>
27.  </body>
28.  </html>
```

Code Explanation

This solves the problem. The code, which runs when the page is loaded:

1. Sets the `innerHTML` of the `output` element to the value of the `range` input.
2. Attaches an event listener to catch any changes of that value and update the `output` accordingly.
3. And it only runs if the browser supports the `range` input type. We used Modernizr to check for that.

min, max, and step attributes

The `min`, `max`, and `step` attributes can be used on the `number`, `range`, and datepicker elements.

`min` and `max` are intuitive. They are used to set the minimum and maximum possible values.

The `step` attribute is used to indicate possible values in between the `min` and `max` values.

These attributes are only supported by:

1. Opera
2. Chrome

Code Sample

html5-forms/Demos/input-number-step.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 12 Omitted-------
13.   <label for="evennumber">Even Number: </label>
14.   <input type="number" name="evennumber" id="evennumber" min="0" max="100"
          >>>   step="2">
      -------Lines 15 through 16 Omitted-------
```

To see how this works:

1. Open html5-forms/Demos/input-number-step.html in Chrome or Opera.
2. Use the spinner to go up and down. Notice only even numbers are shown and that you cannot go below 0 or above 100.
3. Type in 15 or some other odd number. Notice the field turns red.

> Safari can validate the numbers based on min, max, and step, but doesn't provide an easy way to pick a number.

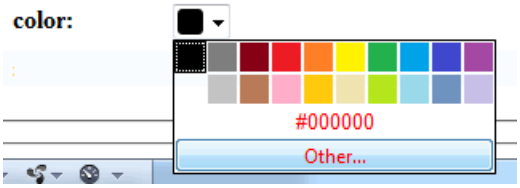
color

Only Opera 11 supports the new HTML5 `color` input type, but, again, as browsers all fall back to `type="text"` when they don't recognize an input type, there is no harm in using the `color` input type now. When other browsers start implementing it, your visitors will likely get a nice color picker.

Chrome and Safari recognize valid color names and hexadecimal color formats, but don't provide an easy way to pick them. As we saw earlier with the `number` type, you can check the validity of the `color` input type with JavaScript. To see this:

1. Open html5-forms/Demos/input-validity.html in Chrome or Safari.
2. In the color field, enter "red" or "#ff0000".
3. Press the **Check Validity** button. You'll receive an alert reading "red is a valid color."
4. Now try it with a bad color name (e.g., "foo") and you'll get an alert reading "foo is NOT a valid color."

Opera's color picker is shown below:



If you click on Other... you get a large color picker with more options.

5.3 HTML5 New Form Attributes

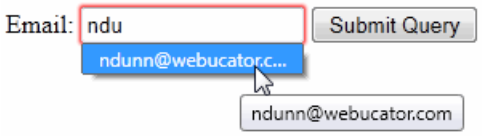
HTML5 introduces two new attributes of the `form` element:

1. `autocomplete` - "on" or "off". When set to "off" the browser should not use built-in features to help a user auto-fill the form.
2. `novalidate` - Boolean. If included, the form should not validate on submission.

autocomplete

Browsers have different ways of choosing data to use for `autocomplete`. For example, Chrome does it based on previous form entries, while Opera uses its built-in contact list. Users can manage autocomplete in their browser's settings. When

`autocomplete` is on, you see something like the behavior shown below in Firefox 4:



When `autocomplete` is off, that behavior is blocked.

novalidate

The purpose of the `novalidate` attribute is to allow users to submit their forms even if the form data is invalid. For example, perhaps they're filling out an online application and you want to allow them to save the current state of their application even though some fields might not be valid.

But only Opera and Firefox 4 currently support validation on submission, so all other browsers are currently exhibiting the `novalidate` behavior by default. However, we can use Opera to illustrate.

Take a look at the following code.

Code Sample

html5-forms/Demos/form-attributes.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 7 Omitted-------
8.    <form method="post" action="" autocomplete="on">
9.     <!--novalidate attribute not included-->
10.    <label for="fullname">Name: </label>
11.    <input type="text" name="fullname" id="fullname" required>
12.    <label for="email">Email: </label>
13.    <input type="email" name="email" id="email">
14.    <input type="submit">
15.   </form>
      -------Lines 16 through 17 Omitted-------
```

Code Explanation

The screen shot below shows this page in Opera. Notice that the `fullname` field is required, but left empty, and the `email` field is of the email type, but is not a

valid email. So, both fields are currently invalid. When submitting this form in Opera, you get the following results:

If we were to include the `novalidate` attribute, the form would submit without erroring.

But Opera's (and Firefox's) behavior is a bit ugly as it only reports the first error found. So at this stage, the promise of validation without JavaScript is yet to be fulfilled. And, unless browsers do a better job than Opera has in reporting form errors on submission, it may remain unfulfilled even after all the browsers technically support the specification.

5.4 Some Other New Form Field Attributes

required

The `required` attribute is used to indicate that a form field must contain data. It is only supported by:

1. Opera
2. Chrome
3. Firefox 4

Code Sample

html5-forms/Demos/required.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.     <label for="fullname">Name: </label>
10.    <input type="text" name="fullname" id="fullname" required>
      -------Lines 11 through 14 Omitted-------
```

There is a bug in WebKit browsers (Chrome and Safari) that prevents `required` from working on `color` input types. Open html5-forms/Demos/webkit-color-required-bug.html in Chrome or Safari to see the bug.

placeholder

The `placeholder` attribute is used to add placeholder text to the form field. It is only supported by:

1. Opera
2. Chrome
3. Firefox 4

Code Sample

html5-forms/Demos/placeholder.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.     <label for="fullname">Name: </label>
10.    <input type="text" name="fullname" id="fullname" placeholder="Enter
          >>>  Firstname">
11.    <input type="submit">
      -------Lines 12 through 14 Omitted-------
```

Code Explanation

Here's what it looks like in Firefox 4:

Name: Enter Firstname

autofocus

The `autofocus` attribute can only go in one field on the page. It instructs the browser to place focus on that field allowing the user to begin typing as soon as the page loads.

Code Sample

html5-forms/Demos/autofocus.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 8 Omitted-------
9.     <label for="fullname">Name: </label>
10.    <input type="text" name="fullname" id="fullname" autofocus>
11.    <label for="email">Email: </label>
12.    <input type="email" name="email" id="email">
13.    <input type="submit">
      -------Lines 14 through 16 Omitted-------
```

We used to accomplish this by adding `onload="document.getElementById('fullname').focus();"` to the `<body>` tag or some other similar scripting method. The HTML5 way is easier and comes with at least two added benefits:

1. It works when scripting is turned off.
2. Browser makers could potentially provide a preference setting for disabling autofocus.

Unfortunately, there's an associated bug in Google Chrome. When one field in a form contains the autofocus attribute, Chrome's number spinboxes break. Check out the following code:

Code Sample

html5-forms/Demos/chrome-autofocus-bug.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 10 Omitted-------
11.    <div>
12.     <label for="fullname">Your name:</label>
13.     <input type="text" name="fullname" id="fullname" autofocus>
14.    </div>
15.    <div>
16.     <label for="age">Your age:</label>
17.     <input type="number" name="age" id="age" min="0" step="1">
18.    </div>
      -------Lines 19 through 21 Omitted-------
```

That `autofocus` attribute in the **fullname** field messes up the **age** field's spinbox in Chrome (testing on v. 8.0.552.215 on Windows 7). When you click on the up or down arrow, nothing happens until the field loses focus. To test this:

1. Click on the age field's up arrow three times. Nothing happens.
2. Click somewhere else on the page to remove focus from the element. The value updates to 2.

Chrome registers all three clicks, but doesn't update the value until focus is removed from the element.

Very strange. If you remove the `autofocus` attribute from the **fullname** element, everything works swimmingly.

So, I think the takeaway for now is that we cannot yet use **autofocus** in any forms that include **number** types.

Accessibility and Autofocus

Autofocusing on a form element (the HTML5 way or through script) can cause problems for people using screen readers. For sighted people, it's generally okay if we provide one focus point for the keyboard (i.e., autofocus) and another one for the eyes (e.g., instructions for filling out the form), but for people using screen readers, there is only one focus point. So be careful not to skip over important contextual content when directing focus to the first form field.

autocomplete

The `autocomplete` attribute is used to override the browser's or form element's autocomplete behavior on a field by field basis. It is widely supported by HTML5-compliant browsers.

form

The `form` attribute is used to associate a form element with a form (by the form's `id`) in which it is **not** nested. This would typically be used for styling purposes; for example, if you wanted to have one or more form elements appear separately from the main form.

Note that the `form` attribute can be used with the `label` and `fieldset` elements as well as all the form fields (e.g., `input`, `textarea`, `select`,...). Unfortunately, only Opera supports it.

pattern

The `pattern` attribute is used to force a specific pattern (via a regular expression[16]) within an form field. It is only supported by:

1. Opera
2. Chrome
3. Firefox 4

Code Sample

html5-forms/Demos/pattern.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 13 Omitted-------
14.    <label for="telephone">Telephone: </label>
15.    <input type="tel" name="telephone" id="telephone" pattern="^\(?([2-
          >>>  9]\d\d)\)?[\-\. ]?([2-9]\d\d)[\-\. ]?(\d{4})$">
      -------Lines 16 through 19 Omitted-------
```

Code Explanation

Open this file in one of the supporting browsers and enter data in the field. It should remain red until the value is a valid 10-digit U.S.-style phone number. It will allow for parentheses around the area code and for dashes, spaces, and dots as separators.

5.5 New Form Elements

HTML5 introduces these new elements, which are often associated with forms:

1. `output`
2. `datalist`
3. `progress`
4. `meter`

We covered the `output` element (see page 85) earlier in this lesson.

The others are covered below.

datalist

The `datalist` element combined with the `list` attribute is used to provide a list of suggestions for an input field. It differs from the `select` element in that the

16. For a great little book on regular expressions, see Sams Teach Yourself Regular Expressions in 10 Minutes (http://www.amazon.com/Teach-Yourself-Regular-Expressions-Minutes/dp/0672325667).

associated `input` can accept values that are not included in the `datalist`. It is only supported by:

1. Opera
2. Firefox 4

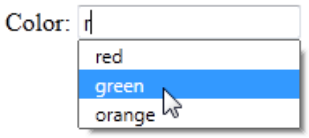
Code Sample

html5-forms/Demos/datalist.html

```
1.   <!DOCTYPE HTML>
     -------Lines 2 through 7 Omitted-------
8.   <label for="color">Color: </label>
9.   <input type="color" name="color" id="color" list="color-list">
10.  <datalist id="color-list">
11.   <option value="red">
12.   <option value="blue">
13.   <option value="green">
14.   <option value="yellow">
15.   <option value="orange">
16.  </datalist>
     -------Lines 17 through 18 Omitted-------
```


Code Explanation

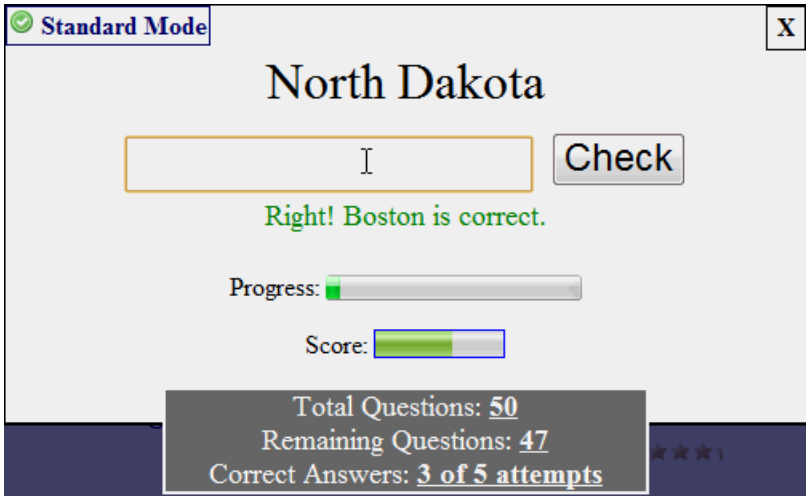
Here's what it looks like in Firefox 4:



progress and meter

HTML5 introduces the new `progress` and `meter` elements. They sometimes get confused, so we'll illustrate with an example.

We use both `progress` and `meter` in our HTML5 Cards (http://www.html5-cards.com) application as shown below:



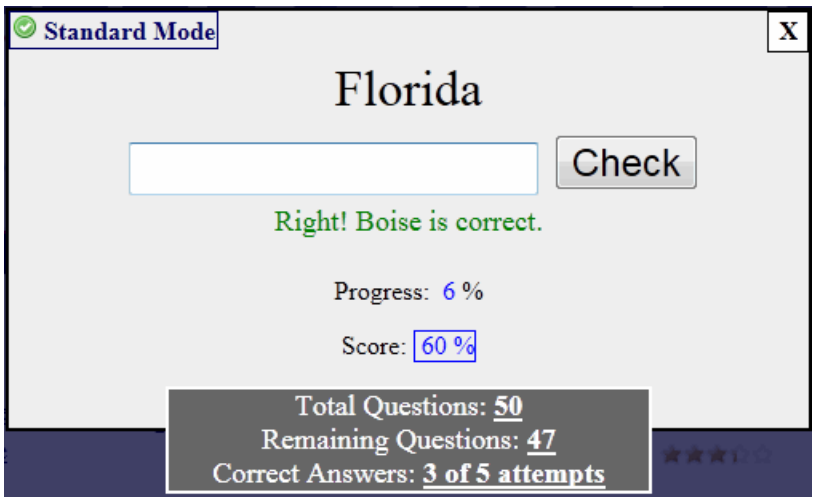
The `progress` element is used to show the progress through some action. In this case, through completing the "deck" of flash cards. Another example would be a file download or completion of a form.

The `meter` element, according to the spec[17], "represents a scalar measurement within a known range". So the `meter` element is only used when you know the minimum and maximum values. We use it in the above HTML5 cards example to show the user's current score (between 0% and 100%). This is different from `progress`, which shows how much of the deck has been completed. The HTML markup for the two tags is shown below:

17. See http://www.w3.org/TR/html5/the-button-element.html#the-meter-element.

```
Progress: <progress id="amountComplete" max="100" val »»
ue="0"><span>0</span>%</progress>
Score: <meter id="score" max="100" value="0"><span>0</span>%</meter>
```

Only Chrome and Opera support the `progress` and `meter` tags, so we use the nested `<span>` tags in combination with JavaScript to provide similar functionality in other browsers. For example, here's what we show in IE9:



The Form Element's change Event Handler

The `progress` and `meter` elements can be updated using the new `change` event handler on the `form`, rather than capturing changes on each individual form element:

```
document.getElementById('my-form').addEventListener("change",up »»
dateProgress,false);
```


Exercise 6 An HTML5 Quiz

30 to 45 minutes

In this exercise, you will create an HTML5 quiz that validates form entries and reports the percentage of both the valid (but not necessarily correct) answers and the percentage of correct answers.

1. Open html5-forms/Exercises/quiz.html in your editor.
2. Make the following changes to the form:
   A. Add placeholders to all questions.
   B. Make all questions required.
   C. Question 1 should only accept valid colors.
   D. Question 2 should only accept integers greater than or equal to 20.
   E. Question 3 should only accept the pattern shown in the footnote below[18] (don't look if you want to figure out the pattern yourself).
   F. Question 4 should only accept valid dates.
   G. Question 5 should only accept valid URLs and should provide a list of common search engines to choose from, but should not limit the answer to those shown in the list.
3. At the bottom of the form:
   A. Add a bar showing the percentage of valid (but not necessarily correct) answers answered. Give it an `id` of "quiz-progress".
   B. Add a bar showing the percentage of correct answers answered. Give it an `id` of "quiz-success".
4. Finish the `updateMeasures()` function so that it correctly updates the two bars added above on every form change. **Hint**: one way to do this is to loop through the input fields stored in the `questions` variables.

18. `[9\+\-\*/]{4,7}`

Exercise Code

html5-forms/Exercises/quiz.html

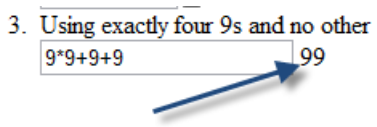
```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>HTML5 Quiz</title>
<link href="style.css" rel="stylesheet" type="text/css">
<script>
 if (window.addEventListener) {
  window.addEventListener("load", addLoadEvents, false);
 }
 function addLoadEvents() {
  document.getElementById('quiz').addEventListener("change",function()
    >>>   {
   updateMeasures();
  }, false);
 }

 function updateMeasures() {
  var questions = document.getElementsByTagName("input");
  var numQuestions = questions.length;
  var numAnswers = 0;
  var numCorrectAnswers = 0;
  var answers = ["orange","23","99+9/9","1963-11-
    >>>  22","http://www.google.com"];
  //finish this function
 }
</script>
</head>
<body>
<h1>Quiz</h1>
<form method="post" action="process.xyz" id="quiz">
 <ol>
  <li>
   <label for="q1">What color do you get when you mix red and yellow?</la »»
    >>>  bel>
   <input type="text" name="q1" id="q1">
  </li>
  <li>
   <label for="q2">What is the first primary number greater than
    >>>  20?</label>
```



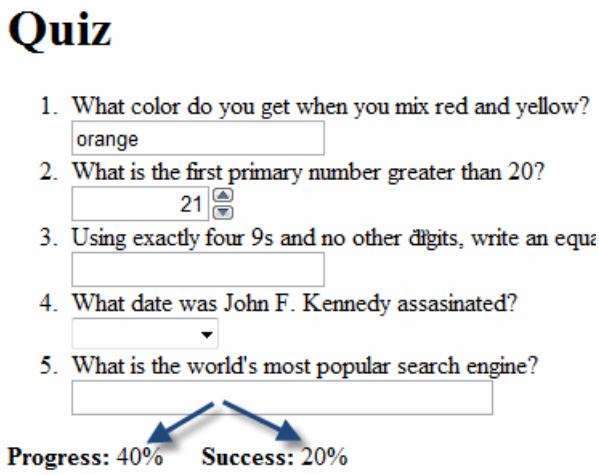
```
37.      <input type="text" name="q2" id="q2">
38.     </li>
39.     <li>
40.      <label for="q3">Using exactly four 9s and no other digits, write an
          >>>  equation which evaluates to 100. You may use addition (+),
          >>> subtraction (-), multiplication (*), and division (/). Do not
          >>>  include spaces.</label>
41.      <input type="text" name="q3" id="q3">
42.     </li>
43.     <li>
44.      <label for="q4">What date was John F. Kennedy assasinated?</label>
45.      <input type="text" name="q4" id="q4">
46.     </li>
47.     <li>
48.      <label for="q5">What is the world's most popular search engine?</la »»
          >>> bel>
49.      <input type="text" name="q5" id="q5">
50.     </li>
51.    </ol>
52.    <!--add bar showing percentage of valid (but not necessarily correct)
          >>>  answers-->
53.    <!--add bar showing percentage of correct answers-->
54.   </form>
55.   </body>
56.   </html>
```

***Challenge**

1. Add code so that the result of the formula the user enters in question 3 is displayed next to the input field like this:



2. Fix the two bars at the bottom of the form so that they present as follows in Opera and other browsers that do not support the `progress` and `meter` elements:



You'll need to change both your HTML and JavaScript to make this work.

Exercise Solution

html5-forms/Solutions/quiz.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 16 Omitted-------
17.    function updateMeasures() {
18.     var questions = document.getElementsByTagName("input");
19.     var numQuestions = questions.length;
20.     var numAnswers = 0;
21.     var numCorrectAnswers = 0;
22.     var answers = ["orange","23","99+9/9","1963-11-
         >>> 22","http://www.google.com"];
23.     for (var i=0; i<numQuestions; i++) {
24.      if (questions[i].validity.valid && questions[i].value.length>0) {
25.       numAnswers++;
26.      }
27.      if (questions[i].value==answers[i]) {
28.       numCorrectAnswers++;
29.      }
30.     }
31.     var progress=Math.round(numAnswers/numQuestions * 100);
32.     var score=Math.round(numCorrectAnswers/numQuestions * 100);
33.     document.getElementById("quiz-progress").value=progress;
34.     document.getElementById("quiz-success").value=score;
35.    }
36.   </script>
37.   </head>
38.   <body>
39.   <h1>Quiz</h1>
40.   <form method="post" action="process.xyz" id="quiz">
41.    <ol>
42.     <li>
43.      <label for="q1">What color do you get when you mix red and yellow?</la »»
         >>> bel>
44.      <input type="color" name="q1" id="q1" required placeholder="Enter
         >>> Color">
45.     </li>
46.     <li>
47.      <label for="q2">What is the first primary number greater than
         >>> 20?</label>
48.      <input type="number" name="q2" id="q2" min="20" step="1" required
         >>> placeholder="Enter Number">
49.     </li>
```



```
50.     <li>
51.      <label for="q3">Using exactly four 9s and no other digits, write an
          >>>  equation which evaluates to 100. You may use addition (+),
          >>> subtraction (-), multiplication (*), and division (/). Do not
          >>>  include spaces.</label>
52.      <input type="text" name="q3" id="q3" required placeholder="Enter
          >>> Equation" pattern="[9\+\-\*/]{4,7}">
53.     </li>
54.     <li>
55.      <label for="q4">What date was John F. Kennedy assasinated?</label>
56.      <input type="date" name="q4" id="q4" required placeholder="Enter
          >>> Date">
57.     </li>
58.     <li>
59.      <label for="q5">What is the world's most popular search engine?</la »»
          >>> bel>
60.      <input type="url" name="q5" id="q5" required placeholder="Enter URL"
          >>>  list="q5list">
61.      <datalist id="q5list">
62.       <option value="http://www.yahoo.com">
63.       <option value="http://www.google.com">
64.       <option value="http://www.excite.com">
65.       <option value="http://www.dogpile.com">
66.      </datalist>
67.     </li>
68.    </ol>
69.    <strong>Progress:</strong>
70.    <progress id="quiz-progress" min="0" max="100" title="Shows percentage
          >>>  of valid answers"></progress>
71.    <strong>Success:</strong>
72.    <meter id="quiz-success" min="0" max="100" title="Shows percentage of
          >>>  correct answers"></meter>
73.   </form>
74.   </body>
75.   </html>
```

Challenge Solution

html5-forms/Solutions/quiz-challenge.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 10 Omitted-------
11.    function addLoadEvents() {
12.     document.getElementById('q3').addEventListener("change",function() {
          >>>
13.      document.getElementById('q3output').innerHTML = eval(this.value);
14.     }, false);
15.     document.getElementById('quiz').addEventListener("change",function()
          >>>   {
16.      updateMeasures();
17.     }, false);
18.    }
19.
20.    function updateMeasures() {
      -------Lines 21 through 35 Omitted-------
36.     document.getElementById("quiz-progress").value=progress;
37.     document.getElementById("quiz-progress").getElementsByTag »»
          >>>  Name("span")[0].innerHTML=progress;
38.     document.getElementById("quiz-success").value=score;
39.     document.getElementById("quiz-success").getElementsByTag »»
          >>>  Name("span")[0].innerHTML=score;
40.    }
      -------Lines 41 through 56 Omitted-------
57.      <input type="text" name="q3" id="q3" required placeholder="Enter
          >>>  Equation" pattern="[9\+\-\*/]{4,7}">
58.      <output id="q3output"></output>
59.     </li>
60.     <li>
      -------Lines 61 through 74 Omitted-------
75.    <strong>Progress:</strong>
76.    <progress id="quiz-progress" min="0" max="100" title="Shows percentage
          >>>   of valid answers"><span>0</span>%</progress>
77.    <strong>Success:</strong>
78.    <meter id="quiz-success" min="0" max="100" title="Shows percentage of
          >>>   correct answers"><span>0</span>%</meter>
      -------Lines 79 through 81 Omitted-------
```

5.6 Conclusion

In this lesson, you have learned about the new HTML5 form elements and attributes and how they are supported by current browsers. You have also learned how to use Modernizr to detect for the support of these new features. Finally, you have learned how to use the new `output`, `progress`, and `meter` elements.

6. HTML5 Web Storage

In this lesson, you will learn...

1. How to use the two client-side storage methods in the W3C's web storage specification.
2. About the past and future of client-side storage.

In this unit you will learn about local storage and session storage and the use cases for each. You will also learn about some other client-side storage methods, one defunct and one up and coming.

6.1 Overview of HTML5 Web Storage

Before HTML5, the standard way of storing data on the client was to use cookies. But cookies weren't really meant to store a lot of data and the JavaScript methods for accessing, modifying and removing them are a bit clumsy.

HTML5 offers a very simple and straightforward way to create, read, update, and delete variables that are stored between user sessions (`localStorage`) or just for a single session (`sessionStorage`). We'll take a look at both of these.

6.2 Web Storage

HTML5 Web storage is used to store key-value pair data throughout a single session (`sessionStorage`) or between sessions (`localStorage`). Browsers typically limit the amount of client-side storage space allocated to a single domain to 5 megabytes and throw a `QUOTA_EXCEEDED_ERR` exception if you try to store more than that.

The methods and properties are the same for working with `sessionStorage` and `localStorage` and are shown below:

Web Storage Methods and Properties

Method/Property	Description
`length`	Holds the number of key/value pairs.
`setItem(key,value)`	Creates or updates a key/value pair.
`getItem(key)`	Gets the value of the specified key.
`key(n)`	Returns the nth key. Useful for iterating through key/value pairs.
`removeItem(key)`	Removes the key/value pair for the given key.
`clear()`	Removes all key/value pairs.

Browser Support

The following browsers support Web Storage:

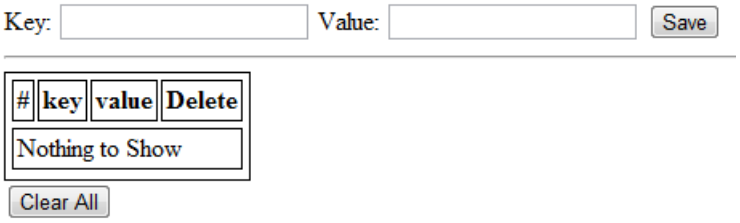
1. Internet Explorer 8+
2. Firefox 3.5+
3. Chrome 4.0+
4. Safari 4.0+
5. Opera 10.5+

Internet Explorer and Firefox do not support web storage unless the files are delivered by a web server. So unless you have a web server (e.g., IIS or Apache) set up locally, you should use Safari, Chrome or Opera for the demos and exercises in this lesson.

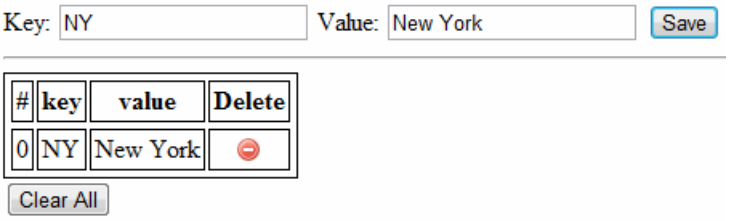
Local Storage

Local storage is used for maintaining state during and between sessions. It is easiest to understand through a simple example:

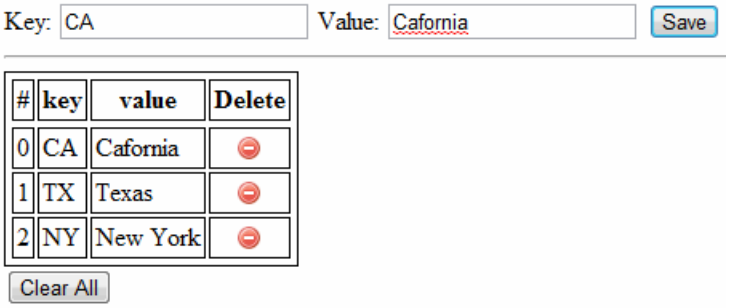
1. Open html5-storage/Demos/local-storage.html in your browser:



2. Enter a key/value pair and press the **Save** button. The table should update:

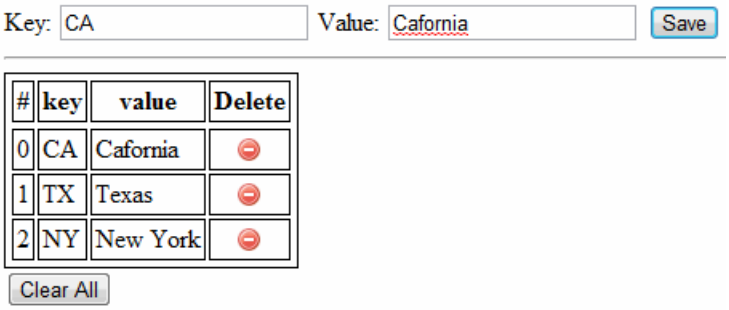


3. Add some more key/value pairs:



4. Refresh your browser. The values you entered should still be in the table.
5. Close and reopen your browser. The values you entered should **still** be in the table.

6. Enter a key that you have already entered, but change the value. Press **Save**. The table should update:



7. Press the **Delete** button next to one of your key/value pairs. It should (after you bypass the warning) be removed from the table and stay removed if you refresh or close and open your browser.
8. Press the **Clear All** button below the table. All key/value pairs should (after you bypass the warning) be removed from the table and stay removed if you refresh or close and open your browser.

Let's take a look at the code starting with the HTML in the `<body>`:

```
<label for="key">Key:</label>
<input type="text" id="key" autofocus autocomplete="off">
<label for="value">Value:</label>
<input type="text" id="value" autocomplete="off">
<button onclick="save()">Save</button>
<hr>
<table>
 <thead>
  <tr>
   <th>#</th>
   <th>key</th>
   <th>value</th>
   <th>Delete</th>
  </tr>
 </thead>
 <tbody id="output">
  <tr>
   <td colspan="4">Nothing to Show</td>
  </tr>
 </tbody>
</table>
<button onclick="clearStorage()">Clear All</button>
```

Things to notice:

1. We will use the `id`s of the `input` elements to get the user-entered values.

2. The **Save** button calls the `save()` function.
3. The **Clear All** button calls the `clearStorage()` function.
4. We will update the rows in the `tbody` element to show the key/value pairs in `localStorage`.

Now let's look at the JavaScript, starting with the `updateTable()` function, which is called when the page loads and each time we set or remove an item from `localStorage`:

```
window.addEventListener("load",updateTable,false);

function updateTable() {
 var tbody = document.getElementById("output");
 while (tbody.getElementsByTagName("tr").length > 0) {
  tbody.deleteRow(0);
 }
 var row;
 if (localStorage.length==0) {
  row = tbody.insertRow(i);
  cell = row.insertCell(0);
  cell.colSpan="4";
  cell.innerHTML = "Nothing to Show";
 }
 for (var i=0; i < localStorage.length; ++i) {
  row = tbody.insertRow(i);
  cell = row.insertCell(0);
  cell.innerHTML = i;
  cell = row.insertCell(1);
  cell.innerHTML = localStorage.key(i);
  cell = row.insertCell(2);
  cell.innerHTML = localStorage.getItem(localStorage.key(i));
  cell = row.insertCell(3);
  cell.innerHTML = "<img src='Images/delete.png'
onclick='deleteItem(\"" + localStorage.key(i) + "\")'>";
 }
}
```

Things to notice:

1. We use the standard table element methods[19] to remove all the table rows in the tbody.
2. If the `localStorage` is empty (`localStorage.length==0`) then we create a single row that reads "Nothing to Show".

19. See http://www.w3.org/TR/html5/tabular-data.html.

3. If the `localStorage` is not empty, we loop through it and create a row for each key/value pair.
4. Note that when the **delete** image is clicked, the `deleteItem()` function is called and the key for that key/value pair is passed in.

Now let's look at the remaining functions:

```
function deleteItem(key) {
 if (!confirm("Are you sure you want to delete this item?")) return;

 localStorage.removeItem(key);
 updateTable();
}

function clearStorage() {
 if (!confirm("Are you sure you want to delete all local storage
for this domain?")) return;
 localStorage.clear();
 updateTable();
}

function save() {
 var key = document.getElementById("key").value;
 var value = document.getElementById("value").value;
 localStorage.setItem(key,value);
 updateTable();
}
```

Things to notice:

1. The `deleteItem()` function uses `localStorage.removeItem(key)` to remove the passed-in key.
2. The `clearStorage()` function uses `localStorage.clear()` to empty `localStorage`.
3. The `save()` function uses `localStorage.setItem(key,value)` to add or update a key/value pair.
4. All three functions call `updateTable()` after they run so that the page gets updated.

And that's all there is to `localStorage`.

Use Case

A good use case for `localStorage` is saving data in a large form (for example - a job application) to finish and submit later. This way, you don't have to clog up your database server with half-completed applications.

Session Storage

And here's the beauty of it: `sessionStorage` is exactly the same. You can literally use find and replace to change "localStorage" to "sessionStorage" and your page will continue to work. The only difference is that your key/value pairs will only be accessible for the life of the session. So, if you close and reopen your browser, your key/value pairs will be gone.

To test this, go through the same process (see page 111) we went through to test `localStorage`, but use html5-storage/Demos/session-storage.html instead.

> Browsers are free to maintain a session longer than a single visit. For example, if the browser crashes, the browser may try to restore the session, in which case it may be able to keep the key/value pairs in `sessionStorage`.

Use Case

A good use case for `sessionStorage` is auto-saving forms so that if a user accidentally refreshes the browser, he/she doesn't lose any data. We'll tackle this in the upcoming exercise.

Prefixing your Keys

As it's possible to have multiple applications fed from the same origin (think URL), it's a good idea to use prefixes to scope your keys to a specific application. Take a look at the following demo:

Code Sample

html5-storage/Demos/questions.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 7 Omitted-------
8.    window.addEventListener("load",updateList,false);
9.
10.   var prefix="q-";
11.
12.   function updateList() {
13.    var questions="<ol>";
14.    for (var i=0; i < localStorage.length; ++i)  {
15.     if (localStorage.key(i).substring(0,prefix.length) == prefix) {
16.      questions+="<li>" + localStorage.key(i).substring(prefix.length) +
         >>>  " <img onclick=\"showAnswer('" + localStorage.key(i) + "')\"
         >>>  src='Images/find.png'> <img onclick=\"deleteQuestion('" + lo »»
         >>>  calStorage.key(i) + "')\" src='Images/delete.png'></li>";
17.     }
18.    }
19.    questions+="</ol>";
20.    document.getElementById("questions").innerHTML = questions;
21.   }
22.
23.   function save() {
24.    var q = prefix + document.getElementById("q").value;
25.    var a = document.getElementById("a").value;
26.    localStorage.setItem(q,a);
27.    document.getElementById("q").value="";
28.    document.getElementById("a").value="";
29.    document.getElementById("q").focus();
30.    updateList();
31.   }
32.
33.   function revealLocalStorage() {
34.    var questions="<ol>";
35.    for (var i=0; i < localStorage.length; ++i)  {
36.     questions+="<li>" + localStorage.key(i) + ": " + localStor »»
         >>>  age.getItem(localStorage.key(i)) + "</li>";
37.    }
38.    questions+="</ol>";
39.    document.getElementById("questions").innerHTML = questions;
40.   }
41.
```



```
42.  function showAnswer(key) {
43.   alert(localStorage.getItem(key));
44.  }
45.
46.  function deleteQuestion(key) {
47.   if (!confirm("Are you sure you want to delete this question?")) return;
        >>>
48.   localStorage.removeItem(key);
49.   updateList();
50.  }
51.
52.  function deleteQuestions() {
53.   if (!confirm("Are you sure you want to delete all questions?")) return;
        >>>
54.
55.   var numItems = localStorage.length;
56.   for (var i=numItems-1; i >=0; --i)  {
57.    if (localStorage.key(i).substring(0,prefix.length) == prefix) {
58.     localStorage.removeItem(localStorage.key(i));
59.    }
60.   }
61.   updateList();
62.  }
63.  </script>
64.  </head>
65.  <body>
66.  <label for="q">Question:</label>
67.  <input type="text" id="q" autofocus autocomplete="off">
68.  <label for="a">Answer:</label>
69.  <input type="text" id="a" autocomplete="off">
70.  <button onclick="save()">Save</button>
71.  <output id="questions"></output>
72.  <menu>
73.   <button onclick="revealLocalStorage()">Show All Local Storage</button>
        >>>
74.   <button onclick="updateList()">Show Questions</button>
75.   <button onclick="deleteQuestions()">Delete All Questions</button>
76.  </menu>
77.  </body>
78.  </html>
```

Code Explanation

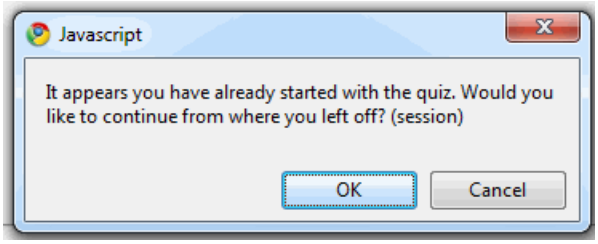
This is very similar to the page we saw earlier. It just prepends a prefix to the key, so that key used in other applications fed from the same domain don't get overwritten. It also adds a convenient feature of emptying the question and answer and replacing the cursor in the question.

Exercise 7 Creating a Quiz Application

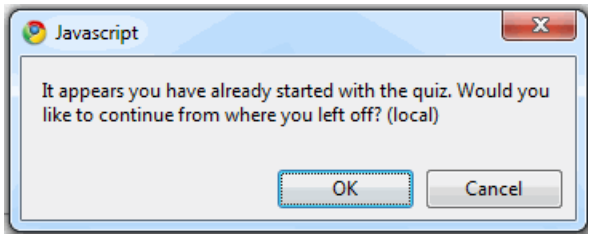
30 to 45 minutes

In this exercise, you will create a quiz application that allows the user to save and resume later. It also protects the user from losing data if he/she accidentally refreshes.

1. Open html5-storage/Solutions/saving-quiz-challenge.html in your browser and play with the application:
   A. Answer one or more questions and then refresh the browser. You will get a dialog giving you the chance to use refill the form with values stored in `sessionStorage`:



   B. Press **OK** and your values get added back.
   C. Refresh again and press **Cancel** on the dialog. Your values do not get added back.
   D. Refresh again. You don't get the JavaScript dialog because the `sessionStorage` key/value pairs were removed when you pressed **Cancel** in the previous step.
   E. Answer one or more questions again and click the **Save My Answers for Later** button.
   F. Close and reopen the browser. You will get a dialog giving you the chance to use refill the form with values stored in `localStorage`:



   G. If you click **Cancel**, the `localStorage` key/value pairs will be removed and you'll have to start the quiz over.
   H. If you click **OK**, the form will be refilled with your previous answers and they will be saved into `sessionStorage`.
   I. Also notice that the footer contains the time and date the quiz answers were last saved:

Answers last saved: 2010-12-15 17:36:23

   This is the challenge to the exercise.
2. Now open html5-storage/Exercises/saving-quiz.html in your editor.

3. In the the `addLoadEvents()` function:
   A. Loop through the `inputs` and add event listeners that capture change events to save the associated key/value pair in `sessionStorage`. Don't forget to use the prefix.
   B. Add an event listener to the **Save** button to capture a click event and call saveAnswers.
   C. Call the `refill()` function at the end.
4. Write the code in the `saveAnswers()` function to save all the answers in `localStorage`.
5. The `refill()` function:
   A. calls `hasAnswers()`, which returns "session", "local", or false, depending on if and where it finds saved answers. If there are no saved answers, it returns without doing anything.
   B. declares some variables:
      A. `confirmed` - we'll change it to true if the user wishes to refill the form.
      B. `msg` - the message to ask the user if he/she wants to refill the form.
      C. `questions` - the question inputs
   C. loops through the inputs. On the first iteration, it prompts the user with the message. If the user clicks **Cancel**, the key/value pairs are deleted (via the `deleteAnswers()` function) and the function returns/ends. Otherwise, we iterate through the questions. This is where you come in...
   D. Add code to populate the question inputs from the appropriate storage location (based on the value of `fillFrom`).

***Challenge**

1. Notice that an external script called dateFormat.js is included. That extends the Date object prototype with a `format()` method, which you use as follows:

```
var now = new Date();
var dateMask = "yyyy-mm-dd H:MM:ss";
var formattedNow = now.format(dateMask);
```

2. Use this to write out the date last saved to the `output` element below the form.
3. You will need to store the date in `sessionStorage` and `localStorage` as appropriate. Don't forget the prefix.
4. Note that the `dateMask` variable is already set in the code.

Exercise Solution

html5-storage/Solutions/saving-quiz.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 12 Omitted-------
13.    function addLoadEvents() {
14.     document.getElementById('quiz').addEventListener("change",function()
          >>>   {
15.      updateMeasures();
16.     }, false);
17.     var questions = document.getElementById("quiz").getElementsByTag »»
          >>>  Name("input");
18.     for (var i=0; i < questions.length; ++i)  {
19.      questions[i].addEventListener("change",function() {
20.       sessionStorage.setItem(prefix+this.id,this.value);
21.      },false);
22.     }
23.     document.getElementById("save").addEventListener("click",saveAn »»
          >>>  swers,false);
24.     refill();
25.    }
26.
27.    function saveAnswers() {
28.     var questions = document.getElementById("quiz").getElementsByTag »»
          >>>  Name("input");
29.     for (var i=0; i < questions.length; ++i)  {
30.      localStorage.setItem(prefix+questions[i].id,questions[i].value);
31.     }
32.    }
33.
34.    function refill() {
35.     var fillFrom = hasAnswers();
36.     if (!fillFrom) return;
37.     var confirmed=false;
38.     var msg="It appears you have already started with the quiz. Would you
          >>>   like to continue from where you left off? (" + fillFrom + ")";
          >>>
39.     var questions = document.getElementById("quiz").getElementsByTag »»
          >>>  Name("input");
40.     for (var i=0; i < questions.length; ++i)  {
41.      if (!confirmed && !confirm(msg)) {
42.       deleteAnswers();
43.       return;
44.      }
```

```
45.      confirmed=true;
46.      if (fillFrom == "session") {
47.       questions[i].value=sessionStorage.getItem(prefix+questions[i].id)
         >>>  || "";
48.       //We need the || "" for IE, which returns null if the key is not
         >>>  found.
49.      } else { //fillFrom == "local"
50.       questions[i].value=localStorage.getItem(prefix+questions[i].id) ||
         >>>   "";
51.       sessionStorage.setItem(prefix+questions[i].id,questions[i].value);
         >>>
52.      }
53.     }
54.     updateMeasures();
55.    }
      -------Lines 56 through 151 Omitted-------
```

Challenge Solution

html5-storage/Solutions/saving-quiz-challenge.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 11 Omitted-------
12.    var dateMask = "yyyy-mm-dd H:MM:ss";
13.    function addLoadEvents() {
14.     document.getElementById('quiz').addEventListener("change",function()
         >>>   {
15.      updateMeasures();
16.     }, false);
17.     var questions = document.getElementById("quiz").getElementsByTag »»
         >>>  Name("input");
18.     for (var i=0; i < questions.length; ++i)  {
19.      questions[i].addEventListener("change",function() {
20.       var now=new Date();
21.       sessionStorage.setItem(prefix+this.id,this.value);
22.       document.getElementById("dateLastSaved").innerHTML=now.for »»
         >>>  mat(dateMask);
23.       sessionStorage.setItem(prefix+"dateLastSaved",now.format(dateMask));
         >>>
24.      },false);
25.     }
26.     document.getElementById("save").addEventListener("click",saveAn »»
         >>>  swers,false);
27.     refill();
28.    }
29.
30.    function saveAnswers() {
31.     var questions = document.getElementById("quiz").getElementsByTag »»
         >>>  Name("input");
32.     var now=new Date();
33.     for (var i=0; i < questions.length; ++i)  {
34.      localStorage.setItem(prefix+questions[i].id,questions[i].value);
35.     }
36.     localStorage.setItem(prefix+"dateLastSaved",now.format(dateMask));
37.    }
38.
39.    function refill() {
40.     var fillFrom = hasAnswers();
41.     if (!fillFrom) return;
42.     var now;
      -------Lines 43 through 60 Omitted-------
61.     if (fillFrom == "session") {
```



```
62.      now = sessionStorage.getItem(prefix+"dateLastSaved");
63.     } else {
64.      now = localStorage.getItem(prefix+"dateLastSaved");
65.      sessionStorage.setItem(prefix+"dateLastSaved",now);
66.     }
67.
68.     document.getElementById("dateLastSaved").innerHTML=now;
69.     updateMeasures();
70.    }
      -------Lines 71 through 163 Omitted-------
164.  <small>Answers last saved: <output id="dateLastSaved">not saved</out »»
         >>>  put></small>
165.  </body>
166.  </html>
```

6.3 Other Storage Methods

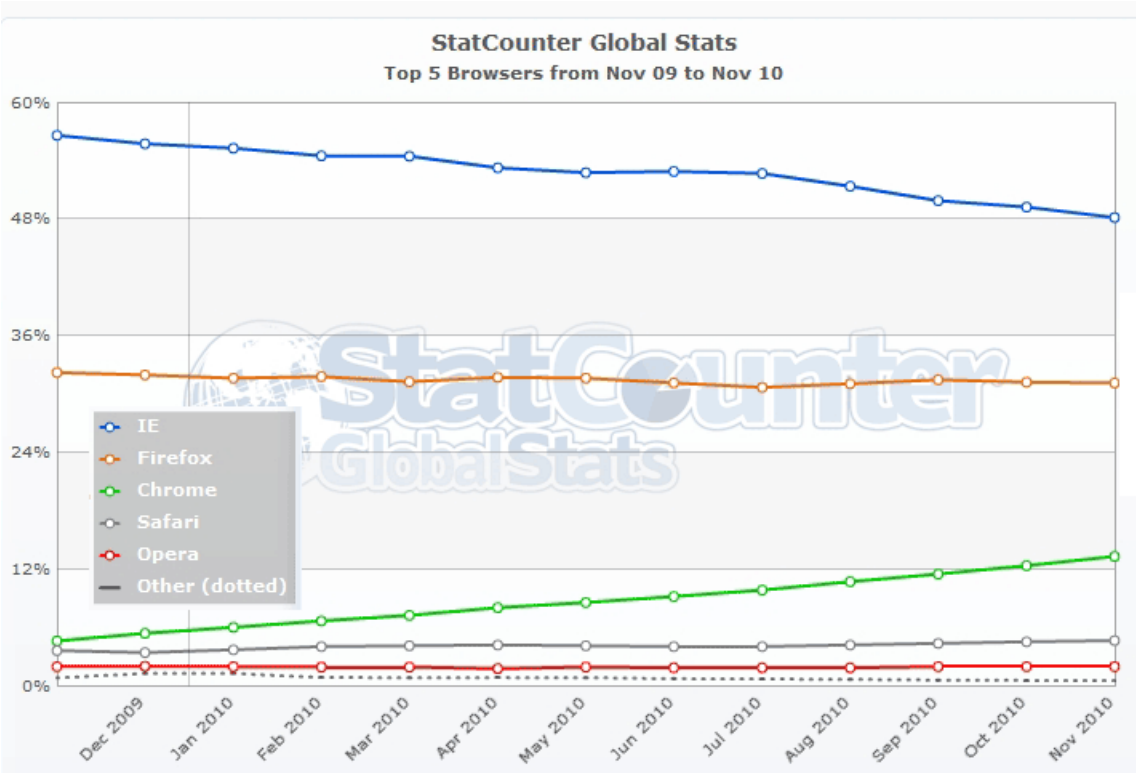
While HTML5 Web Storage makes it a lot simpler to store key/value pairs locally, it does not allow for the same kind of complex data storage that we get on the server via relational databases. One way to get a lot more out of the `sessionStorage` and `localStorage` mechanisms is to use object literals (JSON) to store complex data in a key/value pair. However, the brains behind HTML5 have been thinking bigger.

Web Database Storage

One promising idea, which has actually been implemented by several browsers, was to store data on the client in the same way we usually do it on the server: in a SQL database. The W3C speced it out under the name of Web Database Storage[20] and Chrome, Opera and Safari implemented this with SQLLite. But Mozilla and Microsoft are in favor of a different storage mechanism and, given that together they currently

20. See http://www.w3.org/TR/webdatabase.

have about 80% of the browser market (see chart below), that pretty much killed Web Database Storage.



In November, 2010, the W3C added this disclaimer to the Web SQL Database specification:

Indexed Database API

The storage mechanism preferred by Mozilla and Microsoft is called Indexed Database API and is also a W3C specification (http://www.w3.org/TR/IndexedDB/). It takes an object-approach rather than a relational database-approach to storage and querying. Google has also voiced support for it and has begun working on adding it to Chrome. Currently, only Firefox 4 beta has implemented Indexed Database API, so it will be some time before it's usable in applications.

6.4 Conclusion

In this lesson, you have learned about the two flavors of HTML5 Web Storage that are currently implemented by all the HTML5-compliant browsers.

7. HTML5 Canvas

In this lesson, you will learn...

1. How to get started with canvas.
2. How to draw lines.
3. How to draw rectangles and circles.
4. How to reposition and rotate the canvas.
5. How to create animations.

Canvas is one of the more talked-about new features of HTML5. It makes it possible to create drawings (e.g., for graphs or games) natively in the browser. If you know JavaScript, it is relatively easy to start using Canvas, which allows you to build intricate visual applications without the need of a plugin like Flash or Silverlight.

7.1 Getting Started with Canvas

The `canvas` element takes two attributes: `height` and `width`. Between the open and close tags, you can place fallback content for browsers that do not support canvas, like this:

```
<canvas id="my-canvas" width="500" height="500">Your browser doesn't
 support canvas.</canvas>
```

If you are using canvas to represent something that can be represented with an image as well, you could include an `img` element as your fallback content.

Context

The `<canvas>` tag only creates a drawing surface. To actually do any drawing, you need to use JavaScript. The first step is getting the `context`. Although eventually other contexts will be supported (including **3d**), the only currently supported context is **2d**.

```
var canvas=document.getElementById("my-canvas");
context = canvas.getContext("2d");
```

Since many browsers don't support canvas, they won't be able to get the context, so you should test for context support before beginning to draw:

```
var canvas=document.getElementById("my-canvas");
if (canvas.getContext) {
 context = canvas.getContext("2d");
 //draw here
}
```

7.2 Drawing Lines

Let's start by drawing a simple line. Take a look at the following sample:

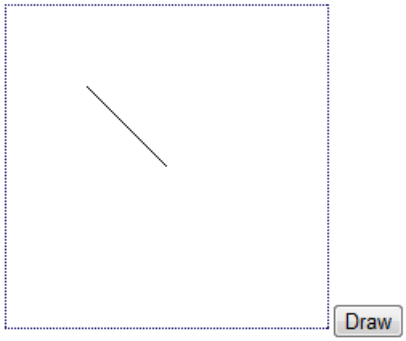
Code Sample

html5-canvas/Demos/path-simple.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>Path - Simple</title>
6.   <link href="style.css" rel="stylesheet" type="text/css">
7.   <script>
8.   function drawPath() {
9.    var canvas=document.getElementById("my-canvas");
10.   if (canvas.getContext) {
11.    context = canvas.getContext("2d");
12.    context.beginPath();
13.    context.moveTo(50,50);
14.    context.lineTo(100,100);
15.    context.stroke();
16.   }
17.  }
18.  </script>
19.  </head>
20.  <body>
21.  <canvas id="my-canvas" height="200" width="200">Your browser doesn't
        >>>  support canvas.</canvas>
22.  <button onclick="drawPath();">Draw</button>
23.  </body>
24.  </html>
```


Code Explanation

After clicking on the **Line** button, a line will get drawn as shown below:



Note that by default the `canvas` element has no border. We've used CSS to add a dotted border.

Let's look at the code:

1. `context.beginPath();` - tells canvas to forget any existing path being worked on and start a brand new path.
2. `context.moveTo(50,50);` - tells canvas to move to the x,y position of the moveTo(x,y) method. No sub-path is created.
3. `context.lineTo(100,100);` - tells canvas to create a sub-path from the current point to the x,y position of the lineTo(x,y) method. Note that this does not draw the sub-path. It just stores it in memory. Each call to `lineTo()` will create an additional sub-path.
4. `context.stroke();` - tells canvas to go ahead and draw the path, which is stored as a list of sub-paths making up one single path. The `stroke()` method is what gets your path on the canvas.

Multiple Sub-Paths

The demo below shows how to use multiple sub-paths to create a triangle:

Code Sample

html5-canvas/Demos/path-multiple-subpaths.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.beginPath();
13.     context.moveTo(50,50);
14.     context.lineTo(100,100);
15.     context.lineTo(100,50);
16.     context.lineTo(50,50);
17.     context.stroke();
18.    }
      -------Lines 19 through 26 Omitted-------
```

We can replace the final `lineTo()` call, which is used to close the triangle, with a call to `closePath()`, which attempts to create a sub-path from the current location to the starting point - the point at which the first sub-path started:

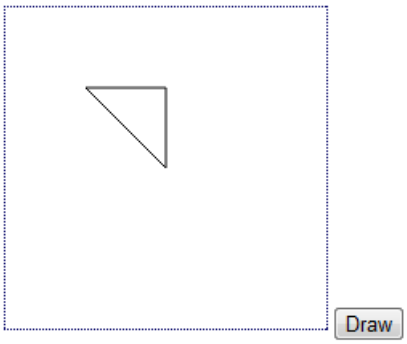
Code Sample

html5-canvas/Demos/path-closepath.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.beginPath();
13.     context.moveTo(50,50);
14.     context.lineTo(100,100);
15.     context.lineTo(100,50);
16.     context.closePath();
17.     context.stroke();
18.    }
      -------Lines 19 through 26 Omitted-------
```


Code Explanation

Both this and the previous example render as follows:



Note that you can also hop around using `context.moveTo(x,y)` to create disconnected sub-paths:

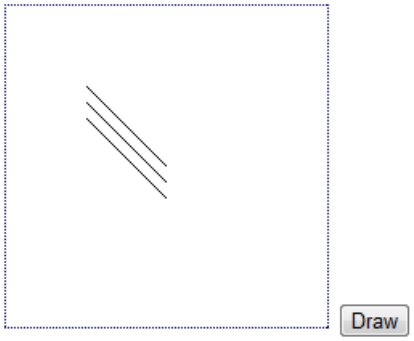
Code Sample

html5-canvas/Demos/path-disconnected.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.beginPath();
13.     context.moveTo(50,50);
14.     context.lineTo(100,100);
15.     context.moveTo(50,60);
16.     context.lineTo(100,110);
17.     context.moveTo(50,70);
18.     context.lineTo(100,120);
19.     context.stroke();
20.    }
      -------Lines 21 through 28 Omitted-------
```

Code Explanation

This will render as follows:



The Path Drawing Process

So, the process for drawing a basic path is as follows:

1. Begin the path: `context.beginPath()`
2. Create the sub-paths: one or more calls to `context.moveTo(x,y)` and `context.lineTo(x,y)`
3. Optionally close the path: `context.closePath()`
4. Draw the path: `context.stroke()`

The fill() Method

We saw the `stroke()` method for drawing the path. In cases where you have multiple connected sub-paths you can use the `fill()` method to fill the area with the current fill color (black by default):

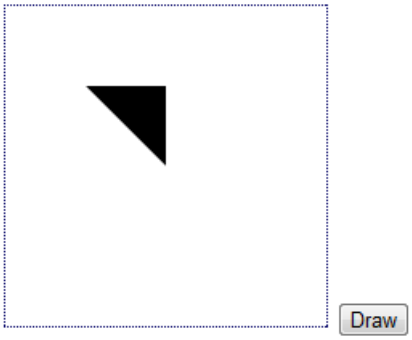
Code Sample

html5-canvas/Demos/path-fill.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.beginPath();
13.     context.moveTo(50,50);
14.     context.lineTo(100,100);
15.     context.lineTo(100,50);
16.     context.fill();
17.    }
      -------Lines 18 through 25 Omitted-------
```


Code Explanation

This will render as follows:



Note that in this example, the path was not closed (with `closePath()`) or drawn (with `stroke()`). The example below uses both `stroke()` and `fill()`. To see where the stroke starts and the fill ends, we have introduced the `fillStyle` and `strokeStyle` properties, which are used to set colors, and the `lineWidth` property, which sets the weight of the stroke.

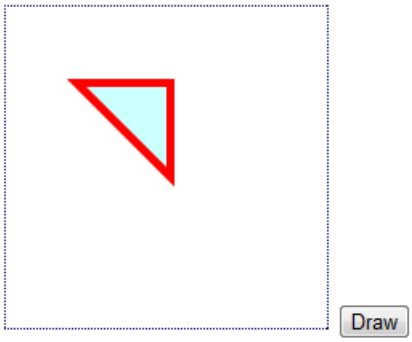
Code Sample

html5-canvas/Demos/path-fill-stroke.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.lineWidth = 10;
13.     context.strokeStyle = "rgb(255,0,0)";
14.     context.fillStyle = "rgb(204,255,255)";
15.     context.beginPath();
16.     context.moveTo(50,50);
17.     context.lineTo(100,100);
18.     context.lineTo(100,50);
19.     context.closePath();
20.     context.stroke();
21.     context.fill();
22.    }
      -------Lines 23 through 30 Omitted-------
```

Code Explanation

This will render as follows:



7.3 Color and Transparency

As we saw above, we can set colors using the `fillStyle` and `strokeStyle` properties. For best support, when naming colors, choose from the following:

1. Color names (e.g., red, green, deeppink)
2. Hex colors (e.g., #ff0000, #008000, #ff1493)
3. Shortened hex colors (e.g., #f00, #080, #f19)
4. RGB functional notation (e.g., rgb(255,0,0), rgb(0,128,0), rgb(255,20,147))

Transparency

It is possible to make our strokes and fills semi-transparent using the `rgba(r,g,b,a)` (a for alpha) functional syntax:

- A value of 1 for `a` means fully opaque.
- A value of 0 for `a` means fully transparent.

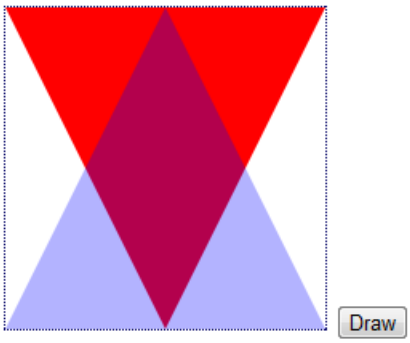
Code Sample

html5-canvas/Demos/transparency.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.fillStyle = "rgba(255,0,0,1)";
13.     context.beginPath();
14.     context.moveTo(0,0);
15.     context.lineTo(200,0);
16.     context.lineTo(100,200);
17.     context.fill();
18.     context.fillStyle = "rgba(0,0,255,.3)";
19.     context.beginPath();
20.     context.moveTo(0,200);
21.     context.lineTo(200,200);
22.     context.lineTo(100,0);
23.     context.fill();
24.    }
      -------Lines 25 through 32 Omitted-------
```

Code Explanation

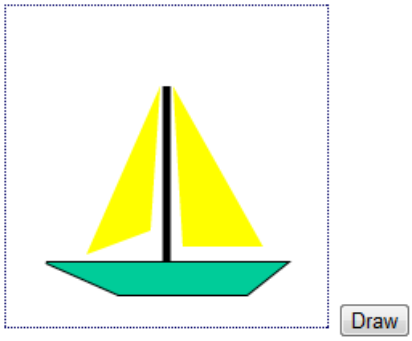
This will render as follows:



Exercise 8 Drawing a Sailboat

20 to 30 minutes

In this exercise, you will use HTML5 canvas to draw a simple sailboat like the one shown below:



1. Open html5-canvas/Exercises/sailboat.html in your editor.
2. Add the JavaScript code necessary to draw the sailboat pictured above.

***Challenge**

Have the left sale blink between different colors. You'll need to use some JavaScript skills to make this happen.

If JavaScript isn't your thing, try adding a little person on your sailboat.

Exercise Solution

html5-canvas/Solutions/sailboat.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 10 Omitted-------
11.     context = canvas.getContext("2d");
12.
13.     //boat
14.     context.fillStyle = "rgb(0,204,153)";
15.     context.lineWidth = 2;
16.     context.beginPath();
17.     context.moveTo(25,160);
18.     context.lineTo(70,180);
19.     context.lineTo(150,180);
20.     context.lineTo(175,160);
21.     context.closePath();
22.     context.stroke();
23.     context.fill();
24.
25.     //pole
26.     context.beginPath();
27.     context.lineWidth = 5;
28.     context.moveTo(100,160);
29.     context.lineTo(100,50);
30.     context.stroke();
31.
32.     //left sail
33.     context.beginPath();
34.     context.fillStyle = "rgb(255,255,0)";
35.     context.moveTo(96,50);
36.     context.lineTo(50,155);
37.     context.lineTo(90,140);
38.     context.fill();
39.
40.     //right sail
41.     context.beginPath();
42.     context.moveTo(104,50);
43.     context.lineTo(160,150);
44.     context.lineTo(110,150);
45.     context.fill();
46.    }
      -------Lines 47 through 54 Omitted-------
```


Challenge Solution

html5-canvas/Solutions/sailboat-challenge.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
      -------Lines 12 through 47 Omitted-------
48.     blink(context,"rgb(255,255,0)");
49.     addSailor(context);
50.    }
51.   }
52.
53.   function blink(context,color) {
54.    context.fillStyle=color;
55.    context.beginPath();
56.    context.moveTo(96,50);
57.    context.lineTo(50,155);
58.    context.lineTo(90,140);
59.    context.fill();
60.    if (color == "rgb(255,255,0)") {
61.     color = "rgb(255,204,0)";
62.    } else {
63.     color = "rgb(255,255,0)";
64.    }
65.    setTimeout(function() {blink(context,color);},250);
66.   }
67.
68.   function addSailor(context) {
69.    context.strokeStyle = "rgb(0,0,0)";
70.    context.lineWidth = 1;
71.    context.beginPath();
72.    //left leg
73.    context.moveTo(130,160);
74.    context.lineTo(135,152);
75.    //right leg
76.    context.lineTo(140,160);
77.    //body
78.    context.moveTo(135,152);
79.    context.lineTo(135,140);
80.    //arms
81.    context.moveTo(130,145);
```

```
82.   context.lineTo(140,145);
83.   context.stroke();
84.   //square head
85.   context.beginPath();
86.   context.lineWidth = 5;
87.   context.moveTo(135,140);
88.   context.lineTo(135,135);
89.   context.stroke();
90.  }
     -------Lines 91 through 97 Omitted-------
```

7.4 Rectangles

To create a rectangle, you specify the top-left starting position (x,y) and the width and height of the rectangle. There are three methods for creating rectangular shapes:

1. `fillRect(x,y,width,height)` - draws a filled rectangle.
2. `strokeRect(x,y,width,height)` - outlines a rectangle.
3. `clearRect(x,y,width,height)` - clears a rectangular area.

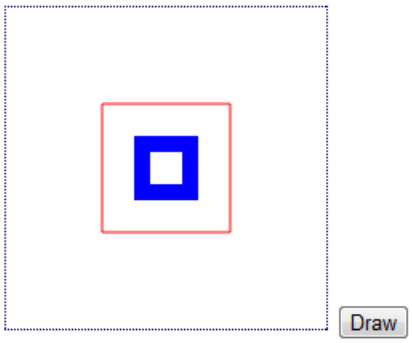
Code Sample

html5-canvas/Demos/rect.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.strokeStyle="red";
13.     context.fillStyle="blue";
14.     context.fillRect(80,80,40,40);
15.     context.strokeRect(60,60,80,80);
16.     context.clearRect(90,90,20,20);
17.    }
      -------Lines 18 through 25 Omitted-------
```

Code Explanation

The above code will render the following:



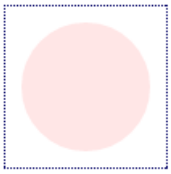
7.5 Circles and Arcs

Note that a circle is just an arc that keeps going around until it gets to its starting point.

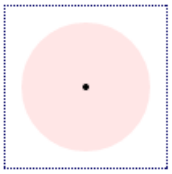
Circles and arcs are created by identifying the center of the circle (x,y) and then choosing a radius. You then set the starting and ending points on the circle and indicate whether to connect them going clockwise or counter-clockwise. The method looks like this:


```
Syntax
arc(x,y,radius,startDegree,endDegree,counterclockwise);
```

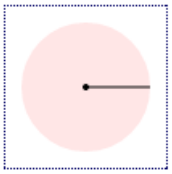
Let's break this apart. Imagine the following circle:



1. Identify the center point:

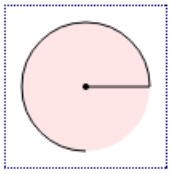


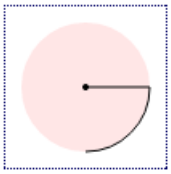
2. Choose a radius (e.g., 40). We draw the radius directly out to the right. Its end marks the 0 degree/radians point of the circle/arc:



3. Choose a starting point along the (usually 0) and an ending point (e.g., 90 degrees). The sixth and final argument indicates whether you want to connect these two points by moving along the circle counter-clockwise (true) or

clockwise (false). The first image below shows the counter-clockwise result and the second shows the clockwise result:





The code for creating the above graphics is shown below:

Code Sample

html5-canvas/Demos/arc-explained.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 7 Omitted-------
8.    window.addEventListener("load",function() {
9.     var canvas=document.getElementById("my-canvas");
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     context.beginPath();
13.     context.moveTo(50,50);
14.     context.arc(50,50,40,0,degreesToRadians(360),true);
15.     context.fillStyle="rgba(255,0,0,.1)";
16.     context.fill();
17.    }
18.   }, false);
19.
20.   function centerPoint() {
21.    var canvas=document.getElementById("my-canvas");
22.    if (canvas.getContext) {
23.     context.fillStyle="rgba(0,0,0,1)";
24.     context = canvas.getContext("2d");
25.     context.beginPath();
26.     context.arc(50,50,2,0,degreesToRadians(360),true);
27.     context.fill();
28.    }
29.   }
30.
31.   function radius() {
32.    var canvas=document.getElementById("my-canvas");
33.    if (canvas.getContext) {
34.     context = canvas.getContext("2d");
35.     context.beginPath();
36.     context.moveTo(50,50);
37.     context.lineTo(50+40,50);
38.     context.stroke();
39.    }
40.   }
41.
42.   function showArc(ccw) {
43.    var canvas=document.getElementById("my-canvas");
44.    if (canvas.getContext) {
```

```
45.     context = canvas.getContext("2d");
46.     context.beginPath();
47.     context.arc(50,50,40,0,degreesToRadians(90),ccw);
48.     context.stroke();
49.    }
50.   }
51.
52.   function degreesToRadians(degrees) {
53.    return (Math.PI/180)*degrees;
54.   }
      -------Lines 55 through 66 Omitted-------
```

Radians

Usually when we (meaning most humans) think of angles, we think in terms of degrees (e.g., a 90-degree angle). Others (meaning some math folks, including the people behind canvas) think in terms of radians. So, the fourth and fifth arguments of the `arc()` method are not in degrees, but in radians.

- A full circle is 2π radians, which is the same as 360 degrees.
- A semi-circle is π radians, or 180 degrees.
- A quarter-circle is π/2 radians, or 90 degrees.

For those of you who would prefer to continue to think in degrees, you can use this simple conversion function:

```
function degreesToRadians(degrees) {
 return (Math.PI/180)*degrees;
}
```

Without the `degreesToRadians()` function, a circle could be created like this:

```
context.arc(100,100,50,0,2*Math.PI,true);
```

Using the function, you would create the same circle like this:

```
context.arc(100,100,50,0,degreesToRadians(360),true);
```

To get a better understanding of radians and degrees and how the `counterclockwise` parameter works, open html5-canvas/Demos/arc-radians.html and html5-canvas/Demos/arc-degrees.html in your browser.

Exercise 9 Drawing a Snowman

20 to 30 minutes

In this exercise, you will use circles and squares to create a snowman like the one pictured below:

1. Open html5-canvas/Exercises/snowman.html in your editor.
2. Add the JavaScript code necessary to draw the snowman pictured above. You will need to add:
   A. A layer of snow on the ground.
   B. Three balls for the body and head.
   C. Eyes, mouth and nose.
   D. A hat.
   E. Arms.
   F. Buttons.
   G. A sun.

***Challenge**

Make the sun gradually disappear and the night grow darker.

Exercise Solution

html5-canvas/Solutions/snowman.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 9 Omitted-------
10.    if (canvas.getContext) {
11.     context = canvas.getContext("2d");
12.     //day
13.     context.canvas.style.backgroundColor = "rgb(153,153,255)";
14.
15.     //ground snow
16.     context.beginPath();
17.     context.fillStyle = "rgb(255,255,255)";
18.     context.fillRect(0,360,400,40);
19.
20.     //bottom ball
21.     context.beginPath();
22.     context.fillStyle = "rgb(255,255,255)";
23.     context.arc(200,300,70,0,degreesToRadians(360),true);
24.     context.fill();
25.
26.     //middle ball
27.     context.beginPath();
28.     context.fillStyle = "rgb(255,255,255)";
29.     context.arc(200,200,50,0,degreesToRadians(360),true);
30.     context.fill();
31.
32.     //head
33.     context.beginPath();
34.     context.fillStyle = "rgb(255,255,255)";
35.     context.arc(200,125,35,0,degreesToRadians(360),true);
36.     context.fill();
37.
38.     //right eye
39.     context.beginPath();
40.     context.fillStyle = "rgb(0,0,0)";
41.     context.arc(190,115,4,0,degreesToRadians(360),true);
42.     context.fill();
43.
44.     //left eye
45.     context.beginPath();
46.     context.fillStyle = "rgb(0,0,0)";
```

```
47.     context.arc(210,115,4,0,degreesToRadians(360),true);
48.     context.fill();
49.
50.     //mouth
51.     context.beginPath();
52.     context.fillStyle = "rgb(255,0,0)";
53.     context.arc(200,125,15,degreesToRadians(45),degreesToRadi »»
         >>>  ans(135),false);
54.     context.fill();
55.
56.     //nose
57.     context.beginPath();
58.     context.fillStyle = "rgb(255,102,0)";
59.     context.moveTo(200,122);
60.     context.lineTo(220,126);
61.     context.lineTo(200,130);
62.     context.fill();
63.
64.     //hat
65.     context.beginPath();
66.     context.fillStyle = "rgb(0,0,0)";
67.     context.arc(200,100,25,0,degreesToRadians(180),true);
68.     context.fill();
69.     context.fillRect(163,94,75,8);
70.
71.     //left arm
72.     context.beginPath();
73.     context.lineWidth=2;
74.     context.strokeStyle = "rgb(155,85,0)";
75.     context.moveTo(220,180);
76.     context.lineTo(300,160);
77.     context.lineTo(315,165);
78.     context.moveTo(300,160);
79.     context.lineTo(315,158);
80.     context.moveTo(300,160);
81.     context.lineTo(315,150);
82.     context.stroke();
83.
84.     //right arm
85.     context.beginPath();
86.     context.moveTo(170,180);
87.     context.lineTo(110,240);
```

```
88.     context.lineTo(115,255);
89.     context.moveTo(110,240);
90.     context.lineTo(95,255);
91.     context.moveTo(110,240);
92.     context.lineTo(85,255);
93.     context.stroke();
94.
95.     //buttons
96.     context.beginPath();
97.     context.fillStyle = "rgb(0,0,0)";
98.     context.arc(200,180,5,0,degreesToRadians(360),true);
99.     context.arc(200,220,5,0,degreesToRadians(360),true);
100.    context.arc(200,260,5,0,degreesToRadians(360),true);
101.    context.fill();
102.
103.    //sun
104.    context.beginPath();
105.    context.fillStyle = "rgb(255,255,0)";
106.    context.arc(400,0,70,0,degreesToRadians(360),true);
107.    context.fill();
108.   }
      -------Lines 109 through 120 Omitted-------
```

Challenge Solution

html5-canvas/Solutions/snowman-challenge.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 7 Omitted-------
8.    var timer = null;
9.    function drawPath() {
10.    var canvas=document.getElementById("my-canvas");
11.    if (canvas.getContext) {
      -------Lines 12 through 110 Omitted-------
111.
112.    darken(context,153,153,255,1);
113.   }
      -------Lines 114 through 115 Omitted-------
116.  function darken(context,r,g,b,sunchop) {
117.   context.canvas.style.backgroundColor = "rgb("+r+","+g+","+b+")";
118.   context.beginPath();
119.   context.strokeStyle = "rgb("+r+","+g+","+b+")";
120.   context.lineWidth = sunchop;
121.   context.arc(400,0,70,0,degreesToRadians(360),true);
122.   context.stroke();
123.   if (r>0) r-=3;
124.   if (g>0) g-=3;
125.   if (b>0) b-=3;
126.   sunchop+=2;
127.   if (r>0 || g>0 || b>0) {
128.    timer = setTimeout(function() {darken(context,r,g,b,sunchop)},100);
         >>>
129.   }
130.  }
      -------Lines 131 through 141 Omitted-------
```

7.6 Quadratic and Bézier Curves

Canvas also includes functions for creating Quadratic and Bézier curves. These methods are shown below:

Syntax

```
quadraticCurveTo(cp1x, cp1y, x, y)
bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
```

If you are new to Quadratic and Bézier curves, they can be difficult to get used to. For both:

1. Start with a line with a stated start point and end point.
   - The start point is the current position, which you can set using the `moveTo(x,y)` method.
   - The end point is set with the last two arguments of the method: x and y.
2. Add control points.
   - Just one for a Quadratic curve.
   - Two for a Bézier curve.
3. Imagine that these control points are tugging at the straight line created by the start and end points. By doing so, they create a curve.

The best way to see how they work is to practice with them a little. We've built small HTML5 applications for doing so.

Practice

We'll start with Quadratic curves:

1. Open html5-canvas/Demos/quadratic.html in your browser. You will see:
   A. A 500x500 Canvas.
   B. A menu in the upper right with as **Start Over** button for clearing the canvas and controls for setting the color.[21]
   C. A `textarea` immediately below the Canvas, which holds the generated code, in case you create a drawing you want to use somewhere else.
   D. A list of the current curve point values to the right of the `textarea`.

To use this application:

1. Click anywhere on the Canvas to set a **start point**.
2. Click a second time to set an **end point**.

21. We would have used a `color` input type for this, but no browser currently supports it.

3. Click a third time to set the **control point**.
4. Click a fourth time to start over; that is, to set a **start point** for a new curve.

You may change the color any time before the fourth click to set a new color for the curve.

Play around with this for a little and then open html5-canvas/Demos/bezier.html in your browser to play with creating Bézier curves. This application is the same, except that the fourth click sets a second control point.

If you generate a drawing you like:

1. Open html5-canvas/Demos/test-bed.html in your editor.
2. Copy the code from the textarea in the curve application.
3. Paste it between the start paste and end paste comments in test-bed.html and save.
4. Open test-bed.html in your browser. You should see your drawing there.

7.7 Images

You can add existing images to your drawing with the `drawImage()` method, which is overloaded and has two signatures:

```
drawImage(image, x, y, width, height) //basic
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)
 //sprites
```

In both cases, you need an image object as the first parameter. You can grab an image from the page (e.g., using `document.getElementById('my-image')`) or you can create an image object using JavaScript:

```
var img = new Image();
img.src = 'images/my-image.gif';
```

It is generally a good idea to hold off trying to display the image in your drawing until you are sure that it has loaded. So, you should wrap your drawing code as shown below:

```
img.onload = function() {
 context.drawImage(/*signature*/);
}
```

Let's look at the different signatures now.

drawImage() - Basic

The basic signature simply places the image at a given x,y position on the canvas. The `width` and `height` parameters are optional and are used for scaling or distorting the original image (generally a bad idea).

drawImage() Parameters (basic)

Parameter	Description
`image`	Image object
`x`	x position on canvas
`y`	y position on canvas
`width`	width of canvas
`height`	height of canvas

Here is a simple example:

Code Sample

html5-canvas/Demos/image-basic.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 11 Omitted-------
12.    if (canvas.getContext) {
13.     var context = canvas.getContext("2d");
14.     var logo = new Image();
15.     logo.src = 'Images/logo.png';
16.     logo.onload = function() {
17.      context.drawImage(logo,50,50);
18.     }
19.    }
      -------Lines 20 through 26 Omitted-------
```

Code Explanation

This will render as follows:

drawImage() - Sprites

A sprite is an image file that contains several graphics used on a web page. By showing different parts of the sprite in different locations, it appears that there are several different images, but they are all contained in a single file, which translates to a single (faster) download.

To get this to work, we have to specify which part of the image (the source) we want to show and where (and how large) we want it to appear on the canvas. The table below shows the parameters the `drawImage()` method takes to create sprites.

drawImage() Parameters (sprites)

Parameter	Description
`image`	Image object
`sx`	X position of image (source)
`sy`	Y position of image (source)
`sWidth`	width of source (from X pos)
`sHeight`	height of source (from Y pos)
`dx`	X position of canvas (destination)
`dy`	Y position of canvas (destination)
`dWidth`	width of destination (for scaling)
`dHeight`	height of destination (for scaling)

Take a look at the following example:

Code Sample

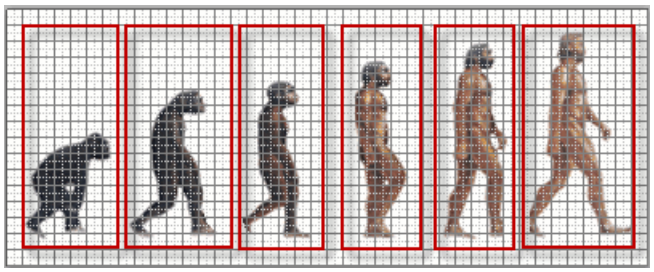
html5-canvas/Demos/image-sprite.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 7 Omitted-------
8.    var img;
9.    window.addEventListener("load",function() {
10.    img = new Image();
11.    img.src = 'Images/evolution.gif';
12.    draw();
13.   },false);
14.
15.   function draw() {
16.    var canvas=document.getElementById("my-canvas");
17.    if (canvas.getContext) {
18.     var context = canvas.getContext("2d");
19.     img.onload = function() {
20.      context.drawImage(img, 10, 10, 60, 140, 60, 20, 60, 140);
21.      document.getElementById("btnEvolve").disabled=false;
22.     }
23.    }
24.   }
25.
26.   function evolve(pic) {
27.    var pics = [
28.     {
29.      "sx" : 10, "w" : 60,
30.     },
31.     {
32.      "sx" : 74, "w" : 66
33.     },
34.     {
35.      "sx" : 143, "w" : 60
36.     },
37.     {
38.      "sx" : 217, "w" : 55
39.     },
40.     {
41.      "sx" : 270, "w" : 55
42.     },
43.     {
44.      "sx" : 324, "w" : 70
```

```
45.     }
46.    ];
47.    var canvas=document.getElementById("my-canvas");
48.    if (canvas.getContext) {
49.     var context = canvas.getContext("2d");
50.     context.clearRect(0,0,200,200);
51.     context.drawImage(img, pics[pic].sx, 10, pics[pic].w, 140, 60, 20,
          >>>  pics[pic].w, 140);
52.    }
53.    if (pic < 5) {
54.     setTimeout(function() {pic++; evolve(pic); }, 250);
55.    }
56.   }
      -------Lines 57 through 64 Omitted-------
```

Code Explanation

This page loads a single image (shown below with a grid overlay):



When the image loads, we show the first character in the image and enable the evolve button, so we can call the `evolve()` function, which recurses (calls itself) every 250 milliseconds passing on the next index of the `pics` array, which stores the X position of image (`sx`) and the width (`w`) for both the source and destination.

7.8 Text

There are two methods for adding text to the canvas:

1. `fillText(text,x,y)` - adds "solid" text at the x,y position.
2. `strokeText(text,x,y)` - adds "hollow" text at the x,y position.

Text Properties

You can set the following text properties:

1. `font` - uses the same syntax as the CSS font property: `font-style font-variant font-weight font-size/line-height font-family`
2. `textAlign` - possible values are "start," "end," "left," "right," and "center."
3. `textBaseline` - possible values are "top," "hanging," "middle," "alphabetic," "ideographic," and "bottom."

The `measureText(text)` method returns an object containing text metrics. Presumably, metrics will be added, but currently it only has one property: `width`. So, to see how wide some text would be in the current font, you would do this:

```
var width = context.measureText(text).width
```

Take a look at the following example:

Code Sample

html5-canvas/Demos/text.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 11 Omitted-------
12.    if (canvas.getContext) {
13.     var context = canvas.getContext("2d");
14.     context.fillStyle="red";
15.     context.strokeStyle="blue";
16.
17.     context.font = "italic small-caps bold 44pt 'Comic Sans MS'";
18.     context.textAlign = "left";
19.     context.strokeText("Hello",10,100);
20.     context.fillText("World!",10+context.measureText("Hello ").width,100);
          >>>
21.    }
      -------Lines 22 through 28 Omitted-------
```


Exercise 10 Images and Text

30 to 40 minutes

In this exercise, you will start with two images found in the html5-canvas/Exercises/Images folder:

1. south-america.gif - a map of South America.
2. flags.png - a picture containing small graphics of country flags.

You will create the following drawing:

Notice the text under the graphics.

1. Open html5-canvas/Exercises/south-america in your editor.
2. Add the JavaScript code necessary to:
   A. Create the image objects and set their source values.
   B. Draw the backdrop (the map).
   C. Place the flags using the sprite method shown earlier. Each flag is 18 pixels wide and 13 pixels high. The source and destination positions are shown in the table below.
   D. Add the country names.

Country	Source X	Source Y	Destination X	Destination Y
Chile	283	88	100	250
Argentina	255	4	130	300
Brazil	171	60	200	170
Paraguay	59	452	170	250
Uruguay	59	564	185	310
Bolivia	59	200	135	210
Peru	31	424	75	170

Exercise Solution

html5-canvas/Solutions/south-america.html

```
1.    <!DOCTYPE HTML>
      -------Lines 2 through 11 Omitted-------
12.    if (canvas.getContext) {
13.     var context = canvas.getContext("2d");
14.     var backdrop = new Image();
15.     var flags = new Image();
16.     backdrop.src = 'Images/south-america.gif';
17.     flags.src = 'Images/flags.png';
18.     backdrop.onload = function() {
19.      context.drawImage(backdrop,0,0);
20.     }
21.     flags.onload = function() {
22.      context.drawImage(flags, 283, 88, 18, 13, 100, 250, 18, 13);
23.      context.fillText("Chile",100, 250);
24.      context.drawImage(flags, 255, 4, 18, 13, 130, 300, 18, 13);
25.      context.fillText("Argentina",130, 300);
26.      context.drawImage(flags, 171, 60, 18, 13, 200, 170, 18, 13);
27.      context.fillText("Brazil",200, 170);
28.      context.drawImage(flags, 59, 452, 18, 13, 170, 250, 18, 13);
29.      context.fillText("Paraguay",170, 250);
30.      context.drawImage(flags, 59, 564, 18, 13, 185, 310, 18, 13);
31.      context.fillText("Uruguay",185, 310);
32.      context.drawImage(flags, 59, 200, 18, 13, 135, 210, 18, 13);
33.      context.fillText("Bolivia",135, 210);
34.      context.drawImage(flags, 31, 424, 18, 13, 75, 170, 18, 13);
35.      context.fillText("Peru",75, 170);
36.     }
37.    }
      -------Lines 38 through 44 Omitted-------
```


7.9 Conclusion

In this lesson, you have learned to use HTML5 canvas to create drawings.

8. Integrated APIs

In this lesson, you will learn...

1. Look at a couple of the new HTML5 APIs.

HTML5 includes a bunch of integrated and associated (e.g., not specifically part of HTML5) APIs. In this lesson, we'll look at the Offline Application API and the Drag and Drop API.

8.1 Offline Application API

The HTML5 Specification includes an API for creating offline applications. The purpose is two-fold:

1. Allow users to access your web application when they are offline.
2. Make your web applications faster by taking advantage of local caching.

There are four steps involved in turning a regular HTML page into an offline application:

1. Create a cache manifest file.
2. Reference the cache manifest file in your HTML5 document.
3. Write JavaScript to manage caching.

Cache Manifest File

The cache manifest file is a plain text file with a .manifest extension. It is structured as follows:

```
CACHE MANIFEST
#Use pound signs for comments

#Explicitly cached entries.
CACHE:
index.html
style.css
script.js
image.png

#Online-only Resources.
NETWORK:
login.php
/online-only/

#Fallback files (use only when can't access online files)
FALLBACK:
online.js offline.js
```

Note that you must configure your web server to deliver .manifest files using the "text/cache-manifest" mime type.

In Apache, you can use `AddType text/cache-manifest .manifest`

In IIS, you can add Mime types through Computer Management.

The HTML File

Referencing the manifest file in your HTML document is easy:

```
<html manifest="example.manifest">
```

Managing Application Cache with JavaScript

You access the application cache through the `window.applicationCache` object. It includes a `status` property indicating the current state of the cache. As the `status` changes, the following events are fired:

1. cached
2. checking
3. downloading
4. error
5. noupdate
6. obsolete

7. progress
8. updateready

You can catch these events using event listeners: `window.applicationCache.addEventListener("error", fnCall, false);`.

A Sample Application

Take a look at our sample application files below:

The HTML Page

Code Sample

html5-apis/Demos/offline.html

```
<!DOCTYPE HTML>
<html manifest="example.manifest">
<head>
<meta charset="UTF-8">
<title>Offline Application API</title>
<script src="offline/script.js" type="text/javascript"></script>
<link href="offline/style.css" rel="stylesheet">
</head>
<body>
<h1>No Heading</h1>
<ol id="output"></ol>
<img src="Images/online.gif" alt="online/offline image">
</body>
</html>
```

The Cache Manifest

Code Sample

html5-apis/Demos/example.manifest

```
CACHE MANIFEST
#Version 1.2
offline.html
offline/style.css
offline/script.js

NETWORK:
*

FALLBACK:
Images/online.gif Images/offline.gif

#COMMENTS
##We could use the following fallback settings to use
##a different stylesheet and script when offline
##style.css offline-style.css
##script.js offline-script.js
```

The JavaScript

Code Sample

html5-apis/Demos/offline/script.js

```
window.addEventListener("load",function() {
 document.getElementsByTagName("h1")[0].innerHTML="Hello World";
},false);

var appCache = window.applicationCache;

appCache.addEventListener("error", function() {
 alert("Cache failed to update");
 document.getElementById("output").innerHTML+="<li>a cache error has occurred</li>";
}, false);

appCache.addEventListener("updateready", function() {
 var refresh = confirm("An updated version is ready.  Press OK refresh your browser.");
 if (refresh) {
  location.reload();
 }
 document.getElementById("output").innerHTML+="<li>cache ready to be updated</li>";
}, false);

appCache.addEventListener("cached", function() {
 document.getElementById("output").innerHTML+="<li>application cached</li>";
}, false);

appCache.addEventListener("checking", function() {
 document.getElementById("output").innerHTML+="<li>checking cache</li>";
}, false);

appCache.addEventListener("downloading", function() {
 document.getElementById("output").innerHTML+="<li>cache download »» ing</li>";
}, false);

appCache.addEventListener("noupdate", function() {
 document.getElementById("output").innerHTML+="<li>no cache update</li>";
```

```
34.  }, false);
35.
36.  appCache.addEventListener("obsolete", function() {
37.   document.getElementById("output").innerHTML+="<li>manifest file returned
        >>>  a 404 or 410</li>";
38.  }, false);
39.
40.  appCache.addEventListener("progress", function() {
41.   document.getElementById("output").innerHTML+="<li>Resource being fetched
        >>>  from cache</li>";
42.  }, false);
```


The CSS

Code Sample

html5-apis/Demos/offline/style.css

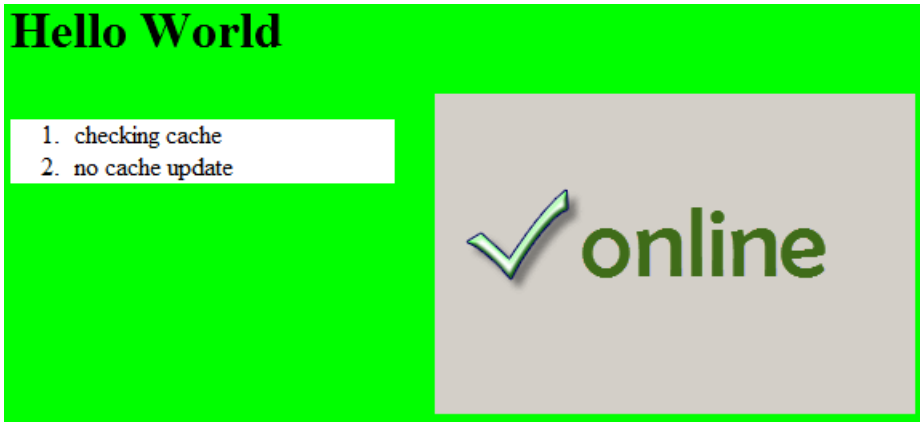
```
1.    body {
2.     background-color:#f00;
3.    }
4.
5.    #output {
6.     background-color:white;
7.     float:left;
8.     width:200px;
9.     margin-right:25px;
10.   }
```

And here are the two images, the first of which is shown if the browser can connect to the site:

To test this application, you must access it through a web server (e.g., localhost). Here's how it works:

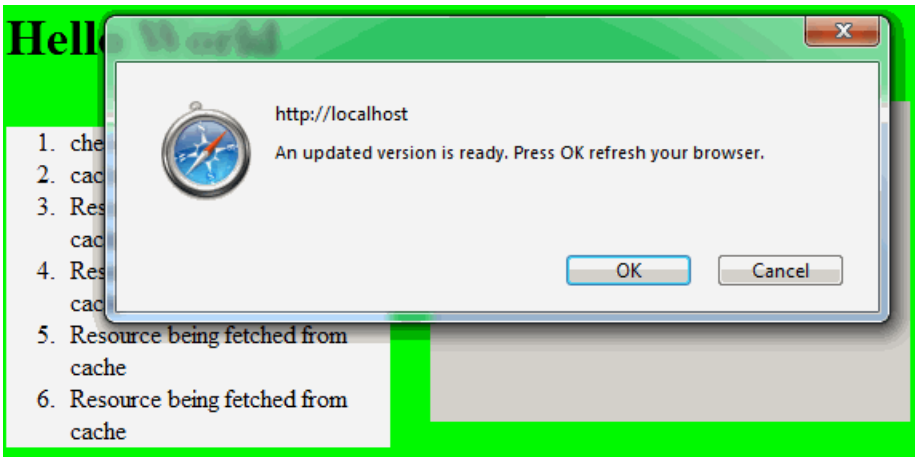
1. The first time you visit the page, all the files will download from the server and get cached:

2. Refresh the browser:



3. Modify the CSS document (e.g., change the background color) and refresh. Nothing changes. All the files were fed from cache.

4. Modify the cache manifest file to force it to be redownloaded (e.g., change the version comment from version 1.0 to version 1.1. Refresh the browser:



Notice that the resources are being fetched again, but the page has not updated.

5. If you press **Cancel** on the confirm dialog, the page will not update. Instead, press **OK** to get the page to update:

6. Our manifest file indicates that offline.gif should be displayed if the browser is offline. To see this, you need to go offline, which is hard to do when you're working locally.

8.2 Drag and Drop API

The HTML5 Drag and Drop API provides a great illustration of the downside of the "paving the cowpaths" approach of HTML5. It standardizes the drag-and-drop API originally created in Internet Explorer 5 (yup, 5). As other browsers implemented the same API, it was decided that this was a cowpath worth paving. We explain

basic usage here, which ironically doesn't work very well in Internet Explorer 9, but does work in the other HTML5-compliant browsers.

To build a drag-and-drop application, you need to do the following:

1. Create a draggable element.
2. Create an element to be your drop zone.
3. Capture and respond to drag-and-drop events.
4. Optionally change styles during drag-and-drop process.

It's easiest to understand by looking at a small demo application:

The HTML Page

Code Sample

html5-apis/Demos/drag-and-drop.html

```
1.   <!DOCTYPE HTML>
2.   <html>
3.   <head>
4.   <meta charset="UTF-8">
5.   <title>Drag and Drop API</title>
6.   <script src="../../html5-common/dateFormat.js"
        >>>  type="text/javascript"></script>
7.   <script src="dragdrop/script.js" type="text/javascript"></script>
8.   <link href="dragdrop/style.css" rel="stylesheet">
9.   </head>
10.  <body>
11.  <h1>Drag and Drop</h1>
12.  <div id="dropzone">Last Dropped: <time>never</time></div>
13.  <div id="drag" draggable="true"></div>
14.  <ol id="output"></ol>
15.  </body>
16.  </html>
```

Code Explanation

We have two `div`s:

1. "dropzone" for capturing the drop events
2. "drag" for dragging - notice the **draggable="true"** attribute

The CSS

Code Sample

html5-apis/Demos/dragdrop/style.css

```
#dropzone {
 border:1px solid black;
 width:100px;
 height:100px;
 float:left;
 margin-right:10px;
 text-align:center;
}

#drag {
 background-color:red;
 width:50px;
 height:50px;
 float:left;
 cursor:move;
}

#output {
 background-color:white;
 float:left;
 width:200px;
 margin-right:25px;
}

time {
 text-decoration:underline;
}
```

Code Explanation

Nothing too exciting here. Just some basic styling.

The JavaScript

Code Sample

html5-apis/Demos/dragdrop/script.js

```
var dropzone, drag;
window.addEventListener("load",dragDrop,false);
function dragDrop() {
 dropzone = document.getElementById("dropzone");
 drag = document.getElementById("drag");
 drag.addEventListener("dragstart", handleDragStart, false);
 drag.addEventListener("drag", handleDrag, false);
 drag.addEventListener("dragend", handleDragEnd, false);
 dropzone.addEventListener("dragenter", handleDragEnter, false);
 dropzone.addEventListener("dragover", handleDragOver, false);
 dropzone.addEventListener("dragleave", handleDragLeave, false);
 dropzone.addEventListener("drop", handleDrop, false);
}

function handleDragStart(e) {
 this.style.backgroundColor="green";
 document.getElementById("output").innerHTML+="<li>Dragging Start »»
    >>> ed</li>";
 e.dataTransfer.setData('Text',this.id);
}

var dragReported=false;
function handleDrag(e) {
 if (!dragReported) document.getElementById("output").inner »»
    >>> HTML+="<li>Dragging</li>";
 dragReported = true;
}

function handleDragEnter(e) {
 document.getElementById("output").innerHTML+="<li>Drag Enter</li>";
 cancel(e);
}

var dragOverReported=false;
function handleDragLeave(e) {
 document.getElementById("output").innerHTML+="<li>Drag Leave</li>";
 dragOverReported = false;
}
```



```
37.
38.  function handleDragOver(e) {
39.   if (!dragOverReported) document.getElementById("output").inner »»
        >>>  HTML+="<li>Dragging Over</li>";
40.   dragOverReported = true;
41.   cancel(e);
42.  }
43.
44.  function handleDrop(e) {
45.   var droppedElem = e.dataTransfer.getData('Text');
46.   document.getElementById("output").innerHTML+="<li><strong>" +
        >>>  droppedElem + "</strong> dropped on <strong>" + this.id +
        >>>  "</strong></li>";
47.   var now = new Date();
48.   this.getElementsByTagName("time")[0].innerHTML = now.format("H:MM:ss");
        >>>
49.   cancel(e);
50.  }
51.
52.  function handleDragEnd(e) {
53.   document.getElementById("output").innerHTML+="<li>Dragging Ended</li>";
        >>>
54.   this.style.backgroundColor="red";
55.  }
56.
57.  function cancel(e) {
58.   if (e.preventDefault) {
59.    e.preventDefault();
60.   }
61.   return false;
62.  }
```

Code Explanation

Here is where the action takes place. We capture these events for the draggable element:

1. `dragstart`
2. `drag`
3. `dragend`

And these events for the drop zone:

1. `dragenter`
2. `dragover`

3. `dragleave`
4. `drop`

Note that the `drag` and `dragover` events fire repeatedly as the draggable element is dragged. That's why we use the booleans `dragReported` and `dragOverReported` to stop the reporting. If you leave those out, you'll likely crash your browser.

The excitement happens in the `handleDrop()` function, which writes out the time of the drop. You could do anything at this point; for example, change the position of the element or add an item to a shopping cart.

You may have noticed the two calls to cancel():

1. In `handleDragOver()`, that allows us to drop the draggable element.
2. In `handleDrop()`, that prevents the browser from trying to navigate to another page when the element is dropped.

The unfortunate reality is that the drag-and-drop API in HTML5 does not make developers' lives easier. If you're working a lot with drag-and-drop, you're almost definitely better off using a framework like jQuery or YUI. However, the concepts covered in this section are the same as you'll see in those libraries.

8.3 Conclusion

In this lesson, you have learned

- about the Offline Application API.
- about the Drag and Drop API.

