Responsive Web Development with HTML5, CSS3, JavaScript and jQuery

Student Labs

Table of Contents

Lab 1 - Create a Basic HTML5 Layout

In this lab, you will create an HTML5 template which will serve as the starting point for creating new HTML files in upcoming labs.

This lab will help you understand what a very simple HTML5 document looks like. Specifically, you will learn to declare the DOCTYPE and character encoding.

When creating HTML5 files, you will use Notepad as a text editor. Of course, you're free to use another text editor if you'd prefer.

Also, we'll be using Google Chrome as a web browser, since it provides the best support for HTML5 in a Windows environment.

Note: On the Mac, Safari provides the best support for HTML5. Feel free to experiment with other web browsers, such as Firefox 4 and Internet Explorer 9. However, we can't guarantee that all the HTML5 pages you create will render properly in those other browsers. If you want to see how well your browser supports HTML5, navigate to **http://www.html5test.com** from within the associated browser.

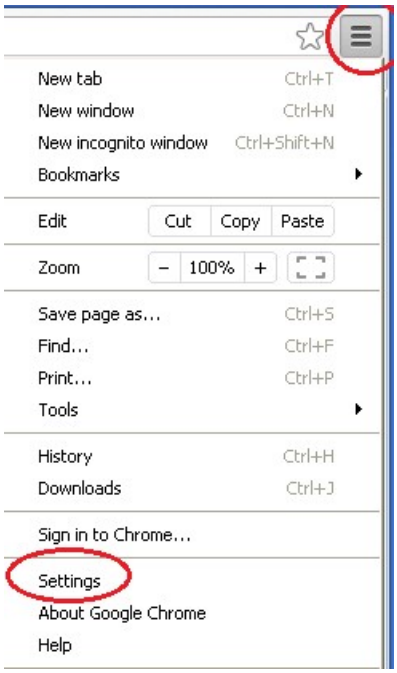
Note: These labs were written using Chrome 23.0.x. If you're using another version of Chrome, you may see slight differences in the GUI.

Part 1 - Clear the Browser Cache and Location Settings

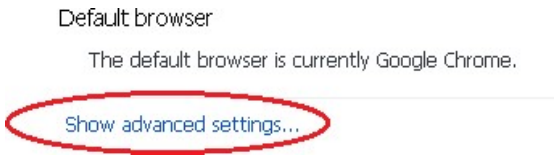
In this part, you will clear Chrome's cache and location settings. This will help ensure that any cached files and stored location settings will not interfere with any of the labs.

__1. Open Chrome by selecting **Start->All Programs->Google Chrome->Google Chrome**.

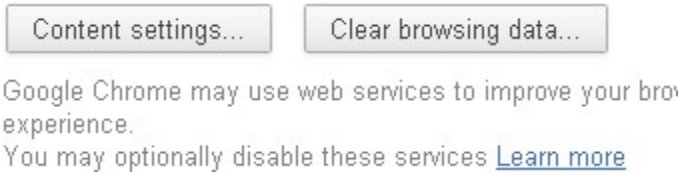
__2. Click the icon to the right of the location bar and select **Settings** from the drop-down menu. Older Chrome versions will show Options instead of Settings.



__3. Under Settings expand Show advanced settings...



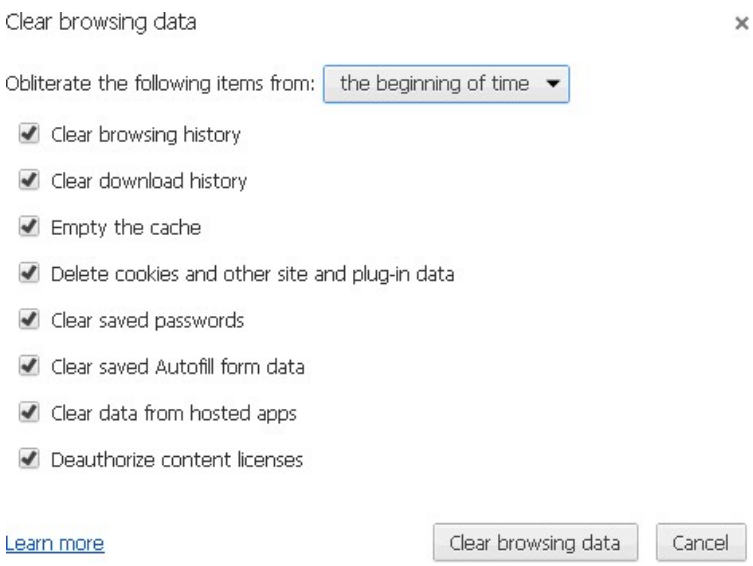
__4. Under Privacy click the **Clear browsing data** button.



__5. Check all the checkboxes.

__6. Select **the beginning of time** option from the **Obliterate the following items from** drop-down box.

__7. Click the **Clear browsing data** button.

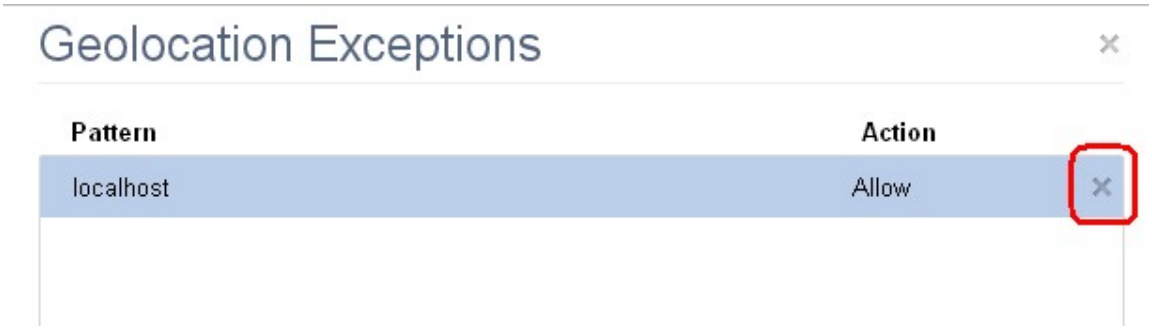


Wait for the clearing process to complete.

__8. Click the **Content settings** button, which is located to the left of the **Clear browsing data** button.

__9. Scroll down in the browser and click the **Manage exceptions** button under the **Location** section.

__10. If any exceptions appear in the panel, select the exception and click the **X** to the right of it to delete the exception.



__11. Click the **X** in the upper right corner of the **Geolocation Exceptions** panel to close the panel.

__12. Click the **X** in the upper right corner of the **Content Settings** panel to close the panel.

__13. Close the browser.

Part 2 - Create the Template

In this part, you will create the HTML5 template. The template will consist of the following elements:

- DOCTYPE
- html
- head
- meta (used to specify character encoding)
- title
- body

__1. Run the following program to run a specialized editor. You do not need to use anything but a regular text editor for the labs but this program has some tools that can make it easier to work with the web pages you will be editing.

```
C:\Software\NotepadPlus\notepad++.exe
```

__2. If you are asked to install plug-ins simply click **Cancel**.

__3. If there is any opened file, close it.

__4. Enter the **DOCTYPE** declaration, followed by the **html** element:

```
<!DOCTYPE html>
<html>
```

The DOCTYPE declaration must be the very first element in your HTML5 document. It even must appear before the html element.

Note: For those of you who are familiar with XML and XHTML (an XML compliant version of HTML), the DOCTYPE element is used to indicate the document type definition (DTD) that an XML document adheres to. A DTD is used to specify the grammar of an XML document.

However, an HTML5 document is not XML, so the DOCTYPE element is not required for this purpose. Instead, it's used to trigger "no-quirks mode" (a.k.a. "standards mode"). "No-quirks mode" implies the browser renders the page according to the HTML5 and CSS standards. "Quirks mode" implies it doesn't and instead tries to maintain backward compatibility with legacy pages created for older browser versions.

Also, notice that in HTML5 the DOCTYPE element is kept very minimal and simply includes the root element name. Normally, when entering a DOCTYPE element in an XHTML/XML document, you would need to specify several other pieces of information, including a URL pointing to the DTD (see example below). However, since the DOCTYPE element is not being used to reference a DTD, we can omit this additional information.

e.g.,

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Note: The DOCTYPE element is case insensitive. You could have written the following instead:

```
<!DOCTYPE HTML>
<!doctype html>
```

__5. Define the document's character encoding inside the head element as shown below.

```
<head>
   <meta charset="utf-8"/>
   <title>HTML 5 Template</title>
</head>
```

Note: The character encoding declaration is shorter than the one you're accustomed to writing in HTML4:

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

Note: HTML5 is not XML. Hence, element names are not case sensitive, elements don't require both a start and end tag, and attribute names don't need to be enclosed in quotes. Hence, you could have expressed the meta element in any of the ways listed below:

```
<META CHARSET="utf-8"/>

<META CHARSET=utf-8/>

<meta charset="utf-8">

<meta charset=utf-8/>

<Meta CharSet=Utf-8>
```

Having said that, we will use XML syntax going forward.

__6. Finally, add the **body** element and the **html** close tag to the document:

```
<body>
   Contents goes here...
</body>

</html>
```

Next, you will save the HTML file to the Apache Web Server document root.

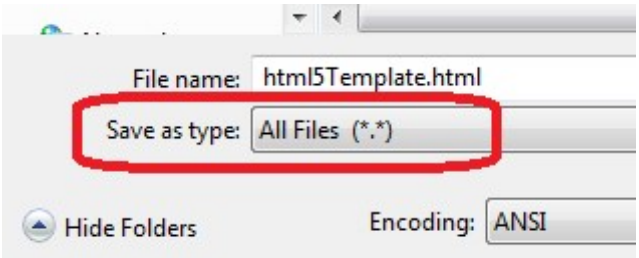
__7. Open Windows Explorer.

__8. Create a folder called **layout** under the following directory (the Apache folder may be installed in another location, check with your instructor):

```
C:\Apache2.2\htdocs
```

__9. Save the HTML file as **html5Template.html** within the layout folder.

Note: If you're using Notepad to save the file, make sure you choose **All Files (*.*)** in the **Save as type** field. Otherwise, a **txt** extension will automatically be added to the file name.



Part 3 - Test

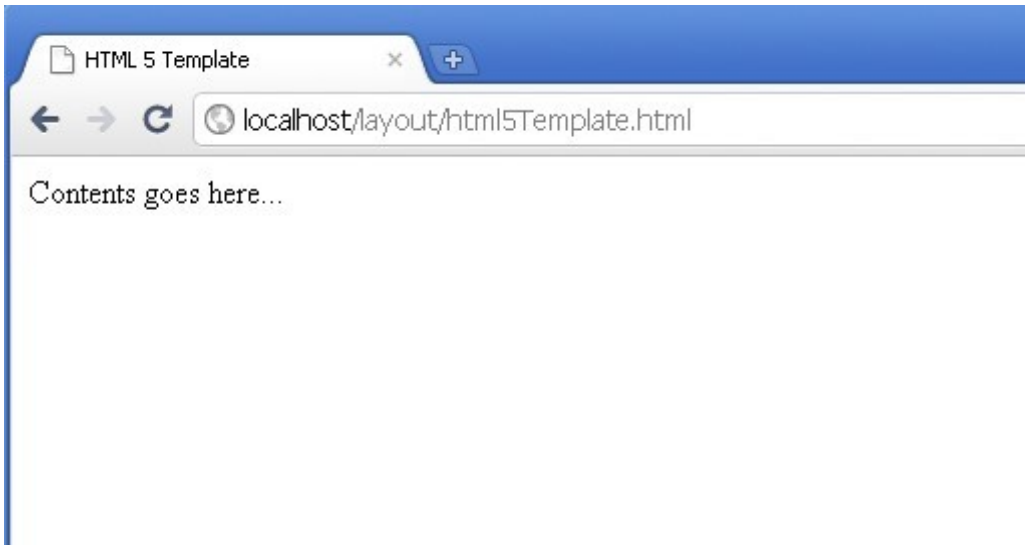
In order to test the HTML5 document, we'll use Google Chrome, since it provides the best support for HTML5.

__1. Open the **Chrome** browser.

__2. Enter the following URL into the address bar:

```
http://localhost/layout/html5Template.html
```

__3. You should see the following screen:



__4. Close the all.

Part 4 - Review

In this lab, you created a template for your HTML5 files. In the process, you learned how to declare DOCTYPE and meta elements.

Lab 2 - Improve Your Forms Using HTML5

An HTML form is used for gathering information from a user and submitting it to a server to be processed. This information normally needs to be validated. This is typically done both on the client side and the server side, since client validation may be turned off in the browser.

Prior to HTML5, the form fields didn't have any built-in validation logic. That meant you were stuck writing JavaScript code to perform client side validation, which was both tedious and error prone. Now in HTML5, new input types were introduced containing built-in validation logic, which eliminates the need for JavaScript code (see note below). In this lab, you will learn to use several of the new input types: **e-mail**, **date**, **number**, and **range**.

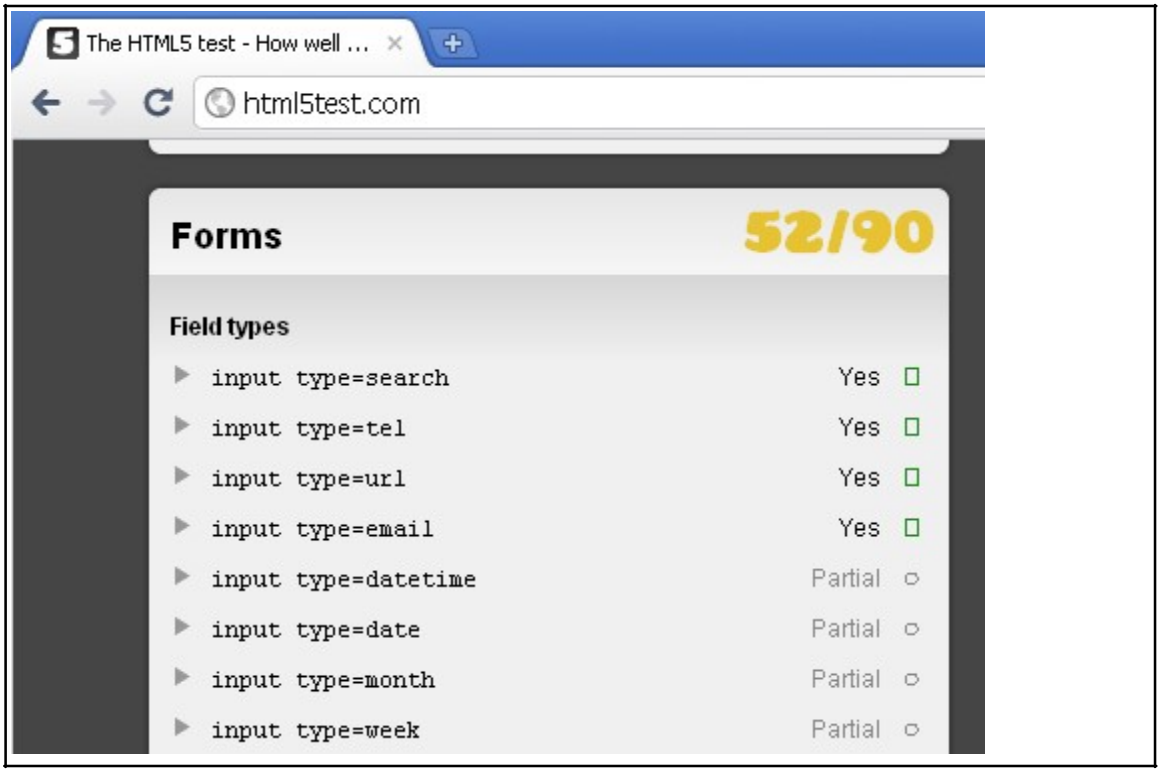
In addition to these new input types, HTML5 also supports three major new features for form fields: the **placeholder**, **autofocus**, and **pattern** attributes. Implementing the features provided by these attributes used to require JavaScript code.

The **placeholder** attribute allows the developer to define the text that will appear in a text box-based control when its contents are empty. When the field gains focus, the text is removed. If the user doesn't type a value into the field and the field loses focus, the text is restored. You may want to use the placeholder attribute when implementing a search field to indicate to the user they should input a value. This way, you don't need to define a label for the search field that's displayed adjacent to it.

The **autofocus** attribute tells a browser to automatically give focus to a particular form field when the page is loaded.

The **pattern** attribute is used to specify a regular expression that a field's value must match.

Note: You'll still want to have JavaScript code inside your pages for older browsers that don't support HTML5 or for newer browsers that don't fully support HTML5. You can navigate to http://www.html5test.com to see how well your browser supports HTML5. Below are the results for Google Chrome v10.0.x., which provides fairly good support for the new HTML5 form features:



Part 1 - Create an Input Form Using the New HTML5 Input Types

In this part, you will create a new HTML file, based on the template, which will contain an input form for registering a new employee. You will make use of several of the new input types which provide built-in validation.

__1. Open **C:\Apache2.2\htdocs\layout\html5Template.html**. Save it as **registerEmployee.html** in the same folder.

__2. Change the **title** content to:

```
<title>Register an Employee Form</title>
```

__3. Remove the content inside the body element.

__4. Add the following code inside the body element:

```
<h1>Employee Registration Form</h1>
```


Next we will create the registration form.

__5. Add the following form element:

```
<form action="thankYou.html">
</form>
```

__6. Add a label and a corresponding text field for the employee's name inside the form element:

```
<label for="fullName">Full Name:</label><br/>
<input type="text" name="fullName" id="fullName" placeholder="Enter
your full name" autofocus required />*<br/>
```

Notice the following:

- We used the **placeholder** attribute to specify text that will appear inside the field when the page is loaded. This gives the user a visual cue that they need to input their name into the text field.
- We used the **autofocus** attribute to indicate that the field should automatically get focus when the page is first loaded. Before HTML5, this could only be done using JavaScript.
- To specify that the employee's name is mandatory, we used the new HTML5 attribute **required**.

Note: When specifying autofocus and required, it's also legal to use XML syntax:

<input type="text" name="fullName" id="fullName" placeholder="Enter your full name" autofocus="autofocus" required="required" />

__7. Add a label and an e-mail field for the employee's e-mail:

```
<label for="email">E-mail:</label><br/>
<input type="email" name="email" id="email" placeholder="Enter a valid
e-mail" required />*<br/>
```


Notice the following:

- To create an e-mail field, we used the input element and specified the new HTML input type **email**.
- To specify that the e-mail is mandatory, we used the new HTML5 attribute **required**. That means the required attribute is not specific to text fields (input type="text"), but can be used with many of the new input types too.

__8. Add a label and a date field for the employee's date of birth.

```
<label for="dateOfBirth">Date of Birth:</label><br/>
<input type="date" name="dateOfBirth" id="dateOfBirth" required />*<br/>
```

Notice the following:

- To create a date field, we used the input element and specified the new HTML input type **date**.

__9. Add a label and a number field for the employee's age:

```
<label for="age">Age:</label><br/>
<input type="number" name="age" id="age" size="6" min="18" max="120"
step="1" value="21" /><br/>
```

Notice the following:

- To create a number field, we used the input element and specified the new HTML input type **number**.
- We used the **size** attribute to indicate how many characters (6) can be input for the employee's age.
- We used the **min** and **max** attributes to specify the minimum age (18) and maximum age (120), respectively.
- We used the **step** attribute to specify the interval (1) between valid values (e.g., 18, 19, 20).
- We used the **value** attribute to specify the default age (21).

__10. Add a label and a range field for the employee's number of years of experience:

```
<label for="rating">Years of Experience:</label><br/>
<input type="range" name="rating" id="rating" min="0" max="100"
step="1" value="0" /><br/>
```


Notice the following:

- To create a range field, we used the input element and specified the new HTML input type **range**.
- We used the **min** and **max** attributes to specify the minimum number of years of experience (0) and maximum number of years of experience (100), respectively.
- We used the **step** attribute to specify the interval (1) between valid values (e.g., 1, 2, 3).
- We used the **value** attribute to specify the default number of years of experience (0).

__11. Add a label and a text field for the employee's phone number:

```
<label for="phone">Phone:</label><br/>
<input type="text" name="phone" id="phone" pattern="[0-9]{3}-[0-9]{3}-
[0-9]{4}" placeholder="XXX-XXX-XXXX" required />*<br/>
```

Notice the following:

- To create a phone number field, we used a standard text field. There is no new input type for phone numbers.
- HTML5 allows us to use regular expressions to limit the set of values that you can input into a field. In this case, we input a regular expression which restricts the set of values to a number that looks like this: XXX-XXX-XXXX
- We used the **placeholder** attribute to display the phone number pattern to the user so they don't mistype it.

Note: The string start `^` and end `$` indicators are not required for the regular expression. This is because HTML5 automatically prefixes the regular expression with **`^`**`(?:` and suffixes it with `)$`.

__12. Finally, add a submit button to allow the user to submit the form and a label indicating that any fields with an asterisk (*) next to them are required.

```
<input type="submit" value="Register" />
<p>Fields marked * are required.</p>
```

__13. Save and close the file.

__14. Copy **C:\LabFiles\thankYou.html** to **C:\Apache2.2\htdocs\layout**, since the form submits to that page.

Part 2 - Test

__1. Open Google Chrome.

__2. Navigate to the following URL:

```
http://localhost/layout/registerEmployee.html
```

__3. You should see the following screen:

Notice the following:

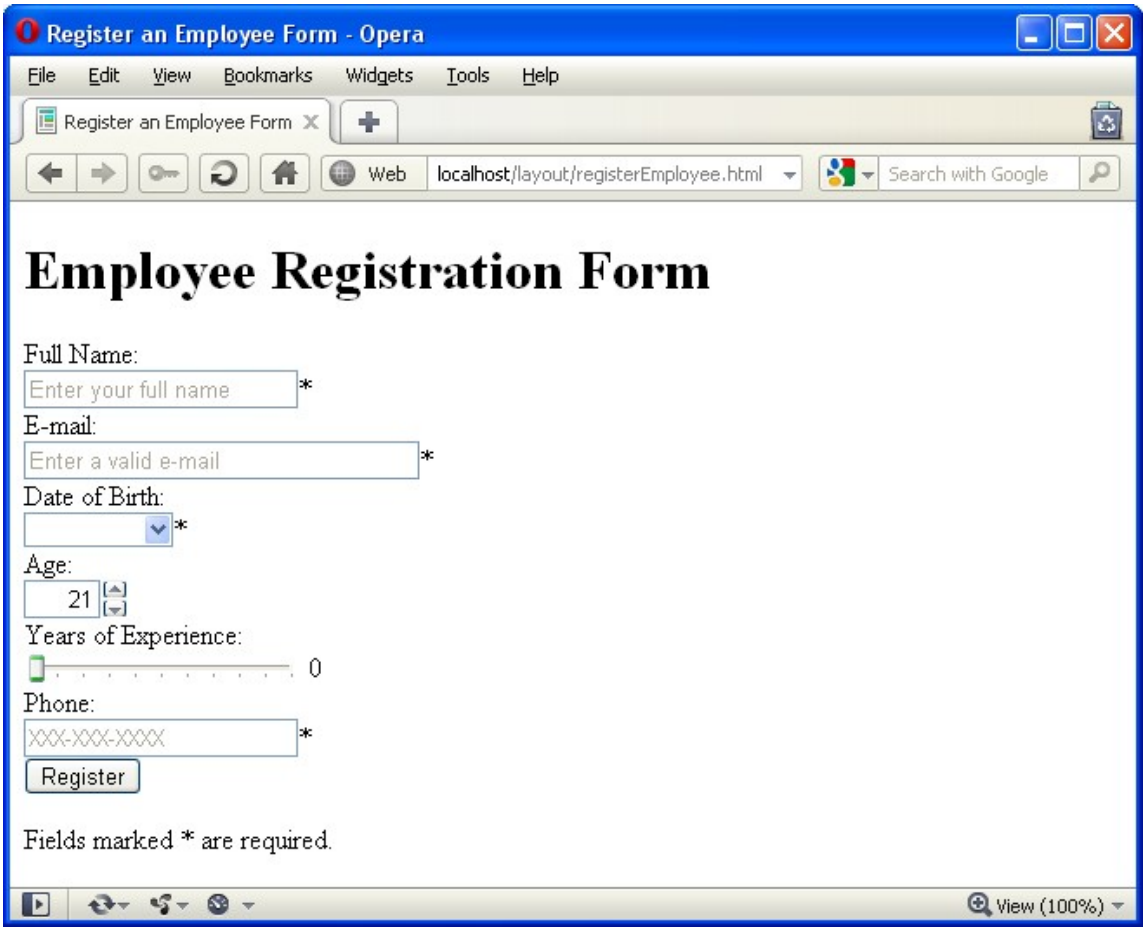
- The **Full Name** field has focus thanks to the autofocus attribute.
- The placeholder text appears inside the fields.

__4. Also, notice the following:

- Chrome rendered the **E-mail** field (input type="email") as a text field. HTML5 does not mandate any default user interface for the new input types, so each browser and device is free to choose how to render them.

- Chrome rendered the **Date of Birth** field (input type="date") as a *spin box* (i.e., a text field with up and down arrows that you click on to change the value) that allows you to increment/decrement the date by a day. Ideally, it would have been preferable if it rendered a date picker. Once again, HTML5 does not specify a default rendering for this input type.
- Chrome also rendered the **Age** field (input type="number") as a spin box. However, when you click the up and down arrows, the age is incremented by the value of the step attribute (1).
- Chrome rendered the **Years of Experience** field (input type="range") as a slider bar. However, we can't tell what the value selected is. We'll address this limitation in the next part of the lab.

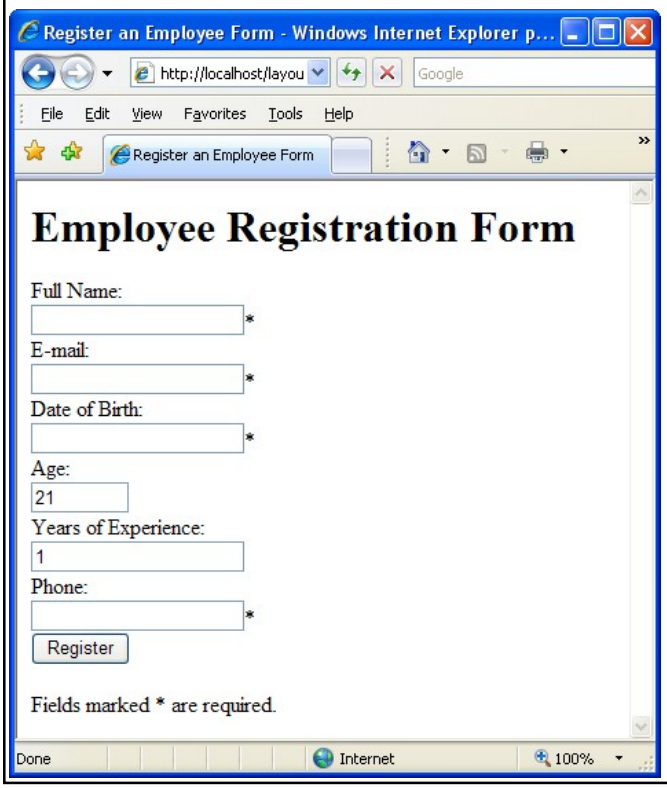
Note: If you loaded this page in Opera 11.01, this is how it would look:



Notice the date field is rendered as a drop-down. If you click on the drop-down, a date picker will be displayed:

Note: If you load a page in an older browser that doesn't support/fully support HTML5, the new input types will be ignored. The input types will gracefully degrade to text fields (type="text").

e.g., Internet Explorer 7



Now let's test how the form works. We'll test the "happy day" scenario first.

__5. Input the following values:

Full Name: Mike Harris

E-mail: mharris@yahoo.com

Date of Birth: 01-23-1980

Age: 21 (default value)

Years of Experience: 0 (default value)

Phone: 215-314-2300

__6. Submit the form by clicking **Register**.

__7. You should see the thank you page.

Now let's try some error scenarios. First, we'll see what happens if you omit a required field.

__8. Click the Back button (left arrow).

__9. Delete the text in the full name field.

__10. Try submitting the form.

__11. You should see the following error message:

Pretty cool!

Let's try another error scenario. Let's see what happens if the user types in an invalid e-mail address.

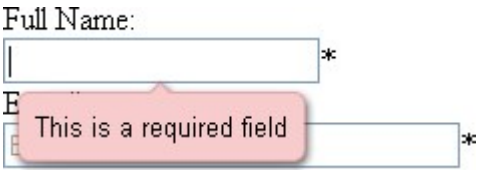
__12. Enter **Mike Harris** back into the full name field.

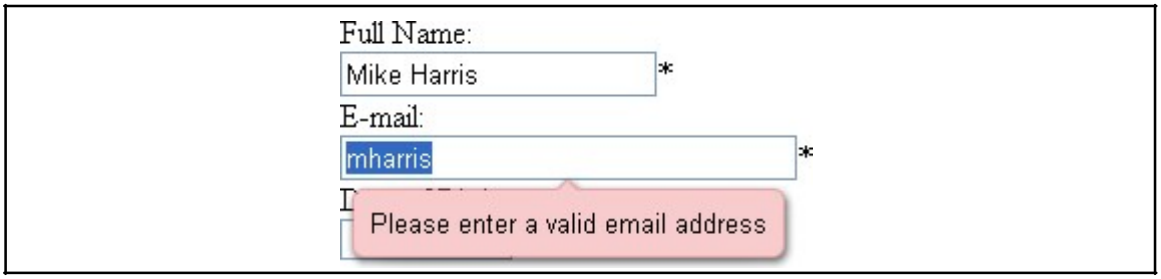
__13. Change the e-mail address from **mharris@yahoo.com** to **mharris**.

__14. Try submitting the form.

__15. You should see the following error message:

Note: Opera will also display a visual cue (callout box) indicating a validation rule has failed when you try submitting the form, as shown below. The error messages though are a little bit different. This is because the HTML5 specification doesn't dictate what error messages should be displayed.





Part 3 - Display the Value of the Range Field

In this part, you will add some JavaScript and HTML code to display the value for the number of years of experience that the user input, since Chrome doesn't automatically display values for range fields.

__1. Open **registerEmployee.html**.

__2. First off, add a span element to the right of the range field. The span element will display the value the user selected for the number of years of experience:

```
<input type="range" name="rating" id="rating" min="0" max="100"
step="1" value="0" />
<span id="ratingValue">0</span><br/>
```

__3. Next, add an onchange event handler to the range field that gets called after the user drags the slider to a new position:

```
<input type="range" name="rating" id="rating" min="0" max="100"
step="1" value="0" onchange="displayRating(this);" />
```

Note: The parameter **this** that's passed to the displayRating function is a reference to the current element (i.e., the input element).

__4. Add a script element before **</head>**.

```
<script>

</script>
```

Note: In HTML5, you no longer need to specify the type attribute to indicate the category of scripting language you're using. It now defaults to "text/javascript".

__5. Add the event handler function, displayRating, inside the script element:

```
function displayRating(slider) {
   document.getElementById("ratingValue").innerHTML = slider.value;
}
```

This function retrieves a reference to the span element and updates the HTML inside with the slider value.

Note: Remember, JavaScript is a case sensitive language so pay attention to case.

__6. Save your changes.

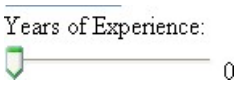
__7. Close all open browsers.

Now let's test the changes.

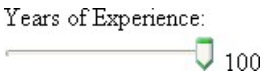
__8. Navigate to the following URL:

```
http://localhost/layout/registerEmployee.html
```

You should now see **0** for the number of years of experience listed to the right of the slider.



__9. Drag the slider all the way to the right. Make sure the value increments in steps of 1 and reaches a maximum value of 100.



__10. Now drag the slider all the way to left. Make sure the value decrements in steps of 1 and reaches a minimum value of 0.

Troubleshooting: If the value doesn't get updated when you drag the slider, refer to the next section for troubleshooting instructions.

Part 4 - Troubleshooting JavaScript Problems in Chrome

In this part, you will learn a little bit about how to troubleshoot JavaScript related problems that crop up when using Chrome.

For example, if you load a web page that has JavaScript syntax errors and do something that invokes that code, Chrome won't display any errors unless you open the Developer Tools. Aside from viewing syntax errors, you can use the Developer Tools for debugging semantic errors.

In our case, we will pretend you made a typo.

__1. Change the code as shown in bold below:

```
function displayRating(slider) {
   document.getelementById("ratingValue").innerHTML = slider.value;
}
```

__2. Save the file.

__3. Refresh the browser to pick up the changes.

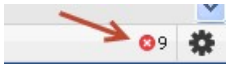
__4. Try dragging the slider.

Notice that the value stays at 0. Let's see why.

__5. Select **Tools->Developer tools** from the menu bar.

__6. The Developer tools should appear in the lower half of the browser window. Click the Sources tab and expand the html code.

Notice the red **X** in the lower right corner of the screen. This indicates there are errors in your page.



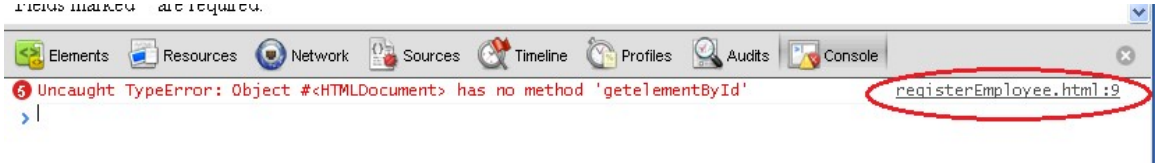
Note: You will probably see a different number of errors than above. The number 9 does not indicate how many errors there are in the HTML file. In our case, there is only 1 error. It just indicates how many times that line of code was executed in error (in this case, 9 times). Since I kept dragging the slider to the right, it kept trying to execute the same line of code to update the value.

__7. Click the Console tab.

It should show you the error message that occurred when executing the code, along with the offending file and line number.

The error message is very clear. The HTMLDocument object doesn't have a method named getelementById.

__8. Click the **registerEmployee.html:9** link.



__9. Chrome will show you the offending line of code in the **Scripts** tab.

__10. Fix the error in the code.

__11. Save your changes and refresh the page.

__12. Now try dragging the slider. The value should be updated and you should see no red Xs in the Developer Tools.

Now let's try debugging using the Developer Tools.

__13. Click on the **Sources** tab inside the *Developer Tools* panel.

__14. Click inside the gray margin to the left of the **document.getElementById()** line of code inside the **displayRating** function.

__15. Verify a breakpoint symbol (blue arrow) appears.

```
4    <meta charset="utf-8"/>
5    <title>Register an Employee Form</title>
6
7  <script>
8  function displayRating(slider) {
9     document.getElementById("ratingValue").innerHTML = slider.value;
10 }
11 </script>
12
13 </head>
```

__16. Try dragging the slider.

__17. Verify that the breakpoint was hit. The line of code should now appear in blue.

```
6    <title>Register an Employee Form</title>
7    <script>
8        function displayRating(slider) {
9            document.getElementById("ratingValue").i
10       }
```

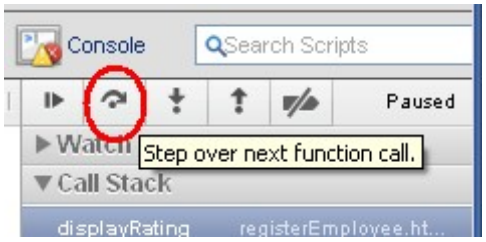
Note: When a breakpoint is hit, it means that the line of code (in our case, document.getElementById()) is about to be executed.

__18. In the right panel, look inside the **Scope Variables** panel and you'll see the **slider** variable listed.

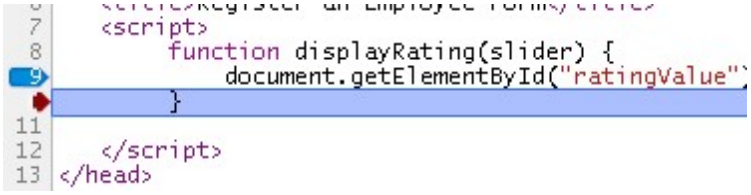
__19. Expand it and you can see the value of its properties.

Ordinarily, being able to see the value of a variable or its properties can be useful in troubleshooting semantic errors. In our case, it doesn't help us, since we don't have any semantic errors.

__20. To execute the line of code, click the **Step over next function call** button in the toolbar at the top of the panel.



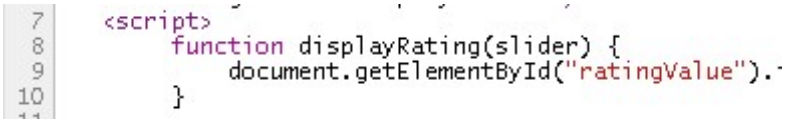
__21. The cursor should advance to the next line of code.



Note: There is also a **Step into next function call** that allows you to step through the code inside of a function that's called by the current function. This can be useful in terms of troubleshooting functions you've written which call other functions.

__22. Click the **Resume script execution** button (to the left of the **Step over next function call** button) to resume running the script.

__23. Remove the breakpoint by clicking on the blue arrow in the margin. The arrow should disappear.



__24. Close the Developer Tools by selecting **Tools->Developer tools** from the menu bar.

Note: You can also press **Ctrl-Shift-I** to open and close the Developer Tools.

Part 5 - Display the Value of the Range Field Using the Output Element

In this part, you will replace the existing span element and JavaScript function that you used to display the slider value with the new output element. The output element was introduced in HTML5 to display the result of a calculation.

__1. Replace the **span** element:

```
<span id="ratingValue">0</span><br/>
```

with the **output** element:

```
<output name=output>0</output><br/>
```

__2. Add the 'oninput' attribute below to the 'rating' input element, so that it looks like so:

```
<input type="range" name="rating" id="rating" min="0" max="100"
step="1" value="0" oninput="output.value=rating.value"/>
```

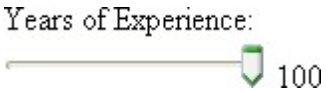
Like the output element, the **oninput** attribute was also introduced in HTML5. It's used to specify what happens when the oninput event is triggered . This event is triggered when the user interacts with an input control. In this case, we're setting the value of the output element to the value of the rating element.

__3. Delete the **script** element, containing the displayRating function, since it's no longer needed.

__4. Save your changes.

__5. Refresh the browser. You should now see **0** for the number of years of experience listed to the right of the slider.

__6. Drag the slider all the way to the right. Make sure the value increments in steps of 1 and reaches a maximum value of 100.



__7. Now drag the slider all the way to left. Make sure the value decrements in steps of 1 and reaches a minimum value of 0.

Part 6 - Style the Error Fields

In this part, you will create some CSS styles so that fields that fail validation show up in pink.

__1. Add the following style element before **</head>**:

```
<style type="text/css">
   :required, :invalid { background-color: pink; }
   :valid { background-color: white; }
</style>
```

HTML5 introduced new pseudo-classes that can be used to style validation errors. In this case, you specified that any required elements or invalid elements should have a background color of pink, whereas any valid elements should have a background color of white.

__2. Save and close the file.

__3. Refresh the page.

You should see the following screen:

Notice that all the required fields (**Full Name**, **E-mail**, **Date of Birth**, and **Phone**) are displayed in pink. This makes sense, given that they're not filled in. It also indicates that Chrome validates the field data even before you submit the form.

__4. Input **Mike Harris** for the full name.

Notice the **Full Name** field turns white. This means the field is valid. Remember that you set the background color for the :valid pseudo-class to white.

__5. Now input **mharris@yahoo.com** for the e-mail.

Now the **E-mail** field should turn white.

__6. Repeat this process for the remaining fields.

__7. Submit the form. You should see the thank you page.

__8. Close the browser.

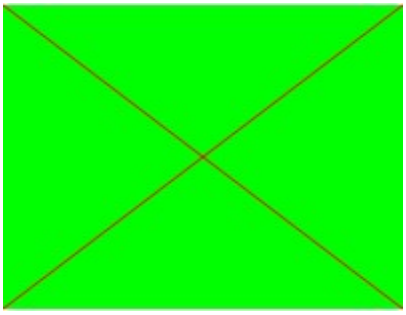
Part 7 - Review

In this lab, you learned how to:

- Use some of the new input types: e-mail, date, number, and range
- Use the autofocus, placeholder, and pattern attributes
- Use the output element and oninput attribute.
- Style the fields that are in error

Lab 3 - Drawing Using the Canvas Element

In this lab, you will learn how to draw 2D graphics using the canvas element introduced in HTML5. The canvas element allows you to draw rectangles, ovals, lines, arcs, text, images, etc. We won't be drawing anything too fancy in this lab. Instead, we'll simply draw a green rectangle containing two red diagonal lines.



Part 1 - Create a 2D Drawing Using the Canvas Element

In this part, you will create a new HTML page that uses a canvas element to render the image above.

__1. Create a new folder called **canvas** within the **C:\Apache2.2\htdocs** folder.

__2. Open **C:\Apache2.2\htdocs\layout\html5Template.html** using Notepad and save it as **drawOnCanvas.html** in the **canvas** folder.

__3. Change the title content to:

```
<title>Draw Something</title>
```

__4. Remove the content inside the body element.

Next we will create our canvas.

__5. Add a canvas element inside the body element:

```
<canvas id="c" width="400" height="300">
   Your browser does not support the canvas element.
</canvas>
```


A canvas doesn't display anything by default. It's just used to define a rectangular region within the page that you can draw on. To draw on it, we'll need to use JavaScript. A canvas element has a corresponding DOM element (called HTMLCanvasElement) that we can use to draw graphics.

Note: Technically, you can have many canvas elements inside your page, each of which contains it own drawing.

__6. Add an onload event handler to the body element:

```
<body onload="drawOnCanvas();">
```

When the page loads, the drawOnCanvas function, which we'll be writing next, gets called.

__7. Add the following script element before **</head>**:

```
<script>

</script>
```

__8. Add the declaration for the drawOnCanvas function before **</script>**:

```
function drawOnCanvas() {

}
```

__9. Add the following variable declarations inside the drawOnCanvas function:

```
var c = document.getElementById("c");
var ctx = c.getContext("2d");
```

The code does the following:

- Retrieves a reference to the canvas element (i.e., the DOM element mentioned above).
- Obtains a reference to a 2D drawing context. The drawing context is used to draw the graphics.

Note: Currently, HTML5 only has support for a 2D graphics context. The HTML5 specification states there may be support for a 3D graphics context in a future version.

__10. Add the following code to draw a green rectangle to the drawOnCanvas function:

```
// set fill color to green
ctx.fillStyle = "#00FF00";
// draw filled rectangle
ctx.fillRect(50, 50, 200, 150);
```

Notice the following:

- We used the **fillStyle** property of the context object to set the fill color to "#00FF00" (green).
- We used the **fillRect** method of the context object to draw a filled rectangle. When drawing a rectangle, the fillRect method uses the current fillStyle, which defaults to black.
- **fillRect** takes 4 arguments: x, y, width, and height. x and y correspond with the upper left hand coordinates of the rectangle.

Note: Remember that the origin (x = 0, y = 0) corresponds with the upper left hand corner of the browser window. As you move to the right (along the x axis), x increases in value. As you move down (along the y axis), y increases in value.

Note: When specifying the fillStyle color, you can specify a CSS color in hexadecimal format (e.g., "#00FF00"), in RGB format (e.g., "rgb(0, 255, 0)"), or using one of the predefined color names (e.g., "red").

Note: If you don't want to create a filled rectangle, you can use the strokeRect method instead. It has the same method signature as the **fillRect** method.

__11. Add the following code to draw the first diagonal to the drawOnCanvas function:

```
// trace first diagonal
ctx.moveTo(50, 50);
ctx.lineTo(250, 200);
```

moveTo(x, y) allow you to specify where to start drawing a line. In this case, the starting point for your line is the upper left corner of the rectangle.

lineTo(x, y) indicates where to draw the line to. In this case, the end point for your line is the lower right corner of the rectangle.

__12. In a similar manner, add the following code to draw the second diagonal:

```
// draw second diagonal
ctx.moveTo(250, 50);
ctx.lineTo(50, 200);
```

__13. Add the following code to render the two lines you drew in red ink to the drawOnCanvas function:

```
// draw diagonals in red
ctx.strokeStyle = "#FF0000";
ctx.stroke();
```

Technically, when you draw lines in HTML5, they're not displayed by default. You need to tell the context object to render them using its stroke method. One analogy that can be made is that you're drawing with a pen which is out of ink. When you call the **stroke** method, you refill the pen with ink and retrace the lines (in this case, red ink, since the **strokeStyle** property was set to red).

Note: Technically, the above rule applies to all *paths* (complex objects). A *path* is a set of subpaths. A *subpath* is a set of one or more points connected by a straight line or a curved line. When you drew the two diagonal lines, you created a path consisting of two subpaths.

__14. Save the file.

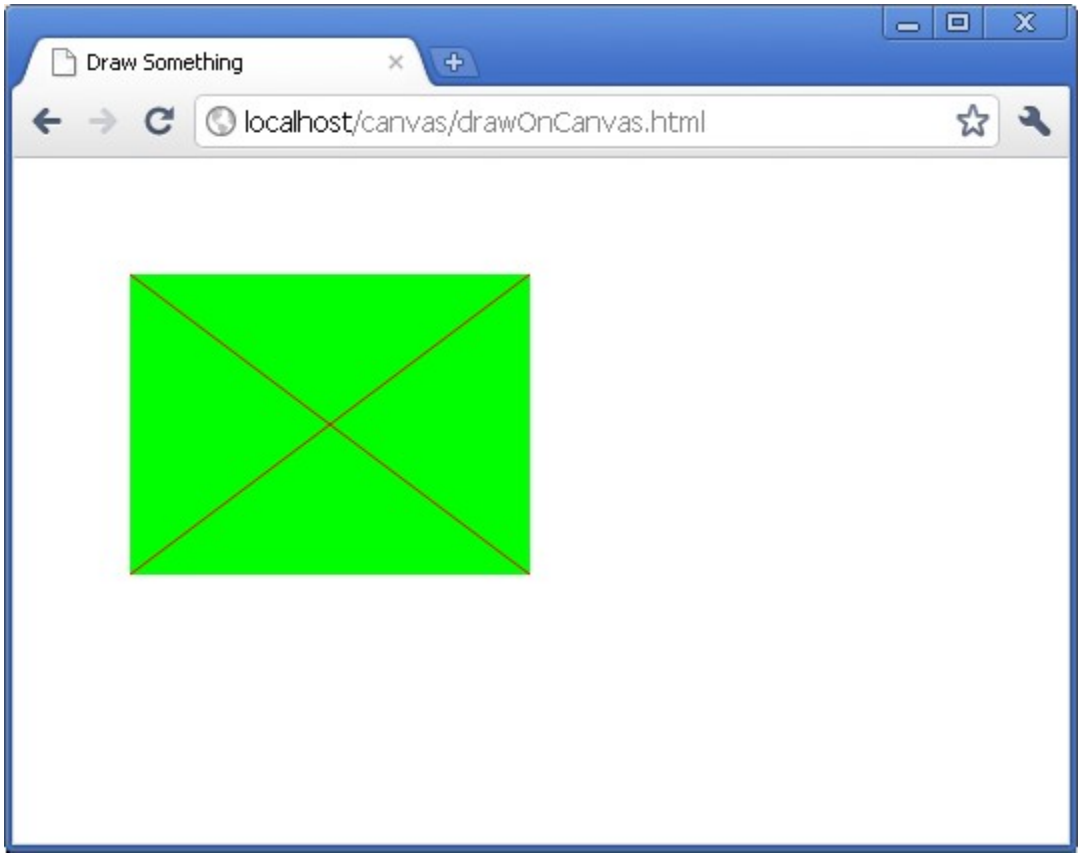
Part 2 - Test

__1. Open Google Chrome.

__2. Navigate to the following URL:

```
http://localhost/canvas/drawOnCanvas.html
```

__3. You should see the following screen:



As expected, the canvas contains the drawing you created in the previous part. However, notice that you can't see the border of the canvas.

__4. Open **drawOnCanvas.html**.

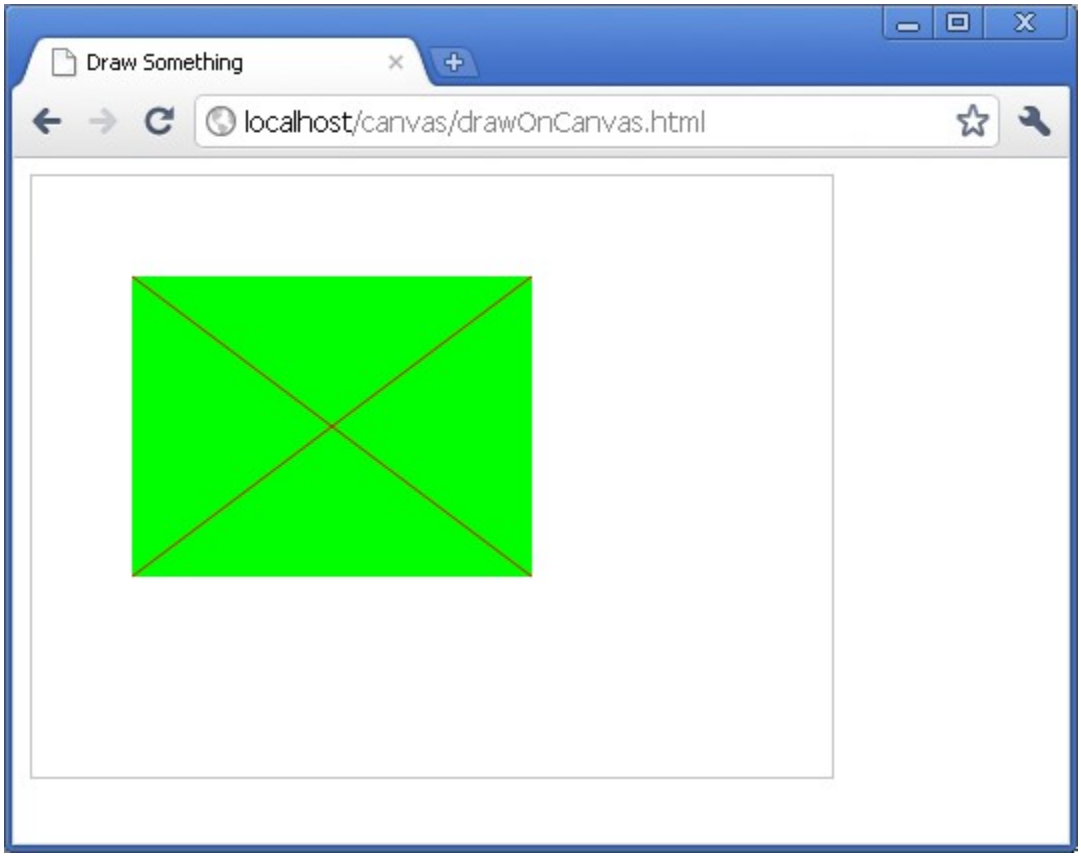
__5. To see the border of the canvas, add the following style attribute to the canvas element:

```
<canvas id="c" width="400" height="300"
  style="border: 1px solid #c3c3c3;">
```

__6. Save the file.

__7. Refresh the page in the browser.

You should now see the edges of the canvas.



Make sure that when you draw inside the canvas, you take into account the size of the canvas. Let's see what happens if you don't.

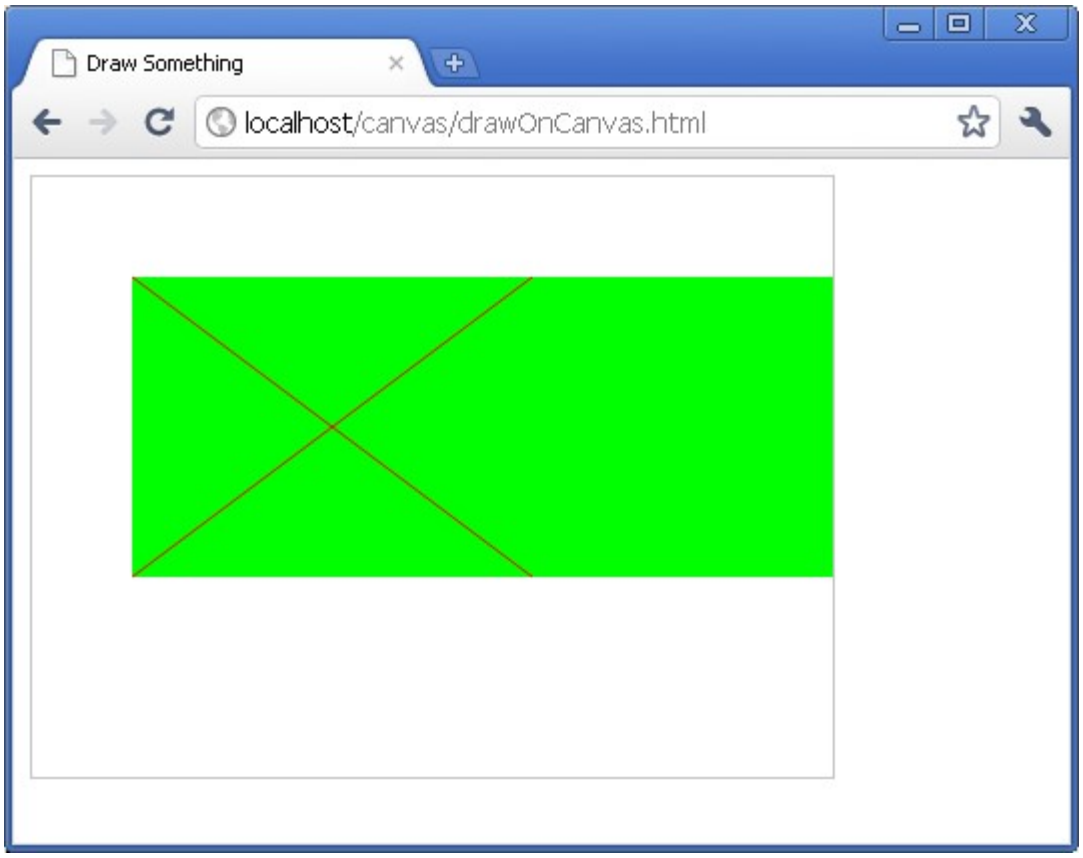
__8. Try changing the size of the rectangle to a value that exceeds the width of the canvas:

```
ctx.fillRect(50, 50, 500, 150);
```

__9. Save your changes.

__10. Refresh the page in the browser.

Notice that the rectangle is clipped. Obviously, this isn't what we wanted.



__11. Change the rectangle width back to its original size and save your changes.

__12. Test again.

__13. Close the browser and the editor.

Part 3 - Review

In this lab, you learned how to use the canvas element to draw a filled rectangle and lines.

Lab 4 - Intro to CSS3

In this lab you will copy some files used in the labs and see some basics about how CSS is used to define styles of web pages.

Part 1 - Copy Files

The lab setup has already installed a web server so the files you edit in labs can be viewed like they would be viewed as they would as part of a web site instead of as local files. This is because the browser may display things slightly differently if the files were not downloaded from a web server.

In order to accelerate the labs somewhat, you will copy a folder of files to the location the web server can find them so you can directly edit these files in the labs.

__1. Open a Windows Explorer file browser window.

__2. Go the 'C:\LabFiles' folder and copy the entire '**css3**' folder. Make sure you copy the correct one as there may be another folder with a similar name.

__3. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder although it could also be under the 'Program Files' folders somewhere.

> **Note:** If your Apache server was installed under the 'Program Files' folder structure somewhere, the setup for the labs may not have been followed correctly. Since security on some versions of Windows may not allow you to modify files in the 'Program Files' folders, which you will need to do for the labs, inform your instructor of this to see if this could create a problem.

__4. Paste the entire '**css3**' folder to create the folder '**C:\Apache2.2\htdocs\css3**'.

> **Note:** Pasting the entire folder will copy files for this and other labs. Other labs will likely have you check for the presence of files used in the lab just to make sure.

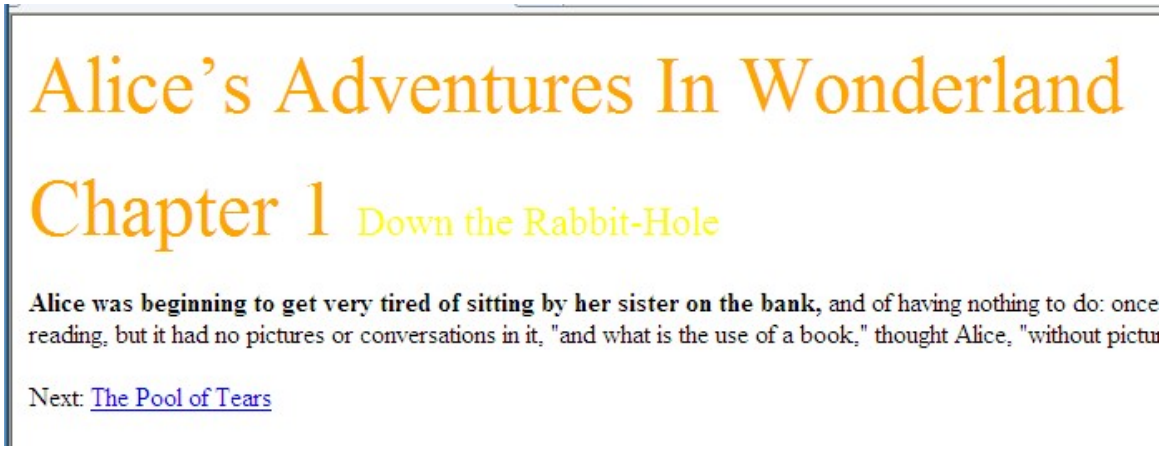
Part 2 - Observe Style Changes

Now that you have copied the required files to the web server, you can modify some styles in a page and observe the effect.

__1. Open a web browser and go to the following URL:

```
http://localhost/css3/intro/intro.html
```


__2. Observe the appearance of the page. Take note of the colors, style and size of the text, etc. These will change somewhat when you alter the styles in the page.



__3. Leave the browser open to the page as you will refresh it later.

__4. Run the following program to run a specialized editor. You do not need to use anything but a regular text editor for the labs but this program has some tools that can make it easier to work with the web pages you will be editing.

```
C:\Software\NotepadPlus\notepad++.exe
```

__5. If you are asked to install plug-ins simply click **Cancel**.

__6. Once you have the **Notepad**++ program running (or another text editor if you prefer), open the following file:

```
C:\Apache2.2\htdocs\css3\intro\intro.html
```

__7. Observe how currently there are some <style> definitions near the top of the page that are modifying how the browser displays the page.

```
<meta charset="UTF-8">
<title>Alice&#8217;s Adventures in Wonderland</title>
<style type="text/css" media="all">
    h1 {
        color: orange;
        font-size: 3em;
        font-weight: normal; }
    h1 .chaptertitle {
            color: yellow;
            font-size: .5em; }
</style>
</head>
```


__8. Modify the '**color**' properties of the two existing styles as shown in bold below.

```
<style type="text/css" media="all">
          h1 {
                    color: red;
                    font-size: 3em;
                    font-weight: normal; }
          h1 .chaptertitle {
                              color: green;
                              font-size: .5em; }
</style>
```

__9. Save the file but leave it open.

__10. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

http://localhost/css3/intro/intro.html

__11. Observe how the color of the text at the top of the page changes colors with the change to the style.

Note: If for some reason you do not see a change, it could be because the browser is caching the page. Use the tools of the browser to clear the cache, close all open browser windows, open a new window and try the URL again. You should try this first anytime you don't see a change you expect to see if the browser cache is not letting you see changes you have made to the pages.

__12. Return to the editor that was editing the page and modify the '**font-weight**' property as shown in bold below.

```
<style type="text/css" media="all">
          h1 {
                    color: red;
                    font-size: 3em;
                    font-weight: bold; }
          h1 .chaptertitle {
                              color: green;
                              font-size: .5em; }
</style>
```

__13. Save the file but leave it open.

__14. Return to the browser showing the page and refresh the page again. Notice how the text that is currently red and the text that is green both become bold.

This is because the '**h1.chaptertitle**' style inherits the '**font-weight**' property of the '**h1**' style and doesn't override it like it does for the color and size.

__15. Close all open files and browsers.

Part 3 - Review

In this lab you just saw some basics about how CSS3 styles modify the display of web pages. You also copied some files and used tools that will be used in other labs to explore CSS3 styles in more depth.

Lab 5 - Applying Basic Styles

In this lab you will see some of the various ways to define styles and link style definitions to elements of a web page. This will include inline styles, style classes, and external stylesheets.

Part 1 - Verify Files

In a previous lab you should have copied several files to a location where the Apache web server can locate them and return them as web pages. This section will verify you have the correct files.

__1. Open a Windows Explorer file browser window.

__2. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder.

__3. Verify that the following sub folder has been copied to where the Apache server is installed. There will be various files within this subfolder.

```
C:\Apache2.2\htdocs\css3\intro
```

Note: If you do not see this folder, you can copy this from the 'C:\LabFiles\css3' folder or extract the appropriate lab solution for the previous lab from the 'C:\LabFiles\Solutions' folder. Be careful which files you copy and where you place them to make sure the web server can locate them with the URLs provided in the labs.

Part 2 - Inline Style

One way to define a style is with the 'style' property of the element you want to modify.

__1. Open a web browser and go to the following URL:

```
http://localhost/css3/intro/intro.html
```


__2. Observe the appearance of the page. Take note of the colors, style and size of the text, etc. These will change somewhat when you alter the styles in the page.

__3. Leave the browser open to the page as you will refresh it later.

__4. Run the Notepad++ program or a regular text editor.

C:\Software\NotepadPlus\notepad++.exe

Note: A file may appear since Notepad++ opens the last used file. To avoid this in future, close the file first and then close Notepad++.

__5. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file:

C:\Apache2.2\htdocs\css3\intro\intro.html

__6. Find the first '<h1>' tag and add the 'style' property as shown in bold below. Make sure you have proper syntax including the quotes and the colon and semi-colon in the style rule.

```
<body>
<h1 style="color:orange;">Alice&#8217;s Adventures In Wonderland</h1>
<h1 id="ch01">Chapter 1 Down the Rabbit-Hole</h1>
```

__7. Save the file but leave it open.

__8. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

http://localhost/css3/intro/intro.html

__9. Observe how the color of the text at the top of the page changes colors with the change to the style.

Note: If for some reason you do not see a change, it could be because the browser is caching the page. Use the tools of the browser to clear the cache, close all open browser windows, open a new window and try the URL again. You should try this first anytime you don't see a change you expect to see if the browser cache is not letting you see changes you have made to the pages.

Part 3 - Embedded Style

Although defining a style inline can be clear how the element is being modified, it is not very reusable, especially if you want multiple elements to have the style applied. It may be better to define an embedded style in the <style> tag of the page to be able to apply the style rule to all of the matching elements of the page.

__1. Return to the editor that was editing the page. Modify the existing '<style>' tag by adding a new style rule for the paragraph (<p>) element as shown in bold below.

```
<style type="text/css" media="all">
          h1 {
                    color: red;
                    font-size: 3em;
                    font-weight: bold; }
          h1 .chaptertitle {
                              color: green;
                              font-size: .5em; }
          p {
                    color: rgb(153,153,153);
                    font-family: perpetua, georgia, serif;
                    line-height: 1.5;
                    text-indent: 3em; }
</style>
</head>
```

__2. Save the file but leave it open.

__3. Return to the browser showing the page and refresh the page again. Notice that now the paragraph text is a different color, has a little more space between lines and is indented.

Part 4 - Style Class

Linking a style definition to a specific type of element may not give the flexibility you need. You can define a style "class" to name a style and link it to various elements with the name of the style class.

__1. Return to the editor that was editing the page. Modify the existing '<style>' tag by adding a new '**chaptertitle**' style class definition as shown in bold below. Make sure to include the '**.**' selector before the name of the class so this can be applied to any element in the page.

```
            p {
                        color: rgb(153,153,153);
                        font-family: perpetua, georgia, serif;
                        line-height: 1.5;
                        text-indent: 3em; }
            .chaptertitle {
                        font-family: perpetua, georgia, serif;
                        font-variant: small-caps;
                        letter-spacing: .1em; }
</style>
</head>
```

__2. Scroll down to the second <h1> element and notice that part of it already has a tag that applies the 'chaptertitle' style. This was so another lab could show two different colors by using the style class. You do not have to modify anything here.

```
<h1 style="color:orange;">Alice&#8217;s Adventures In Wonderland</h1>
<h1 id="ch01">Chapter 1 <span class="chaptertitle">Down the Rabbit-
Hole</span></h1>
<article>
```


__3. Find the link anchor tag (<a>) near the bottom of the page and add the '**class**' property with the name of the new class as shown in bold below.

```
</article>
<footer><nav> Next:
<a class="chaptertitle" href="AAIWL-ch02.html">The Pool of Tears</a>
</nav></footer>
```

__4. Save the file but leave it open.

__5. Return to the browser showing the page and refresh the page again. Notice how the text of the chapter title at the top and on the link at the bottom both change. Probably the easiest way to detect they have changed is they are both in small capital letters now.

__6. Return to the editor that was editing the page. Modify the existing '<style>' tag by adding a new '**a.chaptertitle**' style class definition as shown in bold below. This is a "dependent" class that can only be applied to the anchor (<a>) element.

```
            .chaptertitle {
                        font-family: perpetua, georgia, serif;
                        font-variant: small-caps;
                        letter-spacing: .1em; }
            a.chaptertitle {
                        font-size: 2em;
                        font-style: italic; }
</style>
</head>
```

__7. Save the file but leave it open.

__8. Return to the browser showing the page and refresh the page again. Notice that nothing else on the page changes but the size and style of the link text changes. The dependent class only applied to this element.

Part 5 - Link to Stylesheet

One of the most useful ways to define reusable styles is to define them in external "stylesheets" (.css files) and then link to them on pages. Some simple stylesheets have already been given to you and you will link to them from the page.

__1. Return to the editor that was editing the page. Before the existing <style> tag, add a new **<link>** tag as shown in bold below.

```
<title>Alice&#8217;s Adventures in Wonderland</title>
<link href="global.css" type="text/css" rel="stylesheet" media="all">
<style type="text/css" media="all">
```

__2. Save the file but leave it open.

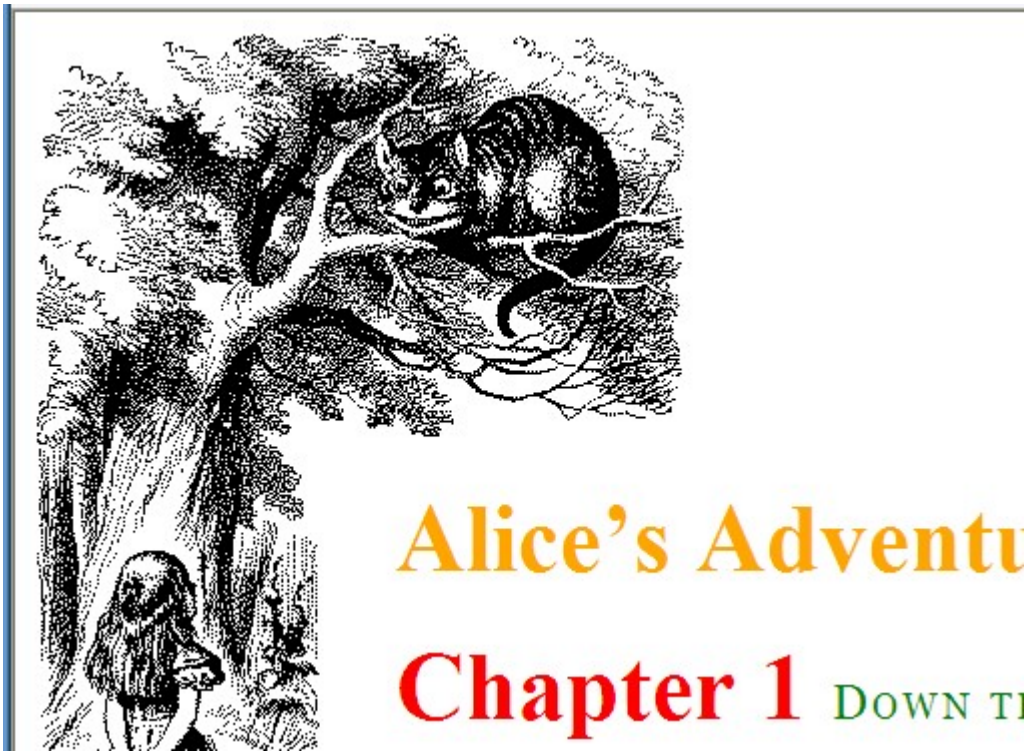
__3. Return to the browser showing the page and refresh the page again. Now the text should have quite a bit of padding on the top and left of the page.

__4. Return to the editor editing the page. Within the existing <style> tag add the following '@import' rule in bold. Make sure it is at the top of the content of the <style> tag.

```
<link href="global.css" type="text/css" rel="stylesheet" media="all">
<style type="text/css" media="all">
@import url('ch01.css');
          h1 {
                    color: red;
                    font-size: 3em;
                    font-weight: bold; }
```

__5. Save the file but leave it open.

__6. Return to the browser showing the page and refresh the page again. Now you should have an image that fills in some of the empty space that was created.



__7. If you want you can open the following files and look at some of the style rules that have made the changes you saw.

```
C:\Apache2.2\htdocs\css3\intro\global.css
C:\Apache2.2\htdocs\css3\intro\ch01.css
```

__8. Close the files when you finished reviewing the files.

Part 6 - Pseudo-Classes

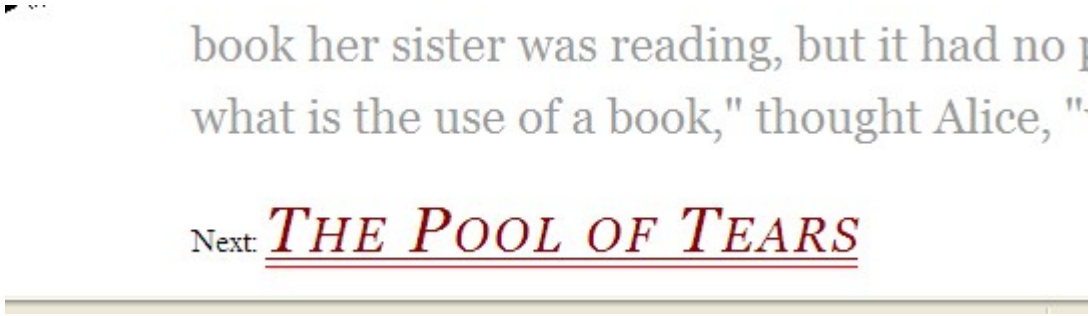
One new feature of CSS3 is the use of "pseudo-classes". These are special states of certain HTML elements that can be styled independently of the other states of the element. One of the best examples of this are the various states of a link.

__1. Return to the editor editing the page. Within the existing <style> tag add the following pseudo-class definitions in bold. Notice that they contain a colon (':') and not a period because the link state the style applies to comes after the colon.

```
            a.chaptertitle {
                        font-size: 2em;
                        font-style: italic; }
            a:link {
                        color: darkred;
                        border-bottom: 1px solid red; }
            a:visited {
                        color: darkred;
                        border-bottom: 1px dashed red;}
            a:hover {
                        color: red;
                        border-bottom: 1px solid pink;}
            a:active {
                        color: pink;
                        border-bottom: 1px solid pink;}
</style>
</head>
```

__2. Save the file but leave it open.

__3. Return to the browser showing the page and refresh the page again. Notice the style of the link has changed as as you hover over it or click it the color changes. The file the link is supposed to point to doesn't exist so you will get an error if you follow the link but you should still see the various changes as the mouse interacts with the link.



__4. Close all open browsers.

__5. If you are using Notepad++ then close all open files and then close Notepad++.

Part 7 - Review

In this lab you saw many different ways to define styles that apply to HTML elements. You could define them inline directly in the element, embedded within the <style> tag on the page, or link to external stylesheets. You also saw some aspects of “pseudo-classes” that are styles for various states of HTML elements.

Lab 6 - Modify Text Styles

In this lab you will modify various text properties with font styles. Since text can be a large part of the content of a page, these styles can have a big impact on the overall style of a page.

Part 1 - Verify Files

In a previous lab you should have copied several files to a location where the Apache web server can locate them and return them as web pages. This section will verify you have the correct files.

__1. Open a Windows Explorer file browser window.

__2. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder.

__3. Verify that the following sub folder has been copied to where the Apache server is installed. There will be various files within this subfolder.

```
C:\Apache2.2\htdocs\css3\text
```

Note: If you do not see this folder, you can copy this from the 'C:\LabFiles\css3' folder or extract the appropriate lab solution for the previous lab from the 'C:\LabFiles\Solutions' folder. Be careful which files you copy and where you place them to make sure the web server can locate them with the URLs provided in the labs.

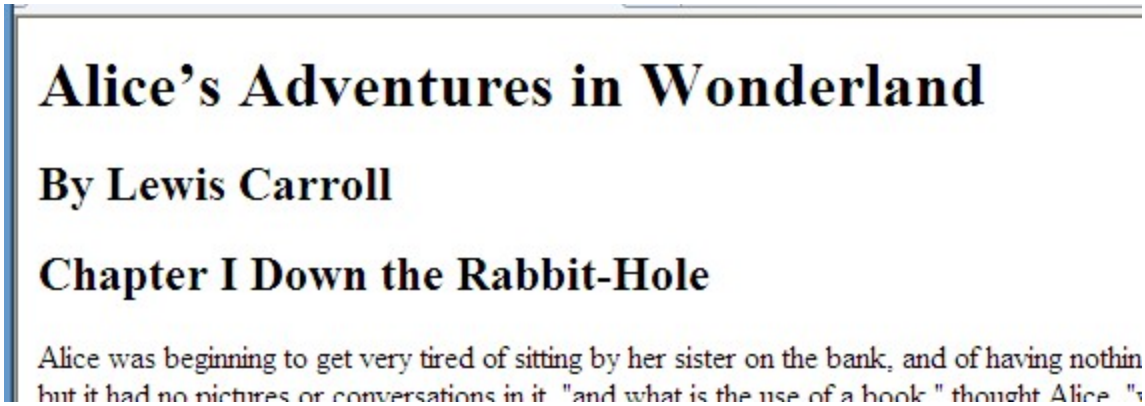
Part 2 - Define Font Family

One of the most important text styles is the font used to display text. With web pages you can define a list of different fonts you want the browser to use with the 'font-family' style property.

__1. Open a web browser and go to the following URL:

```
http://localhost/css3/text/index.html
```


__2. Observe that right now the text does not really have much style applied to it but this will change as various style properties are added.



__3. Leave the browser open to the page as you will refresh it later.

__4. Run the Notepad++ program or a regular text editor.

```
C:\Software\NotepadPlus\notepad++.exe
```

Note: A file may appear since Notepad++ opens the last used file. To avoid this in future, close the file first and then close Notepad++.

__5. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file:

```
C:\Apache2.2\htdocs\css3\text\index.html
```

__6. Notice that the page already has a link to a stylesheet of the name 'font-properties.css'. Rather than defining styles directly in the page we will edit this file.

```
<title>Alice&#8217;s Adventures In Wonderland | Chapter I</title>
<link href="font-properties.css" type="text/css" rel="stylesheet"
media="all">
```

__7. Close the 'index.html' file.

__8. Open the following file with the text editor:

```
C:\Apache2.2\htdocs\css3\text\font-properties.css
```


__9. Add the 'font-family' property to the 'body' style as shown in bold below. Make sure the entire name of the 'Times New Roman' font is in quotes.

```
body {
            font-family: Constantia, Georgia, "Times New Roman", serif;
}
```

__10. Save the file but leave it open.

__11. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

http://localhost/css3/text/index.html

__12. You may not see a big change but there should be some subtle differences in the font. This may be because your browser is now using one of the fonts in the 'font family' list instead of the default font for the browser.

Note: Often 'Times New Roman' is a default font of the browser so unless our 'font-family' property had a different font available to the browser before this you might not see any change.

__13. Return to the editor that was editing the 'font-properties.css' file. Modify the other style rules that are currently empty to add the 'font-family' property shown in bold below.

```
h1, h2, .author {
            font-family: Corbel, Helvetica, Arial, sans-serif;
}
h2.chaptertitle {
            font-family: Corbel, Helvetica, Arial, sans-serif;
}
```


__14. Save the file but leave it open.

__15. Return to the browser that was showing the page and refresh the page.

__16. Observe that the font styles used by the various headers at the top of the page change as they are now using some type of “sans-serif” font.

Part 3 - Use Web Fonts

With CSS3, you are no longer restricted to depending on the browser to support one of the fonts named with the 'font-family' property. You can define a “Web Font” rule which allows the browser to download the font directly from the server. This can help make the styles of web pages more uniform across multiple browser types.

__1. Open the following file in the editor:

C:\Apache2.2\htdocs\css3\text\index.html

__2. Modify the name of the file linked to as the stylesheet as shown in bold below. Since web fonts require quite long style rules we will provide much of it for you to avoid typos.

```
<title>Alice&#8217;s Adventures In Wonderland | Chapter I</title>
<link href="font-properties-webfont.css" type="text/css"
rel="stylesheet" media="all">
```

__3. Save the 'index.html' file.

__4. Open the following file with the text editor:

C:\Apache2.2\htdocs\css3\text\font-properties-webfont.css

__5. Observe the various '@font-face' rules, one example of which is shown below. The most important property is the 'font-family' property which can be a name of your choosing. We will see this show up later. Also notice that there are various 'url(...)' functions linking to files that were copied to the web server. There are several different URLs because different browsers (and versions) may use different file formats for web fonts.

```
@font-face {
    font-family: 'BodyCopy';
    src: url('fonts/DVS/DejaVuSerif.eot');
    src: url('fonts/DVS/DejaVuSerif.eot?#iefix')
          format('embedded-opentype'),
         url('fonts/DVS/DejaVuSerif.woff') format('woff'),
         url('fonts/DVS/DejaVuSerif.ttf') format('truetype');
    font-weight: normal;
    font-style: normal; }
```

Note: The “Font Squirrel” web site, has useful tools for downloading free web fonts and provides sample stylesheet code for the '@font-face' rules that can be used to link to those fonts. That sample code would be similar to that shown above with the only difference being the relative URL to access the font files where you decide to place them on the server.

http://www.fontsquirrel.com/

__6. Scroll to the bottom of the 'font-properties-webfont.css' file and find the various styles that had previously been defined in the other stylesheet file. Currently they are the same as they were in that file.

__7. To use the web fonts we must reference the name given to it in the '@font-face' rules so add the following bold entries to the values of the various 'font-family' style properties. Make sure to also add the comma between the new font name and the fonts that were already present. Also double check that the font name you add matches exactly the 'font-family' property defined with the various '@font-face' rules.

```
body {
          font-family: 'BodyCopy', Constantia, Georgia,
                    'Times New Roman', serif;
}
h1, h2, .author  {
          font-family: 'TitleCopy', Corbel, Helvetica, Arial,
                    sans-serif;
}
h2 .chaptertitle {
          font-family: 'TitleCopy', Corbel, Helvetica, Arial,
                    sans-serif;
}
```


Note: When using web fonts it is common to list the name given to the web font by the '@font-face' rules as the first font referenced in a 'font-family' property. If other fonts are listed first, the browser may end up using a local font instead of the web font which might defeat the purpose of the uniformity across browsers which is part of the appeal of web fonts.

__8. Save the file but leave it open.

__9. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

http://localhost/css3/text/index.html

__10. You should see that the fonts of the body and the headers change quite drastically as the web font is picked up instead of the “standard” fonts used before.

Part 4 - Apply Font Effects

Besides just controlling the font family, there are various font effects that can be applied.

__1. Return to the editor for the 'font-properties-webfont.css' file. Add the following new style rule in bold to display the '<h1>' elements in italic text.

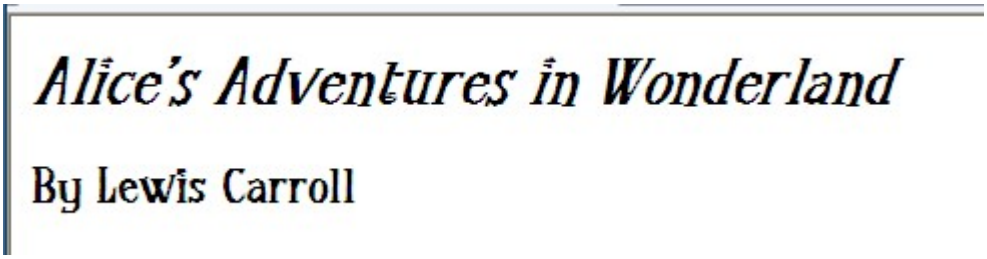
```
h1, h2, .author  {
          font-family: 'TitleCopy', Corbel, Helvetica, Arial,
                    sans-serif;
}
h1 {
          font-style: italic;
}
```


__2. Save the file but leave it open.

__3. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

```
http://localhost/css3/text/index.html
```

__4. You should see the first line of text, the only thing to use the <h1> element, is now italic.



Note: It is important that because this is a web font that the definition of the related '@font-face' rule did NOT indicate that the web font only applied for “normal” text. Notice that the 'TitleCopy' @font-face does not have the 'font-style' property while the 'BodyCopy' @font-face does. Depending on the web font you might need different files to download for different font styles so this depends somewhat on the exact web font being used.

In this example the 'LTG' font used for the 'TitleCopy' font family has one set of files for all styles (bold, italic, bold-italic) of the font. So for this web font we do not need to define the 'font-weight' or 'font-style' properties in the rule because it can be used for all.

In the example of the 'DVS' font used for 'BodyCopy', there are different files for each style of font. So for these 4 'font-face' rules, the 'font-weight' and 'font-style' properties are used so that depending on the style used within the page the appropriate rule will be matched. In this case the value of the 'font-family' property for all 4 is set to 'BodyCopy'. This just makes it so that you do not need to change the name of the font to switch to a different style. We could have also used different names like 'BodyCopyBold', 'BodyCopyItalic', etc but this would have required you to switch to the different name of the font when you wanted to change the style. Either way would be valid and it is a preference which approach to take.

__5. Return to the editor for the 'font-properties-webfont.css' file. Add the following new style rule in bold to display the 'author' style elements in all uppercase.

```
h1, h2, .author  {
          font-family: 'TitleCopy', Corbel, Helvetica, Arial,
                    sans-serif;
}
.author {
          text-transform: uppercase;
}
```

Note: The 'author' style is already applied in the page but hadn't yet had any unique style properties defined.

__6. Save the file but leave it open.

__7. Return to the browser that was showing the page and refresh the page.

__8. You should see a change to the style of the authors name.

Note: It seems like the web font being used for the title text does not really have support for bold. If we were using a “standard” font for the header elements we could add properties like 'font-weight: bold;' to use bold text.

Part 5 - Modify Text Alignment

Controlling how the text is aligned can be another important style property.

__1. Return to the editor for the 'font-properties-webfont.css' file. Modify the existing 'body' style rule to include the properties in bold below to justify text.

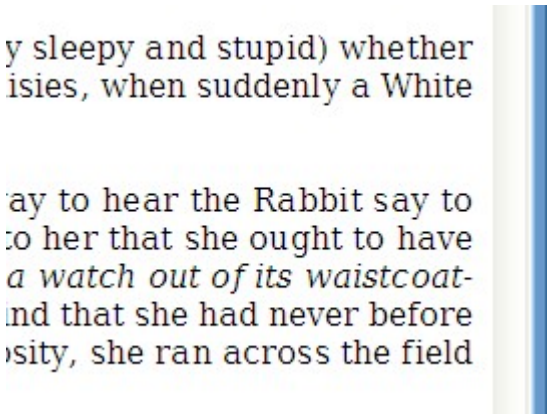
```
body {
            font-family: 'BodyCopy', Constantia, Georgia,
                        'Times New Roman', serif;
            text-align: justify;
            text-justify: distribute;
}
```

Note: The most important property is the 'text-align' property as the manner in which the justification is done (controlled by the 'text-justify' property) may be difficult to notice differences.

__2. Save the file but leave it open.

__3. Return to the browser that was showing the page and refresh the page.

__4. You should see body text in the main content of the page is now justified instead of left-aligned.



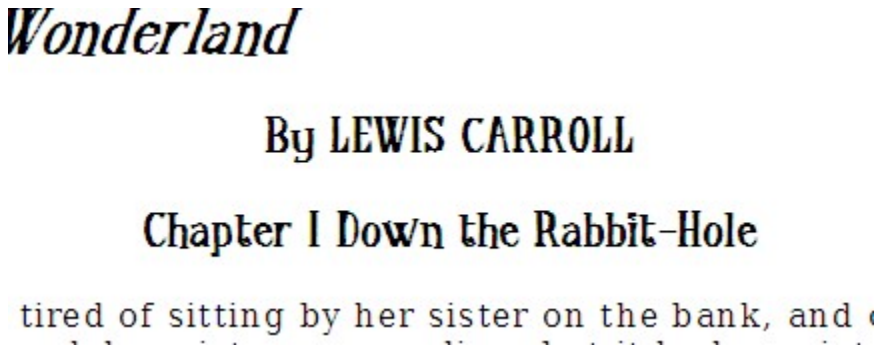
__5. Return to the editor for the 'font-properties-webfont.css' file. Add the following new style rule in bold to center <h2> elements.

```
h1, h2, .author  {
            font-family: 'TitleCopy', Corbel, Helvetica, Arial,
                        sans-serif;
}
h2 {
            text-align: center;
}
```


__6. Save the file but leave it open.

__7. Return to the browser that was showing the page and refresh the page.

__8. You should see the <h2> header text centered on the page.



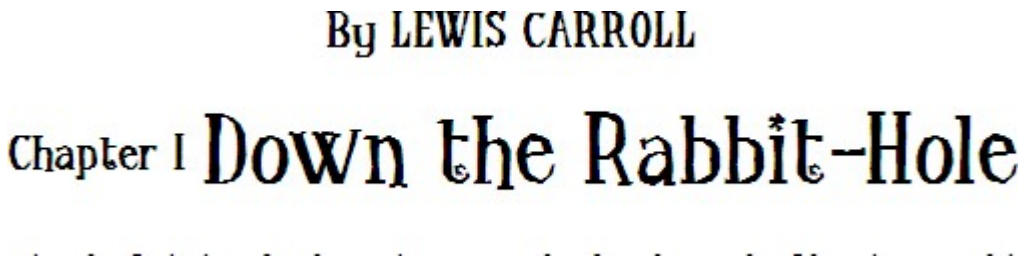
__9. Return to the editor for the 'font-properties-webfont.css' file. Modify the existing 'h2.chaptertitle' style rule to include the properties in bold below to adjust the size and vertical alignment.

```
h2 .chaptertitle {
          font-family: 'TitleCopy', Corbel, Helvetica, Arial,
                    sans-serif;
          vertical-align: middle;
          font-size: 200%;
}
```

__10. Save the file but leave it open.

__11. Return to the browser that was showing the page and refresh the page.

__12. The chapter title is now bigger but middle aligned with the chapter number.



__13. Close all open browsers.

__14. If you are using Notepad++ then close all open files and then close Notepad++.

Part 6 - Review

There are many different style properties that can affect display of text. This starts with what font is used but also includes other properties like style, weight, size, and alignment. Web Fonts are also new with CSS3 and can allow you to more closely control the style across browsers and platforms by allowing the browser to download font format files instead of relying on linking to fonts already local to the browser and machine.

Lab 7 - Control Element Spacing with Box Properties

The “Box” around an element controls how elements are spaced or grouped with other elements on the page. In this lab you will control some of the common box style properties.

Part 1 - Verify Files

In a previous lab you should have copied several files to a location where the Apache web server can locate them and return them as web pages. This section will verify you have the correct files.

__1. Open a Windows Explorer file browser window.

__2. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder.

__3. Verify that the following sub folder has been copied to where the Apache server is installed. There will be various files within this subfolder.

```
C:\Apache2.2\htdocs\css3\box
```

Note: If you do not see this folder, you can copy this from the 'C:\LabFiles\css3' folder or extract the appropriate lab solution for the previous lab from the 'C:\LabFiles\Solutions' folder. Be careful which files you copy and where you place them to make sure the web server can locate them with the URLs provided in the labs.

Part 2 - Control Height and Width

Some of the most common box style properties to control are the height and width of elements.

__1. Open a web browser and go to the following URL:

```
http://localhost/css3/box/index.html
```

Note: The page in this example uses some HTML5 elements that may not be recognized directly in older Internet Explorer browsers. This might cause the styles to not be displayed properly to some elements. There is a JavaScript trick included in this page that can enable HTML5 for some older Internet Explorer versions though.

If you do try testing with Internet Explorer and don't see the behavior the style should be creating try testing with Firefox or Chrome to see if it is different.

Screenshots in the lab will show how things should appear in Firefox.

__2. Get familiar with the current style of the page. In particular notice there are some images (HTML5 <figure> tags) that we will control text flow around in one part of the lab.

An unusually large saucepan flew close by it, and very nearly carried it off

"Which would *not* be an advantage," said Alice, who felt very glad to ge

think what work it would make with the day and night! You see the earth take

__3. Leave the browser open to the page as you will refresh it later.

__4. Run the Notepad++ program or a regular text editor.

```
C:\Software\NotepadPlus\notepad++.exe
```

Note: A file may appear since Notepad++ opens the last used file. To avoid this in future, close the file first and then close Notepad++.

__5. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file. This is a stylesheet currently linked from the index.html file (along with some other stylesheets for font and text styles).

```
C:\Apache2.2\htdocs\css3\box\box-properties.css
```

__6. Add the following width and height properties in bold to the currently empty 'header' style.

```
header {
            width: 100%;
            height: 135px;
}
```

__7. Save the file but leave it open.

__8. Return to the browser that was showing the page, make sure you are viewing the top headers of the page, and refresh the page. Open a new one to the following URL if you closed it:

```
http://localhost/css3/box/index.html
```

__9. You should see that the chapter number and title are a little closer vertically than they were to the author name on the right. If you were not viewing the top of the page before you did the refresh this might be hard to detect.

__10. If your browser window is currently maximized, used the 'restore' button so you can resize the browser window.

__11. Drag the width of the browser window to make it much narrower and see if the various elements in the header start overlapping badly with each other. This is because the word wrapping of the header is now causing it to want to use more height than we allowed and it is overlapping the next element. You might also see that the chapter title is wrapping as well.

__12. Return to the editor that was editing the 'box-properties.css' file. Add the following properties in bold to the currently empty 'h1' and 'article' styles.

```
h1 {
            width: 95%;
            max-width: 980px;
            min-width: 760px;
}

article {
            max-width: 980px;
            min-width: 660px;
}
```

__13. Save the file but leave it open.

__14. Return to the browser that was showing the page and don't change the width of the browser window but refresh the page.

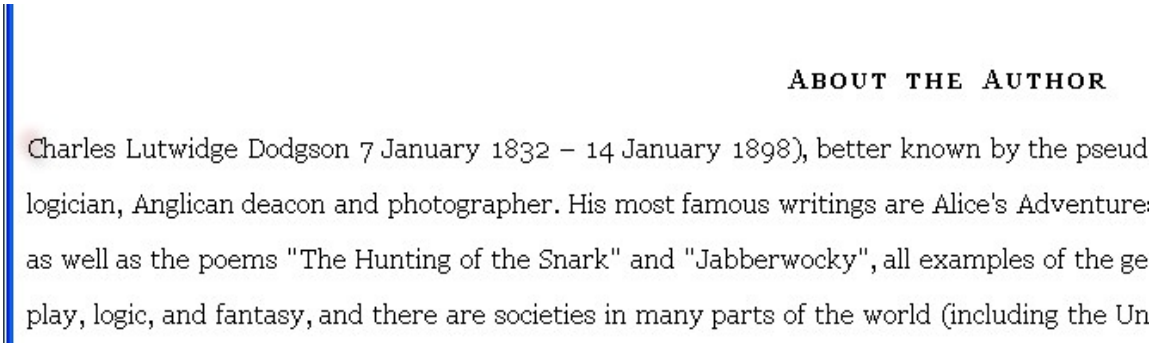
__15. You should see that the various text at the top of the page doesn't do as much word wrapping because of the 'min-width' properties in particular so things likely will not overlap anymore.

Note: Setting height and width in a stylesheet is often only done for structural elements like headers and body. Height and width of images especially is often done inline within the HTML element for the image.

__16. Gradually expand the width of the browser window and you should see the chapter title and number move right a little once the window is wider than the minimum width of the <article> content. Depending on your screen size you may also see that eventually the content starts leaving more empty space on the right because you have hit the maximum width value.

Note: Depending on how narrow you had the window to start with and your screen size the behavior here may be slightly different. The key thing is that the text does not wrap as horribly as before you set a minimum width.

__17. Scroll to the very bottom of the page and notice there is an 'About the Author' section which is an HTML5 <aside> element.



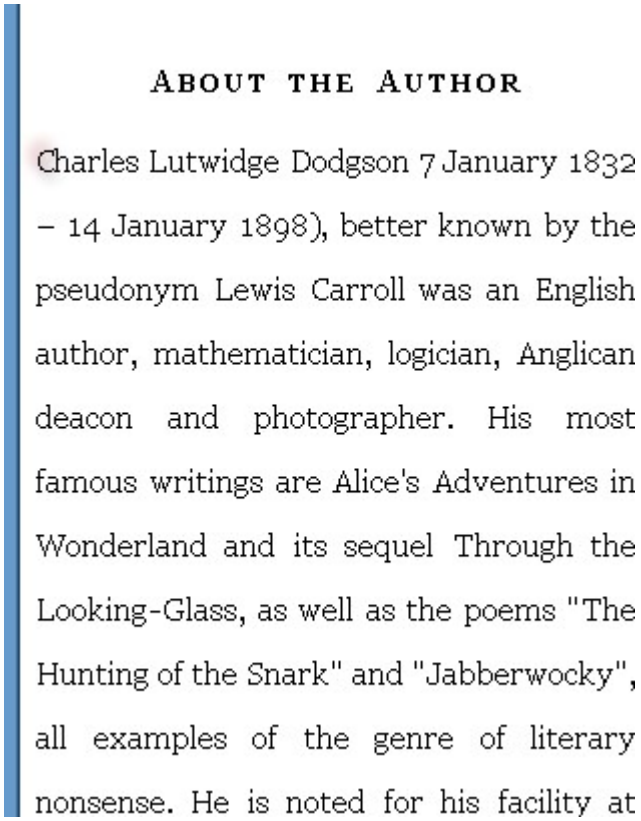
__18. Return to the editor that was editing the 'box-properties.css' file. Add the following properties in bold to the currently empty 'aside' style.

```
aside {
            width: 300px;
            height: 400px;
}
```

__19. Save the file but leave it open.

__20. Return to the browser that was showing the page and refresh the page while still looking at the bottom of the page.

__21. You should now see that this section is narrower. It is hard to tell but the height should be taller than the value of the 'height' property.



ABOUT THE AUTHOR

Charles Lutwidge Dodgson 7 January 1832 – 14 January 1898), better known by the pseudonym Lewis Carroll was an English author, mathematician, logician, Anglican deacon and photographer. His most famous writings are Alice's Adventures in Wonderland and its sequel Through the Looking-Glass, as well as the poems "The Hunting of the Snark" and "Jabberwocky", all examples of the genre of literary nonsense. He is noted for his facility at

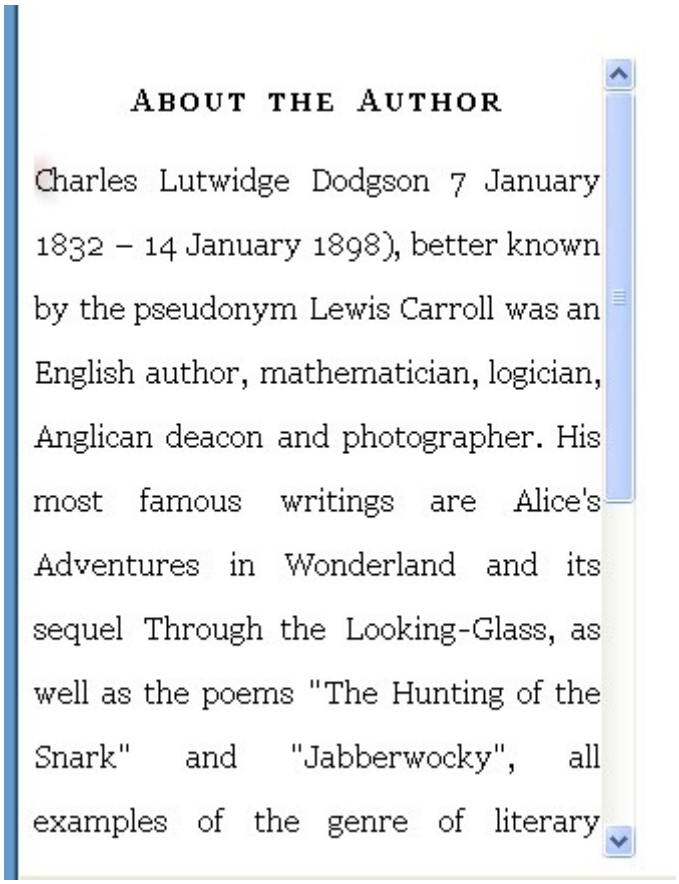
__22. Return to the editor that was editing the 'box-properties.css' file. Add the following new 'overflow' property in bold to the 'aside' style.

```
aside {
          width: 300px;
          height: 400px;
          overflow: auto;
}
```

__23. Save the file but leave it open.

__24. Return to the browser that was showing the page and refresh the page while still looking at the bottom of the page.

__25. You should now see that the 'height' style property is honored and a scrollbar is added to scroll through the text within the allowed space.



Note: The narrow width was used not really because it looks good but to show the effect of “overflow” content. Although you might think the default value of the 'overflow' property was 'auto' it is actually 'visible' which tells the browser to ignore the height if needed to display the entire content. There is also a 'scroll' value but this will often add scrollbars even if they aren't needed. A 'hidden' value will not display overflow content but provide no way to view it so 'visible' and 'auto' are the two most popular values of this property.

Part 3 - Text Flow

Right now the text of the document does not flow around the images. This section will change that.

__1. Return to the editor that was editing the 'box-properties.css' file.

__2. Add the following new style classes in bold to the stylesheet. Make sure they are not embedded within the brackets of another style.

```
.floatleft {
          float: left;
}

.floatright {
          float: right;
}
```

__3. Save the file but leave it open.

__4. Return to the browser that was showing the page and refresh the page.

__5. Find the images in the file and notice now that text flows around them. Before there was blank space next to the images and the text didn't pick up again until after the image. The images had already indicated they were using the 'floatleft' or 'floatright' style classes but without these classes being defined that was ignored. Now that you defined that class and used the 'float' property the text flows differently.

: be murder to leave it behind?" She said the last words out loud, and the little thi

Don't grunt," said Alice; "that's not at all a proper way of expressing yourself."

inxiously into its

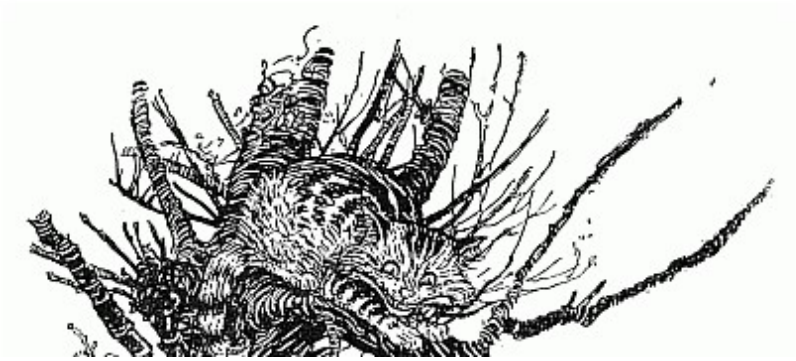
no doubt that it

a real nose; also

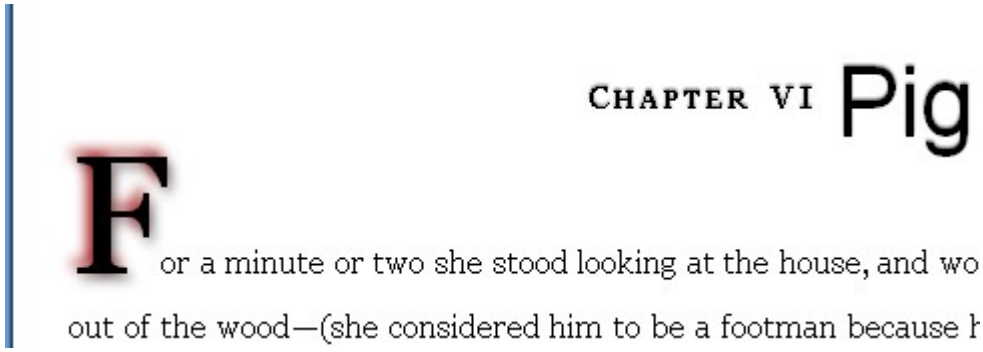
gether Alice did

s only sobbing,"

there were any



__6. Scroll to the top of the page and look at the first letter of the first paragraph. There is an existing style that makes the letter larger and with a shadow but notice it sticks up above the top of the line for the rest of the text. You may not see this in some Internet Explorer versions so use Firefox or Chrome if needed.



__7. Return to the editor that was editing the 'box-properties.css' file.

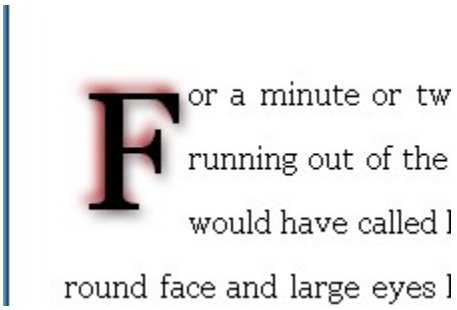
__8. Add the following new style in bold to the stylesheet. Be careful with the syntax of the name of the style as it must select the same element as the existing style (in text-properties.css) that creates the larger first letter.

```
h2 + p:first-letter {
            float: left;
            margin: .2em .1em .1em;
}
```

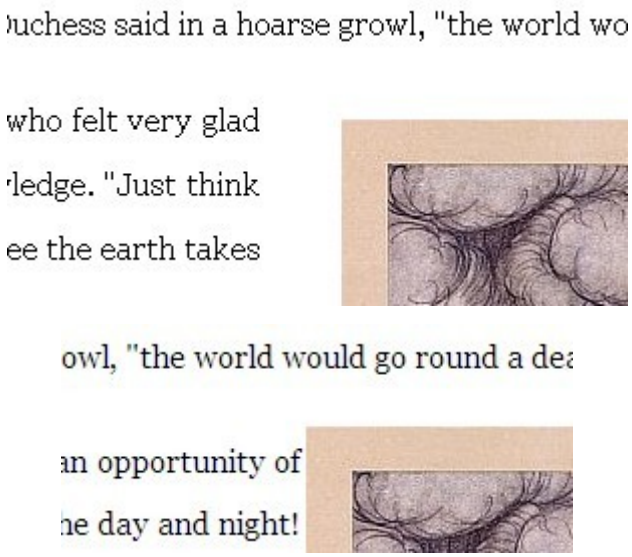
__9. Save the file but leave it open.

__10. Return to the browser that was showing the page and refresh the page.

__11. You should now see that the letter drops down with the first few lines of text flowing around it and a very small margin. Depending on the browser you may see that the letter is aligned in different way. Using three values for the 'margin' property we are setting top, left/right, and bottom.



__12. In the browser find one of the figures with the text that now flows around it. Depending on the browser you may see that the text is either close to or further away from the figure. Try the page in different browsers to see this.



__13. Return to the editor that was editing the 'box-properties.css' file.

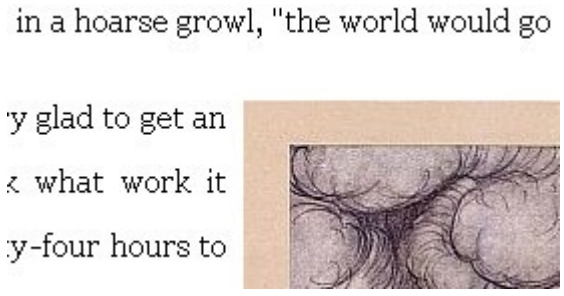
__14. Add the following new style in bold to the stylesheet.

```
figure {
          margin: .5em;
}
```

__15. Save the file but leave it open.

__16. Return to the browser that was showing the page and refresh the page.

__17. You should now see that the margin around the figure is more consistent between browsers.



Note: Depending on the browser and version you may not actually see much change with the rule above. Know that setting the margin yourself and not using the browser default will be a good thing and bring more consistency.

Part 4 - Border and Padding

Some of the last most popular box style properties to use involve borders around elements and padding.

__1. Return to the editor that was editing the 'box-properties.css' file.

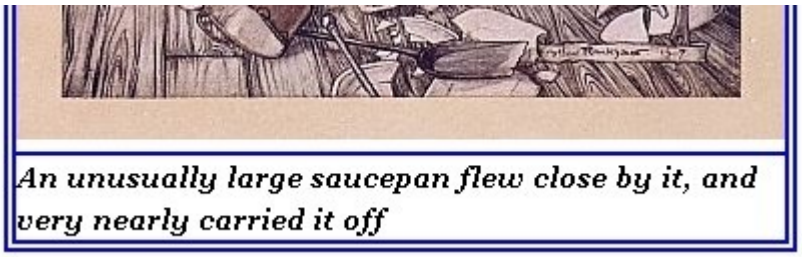
__2. Add a new property to the 'figure' style and a new 'figcaption' style as shown in bold below.

```
figure {
          margin: .5em;
          border: 6px double blue;
}

figcaption {
          border-top: 2px solid blue;
          display: block;
}
```

__3. Save the file but leave it open.

__4. Return to the browser that was showing the page and refresh the page. Find one of the images if you do not already see one.

__5. You should see some blue borders around the image and the caption.



__6. Return to the editor that was editing the 'box-properties.css' file.

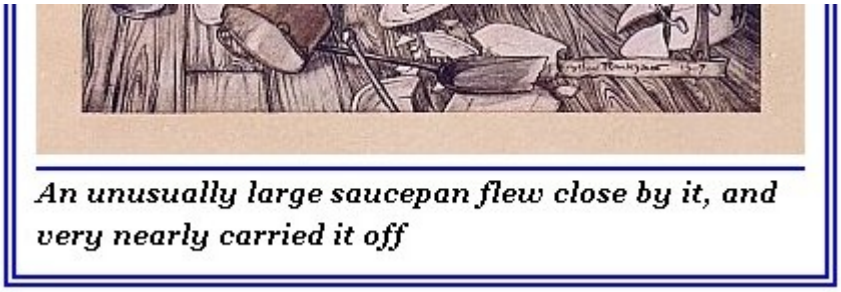
__7. Add the 'padding' property to the existing 'figure' class as shown in bold below.

```
figure {
          margin: .5em;
          border: 6px double blue;
          padding: 10px;
}
```


__8. Save the file but leave it open.

__9. Return to the browser that was showing the page and refresh the page.

__10. You should now see a little space between the inside of the element border and the items inside of it. This is different than the 'margin' property which is the space between the element border and the items outside of it.



An unusually large saucepan flew close by it, and very nearly carried it off

__11. Close all open browsers.

__12. If you are using Notepad++ then close all open files and then close Notepad++.

Part 5 - Review

There are various style properties that control how the size, placement, and spacing of an element relates to other elements on the page. Controlling width and height is common as well as text flow around elements, margins, borders, and padding within the border. Although the browser will often have default values of what some of these properties would be, setting them with your own styles can help with consistent display between browsers.

Lab 8 - Intro to JavaScript

In this lab you will copy some files used in the labs and add some basic JavaScript code to a page.

Part 1 - Copy Files

The lab setup has already installed a web server so the files you edit in labs can be viewed like they would be viewed as they would as part of a web site instead of as local files. This is because the browser may display things slightly differently if the files were not downloaded from a web server.

In order to accelerate the labs somewhat, you will copy a folder of files to the location the web server can find them so you can directly edit these files in the labs.

__1. Open a Windows Explorer file browser window.

__2. Go the 'C:\LabFiles' folder and copy the entire '**JavaScript**' folder. Make sure you copy the correct one as there may be another folder with a similar name.

__3. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder although it could also be under the 'Program Files' folders somewhere.

Note: If your Apache server was installed under the 'Program Files' folder structure somewhere, the setup for the labs may not have been followed correctly. Since security on some versions of Windows may not allow you to modify files in the 'Program Files' folders, which you will need to do for the labs, inform your instructor of this to see if this could create a problem.

__4. Paste the entire '**JavaScript**' folder to create the folder '**C:\Apache2.2\htdocs\JavaScript**'.

Note: Pasting the entire folder will copy files for this and other labs. Other labs will likely have you check for the presence of files used in the lab just to make sure.

Part 2 - Add Basic JavaScript Code

In this part you will add some basic JavaScript code to a page.

__1. Run the following program to run a specialized editor. You do not need to use anything but a regular text editor for the labs but this program has some tools that can make it easier to work with the web pages and JavaScript you will be editing.

```
C:\Software\NotepadPlus\notepad++.exe
```

__2. If you are asked to install plug-ins simply click **Cancel**.

__3. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file:

C:\Apache2.2\htdocs\JavaScript\intro\intro.html

__4. After the existing <noscript> tag but before the ending </body> tag, add the following bold code. This will use JavaScript to write out a simple message.

```
<body>
<noscript>
          <h2>This page requires JavaScript.</h2>
</noscript>
<h1>
          <script>
          document.write("Hello, world!");
          </script>
</h1>
</body>
</html>
```

__5. Save the file but leave it open.

__6. Open a web browser and go to the following URL:

http://localhost/JavaScript/intro/intro.html

__7. Observe that the JavaScript code has written out the message to the content of the page.

__8. Leave the browser open as you will refresh the same page in the next part.

Part 3 - Link to External Script File

Having the JavaScript code contained within a <script> tag within the page itself is just one way to define JavaScript code. You can also link the page to an external script file which can help with reusing the code.

__1. Return to the editor that was editing the 'intro.html' page.

__2. Add the following link to the external script file in bold in the <head> section of the page.

```
<html>
<head>
          <title>My first script</title>
          <script src="intro.js"></script>
</head>
<body>
```

__3. Save the file but leave it open.

__4. In the **Notepad++** program or another text editor open the following file:

```
C:\Apache2.2\htdocs\JavaScript\intro\intro.js
```

__5. Into the currently empty file add the following line.

```
alert("Hello, world from external script!");
```

__6. Save the file but leave it open.

__7. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

```
http://localhost/JavaScript/intro/intro.html
```

__8. Immediately after you reload the page you should see a dialog box with the message. The dialog itself may vary slightly depending on the browser but you should easily recognize the message.

__9. Click the **OK** button to close the dialog and you should still see the message printed by the code directly in the page.

__10. Close all open browsers.

__11. If you are using Notepad++ then close all open files and then close Notepad++.

Part 4 - Review

In this lab you saw that JavaScript code is added to a page by the <script> tag. This can either contain direct JavaScript code or link to an external file with code. You also copied some files and used tools that will be used in other labs to explore JavaScript in more depth.

Lab 9 - Basic JavaScript Syntax

In this lab you will add some more JavaScript code to various pages to explore more the basic syntax of JavaScript. This will include declaring and using variables, some basic control structures, and calling simple functions.

Part 1 - Verify Files

In a previous lab you should have copied several files to a location where the Apache web server can locate them and return them as web pages. This section will verify you have the correct files.

__1. Open a Windows Explorer file browser window.

__2. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder.

__3. Verify that the following sub folder has been copied to where the Apache server is installed. There will be various files within this subfolder.

```
C:\Apache2.2\htdocs\JavaScript\basic
```

> **Note:** If you do not see this folder, you can copy this from the 'C:\LabFiles\JavaScript' folder or extract the appropriate lab solution for the previous lab from the 'C:\LabFiles\Solutions' folder. Be careful which files you copy and where you place them to make sure the web server can locate them with the URLs provided in the labs.

Part 2 - Variables

One of the most basic things you will do in JavaScript code is declare, initialize, and use variables.

__1. Run the Notepad++ program or a regular text editor.

```
C:\Software\NotepadPlus\notepad++.exe
```

> **Note:** A file may appear since Notepad++ opens the last used file. To avoid this in future, close the file first and then close Notepad++.

__2. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file:

```
C:\Apache2.2\htdocs\JavaScript\basic\prompt.js
```


__3. There is already an empty function declared that will be called when the page loads. To the top of this empty function add the following line in bold.

```
window.onload = initAll;

function initAll() {
          var ans = prompt("Are you sure you want to do that?","");
```

Note: The 'prompt' function is a built-in function that will popup a small dialog window to collect input from the user. The message and any default value are what are used for the function parameters.

__4. To complete the function add the following bold code. This will check if there was any value returned by the prompt dialog and use it if there was.

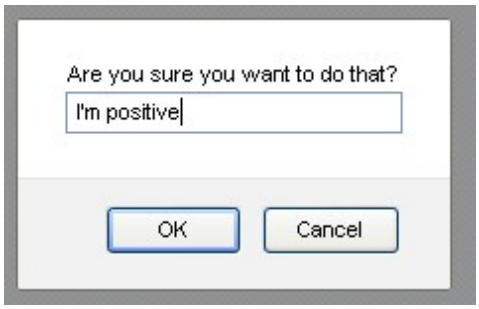
```
function initAll() {
          var ans = prompt("Are you sure you want to do that?","");
          if (ans) {
                    alert("You said " + ans);
                    document.write("You said " + ans);
          }
          else {
                    alert("You refused to answer");
          }
}
```

__5. Save the file but leave it open.

__6. Open a Firefox or Chrome web browser and go to the following URL:

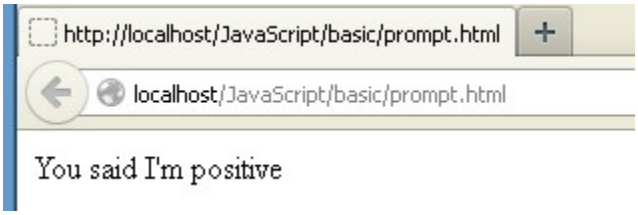
http://localhost/JavaScript/basic/prompt.html

__7. When the page first loads you should see a prompt. Fill in a value and click the **OK** button.



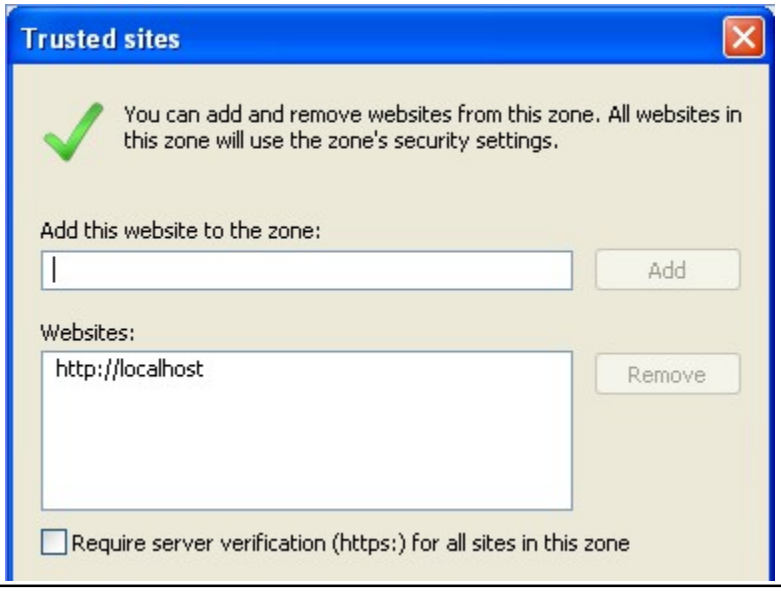
__8. You should see the dialog change to the show the message. Click the **OK** button.

__9. Check that you have the same message written out to the page.



Note: By default, recent versions of Internet Explorer will not allow prompt dialogs from JavaScript. Even though it looks like there might be a way to "temporarily" allow the dialog, it appears that when you refresh the page it reverts back to blocking it so that won't help for this page that only displays the dialog when the page is loaded.

If you wanted to try the page with Internet Explorer you could go into the 'Internet Options' settings and under the security settings add 'http://localhost' to the list of trusted sites. The prompt would then work in Internet Explorer. Changing the security settings of the browser may not be desirable though so we will just use another browser.



__10. You can refresh the page and supply a different message.

Part 3 - Handle Errors

In some situations you might want to check for certain conditions and raise an error if something is incorrect. This is done with the “try/catch” syntax you will see in this part. You will check that the user gave you a positive number before finding the square root of that number.

__1. Run the Notepad++ program or a regular text editor and open the following file:

C:\Apache2.2\htdocs\JavaScript\basic\catchError.js

__2. Add the following bold code to the existing 'initAll' function.

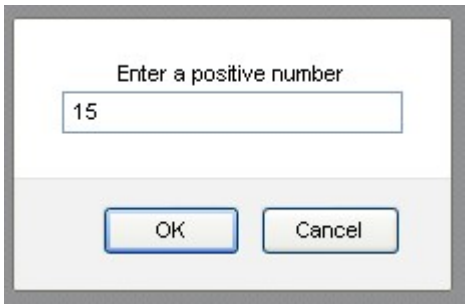
```
function initAll() {
    var ans = prompt("Enter a positive number","");
    try {
        if (!ans || isNaN(ans) || ans<0) {
            throw new Error("Not a valid number");
        }
        var sqrt = Math.sqrt(ans);
        var message = "The square root of " + ans + " is " + sqrt;
        alert(message);
        document.write(message);
    }
    catch (errMsg) {
        alert(errMsg.message);
    }
}
```

__3. Save the file but leave it open.

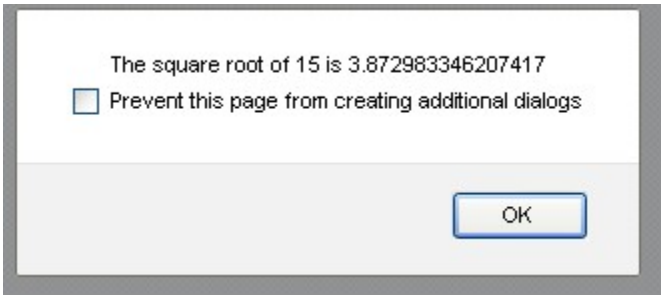
__4. Open a Firefox or Chrome web browser and go to the following URL:

http://localhost/JavaScript/basic/catchError.html

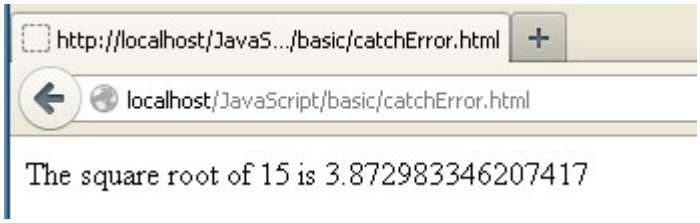
__5. When the page first loads you should see a prompt. Fill in a value and click the **OK** button. Start by using a valid positive number.



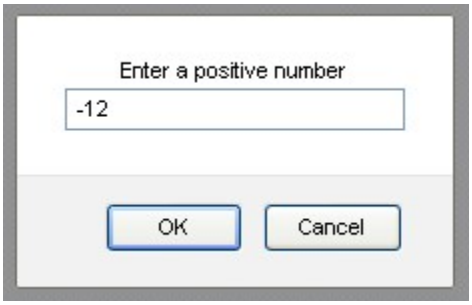
__6. You should see a dialog that shows the square root of your number.



__7. Click the **OK** button on the dialog and you should see the same message printed to the page.



__8. Reload or refresh the page and this time enter an invalid response like a negative number or text that is not a number.



__9. You should see a dialog that says the entry is not a valid number and nothing should be displayed on the page.

Part 4 - Modify Style of Page Elements

JavaScript code can also access elements of a page using the DOM API. In this part you will see just some basics about doing this to modify the style of an element.

__1. Run the Notepad++ program or a regular text editor and open the following file:

C:\Apache2.2\htdocs\JavaScript\basic\find.html

__2. Observe that the main HTML code for the body of the page is a text box for the user to enter a value in and a button. This button will trigger a JavaScript function. There are also some other page elements with text that will have the styles changed. Notice all the elements have the 'id' attribute.

```
<input type="text" id="id_value" size="20" />
<button id="findById">Find by ID</button>

<p id="p1">This is para1 (p1)</p>
<div id="error1">Error message 1 (error1)</div>
<p id="p2">This is para2 (p2)</p>
<div id="error2">Error message 2 (error2)</div>
<p id="p3">This is para3 (p3)</p>
```

__3. Close the file without making any changes.

__4. In the Notepad++ program or a regular text editor and open the following file:

C:\Apache2.2\htdocs\JavaScript\basic\find.js

__5. In the currently empty '**initAll**' function, add the following bold line of code. This will modify the button so that when it is clicked it will call the other function in this file.

```
function initAll() {
          document.getElementById("findById").onclick = findById;
}
```


__6. In the currently empty '**findById**' function, add the following bold lines of code. This uses the 'getElementById' function of the built-in 'document' object which is a key part of the DOM API. This code finds out what the user filled in for the textbox and then finds the element with that id to modify the style to yellow background.

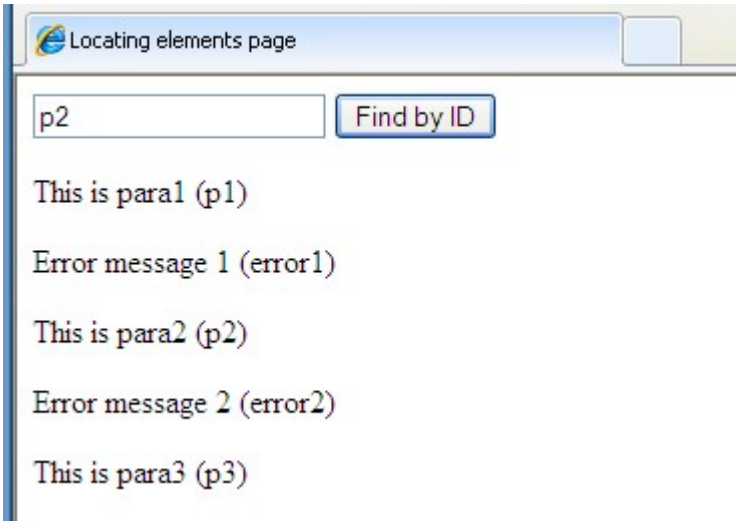
```
function findById() {
            var id = document.getElementById("id_value").value;
            var e = document.getElementById(id);

            if (e != null) {
                        e.style.backgroundColor = "yellow";
            } else {
                        alert("Could not find element by ID: " + id);
            }
}
```

__7. Save the file but leave it open.

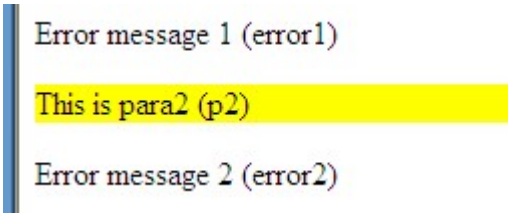
__8. Open a web browser (any type) and go to the following URL:

http://localhost/JavaScript/basic/find.html

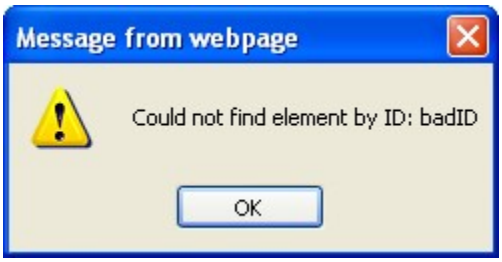
__9. In the textbox of the page enter the id of one of the elements (these are in parenthesis) and then click the '**Find by ID**' button.



__10. You should see that the background of the matching element changes.



__11. Try entering the id of some of the other elements. Also try a value that is not a valid element id and verify the error dialog appears.



Part 5 - Object Detection

One thing that is often desirable when writing JavaScript code is to try and ensure that it will function properly on the various browsers and versions in use by Internet users. Although it is possible to try and detect what kind of browser the user is using and decide if it supports what you need this is complicated and error-prone. An easier mechanism is "object detection" where you write code that detects if the browser supports features you need to use.

This part of the lab will use the ability of the browser to return the date and time in a "locale specific" way. This often uses the calendar and time zone of the computer running the browser. This ability was added to JavaScript a little after other features (although supported on all of the last several versions of the major browsers) so it might be a good candidate for object detection.

__1. In the Notepad++ program or a regular text editor and open the following file:

`C:\Apache2.2\htdocs\JavaScript\basic\date.js`

__2. In the current '**initAll**' function, add the following if/else statements in bold. Leave some space in the 'if' block as more code will be added there.

```
function initAll() {
    var now = new Date();
    if (now.toLocaleDateString) {


    }
    else {
        alert("Your browser does not give a locale-specific date/time");
    }
}
```

Note: The 'toLocaleDateString' function on the Date object that was created is a function that may not be supported by all browsers so we test to see if it exists first.

Normally, you would write code that might provide an alternative version of the page if the feature is not supported. Here we are just adding code to show a popup dialog just to show the pattern of this technique.

__3. Within the 'if' block you just added, add the following code in bold. Make sure that you put in the correct number of curly brackets so they match.

```
    var now = new Date();
    if (now.toLocaleDateString) {
        var date = now.toLocaleDateString();
        var time = now.toLocaleTimeString();
        alert("It is: " + time + " on " + date );
        if (document.getElementById) {
            document.getElementById("dateString").innerHTML =
                "It is: " + time + " on " + date;
        }
        else {
            document.write("It is: " + time + " on " + date );
        }
    }
```

Note: This code uses the functions 'toLocaleDateString' and 'toLocaleTimeString'. Technically we only tested for the presence of one of these but they were both added to JavaScript at the same time.

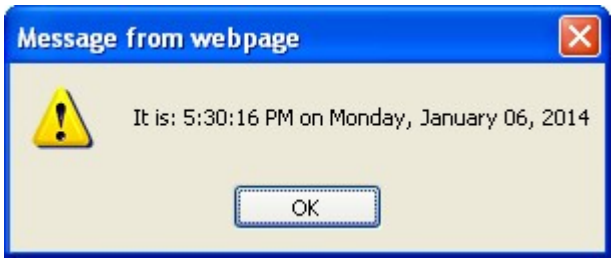
This code also tests for the presence of the 'document.getElementById' function. Even though this is part of the DOM API it also was added a little after some of the other JavaScript basics. Although all major browsers released over the last decade support this function if you want truly cross-browser compatible code you might want to check for it also.

__4. Save the file but leave it open.

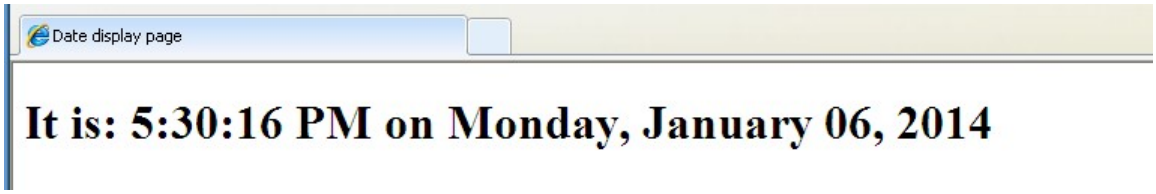
__5. Open a web browser (any type) and go to the following URL:

```
http://localhost/JavaScript/basic/date.html
```

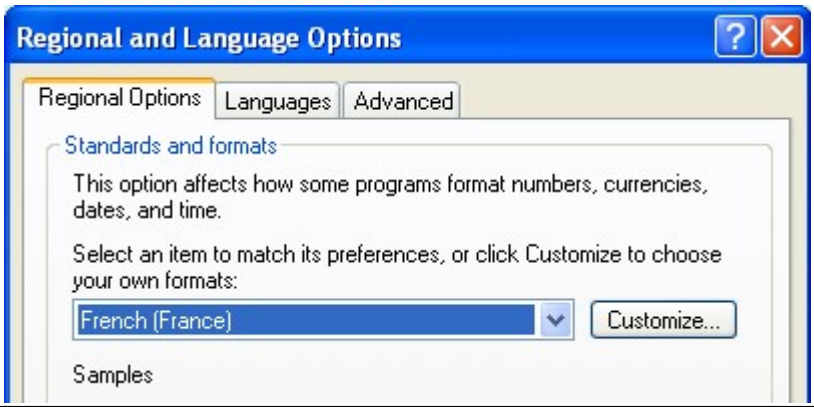
__6. Verify that you get a popup dialog that shows the current time and date.



__7. After clicking the **OK** button you should also see this printed to the page, likely in a large bold <h1> element.



Note: If you want you might be able to experiment with the regional settings of your computer and the date/time formatting to see the date and time displayed differently. Do this with care however to make sure you can switch it back and do not accidentally alter any important system settings. One thing you should not do is change the language of the entire operating system.



__8. Close all open browsers.

__9. If you are using Notepad++ then close all open files and then close Notepad++.

Part 6 - Review

In this lab you saw some more about various basics of JavaScript. This included declaring and using variables, basic statements, simple control structures like if/else and try/catch, and calling various built-in JavaScript functions. These features will build the foundation of most of the JavaScript code that you write.

Lab 10 - JavaScript Functions

Although you may have already seen some simple JavaScript functions we have not yet focused on them much. In this lab you will add complete function definitions and the functions will be a little more complex by accepting parameters and returning values.

Part 1 - Verify Files

In a previous lab you should have copied several files to a location where the Apache web server can locate them and return them as web pages. This section will verify you have the correct files.

__1. Open a Windows Explorer file browser window.

__2. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder.

__3. Verify that the following sub folder has been copied to where the Apache server is installed. There will be various files within this subfolder.

```
C:\Apache2.2\htdocs\JavaScript\function
```

Note: If you do not see this folder, you can copy this from the 'C:\LabFiles\JavaScript' folder or extract the appropriate lab solution for the previous lab from the 'C:\LabFiles\Solutions' folder. Be careful which files you copy and where you place them to make sure the web server can locate them with the URLs provided in the labs.

Part 2 - Return Result from Function

Often functions will perform some calculation and then return the result to where the function was called from.

__1. Run the Notepad++ program or a regular text editor.

```
C:\Software\NotepadPlus\notepad++.exe
```

Note: A file may appear since Notepad++ opens the last used file. To avoid this in future, close the file first and then close Notepad++.

__2. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file:

```
C:\Apache2.2\htdocs\JavaScript\function\bingo.js
```


__3. To the existing '**generateNumber**' function add the following bold code. This will return a bingo number which are between 1-75.

```
function generateNumber() {
            return Math.floor(Math.random() * 75) + 1;
}
```

__4. To the existing '**displayNumber**' function add the following bold code. This will call the other function and store what is returned in a variable. The result will then be used to display a message to the page.

```
function displayNumber() {
            var number = generateNumber();

            document.getElementById("number").innerHTML =
                        "The next BINGO number is: " + number;
}
```

__5. Save the file but leave it open.

__6. Open a web browser (any type) and go to the following URL:

http://localhost/JavaScript/function/bingo.html

__7. Click the 'Pick a BINGO Number' button and verify a number is displayed. You can click several times to verify they are randomly picked and always between 1-75.

__8. Leave the browser displaying the page open.

__9. Return to the editor for the 'bingo.js' file.

__10. Modify the entire content of the '**generateNumber**' function with the new code in bold below. This will return an array with the letter for the column along with the number picked.

```
function generateNumber() {
          var columnLabel = new Array("B", "I", "N", "G", "O");
          var columnIndex = Math.floor(Math.random() * 5);
          var newNum = Math.floor(Math.random() * 15) + 1
                    + (columnIndex * 15);

          return new Array(columnLabel[columnIndex], newNum);
}
```

__11. Make the following small modification to the end of the '**displayNumber**' function. Notice that most of the code is the same except where the message is altered with the number that is returned. Now the column letter and number will be displayed.

```
function displayNumber() {
          var number = generateNumber();

          document.getElementById("number").innerHTML =
                    "The next BINGO number is: " +
                    number[0] + " - " + number[1];
}
```

__12. Save the file.

__13. Return to the browser showing the page and refresh the page. Open a new window to the following location if you closed it.

http://localhost/JavaScript/function/bingo.html

__14. Click the 'Pick a BINGO Number' button and verify that now the column letter and the number are displayed. You can click several times to verify that the number is always within the appropriate range for the column chosen.

Part 3 - Pass Function Parameters

All of the functions you have seen until now have not needed any additional information passed in from where the function is called. Now you will define some functions that will take parameters to use in calculations.

__1. Run the Notepad++ program or a regular text editor and open the following file:

C:\Apache2.2\htdocs\JavaScript\function\primes.js

__2. Examine the existing code in the 'initAll' function. It will prompt for a number and check if it is a positive integer number entered by the user.

__3. Add the following bold line of code where the comment indicates. This will call a function that you will define next. Notice it will take a parameter for the value the user entered.

```
function initAll() {
    var ans = prompt("Enter a positive integer greater than 1","10");
    try {
        if (!ans || isNaN(ans) || ans <= 0 || Math.round(ans) != ans) {
            throw new Error("Not a valid number");
        }
        // call function here
        calculatePrimes(ans);
    }
    catch (errMsg) {
        alert(errMsg.message);
    }
}
```

__4. At the end of the file, add the following declaration of the new function. Make sure to add it outside of any other current function and add the matching curly bracket on the end of the function. Notice this function declares the parameter that it takes but does not declare the type of data it might hold like some other programming languages might.

```
function calculatePrimes(maxNumber) {

}
```


__5. Within the brackets for the new function declaration, add the following bold code. This will check each number within the range by calling another function to test that specific number. The number is added to a String of all the prime numbers if it is found to be prime.

```
function calculatePrimes(maxNumber) {
    var primeString = "";
    for (var testing = 2; testing <= maxNumber; testing++) {
        if (testForPrime(testing)) {
            primeString += testing + ", ";
        }
    }

}
```

__6. To complete the 'calculatePrimes' function, add the bold code below. This will remove the trailing comma at the end of the prime number list amd call another function to display the list on the page.

```
function calculatePrimes(maxNumber) {
    var primeString = "";
    for (var testing = 2; testing <= maxNumber; testing++) {
        if (testForPrime(testing)) {
            primeString += testing + ", ";
        }
    }
    // remove last comma
    var lastCommaIndex = primeString.lastIndexOf(",");
    if (lastCommaIndex != -1) {
        primeString = primeString.substring(0, lastCommaIndex);
    }
    displayPrimes(maxNumber, primeString);
}
```

__7. At the end of the file, add the following declaration of a new function. Make sure to add it outside of any other current function and add the matching curly bracket on the end of the function.

```
function testForPrime(numberToTest) {

}
```


__8. Within the brackets for the new function declaration, add the following bold code. This code will check to see if it is possible to evenly divide the number being checked by any of the numbers less than or equal to the square root of a number. If it is found to be divisible a response is immediately returned that it is not a prime number. If the loop completes and no divisors have been found it is a prime number.

```
function testForPrime(numberToTest) {
          var modval = Math.ceil(Math.sqrt(numberToTest));
          // Check all numbers lower than the square root
          for (; modval > 1; modval--) {
                    if (numberToTest % modval == 0) {
                              return false;
                    }
          }

          // after loop number must be prime
          return true;
}
```

__9. At the end of the file, add the following declaration of a new function. This will be the function to display the list of prime numbers on the page. Notice that it takes two parameters which will be used in the output.

```
function displayPrimes(maxNumber, primeString) {

}
```

__10. Within the brackets for the new function declaration, add the following bold code. This code will display the number the user entered to use as the maximum number to check and display a message. Usually the message will be the list of prime numbers but another message will be displayed if there are none.

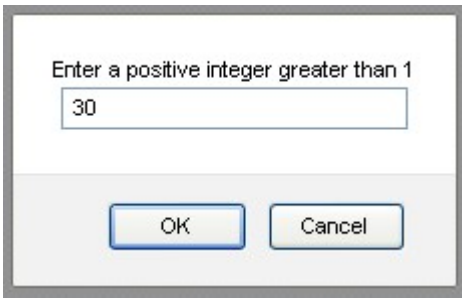
```
function displayPrimes(maxNumber, primeString) {
    document.write("Your maximum to check for primes was: " +
        maxNumber + "<br/>");
    // check if there are any primes
    if (primeString.length == 0) {
        document.write(
            "There are no prime numbers within your maximum");
    }
    else {
        document.write("The prime numbers up to your maximum are: " +
            primeString);
    }
}
```


__11. Save the file but leave it open.

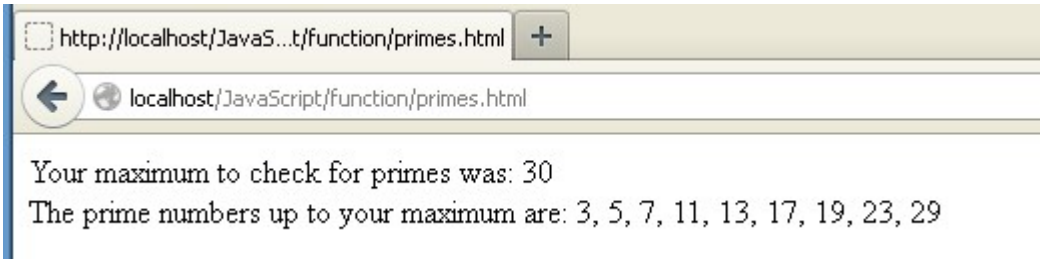
__12. Open a Firefox or Chrome web browser and go to the following URL:

```
http://localhost/JavaScript/function/primes.html
```

__13. Enter a positive number in the dialog and click the **OK** button.



__14. Check that you get the expected output printed to the page including a list that has no comma after the last number.



__15. You may refresh the page and enter another maximum number to check for prime numbers.

__16. Close all open browsers.

__17. If you are using Notepad++ then close all open files and then close Notepad++.

Part 4 - Review

In this lab you defined and used a few functions within your JavaScript code. These functions often returned a value to the location they were called from and took some parameters which were used to modify the behavior of the code being executed. You should notice that unlike some other programming languages, JavaScript does not require you to declare that a function returns a value or the type of the function parameter.

Lab 11 - Arrays in JavaScript

Arrays are useful when managing large sets of data within JavaScript code. This lab will show a few examples of using arrays and how using them can make programming certain algorithms less complex.

Part 1 - Verify Files

In a previous lab you should have copied several files to a location where the Apache web server can locate them and return them as web pages. This section will verify you have the correct files.

__1. Open a Windows Explorer file browser window.

__2. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder.

__3. Verify that the following sub folder has been copied to where the Apache server is installed. There will be various files within this subfolder.

```
C:\Apache2.2\htdocs\JavaScript\array
```

Note: If you do not see this folder, you can copy this from the 'C:\LabFiles\JavaScript' folder or extract the appropriate lab solution for the previous lab from the 'C:\LabFiles\Solutions' folder. Be careful which files you copy and where you place them to make sure the web server can locate them with the URLs provided in the labs.

Part 2 - Use Array to Track Bingo Card Numbers

Although generating random numbers for a bingo card is something that could be done without arrays, it is likely that sometimes the numbers picked would be duplicated on the same card. The easiest way to avoid this is if a number has already been picked for a particular bingo card to discard it and pick another one. An array can be used to track which numbers have already been picked for a card.

__1. Run the Notepad++ program or a regular text editor.

```
C:\Software\NotepadPlus\notepad++.exe
```

__2. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file:

```
C:\Apache2.2\htdocs\JavaScript\array\bingoCard.html
```


__3. Most of this HTML code is standard with a link to a CSS stylesheet and the external JavaScript file you will edit in a few steps. The important part of this code is that there is a table with 5 columns and table cells that have an id of 'squareXX' where 'XX' is a number between 0-23. These would be the valid array indexes in an array with 24 elements which is what we need since the middle square is a “free” space and doesn't need a number.

```
<tr>
            <th>B</th>
            <th>I</th>
            <th>N</th>
            <th>G</th>
            <th>O</th>
</tr>
<tr>
            <td id="square0"> </td>
            <td id="square5"> </td>
            <td id="square10"> </td>
            <td id="square14"> </td>
            <td id="square19"> </td>
</tr>
```

__4. Close the file without making any changes.

__5. In the Notepad++ program or a regular text editor and open the following file:

C:\Apache2.2\htdocs\JavaScript\array\bingoCard.js

__6. Look for the code at the very top of the file and add the following bold line where the comment mentions a global array. This array will be used to store whether or not a particular number has been used for this bingo card.

```
window.onload = initAll;
// global used numbers array
var usedNums = new Array(76);
```

Note: The array has been declared with a size of 76 so that the legal bingo numbers (1-75) can be used directly as array indices. The length of the array will not be important so having the “extra” spot at index '0' will not be an issue. In this case it will be better to simplify code by preventing the need to always subtract 1 from each number to find the related array index. In a situation where the length of the array was important we might not be able to do this.

__7. Examine the first few lines of the '**setSquare**' function. This function is called to initialize the number for each of the squares in the bingo card. The '**colPlace**' array is created and initialized with various numbers 0-4. These represent the column index the various squares are related to.

```
function setSquare(thisSquare) {
    var currSquare = "square" + thisSquare;
    var colPlace = new
        Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
```

Note: Notice that there are 5 places with each of the values except that the value '2' only occurs 4 times because of the "free" spot in the middle column. This array was provided for you because typing out the code to initialize the array may have been error-prone and lead to tough to debug issues if initialized incorrectly. Storing the column index in this array and linking the value stored in the array to the column index the square is associated with will be key in calculating the number in the next calculations.

__8. Add the following bold line of code to calculate the "column basis". This is a number that will be added to a random number generated in later code and represents the range of numbers allowed in a particular column. Notice the square number is used to find the multiplier from the 'colPlace' array.

```
function setSquare(thisSquare) {
    var currSquare = "square" + thisSquare;
    var colPlace = new
        Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
    // calculate column basis
    var colBasis = colPlace[thisSquare] * 15;
```

__9. Fill in the condition of the do/while loop to use the value stored in the 'usedNums' array to indicate if the number has been used in this bingo card yet or not. Also add code after the loop to mark the position in this 'usedNums' array as being used. Notice how the random number generated is used as the index into this array which is why we wanted to be valid indices in this array.

```
var newNum;

// loop until an unused number is found
do {
          newNum = colBasis + getNewNum();
}
while ( usedNums[newNum] );
// mark number used
usedNums[newNum] = true;
```


Note: The previous code might be a little confusing at first but remember this function is going to be called several times. If the number that is picked has not been used yet, the condition will be false and the loop will quit after picking an unused number. If a number has been used before, the position in the 'usedNums' array will be true meaning the loop will repeat and pick another number until one that has not been used is found.

__10. Find the '**anotherCard**' function and add the following 'for' loop in bold. This will reset all of the positions in the 'usedNums' array to be 'false' when generating another bingo card.

```
function anotherCard() {
          // clear used numbers
          for (var i = 1; i < usedNums.length; i++) {
                    usedNums[i] = false;
          }

          newCard();
          return false;
}
```

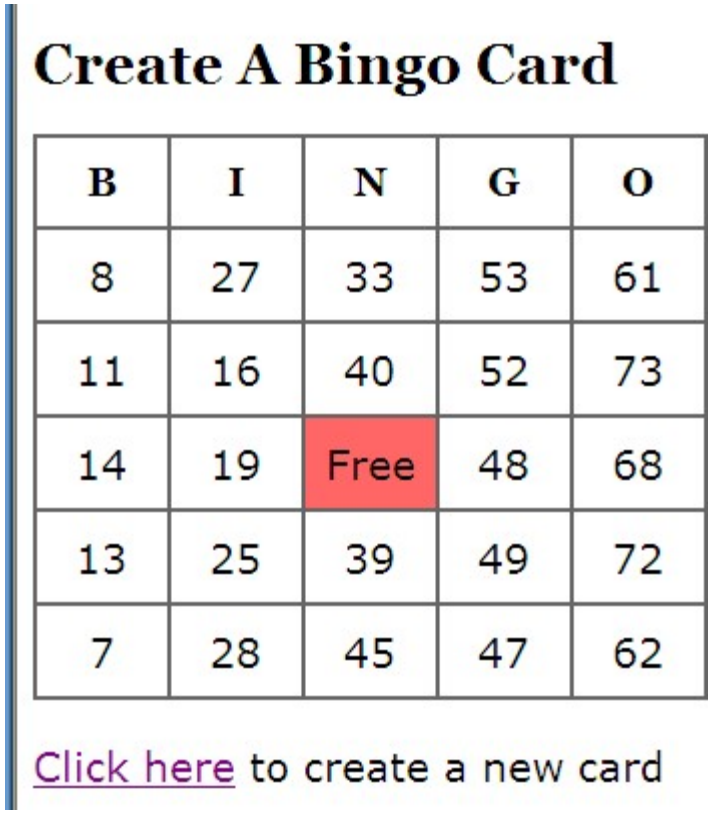
Note: Remember that no matter how big the array, the value for the length of the array is never a valid index. This is why the '<' is used in the conditional test. In our case the length is 76 and the last index that will be reset is 75 which is important to make sure all the numbers will be marked as available again. Often in using arrays operations at the very first position and last position may not be performed correctly and should warrant special attention in testing. In this case we will want to make sure numbers 1 & 75 can get picked for the bingo card.

__11. Save the file but leave it open.

__12. Open a web browser and go to the following URL:

http://localhost/JavaScript/array/bingoCard.html

__13. Verify that you see the bingo card displayed. Check that all squares are filled in with numbers that are legal for that column (1-15 for B, 16-30 for I, etc).



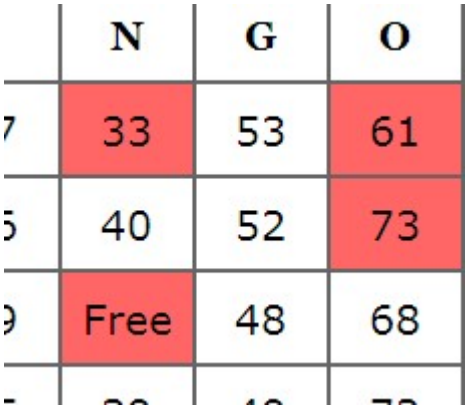
Create A Bingo Card

B	I	N	G	O
8	27	33	53	61
11	16	40	52	73
14	19	Free	48	68
13	25	39	49	72
7	28	45	47	62

Click here to create a new card

__14. Click on the “Click here” link a few times to generate a new bingo card and ensure that functions properly. You should check that numbers that had been used on the previous cards are also used on future cards to verify the array for used numbers is being reset. Also check that numbers from the edges of the column range (1 & 15, 16 & 30, etc) are being used.

__15. If you want you can even click on a square to use an existing function that will mark the square red.



	N	G	O
7	33	53	61
5	40	52	73
9	Free	48	68

Part 3 - Use Array for Prime Numbers

Calculating prime numbers, especially a large number of them, can be a brute force operation. Although it is possible to simply divide a number by every number less than it to see if it is a prime number there are a few facts that can be used to optimize this operation.

- You only need to check for divisors up to the square root of the number you are checking. If there is a number bigger than the square root of a number that divides evenly into the number there will also be a number less than the square root, that will have already been checked, that will divide into it. So if no numbers less than the square root of a number are found that divide a number evenly you do not need to check the numbers above the square root to prove the number is a prime number.
- If a number is NOT a prime number it will be divisible by at least one number less than it that IS a prime number. Therefore in order to check if a number is prime you only need to try dividing it by the other prime numbers you have already found.

In order to use the second fact to make the calculation of prime numbers more efficient, you would need to track the current set of prime numbers that have been found. This will be the perfect use for an array of the prime numbers that have been found so far.

__1. Run the Notepad++ program or a regular text editor and open the following file:

```
C:\Apache2.2\htdocs\JavaScript\array\primesArray.js
```

__2. Examine the logic of the '**initAll**' function but don't make any changes. This code will prompt the user for how many prime numbers to calculate and make sure they enter a whole number that is greater than or equal to 1.

__3. Find the '**calculatePrimesArray**' function and add the bold line of code below at the beginning to declare and initialize an empty array to store the prime numbers that are found.

```
function calculatePrimesArray(numberOfPrimes) {
          // array for primes
          var primes = new Array(0);
```


__4. Find the 'if' test that comes next and add the bold line below to manually add the value '2' as the first prime number. Manually adding the first prime number will make other code simpler because it will not have to deal with the possibility that the array is empty (zero length).

```
function calculatePrimesArray(numberOfPrimes) {
          // array for primes
          var primes = new Array(0);

          // add 2 to prime array
          if (numberOfPrimes >= 1) {
                    primes.push(2);
          }
```

__5. Further down in the '**calculatePrimesArray**' function, fill in the condition of the 'while' loop as shown in bold below so it checks the length of the array to see if enough prime numbers have been found. Since the first prime number is added manually, it is important the loop condition does not use '<=' so that it will behave properly (by skipping the loop) if the user only asks for one prime number.

```
          var lastTested = 2;
          // loop while array needs primes added
          while ( primes.length < numberOfPrimes ) {
                    // increment tested number
                    lastTested++;
```

__6. Near the end of the 'while' loop add the bold line of code below to push the value into the array of primes if the function that is called to test the number returns that it is a prime number. You will add some code to this function next.

```
          while ( primes.length < numberOfPrimes ) {
                    // increment tested number
                    lastTested++;

                    // pass to function to test
                    var isPrime = testForPrime(lastTested, primes);

                    // if prime add to array
                    if (isPrime) {
                              primes.push(lastTested);
                    }
          }
```

__7. Notice the end of this function calls another function to display the prime numbers and passes in the array as a parameter. You do not need to change this code.

```
          displayPrimes(numberOfPrimes, primes);
```


__8. Find the '**testForPrime**' function and notice that the function declaration takes the number to test and the current array of prime numbers. This is key to using the array of prime numbers to make the test more efficient.

```
function testForPrime(numberToTest, primes) {
```

__9. Fill in the conditions for the 'for' loop and 'if' statement as shown in bold below using the elements of the 'primes' array and the index 'I' declared by the 'for' loop. The 'for' loop condition checks if the current prime number being used is less than or equal to the square root of the number being tested. The 'if' statement condition checks if the current number being tested is evenly divisible by the current prime number and if it is immediately returns 'false' because the number being tested is not a prime number.

```
function testForPrime(numberToTest, primes) {
    var upperLimit = Math.ceil(Math.sqrt(numberToTest));

    // loop until prime divisor is greater than upper limit to test
    for (var i = 0;  primes[i] <= upperLimit  ; i++) {
        if ( numberToTest % primes[i] == 0 ) {
            return false;
        }
    }
    // after loop number must be prime
    return true;
}
```

__10. Find the '**displayPrimes**' function and add the code in bold at the end of the function to add the array of primes to the display text.

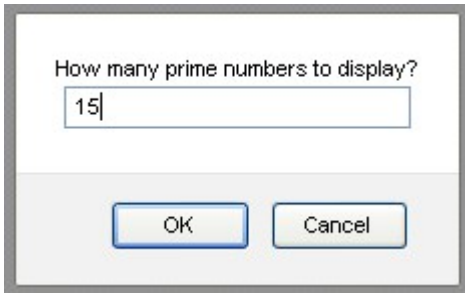
```
function displayPrimes(numberOfPrimes, primes) {
          document.write("You wanted this many primes: " +
                    numberOfPrimes + "<br/>");

          // add primes to end of display text
          document.write("The first " + numberOfPrimes +
                    " prime numbers are: " + primes);
}
```

__11. Save the file but leave it open.

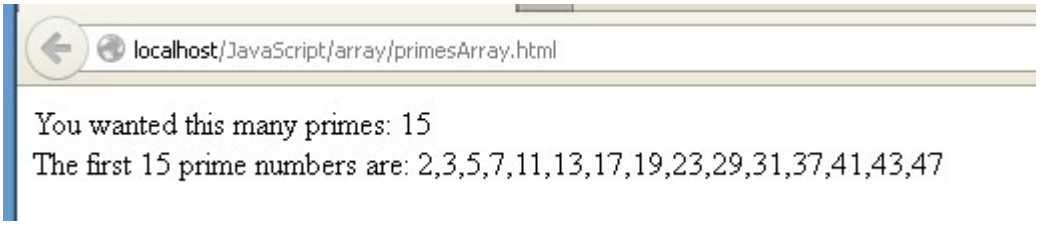
__12. Open a Firefox or Chrome web browser and go to the following URL:

http://localhost/JavaScript/array/primesArray.html

__13. When the page first loads you should see a prompt. Fill in a value and click the **OK** button.

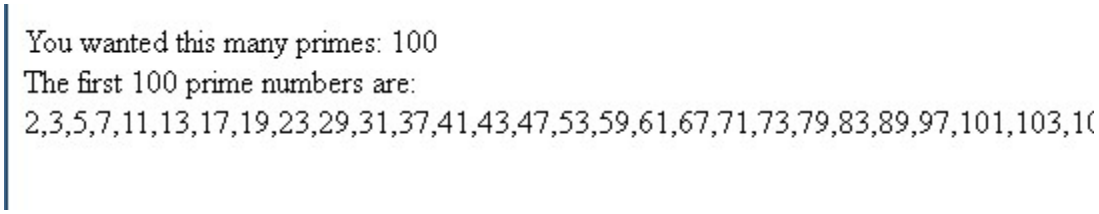


__14. You should see the expected output which includes the list of prime numbers. Notice how simply adding the entire array to the display text converted the array values into a comma-separated list.



__15. Refresh the page and when the prompt displays enter a large value like 100 for the number of prime numbers to calculate.

__16. You should notice that all the numbers remain on one line which causes the numbers to go well off the edge of the screen. If you wanted to see all of the numbers you would need to scroll to the right on the page. This is because there are no spaces for the browser to break the line at.



__17. Return to the editor for the 'primeArrays.js' file and open it again if you closed it.

C:\Apache2.2\htdocs\JavaScript\array\primesArray.js

__18. Find the '**displayPrimes**' function again and add the following code to create a comma-separated String of the array of prime numbers that has a space after each comma.

```
function displayPrimes(numberOfPrimes, primes) {
          document.write("You wanted this many primes: " +
                    numberOfPrimes + "<br/>");

          var primeString = "";
          for (var i = 0; i < primes.length - 1; i++) {
                    primeString += primes[i] + ", ";
          }
          primeString += primes[primes.length - 1];

          // add primes to end of display text
```

Note: Notice the condition for the 'for' loop subtracts one from the length of the array. This is so a comma is not added after the last prime number and that is added to the end of the String after the loop. It is important that we know the array will always have at least one element because otherwise this subtraction might give a '-1' which would not be a valid array index and cause an error.

__19. Change the last line of code of the function as shown in bold below so that the String of prime numbers is added to the display text instead of the array itself.

```
          // add primes to end of display text
          document.write("The first " + numberOfPrimes +
                    " prime numbers are: " + primeString);
}
```

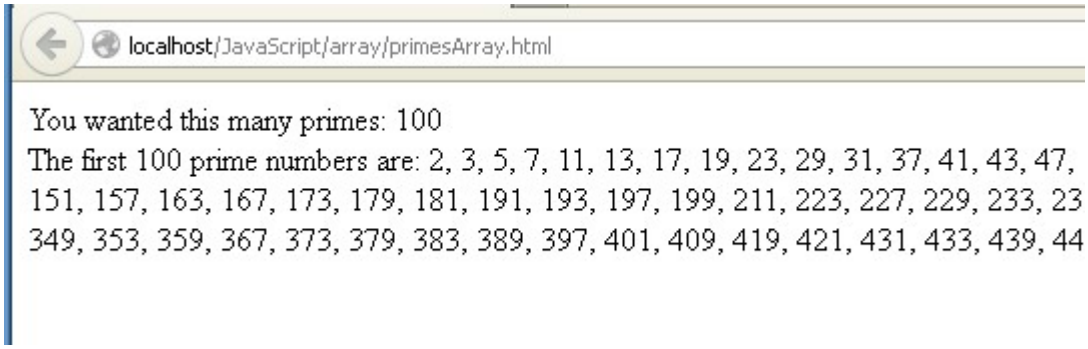
__20. Save the file.

__21. Return to the browser showing the page and refresh the page. Open a new window to the following location if you closed it.

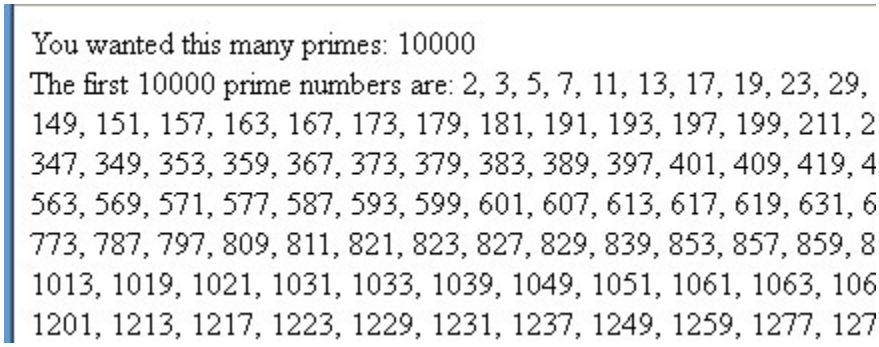
http://localhost/JavaScript/array/primesArray.html

__22. When the prompt displays enter a large value like 100 for the number of prime numbers to calculate.

__23. Verify that now the output off the prime numbers breaks across multiple lines and you do not need to scroll to the right. This is because with the spaces in the String, the browser is able to break the String to the next line when it reaches the right edge of the window.



__24. Refresh the page again and enter a very large number (about 10,000) for the number of prime numbers to calculate. You should see that the page is still displayed relatively quickly thanks to the optimizations made.



__25. Close all open browsers.

__26. If you are using Notepad++ then close all open files and then close Notepad++.

Part 4 - Review

This lab showed you various aspects of using arrays in JavaScript. You saw that in some situations arrays are very useful and the features of an array can even be used to the advantage of simplifying the code. In the bingo card example the array that tracked if a number had been used could use the number directly as the array index which made the code simple. The prime number example showed how you could build up the array of prime numbers as they were discovered and even use that array to make the calculation more efficient.

Lab 12 - Getting Started with jQuery

In this lab, we will setup the basic development environment. We will also learn how to install jQuery and learn its basic usage.

Part 1 - A Note About Browsers

As a best practice, you should always test your application using multiple browsers and versions. For these labs, we recommend that you use the latest versions of these browsers:

1. IE
2. FireFox
3. Chrome

Some of the labs will not work with very old browsers, like IE 6.

Part 2 - Looking at Apache

You will use Apache 2.2.x as the web server for this class. This is already installed for you. Please verify the installation folder. It should be **C:\Apache2.2**

From now on, we will refer to this folder as APACHE_DIR.

We will now validate that Apache is working properly.

__1. Using Notepad, open **APACHE_DIR/htdocs/index.html**.

__2. Note that the file currently contains one line:

```
<html><body><h1>It works!</h1></body></html>
```

__3. From a web browser, enter the URL: **http://localhost**

__4. Make sure that you see:

It works!

Part 3 - Using a Text Editor

You can use any text editor for this class. You can use Notepad or Wordpad. You can also download and install your favorite text editor.

Part 4 - Install jQuery

For our labs, we will need jQuery and jQuery UI JavaScript libraries. Those have already been downloaded for you. All you need to do is copy them over to the right folder.

__1. In the **APACHE_DIR/htdocs** folder, create a new folder called **jquery**

In Windows XP and Vista, you may have to be an Administrator to create folders within APACHE_DIR.

__2. From folder **C:\LabFiles** copy **jquery-1.10.2.min.js** and **jquery-ui.js** to the **APACHE_DIR/htdocs/jquery** folder.

Part 5 - Write a Basic jQuery Application

__1. In the **APACHE_DIR/htdocs** folder, create a new folder called **lab**

__2. In the lab folder, create a new file called **basic.html**

__3. Open the file with an editor and write the basic structure.

```
<html>

<head>
</head>

<body>
</body>

</html>
```

__4. Import jQuery JavaScript file as shown in bold face.

```
<head>
<script type="text/javascript"
    src="/jquery/jquery-1.10.2.min.js"></script>
</head>
```

__5. Add an empty paragraph within the body.

```
<body>
    <p id="p1"></p>
</body>
```


__6. Below the existing <script> tag, add another one.

```
<script type="text/javascript">
    $(function(){
        $("#p1").text("Hello World");
    });
</script>
```

We will inspect the code later. For now, continue with testing.

__7. Save changes.

Part 6 - Test

__1. From the browser, enter the URL: **http://localhost/lab/basic.html**

__2. Make sure that you see:

Hello World

Part 7 - What Just Happened?

All the magic happens in the script:

```
<script type="text/javascript">
    $(function(){
        $("#p1").text("Hello World");
    });
</script>
```

First, let's talk about the core function:

```
$()
```

Almost all jQuery API is accessible through this one single function. You can also call the core function through these names:

```
jQuery()
window.jQuery()
```

But, $() is short and sweet and that's what most developers use.

The core function behaves completely differently depending on the input parameters you pass to it. In our case, we passed an anonymous function:

```
function(){
    $("#p1").text("Hello World");
}
```

When you pass a function as input argument, jQuery treats it as a DOM ready event handler. This event is fired when the DOM document of the page is fully ready. This can happen before the "onload" event of the body. For example, if there are many images in the document, the DOM ready event fires before all the images are actually downloaded.

You can start manipulating the DOM as soon as the DOM is ready.

When the DOM document is ready, FireFox fires the “DOMContentLoaded” event and IE the “onreadystatechange” event for the document object. jQuery makes it easy to attach an handler for that event in a browser independent manner.

Now, let's look at what our DOM ready event handler function does. Firstly, we see that the core function is used again.

```
$("#p1")
```

This time, the core function is taking a string as an argument. This signifies to the core function that the argument is an element selector. jQuery borrows heavily from CSS selector model. According to CSS, "#p1" select a DOM element with the ID of "p1". That will be the empty paragraph we had added earlier.

$("#p1") will return a jQuery object for the paragraph. We then call the text() function for the object. This sets the inner text of the element to "Hello World".

What does the core function - $("#p1") – actually return? It returns a jQuery object that is a collection of all DOM element objects that match the supplied selector rule. This jQuery object builds on top of the JavaScript array API. For example, you can obtain the length: **$("p").length;** //Number of paragraphs in the page. To obtain a DOM element object in the collection, simply call get(index). **alert($("p").get(1).innerHTML);** //Show inner HTML of the second paragraph. When you invoke a method of the collection object, jQuery internally applies it to all DOM elements in the collection: **$("p").text("Hello!");** //Change text of all paragraphs to Hello!. A getter function normally works on the first DOM element in the collection only. **$("p").text();** //Returns the text of the first paragraph only.

To convert a DOM element object into a jQuery collection do:

$(elementObject)

For example:

$(document.getElementById("p1")); //Same as $("#p1").

Part 8 - Something More Advanced

So far, $("#p1") returned a single object. This expected since an element always has a unique ID. Now, we will use selector to return a collection of jQuery objects and work on them.

__1. Below the current paragraph add another one as shown in bold face below.

```
<p id="p1"></p>
<p id="p2"></p>
```

__2. Change the DOM ready event handler as shown below.

```
$("p").text("Hello World");
```

Now, $("p") will select all DOM elements with a <p> tag. Once again, jQuery borrows this from the CSS selector model.

__3. Save changes.

__4. Refresh the browser. Now, both paragraphs will say "Hello World".

Hello World

Hello World

What this means is that the text() function was called for the entire collection of jQuery objects returned by $("p"). Trying to do this using raw JavaScript will take quite a few lines of tedious coding.

Part 9 - Review

In this lab, we installed jQuery and wrote a very basic application. We learned about the core function $(). We wrote a DOM ready handler. We used a simple ID based selector first. And then a tag based selector.

Lab 13 - More on Selectors

In this lab, we will learn a few more advanced uses of selectors. We will also learn about how to manipulate the CSS styles of elements.

Part 1 - Iterating Over a Selected Collection

We know that the core function returns a collection of DOM element objects. We can iterate over the collection using the each() function.

We currently have two paragraphs in basic.html.

```
<p id="p1"></p>
<p id="p2"></p>
```

Let us say that we want to iterate over them and change their text.

__1. Change the line:

```
$("p").text("Hello World");
```

As follows.

```
$("p").each(function (index) {
    $(this).text("Your ID is: " + this.id + " and index: " + index);
});
```

__2. Save changes.

__3. Refresh the browser or enter the URL: **http://localhost/lab/basic.html**

Your ID is: p1 and index: 0

Your ID is: p2 and index: 1

There is a lot going on with the each() function. It takes as argument a function object. This function is called repeatedly, once for each jQuery object in the collection. The function receives the index of the jQuery object in the collection starting with 0.

Important: The "this" keyword within the iteration function points to the DOM element. We can not directly call the text() method for it since that is not a DOM API. We must obtain the jQuery object using $(this) and then call the text() function to set the text. Since, "this" is a DOM element, we can call this.id to get it's ID.

Part 2 - Class Selector

One of the common uses of selector is to select all elements that belong to a CSS class. This is done using the format $(.class_name). This is same as the CSS class selector.

Now, we will select one of the paragraphs by its class name and change its background color.

__1. Set the class of the second paragraph to "highlighted".

```
<p id="p2" class="highlighted"></p>
```

__2. At the end of the DOM ready handler function add the line shown in bold face.

```
$(function(){
    $("p").each(function (index) {
        $(this).text("Your ID is: " + this.id + " and index: " + index);
    });
    $(".highlighted").css("background", "yellow");
});
```

__3. Save changes.

__4. Refresh the browser. The second paragraph will now have a yellow background.

Your ID is: p1 and index: 0

Your ID is: p2 and index: 1

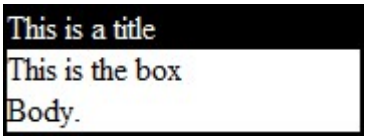
This was also a good example of how to tweak the style of an element using the css() function.

If you know that it is a "p" tag that has the "highlighted" class, you can help jQuery a little by using $(p.highlighted). This will have a performance improvement.

Part 3 - Child Pseudo Selector

So far, we have been using selectors that follow the same syntax as the CSS selector. Now, we will use pseudo selectors that are introduced by jQuery. A pseudo selector is always prefixed by ":".

We will now use jQuery to create a box widget that looks like this.



__1. Copy **C:\LabFiles\template.html** to **APACHE_DIR/htdocs/lab/**.

__2. Rename **template.html** to **box.html**

__3. Open **box.html** in an editor.

__4. Below the line:

```
<!-- Body HTML here -->
```

Add:

```
<div class="box">
<div>This is a title</div>
<div>
This is the box<br/>
Body.
</div>
</div>
```

Basically, any element with style "box" will be converted into a box. The first child element of the box will be converted into the title.

First, we will draw the border around the box.

__5. Below the line:

```
//jQuery code here
```

Add:

```
$("div.box").css("border-style", "solid");
$("div.box").css("border-width", "2px");
```

Now, we will change the background color of the first child element to black. We will change its foreground to white.

If we use $("div div") that will select all div child elements of another div. We don't want that. We only wish to select the first child.

__6. Continue to develop the DOM ready function by adding these lines:

```
$("div.box div:first-child").css("background", "black");
$("div.box div:first-child").css("color", "white");
```

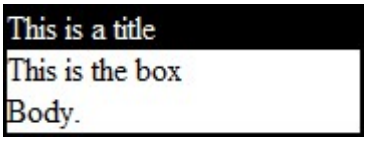
Here, we use the "first-child" pseudo selector. This will select the first child element of any div element that has the box class.

__7. Save changes.

__8. In the browser, enter the URL:

http://localhost/lab/box.html

__9. You should see the page as follows.

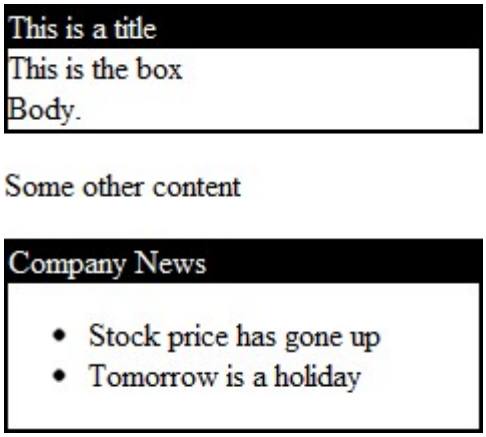


It's now easy to add as many boxes as we want.

__10. Below the last tag, enter:

```
<p>Some other content</p>

<div class="box">
<div>Company News</div>
<ul>
          <li>Stock price has gone up</li>
          <li>Tomorrow is a holiday</li>
</ul>
</div>
```

__11. Save changes and refresh the browser.



Part 4 - Optimize the Code

Selectors require a lot of DOM tree navigation. This can be slow for a large document. Once you have retrieved a collection, you should reuse it as much as possible. In our case, we are running the same selector multiple times. We should be able to optimize that.

__1. Change the DOM ready function as follows.

```
$(function(){
	//jQuery code here
	var box = $("div.box");
	box.css("border-style", "solid");
	box.css("border-width", "2px");
	var title = $("div.box div:first-child");
	title.css("background", "black");
	title.css("color", "white");

});
```

Now, a selector is run only once.

__2. Save changes.

__3. Refresh the browser. It will look same as before.

Part 5 - Pseudo Selector for Table Rows

jQuery has :odd and :even pseudo selectors that apply to table rows. They are great for styling alternate rows differently.

__1. Copy **C:\LabFiles\table.html** to **APACHE_DIR/htdocs/lab**.

__2. Open **APACHE_DIR/htdocs/lab/table.html** in your editor.

__3. Note that the header is created using <thead> and the body using <tbody>.

__4. View the page in a browser:

```
http://localhost/lab/table.html
```

Name	Age
John Doe	19
Jane Doe	21
Mary Doe	22
Superman	222

Nothing spectacular. Now, we will apply different colors to every alternate rows.

__5. Below the line:

```
//jQuery code here
```

Add:

```
$("tr:even").css("background-color", "grey");
$("tr:odd").css("background-color", "yellow");
```

__6. Save.

__7. Refresh the browser.

Name	Age
John Doe	19
Jane Doe	21
Mary Doe	22
Superman	222

It worked. But, the <tr> elements within <thead> are also being colored. We can fix that by combining the even and odd selector with child selector. We want only the <tr> elements that are children of <tbody> to be colored.

__8. Change the code as follows.

```
$("tbody tr:even").css("background-color", "grey");
$("tbody tr:odd").css("background-color", "yellow");
```

__9. Save and test again.

Name	Age
John Doe	19
Jane Doe	21
Mary Doe	22
Superman	222

Now the header is not colored any more. In fact, while we are at it, let's style the header row.

__10. Add this code.

```
$("thead tr").css("background-color", "black");
$("thead tr").css("color", "white");
```

__11. Save and test.

Name	Age
John Doe	19
Jane Doe	21
Mary Doe	22
Superman	222

__12. Close all your editors and web browsers.

Part 6 - Review

In this lab we played with more selectors. We also learned how to apply basic styling using the css() function. As you saw, with only a few lines of code, you can apply consistent look and feel throughout the site.

Lab 14 - Dynamic Style Class Assignment

In this lab, we will learn how to add or remove style class to form elements. We will also learn how to combine multiple selector rules into one.

We will work with a simple login form. When user submits the form, we will validate it and apply styling to invalid input fields.

Part 1 - Import the File

__1. Copy **C:\LabFiles\form.html** to **APACHE_DIR/htdocs/lab**

__2. Open a browser and test the file:

```
http://localhost/lab/form.html
```

__3. Open the file in editor. Note the following things:

- A style class called inputErr has been defined withing <style>.
- The onsubmit event handler for the form is checkForm();
- The form has a text and a password input field. We will validate the form and make sure that these fields are not empty.

Part 2 - Develop the Form Validation Function

Now, we will develop the checkForm() function. This function will return false if any of the input fields are empty. It will also set the style of the invalid fields to inputErr.

__1. Above the </script> line and below the DOM ready handler function, add a new function.

```
function checkForm() {

}
```


__2. Within the checkForm() function, add a variable:

```
var result = true;
```

__3. Add a compound selector as follows.

```
$("input[type='text'], input[type='password']")
```

A compound selector has a comma separated list of selectors. In this case, we are selecting all <input> elements that have the "type" attribute set to either "text" or "password".

__4. Add the each() function to the selected elements as shown in bold face below.

```
$("input[type='text'], input[type='password']").each(function (idx) {
});
```

__5. Fill out the body of the iteration function as follows.

```
$("input[type='text'], input[type='password']").each(function (idx) {
    if (this.value.length == 0) {
        result = false;
        $(this).addClass("inputErr");
    } else {
        $(this).removeClass("inputErr");
    }
});
```

Here, we are using this.value.length to check if the field is empty. Recall that "this" within an iteration handler points to a DOM element and not a jQuery object.

If the field is empty, we are calling addClass() method of the jQuery object and setting the class to inputErr. If the field is not empty, we are removing the class.

__6. Finally, at the end of the checkForm() function, add the line:

```
return result;
```

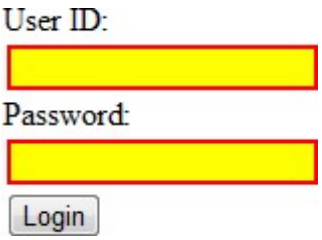
__7. Save changes.

Part 3 - Test

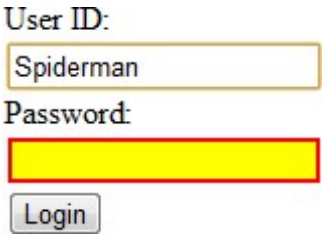
__1. Refresh the browser or enter the URL:

```
http://localhost/lab/form.html
```

__2. Leave both fields empty and submit the form. The color should change for both.



__3. Now, leave one field empty at a time and submit the form. Make sure that only the empty field is getting highlighted.



__4. Close all editors and browsers.

Part 4 - Review

In this lab, we learned how to dynamically add or remove CSS class for an element. We also saw how compound selectors work.

Lab 15 - DOM Manipulation

jQuery makes it super easy to add, remove and move around elements within the document. We will learn how to add and remove elements in this lab.

We will keep building on top of the log in form that we had worked on previously.

Part 1 - Showing Error Message

So far, we have been highlighting invalid form input fields. Now, we will also display error messages. To do that, we will use the append() jQuery function.

First, we will add a container for all error messages.

__1. Open **form.html** in an editor.

__2. Above the <form> tag, add these lines:

```
<div id="messages">
</div>
```

Now, we will work on the checkForm() function. First, we will clear all existing error messages.

__3. Below the line:

```
var result = true;
```

Add:

```
$(".errorMsg").remove();
```

This will wipe out any element with the CSS style class "errorMsg". That means, when we dynamically create the error message elements, they must have the class "errorMsg".

Now, we will create the error messages.

__4. When the input field is empty, use the append() function to add a new error message as shown below.

```
if (this.value.length == 0) {
    $("#messages").append(
        "<p class='errorMsg'>Input can not be empty: " + this.name
        + "</p>");
    result = false;
    $(this).addClass("inputErr");
} else {
```

Note that the new <p> elements will be added to the element with the ID "messages". Also note that we are setting the class of the <p> tags to errorMsg.

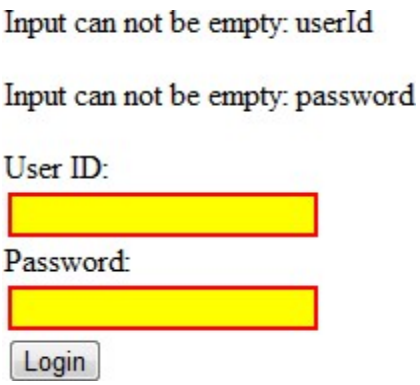
__5. Save changes.

Part 2 - Test

__1. In a browser enter the URL:

http://localhost/lab/form.html

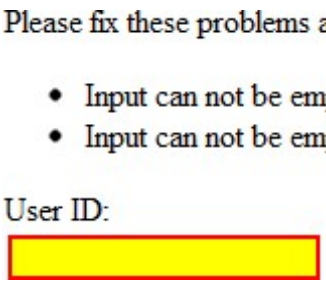
__2. Leave all input fields empty. Submit the form.



__3. Now, leave one field empty at a time and submit the form. Make sure that the correct error message is displayed.

Part 3 - Improve the Error Message Display

Right now the messages look very disjointed. Also there is no instructions to the user what needs to be done. We will now change things so that the messages look like this.



__1. Change the messages div tag as follows.

```
<div id="messages">
Please fix these problems and submit the form again:
<ul id="messageList">
</ul>
</div>
```

Now, the error messages have to be added to the <ul> element.

__2. Change the line:

```
$("#messages").append(
    "<p class='errorMsg'>Input can not be empty: " + this.name
    + "</p>");
```

To add <li> elements to the messageList <ul> element. This is shown below.

```
$("#messageList").append(
    "<li class='errorMsg'>Input can not be empty: " + this.name
    + "</li>");
```

Finally, we need to show or hide the messages div tag depending on if there are error messages.

__3. Above the "return result;" line add:

```
if (result == false) {
          $("#messages").show();
} else {
          $("#messages").hide();
}

return result;
```

__4. By default, when the page is first loaded, the messages div element should be hidden. Do that from the DOM ready handler.

```
$(function(){
    //jQuery code here
    $("#messages").hide();
});
```

__5. Save changes.

__6. Refresh the browser and test the form. Make sure that you get the effect shown in the screenshot earlier.

__7. Close all editors and browsers.

Part 4 - Review

In this lab, we learned how to add or remove elements. We also saw how to show or hide elements.

Lab 16 - Form Event Handling

jQuery makes it easy to attach event handlers to elements. Basically, you do not use the "onsubmit" "onclick" etc attributes for the elements but use jQuery API to attach handlers. jQuery uses underlying DOM API of the browser to get the job done. This keeps HTML clean.

We will continue to work on the login form.

Part 1 - Attach onblur Event Handler

Now, we will enable form validation as soon as a user moves focus away from an input field.

__1. Open **form.html** in an editor.

__2. In the DOM ready handler, add the lines shown in bold face.

```
//jQuery code here
$("#messages").hide();
$("input[type='text'], input[type='password']").blur(function() {
            checkForm();
});
```

This attaches an "onblur" event handler function for all text and password input elements.

__3. Save changes.

__4. In a browser enter the URL:

```
http://localhost/lab/form.html
```

__5. Test by clicking on a text box and then clicking or tabbing away from it. Error messages will be displayed right away.

Part 2 - Attach onsubmit Handler

Right now, the onsubmit handler for the form is set in HTML. We will extract it out from there.

__1. Below the line

```
//JQuery code here
```

Add:

```
$("form").submit(checkForm);
```

This will attach the checkForm function object as the handler for all forms in the page. In real life, to attach the handler to a specific form, you can use the ID sector, for example:

$("#myForm")

__2. Remove the onsubmit attribute for the form. This will look like.

```
<form method="post">
```

__3. Save changes.

__4. Refresh the browser.

__5. Test by submitting a form with invalid input. It should not submit.

__6. Close all editors and browsers.

Part 3 - Review

In this lab, we learned how to attach DOM event handler using jQuery. This extracts event handling code completely away from HTML.

Lab 17 - Basic Ajax

In this lab, we will use the jQuery load() function to fetch HTML from the server and display it. We will build a tabbed sheet like widget.

When a link is clicked, some HTML content will be fetched and displayed below it.

Part 1 - Get Started

__1. Copy these files from **C:\LabFiles** to **APACHE_DIR/htdocs/lab**:

- event.html
- news.html
- safety.html

__2. Copy **C:\LabFiles\template.html** to **APACHE_DIR/htdocs/lab/**

__3. Rename **template.html** as **tabs.html**

__4. Open **tabs.html** in editor.

__5. Below the line:

```
<!-- Body HTML here -->
```

Add:

```
<div class="tabbed">

<a href="news.html">News</a> <a href="event.html">Event</a> <a
href="safety.html">Safety</a>

<div>
<!--Content will go here-->
</div>

</div>
```


Basically, the element will work as the whole tabbed widget. Each <a> element within it will act as a tab. When user clicks on a link, we will fetch the content from the file mentioned in "href" and show it in the inner element.

__6. Save changes.

Part 2 - Add Tab Event Handler

We will now attach an onlcick event handler for all <a> elements within .

__1. Below the line:

```
//jQuery code here
```

Add:

```
$("div.tabbed a").click(function() {
    $(this).parent().children().last().load(this.href);
    return false;
});
```

The $("div.tabbed a") selects all <a> elements. The click() function attaches an onclick event handler. $(this) points to the <a> element that was clicked. parent().children().last() basically picks the inner element.

Finally, the load() function loads the contents of the URL in href attribute.

We have to make sure that we return false. That way, the browser will not follow the link and load a separate page.

__2. Save changes.

Part 3 - Test

__1. Open a browser and enter:

http://localhost/lab/tabs.html

__2. Click **News** link.

__3. Similarly, click **Event** and **Safety** link. Make sure that the right content is being displayed.

__4. Refresh the page. The content will go blank. That is because, we do not select a tab by default when the page is loaded. We will fix that next.

Part 4 - Select the First Tab by Default

When the page is loaded, we will select the first tab by default.

__1. Below the click() function, add a new line as shown below.

```
$(function(){
    //jQuery code here
    $("div.tabbed a").click(function() {
        $(this).parent().children().last().load(this.href);
        return false;
    });
    $("div.tabbed a").first().click();
});
```

That line will fire the onclick event for the first <a> element. Note, $("div.tabbed a")will return all <a> elements that are children of any element. The first() function picks the first jQuery object in the returned collection.

__2. Save changes.

__3. Refresh the browser. Now, the News content will be displayed by default.

Part 5 - Style Selected Tab

Our widget is working well. But, it is difficult to tell which one was last clicked or selected. We will fix that now.

__1. Somewhere within the <head> section, add a <style> element as follows.

```
<style type="text/css">
.tabSelected {
          background: grey;
          border-style: solid;
          border-width: 1px;
          border-color: black;
}
</style>
```


__2. In the onclick event handler function add the lines shown in boldface.

```
$("div.tabbed a").click(function() {
    $(this).parent().children().last().load(this.href);

    var last = $(this).parent().data("last_selected");
    if (last) last.removeClass("tabSelected");
    $(this).parent().data("last_selected", $(this));
    $(this).addClass("tabSelected");

    return false;
});
```

Here, we are taking advantage of the data() method to attach arbitrary JavaScript object to a jQuery object. We save the jQuery object for the currently selected <a> element with the parent using the name "last_selected".

__3. Save changes.

__4. Refresh the browser. Make sure that the selected tab is highlighted.

Part 6 - Instantiate Another Widget

When you design a widget, you should always test with multiple instances of it in the same page.

__1. Copy the entire ** ** element and all its children. Paste below it.

__2. Reorder the tabs (the <a> elements) for the newly copied .

```
<div class="tabbed">
<a href="event.html">Event</a> <a href="news.html">News</a> <a
href="safety.html">Safety</a>
<div>
</div>
</div>
```

__3. Save changes.

__4. Refresh the page.

Does everything look good? No! The first tab of the second widget was not selected by default. What went wrong?

Look at the code that selects the default tab:

```
$("div.tabbed a").first().click();
```

It selects the first one of all <a> elements in the page that are children of . This is why the tab of the second widget is not getting selected.

__5. Change the line that selects a tab by default to:

```
$("div.tabbed a:first-child").click();
```

Here, $("div.tabbed a:first-child") will select every first <a> child element of . That is perfect for us.

__6. Save changes.

__7. Refresh the browser.

Now, default tab for both widgets will be selected.

__8. Work with both tabbed sheet widgets. Make sure that each manages its own state (selected tab) separately.

__9. Close all editors and browsers.

Part 7 - Review

With only a few lines of code, we created a pretty complete tabbed sheet widget. We used the load() function to fetch HTML data using Ajax. We also learned to use the data() function to manage state.

Lab 18 - Submitting Form Using Ajax

Recall the login application we have developed in form.html.

At this stage, it does a good job validating user input. But, it does not really submit the form in any meaningful way. In this lab, we will use Ajax to submit the form.

Part 1 - View Serialized Data

jQuery provides the serialize() method that creates a request query string from the input elements of a form. You can then use that to form a HTTP request.

__1. Open **APACHE_DIR/htdocs/lab/form.html**.

__2. The form submit handler is currently attached as follows.

```
$("form").submit(checkForm);
```

__3. Change the code as follows.

```
$("form").submit(function(){
          if (checkForm()) {
                    var data = $(this).serialize();
                    alert(data);
          }
          return false;
});
```

Note that the function must return false at all times to prevent the browser from submitting the request. We will use Ajax to send the HTTP request.

__4. Now remove the following code so we don't have to deal with the validation of the fields anymore when you leave the field, for the following testing this will help not having this validation. You can leave it if you want, doesn't change the behavior of the serialized method that we just created.

```
$("input[type='text'], input[type='password']").blur(function() {
            checkForm();
});
```

__5. Save the file.

__6. Open a browser and enter the URL:

http://localhost/lab/form.html

__7. Enter some data in both fields and submit the form. The alert dialog will show how the input data is serialized.

__8. Click OK.

Part 2 - Submit the Form Using Ajax

Now, we will post the form to a server side script.

__1. Copy **C:\LabFiles\login.exe** to **APACHE_DIR/cgi-bin**

__2. In the browser, test the script by entering the URL:

```
http://localhost/cgi-bin/login.exe
```

```
<!-- Form data was:

-->
Invalid user name. Please try again.
```


__3. Back in form.html, replace the line:

```
alert(data);
```

With:

```
$.get("/cgi-bin/login.exe", data, function(reply) {
          alert(reply);
});
```

The first argument to $.get() is the URI that is requested. The second one contains the input data. The third one is a function that is called when a reply is successfully obtained.

The $.post() method works the same way. Unfortunately, our CGI script only works with GET requests.

__4. Save changes.

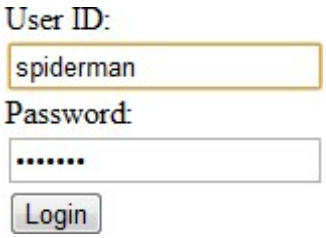
Part 3 - Test

The login script will only successfully authenticate user "bob" with password "bobcat".

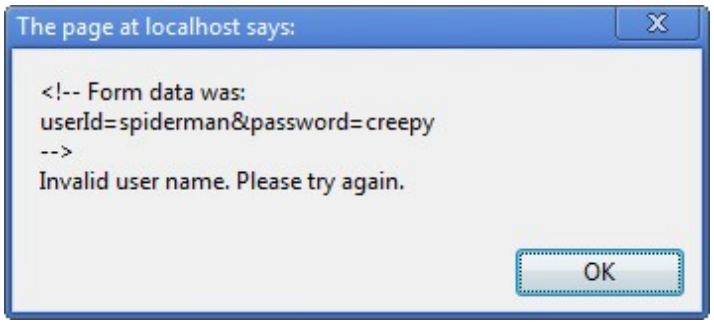
__1. In the browser, enter the URL:

`http://localhost/lab/form.html`

__2. Enter an invalid user ID and password.



__3. Submit the form. The alert window will look like this.



__4. Click OK.

__5. Enter valid user Id (bob) and password (bobcat) and submit the form.

__6. Click OK.

Part 4 - Show the Reply Data

Right now, we are using alert() to show the reply message. Now, we will show it in HTML.

__1. Above the line:

```
<form method="post" >
```

Add:

```
<div id="replyMsg"></div>
```

__2. In the JavaScript code, replace the line:

```
alert(reply);
```

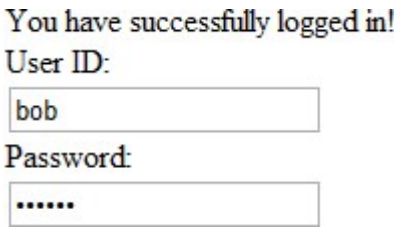
With:

```
$("#replyMsg").html(reply);
```

This will show the reply from the server in .

__3. Save changes.

__4. Refresh the browser and test the form.



Part 5 - Show Progress Indicator

Right now, there is no visual indication that an Ajax request has been sent. This can confuse the user. We will fix that now.

__1. Below the line:

```
//jQuery code here
```

Add:

```
$(document).ajaxStart(function() {
    $("#replyMsg").text("Please wait...");
});
```

The ajaxStart event is fired before any Ajax request is sent by jQuery. We are handling that and showing a progress message.

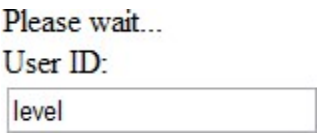
__2. Save changes.

__3. Close all open browsers. We do not wish to hit the browser cache.

__4. Open a new browser and enter the URL:

`http://localhost/lab/form.html`

__5. Enter some data and submit the form. Make sure that you see the "Please wait" message as shown below and after a moment the validation login message.



In real life, you can use an animated image file to show the progress.

Part 6 - Error Handling

When an HTTP request is submitted using Ajax, the browser is not responsible for error handling. The application code must display a message.

__1. Below the line:

```
//jQuery code here
```

Add:

```
$(document).ajaxError(function(event, request, settings){
   $("#replyMsg").html("<b>Error requesting page " + settings.url +
"</b>");
});
```

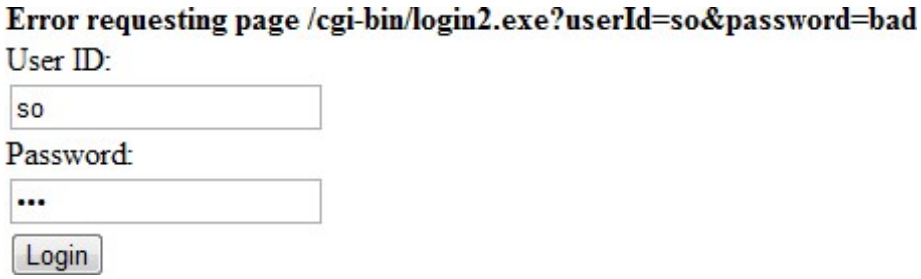
__2. Change the URI for the script to a wrong one.

```
$.get("/cgi-bin/login2.exe", data, function(reply) {
```

__3. Save changes.

__4. Refresh the browser.

__5. Enter some data and submit the form.



__6. Close all editors and browsers.

Part 7 - Review

In this lab, we learned how to use the $.get() method to make an Ajax request. The $.post() method works the same way. We also learned how to show progress indicator and do error handling.

Lab 19 - Build a Drag and Drop Application

In this lab, we will learn how to implement Drag & Drop interactions in your web pages using jQuery UI JavaScript library. The functionality of the lab is built around two jQuery UI interactions : Draggable and Droppable that allow creation of dynamic and interactive web pages and applications using the familiar desktop drag & drop visual paradigm.

Part 1 - Get the Basic Draggable Functionality

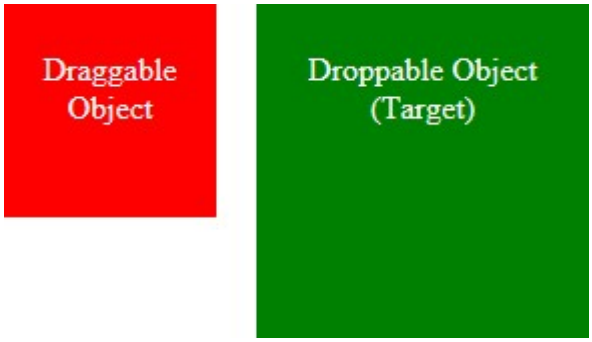
jQuery UI does a great job of aiding developers in creating objects that can be dragged around their web pages with the mouse. Let's see how we can implement this kind of functionality using the jQuery UI goodies bag.

__1. Copy **C:\LabFiles\drag_and_drop.html** to **APACHE_DIR/htdocs/lab/**

__2. Open a browser and enter:

```
http://localhost/lab/drag_and_drop.html
```

You should see a web page with the following content:



If you have not done it yet, do it now: use your mouse and drag the Draggable Object around! Feels great, doesn't it? The power of jQuery UI.

Drop the Draggable Object on the Droppable Object (Target). Nothing happens...

We are going to add animation to this action (more precisely, *drop* event) and demonstrate that Drag & Drop type of actions are possible with jQuery UI.

Part 2 - Adding Droppable Functionality

When you drop a Draggable object in the area of the Droppable one, the droppable call-back function usually provides some feed-back to the user on the acceptance of the object, confirming that the *drop* event, in fact, took place. This can be as simple as visually informing the user of the fact or sending an Ajax call to a back-end system with the user choice in case when there are multiple draggable objects that represent different available options for the user to choose from (e.g. products or services), and so on.

We are going to visually notify the user that the drop event has occurred by styling the Droppable Object's area (which is simply an HTML div section) – we will change the background color from the original green to blue.

__1. Open **APACHE_DIR/lab/drag_and_drop.html** in your editor of choice

__2. Locate the following line:

```
// Add droppable object styling code here
```

__3. Below it paste the following code:

```
$( this ).addClass( "highlighted" );
```

This piece of code will apply the *highlighted* CSS class to the *this* object which represents our Droppable Object (the *this* pointer is very polymorphic: in our context it points to the Droppable object since it sits inside the call-back function that handles the *drop* event and is attached to the Droppable via the "#droppable" selector.

Note: `drop: function( event, ui )` notation denotes that the *drop* event is bound to the event call-back function that takes the *event* object and the *ui* element as its parameters.

__4. Now we need the *highlighted* CSS class.

__5. Locate the following line:

```
   /* Add droppable object's style here */
```

__6. Right below it, add the following line:

```
  .highlighted {background:blue;}
```

The style will set the background color of the target element to blue.

__7. Save changes.

__8. Refresh the browser or enter:

```
http://localhost/lab/drag_and_drop.html
```

__9. Notice, now the background color of the Droppable Object changes when you drop the Droppable Object on it.

Part 3 - Adding Another Event Handler

Right now there is only one event that we are handling, namely, the *drop* event. The API for Draggable and Droppable interactions allows for multiple event handlers. In code, this is manifested by the presence of the curly braces in the .droppable () method's parameter list highlighter below:

```
$( "#droppable" ).droppable({
  drop: function( event, ui ) {
        // Add droppable object styling code here
    $( this ).addClass( "highlighted" );
   }
});
```

jQuery UI web site lists the following supported events for Droppable (http://api.jqueryui.com/droppable/):

```
activate
create
deactivate
drop
out
over
```

Let's incorporate in our animation the *out* event.

__1. Switch back to the editor with **APACHE_DIR/lab/drag_and_drop.html**

__2. Just above the line for the drop event handler:

```
drop: function( event, ui ) {
```


__3. Insert the following fragment:

```
out: function( event, ui ) {
  alert ("I'm out of here!");
},
```

Don't forget the *},* at the end!

You should have the following script section of your page:

```
<script>
$(function() {
  $( "#draggable" ).draggable();
  $( "#droppable" ).droppable({
    out: function( event, ui ) {
              alert ("I'm out of here!");
          },
    drop: function( event, ui ) {
          // Add droppable object styling code here
      $( this ).addClass( "highlighted" );
          }
  });
});
</script>
```

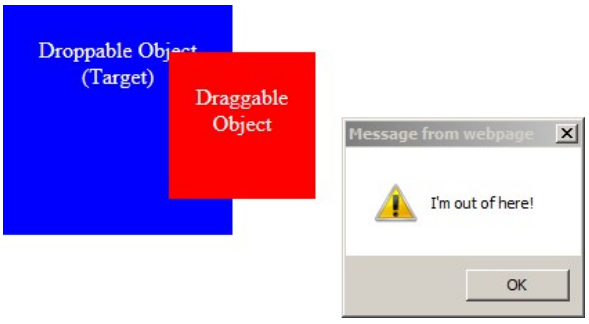
Note: The sequence of event handlers is not important here as the jQuery UI system will find the one that matches the event triggered by the user action automatically.

__4. Save changes.

__5. Refresh the browser or enter: **http://localhost/lab/drag_and_drop.html**

__6. Drop the Draggable on the Droppable. As expected, the background color of the Droppable Object changes from green to blue.

__7. Move the Draggable Object out of the Droppable Object's area. You should be prompted with the Alert dialog that is triggered by the *out* event and handled by the attached event handler we just added



Of course, there is much more that you can do with this powerful API to make your web pages really interactive!

This completes our lab.

__8. Close the editor and browser.

Part 4 - Review

In this lab, we learned how to use Draggable and Droppable interactions offered by the jQuery UI JavaScript library.

Lab 20 - Build a Slide Show Viewer

In this lab, we will use many of the techniques learned from previous labs to build a picture slide show tool. The requirements for the tool are as follows:

1. We should be able to specify the list of images in the HTML page or get it dynamically from an Ajax request.
2. An image can have caption which needs to be displayed.
3. An image needs to be fit within the screen area.

Some of the features of this lab will not work for IE8 or older. Please use FireFox or Google Chrome.

Part 1 - Getting Started

__1. Copy **C:\LabFiles\template.html** to **APACHE_DIR/htdocs/lab**

__2. Rename the file as **slideshow.html**

__3. Open **slideshow.html**

__4. Add a element as follows.

```
<!-- Body HTML here -->
<div id="gallery">
    <img src="#"/>
</div>
```

To use the tool, this is all an HTML designer has to add to the page. The rest will be done from JavaScript.

__5. Add the following styles within the head section.

```
<style type="text/css">
* {padding: 0; margin: 0;}
body {background: black}
#gallery {
          text-align:center;
          margin-top:0px;
          margin-bottom:0px;
          padding:0px;
}
</style>
```

This is pretty basic styling. All we are doing is eliminating and padding or margin space. This way, images will take up the entire screen space.

__6. Save changes.

__7. In a browser navigate to:

```
http://localhost/lab/slideshow.html
```

It will be just a black screen.

Part 2 - The Image List

__1. Copy **C:\LabFiles\images** folder into **APACHE_DIR/htdocs/lab**.

We will now supply the list of images using a JavaScript array object. This can be hard coded in the page or retrieved using Ajax. For now, we will hard code the list.

__2. Above the line:

```
</script>
```

Add:

```
var imageList = [
  {src: "/lab/images/annie.jpg", caption: "Annie the cat"},
  {src: "/lab/images/cave.jpg", caption: "Sea cave"},
  {src: "/lab/images/desert.jpg", caption: "Neat desert scene"},
  {src: "/lab/images/poppy.jpg", caption: "California poppy field"}
];
```

Caveat: Make sure that the last element in the array does not end with a ",". If it does, IE 8 considers that another element follows it. That is, IE will think there are 5 images in the array.

__3. Save changes.

Part 3 - Showing the Images

We will now develop code that shows the next image in sequence within imageList array.

__1. Add the following code.

```
var idx =  -1;

function showImage() {
          ++idx;

          if (idx >= imageList.length) {
                    idx = 0;
          }

          if (idx < imageList.length) {
                    var imgE = $("#gallery img").get(0);

                    imgE.src = imageList[idx].src;
          }
}
```

Nothing too complicated. Except, we had to get the DOM object for the <img> element to set its src property. We did that using $("#gallery img").get(0).

Every time showImage() is called, the next image from the list will be displayed. We will now call that function when the DOM is ready to show the first image by default.

__2. Below the line:

```
//jQuery code here
```

Add:

```
$("#gallery img").click(showImage);
showImage(); //Show first image
```

Here, we are setting up the onclick event handler for the img element. When the image I clicked, we will go forward and show the next image.

__3. Save changes.

__4. Refresh the browser or enter the URL:

http://localhost/lab/slideshow.html

Make sure that the picture is centered. That is what we had set in the style for the gallery div.

__5. Click the image. It should show the next image and so on. Eventually, the application will wrap around and show the first image.

Part 4 - Scaling Images

Our slide show is basically functional. One problem is that the larger images do not fit within the browser window. Browser deals with that by showing scrollbars. This can be annoying for the users. We will now scale the images down to fit the window.

__1. Add a new function that will scale images.

```
function sizeImage() {
    var imgE = $("#gallery img").get(0);

    var yScale = imgE.height / window.innerHeight;
    var xScale = imgE.width / window.innerWidth;

    //Pick the maximum scale
    if (yScale >= xScale) {
        //Height needs to be fit
        imgE.style.height = window.innerHeight;
        imgE.style.width = 'auto';
    } else {
        //Width needs to be fit
        imgE.style.width =  window.innerWidth;
        imgE.style.height = 'auto';
    }
}
```


This function basically scales an image only as much it has to show the entire content within the screen. It preserves the aspect ratio.

But, when should we call this function? We are dynamically setting the src attribute of the img element. Images are loaded asynchronously. Unless the image is fully loaded, the browser has no idea about the width and height of the source image.

The solution is to attach an onload event handler for the img element. This handler will be called when the image is fully loaded. From there we can safely resize the image.

__2. In the DOM ready function, attach an onload event handler for the img element.

```
$(function(){
            //jQuery code here
            $("#gallery img").load(sizeImage);
            $("#gallery img").click(showImage);
            showImage();
});
```

From the showImage() function, we are setting the src attribute to display another image. The height and width attributes of the image will remain same as the previous image. This can throw off the scaling calculation. To get the true width and height of the image, we must set the values to "auto". We will do that now.

__3. Change the showImage() method as shown in bold face below.

```
if (idx < imageList.length) {
    var imgE = $("#gallery img").get(0);
    imgE.style.height = "auto";
    imgE.style.width = "auto";
    imgE.src = imageList[idx].src;
}
```

Now from sizeImage(), we will get the true width and height of the image.

__4. Save changes.

__5. Make sure that you are using FireFox or Chrome. Refresh the browser. Now, as you click through the images, entire images will be shown in the screen. Try resizing the browser size. Clicking the image will show the next image that will fit in the window.

The scaling code shown here does not work in IE8 or older.

There is a slight problem. When an image is smaller than the screen, we scale it up. Scaling images up make them look bad. We should only scale images down but never up.

The solution is simple.

__6. Change the scaling code as shown in bold face.

```
//Pick the maximum scale
if (yScale >= xScale) {
    if (yScale > 1) {
        //Height needs to be fit
        imgE.style.height = window.innerHeight;
        imgE.style.width = 'auto';
    }
} else {
    if (xScale > 1) {
        //Width needs to be fit
        imgE.style.width =  window.innerWidth;
        imgE.style.height = 'auto';
    }
}
```

__7. Save changes.

__8. Maximize the browser so that some of the images are smaller than the screen size.

__9. Refresh the browser. Now, smaller images will not be scaled up. but, large images will be scaled down.

Part 5 - Showing the Caption

We will like to show the caption at the bottom of the image. No matter how high the image is, the caption should show at the same spot relative to the bottom of the image. See the desired result below.

__1. Below the img element add another div for the caption.

```
<div id="gallery">
    <img src="#"/>
    <div>Caption here</div>
</div>
```

__2. Add the following style.

```
#gallery div {
    width: 300px;
    position: relative;
    top: -50px;
    left: 50%;
    margin-left: -150px;
    border-width: 2px;
    border-color: white;
    border-style: solid;
    border-radius: 5px;
    z-index: 99;
    color: white;

    text-shadow: black 2px 2px 0px;
    font-family: Sans-serif;
    font-style: bold;
    font-size: 1.5em;
}
```


Here, we are positioning the caption div relative to where it would be normally positioned. It's normal position will be right below the img element. We are moving it up setting "top: -50px". This way, the caption will be shown right on top of the image. It is important that we set "z-index: 99" so that the image does not hide the caption. The "left" and "margin-left" styles are cleverly set so that the div is centered on the screen.

__3. Save changes.

__4. Refresh the browser.

__5. Click through the images and make sure that the caption appears at the same spot relative to the bottom of the image.

__6. Now, we will dynamically set the caption. At the very end of sizeImage() function, add these lines:

```
var caption = $("#gallery div");
caption.text(imageList[idx].caption);
caption.fadeIn(300).delay(3000).fadeOut(300);
```

Note how fadeIn, delay and fadeOut are chained together. After the caption is faded in, jQuery will wait for 3 seconds. Then the caption will be faded out.

__7. Save changes.

__8. Refresh the browser. Now the caption will pop open and then disappear after a few seconds.

Part 6 - Getting Image List Using Ajax

We can easily convert the application to get the image list dynamically using Ajax.

__1. Copy **C:\LabFiles\image_list.txt** to **APACHE_DIR/htdocs/lab folder**.

__2. Open the file and notice how the data is specified there as JSON.

```
[
  {"src": "/lab/images/annie.jpg", "caption": "Annie the cat"},
  {"src": "/lab/images/cave.jpg", "caption": "Sea cave"},
  {"src": "/lab/images/desert.jpg", "caption": "Neat desert scene"},
  {"src": "/lab/images/poppy.jpg", "caption": "California poppy field"}
]
```

Important: jQuery's handling of JSON data is extremely picky. Note that the property names "src" and "caption" are in double quotes. If you do not quote them or use single quote, the $.getJSON() method will fail.

__3. Back in slideshow.html, set the imageList variable to null.

```
var imageList = null;
```

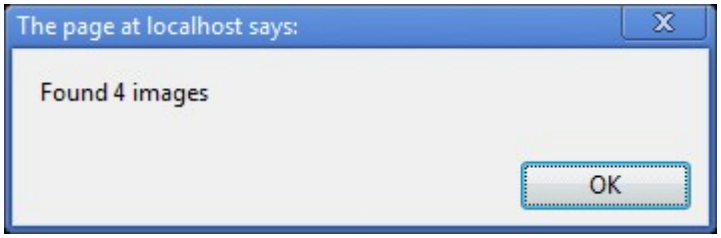
__4. In the DOM ready handler function, comment out showImage(). In it's place use the $.get() method to load the data. This is shown below in bold face.

```
$(function(){
    //jQuery code here
    $("#gallery img").load(sizeImage);
    $("#gallery img").click(showImage);
    //showImage();
    $.getJSON("/lab/image_list.txt", function(data) {
        imageList = data;

        alert("Found " + imageList.length + " images");
        showImage();
    });
});
```

Note: In this example, we have static data in image_list.txt. In real life, the list will be dynamically created from database.

__5. Save changes.

__6. Refresh the browser. You should see this message.



Rest of the application will work same as before.

__7. Close all.

Part 7 - Review

In this lab, we built a full screen slide show application. We paid a lot of attention to scale large images down to size so that the whole image fits in the screen. We avoided scaling small images up.

We used fade in and fade out transitions to show the caption.

Finally, we got the image list data from the server using JSON format. We used the $.getJSON() method for that. Keep in mind, for $.getJSON() to work, the JSON data must be formatted according to the specification.

Lab 21 - Develop a Simple Plugin

In this lab, we will develop a very simple plugin. The goal is to understand the basic mechanism of writing a plugin.

Initially our plugin will do nothing more than open an alert dialog. This is a "hello world" type simple plugin.

Part 1 - Create the Plugin File

In this part, we will create the JavaScript file for the plugin.

__1. In the **APACHE_DIR/lab** folder create a new file called **simple_plugin.js**

__2. Open **simple_plugin.js**

__3. Write the self calling function that wraps the plugin code.

```
(function($){
    //Plugin code below

}(jQuery));
```

This creates an anonymous function that is called immediately. The function takes as parameter the jQuery global object. From within the function, the jQuery object can be accessed as $ since that is the name of the input variable. This is just a shortcut way of referring to the jQuery object. We take the trouble of writing this wrapper function so that the variable name $ will not collide with any global variable by that name. Other toolkits like Prototype may declare a variable by name $.

If you don't wish to use the short cut name for jQuery object, you don't really need this wrapper code.

__4. At the heart of the plugin code is registration of a new plugin function in the prototype of the jQuery object. Write the lines in bold face.

```
(function($){
    //Plugin code below
    jQuery.prototype.simplePlugin = function() {
        alert("Cool plugin");
    };
}(jQuery));
```

This will dynamically add a new method to the jQuery object called simplePlugin.

__5. In the spirit of the brevity of jQuery, we will now use a few short cuts. First, as we already discussed, jQuery global variable can be replaced with $. Hence, the line can now become.

```
$.prototype.simplePlugin = function() {
```

__6. Finally, jQuery toolkit creates a property for the jQuery object called "fn" that is nothing other than its prototype. So, we can further shorten the code as follows.

```
$.fn.simplePlugin = function() {
```

If you are a JavaScript purist and don't wish to use synonyms like fn, feel free to keep using the fully spelled out code we entered first. But, most plugin developers use the shortened version.

__7. Your code should look like this:

```
(function($){
    //Plugin code below
     $.fn.simplePlugin = function() {
        alert("Cool plugin");
    };

}(jQuery));
```

__8. Save the file.

Part 2 - Use the Plugin

__1. Copy the file **C:\LabFiles\template.html** to **APACHE_DIR/lab**

__2. Rename the file as **testplugin.html**

__3. Open **testplugin.html** in an editor.

__4. Import the new JavaScript file for your plugin as shown in bold face.

```
<script type="text/javascript" src="/jquery/jquery-1.10.2.min.js">
</script>

<script type="text/javascript" src="simple_plugin.js"></script>
```


__5. Below the line:

```
//jQuery code here
```

Call the plugin function:

```
$().simplePlugin();
```

Here, we are calling the simplePlugin() function of the jQuery object. We added that function to the prototype of jQuery in the previous part.

In the $() function, we provide no selector rule as input. Hence, the function will return a jQuery object with empty collection of DOM element. That is fine for us.

__6. Save changes.

Part 3 - Test

__1. Open a browser and enter the URL:

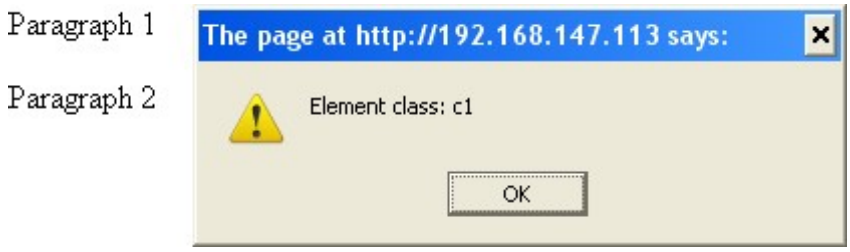
http://localhost/lab/testplugin.html

__2. You should see the alert dialog. This proves that our little plugin is working fine.

Part 4 - Do Something Useful Now

Our plugin is working but does nothing useful to anyone. We will now extend the plugin code. The business logic will be as follows.

Our plugin will help us find out the name of the CSS class associated with any DOM element in the page. When you click on an element, the plugin will show the class name in an alert dialog.



This will show us how to handle event from a plugin and how to iterate over all DOM elements selected by a jQuery selector.

__1. In **simple_plugin.js**, comment out the line:

```
//alert("Cool plugin");
```

__2. Below that line, enter:

```
this.each(function() {
    $(this).click(function(e) {
        alert("Element class: " + this.className);
    });
});
```

Here, we are using "this" keyword a lot. Depending on the context, "this" either points to a jQuery object or a DOM element. In the first line (this.each()), "this" points to the jQuery object. The each() function iterates over all selected DOM elements and calls the iteration handler function for every element.

From the iteration handler function, we are calling $(this).click(). Here, "this" is the DOM element object for which the the iteration handler function is called. Hence, the jQuery wrapper object for it is $(this). By calling click() function we are registering the onclick event handler. Within the event handler function, "this" points to the DOM element where the mouse is clicked. We can use the className property of the DOM element.

__3. Save changes.

__4. In the **testplugin.html** file, add the lines in bold face. This will add a few elements to the body.

```
<!-- Body HTML here -->
<p class="c1">Paragraph 1</p>
<p class="c2">Paragraph 2</p>
```

Feel free to add any more elements and give them a class name.

__5. Now, when we call our plugin function, we can supply a selector rule. Change the code as follows.

```
//jQuery code here
$("p").simplePlugin();
```

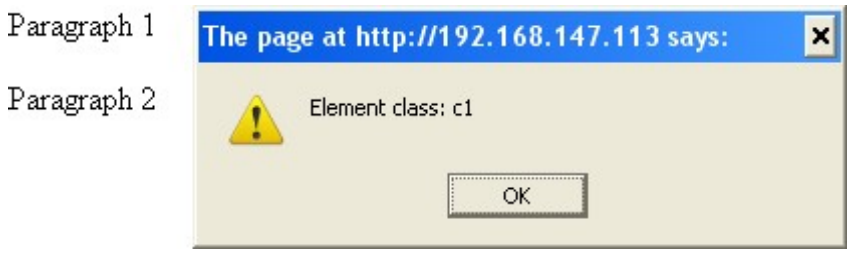
This will show the class of all "p" elements. If you wish to peek at every element, you can also use $("*").simplePlugin().

__6. Save changes.

Part 5 - Test

__1. Refresh the browser.

__2. Click one of the paragraphs. Make sure the alert dialog shows the class of the element.



__3. Close all.

Part 6 - Review

In this lab we developed a very simple plugin. We learned how to iterate over each selected DOM element and then attach an event handler. This is at the core of what most plugins need to do. You can then use jQuery API to create amazing plugins.

Lab 22 - Media Queries and Responsive Design

Following the principles of responsive web design, we want to adjust the style of a page dependent on the environment it is being viewed in. There are various techniques, including querying the browser, that will be explored in this lab.

Part 1 - Verify Files

In a previous lab you should have copied several files to a location where the Apache web server can locate them and return them as web pages. This section will verify you have the correct files.

__1. Open a Windows Explorer file browser window.

__2. Navigate to the '**htdocs**' folder of where the Apache web server was installed. This should be the '**C:\Apache2.2\htdocs**' folder.

__3. Verify that the following sub folder has been copied to where the Apache server is installed. There will be various files within this subfolder.

```
C:\Apache2.2\htdocs\css3\media
```

Note: If you do not see this folder, you can copy this from the 'C:\LabFiles\css3' folder or extract the appropriate lab solution for the previous lab from the 'C:\LabFiles\Solutions' folder. Be careful which files you copy and where you place them to make sure the web server can locate them with the URLs provided in the labs.

Part 2 - Reset Styles

One technique that can help provide a more consistent design across different types of browsers is a "style reset". Rather than using the default properties of some page elements that may be different between browsers, and therefore make the page look different, we will apply a style sheet to "reset" many of these properties by setting them explicitly instead of using the browser default values. Then when other style sheets apply property values to modify the style of the page they will be working from a consistent baseline no matter the browser being used.

__1. Open a web browser and go to the following URL:

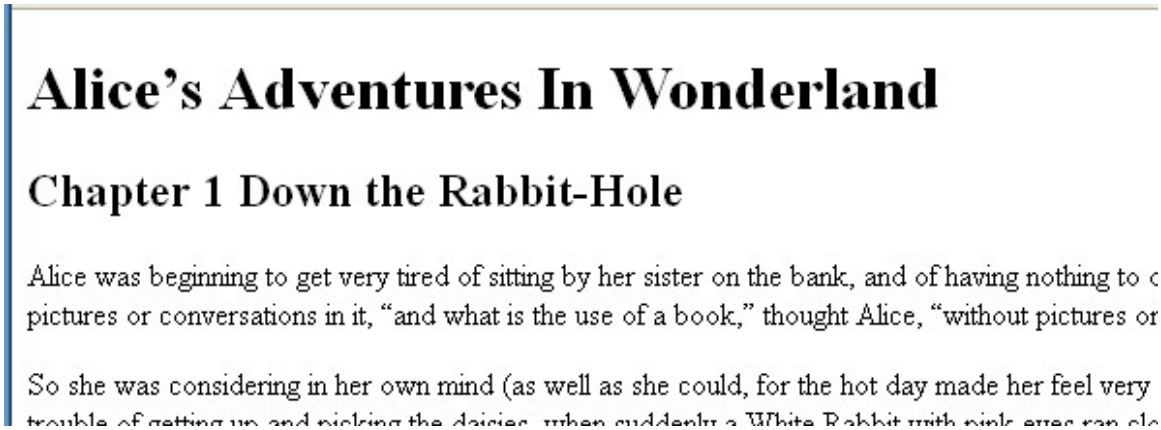
```
http://localhost/css3/media/index.html
```

Note: The page in this example uses some HTML5 elements that may not be recognized directly in older Internet Explorer browsers. This might cause the styles to not be displayed properly to some elements. You will add JavaScript in future steps to fix this.

If you do try testing with Internet Explorer and don't see the behavior the style should be creating try testing with Firefox or Chrome to see if it is different.

Screenshots in the lab will show how things should appear in Firefox.

__2. Get familiar with the current style of the page so you can notice as changes are made.



__3. Leave the browser open to the page as you will refresh it later.

__4. Run the Notepad++ program or a regular text editor.

```
C:\Software\NotepadPlus\notepad++.exe
```

Note: A file may appear since Notepad++ opens the last used file. To avoid this in future, close the file first and then close Notepad++.

__5. Once you have the **Notepad++** program running (or another text editor if you prefer), open the following file.

```
C:\Apache2.2\htdocs\css3\media\cssreset-ericmeyer.css
```

__6. Don't make any changes but notice that this style sheet has many properties that are being "reset" to certain values. These are the properties that might have different default values between browser types so applying this style sheet will override these defaults so they are consistent.

```
table, caption, tbody, tfoot, thead, tr, th,
td {
    margin: 0;
    padding: 0;
```

__7. Close the style sheet without making any changes.

__8. In the Notepad++ program or a regular text editor, open the following file:

```
C:\Apache2.2\htdocs\css3\media\index.html
```

__9. Near the top of the page and after the <title> but before the end of the </head> section, add the link to the reset style sheet as shown in bold below.

```
<title>Alice&#8217;s Adventures in Wonderland</title>
<link type="text/css" rel="stylesheet" media="all"
          href="cssreset-ericmeyer.css" />

</head>
```

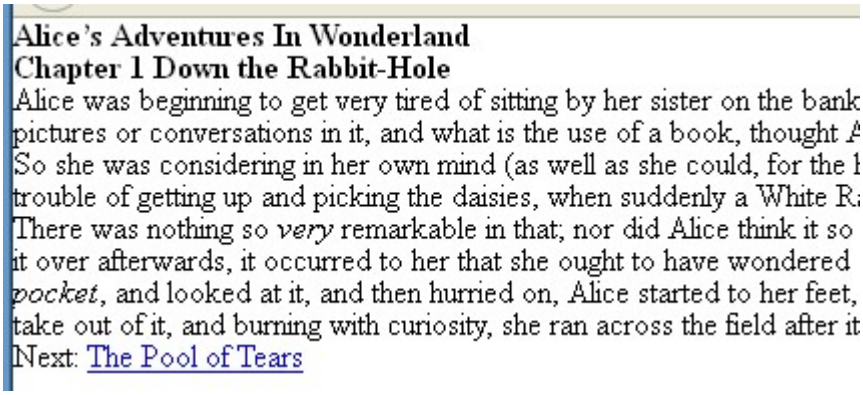
Note: You will always want any "reset" style sheet to be the first style sheet referenced. This is so other style sheets or <style> tags can override the reset values if needed.

__10. Save the file but leave it open.

__11. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

```
http://localhost/css3/media/index.html
```

__12. What you should see is that the styles of the page are very different because of the "reset" style sheet.



Note: Remember the goal of the "reset" style sheet is not necessarily to make the page readable and styled well. You will apply other styles later to override these reset values and provide other styles to improve the page.

__13. Leave the browser open to the page as you will refresh it later.

__14. Return to the editor that was editing the 'index.html' file.

__15. Add the following bold code after the "reset" style sheet but again before the end of the </head> section. This will add a link to the "default" style sheet to apply to the page.

```
<link type="text/css" rel="stylesheet" media="all"
          href="cssreset-ericmeyer.css" />
<link rel="stylesheet" media="screen" href="default.css" />

</head>
```

__16. Save the file but leave it open.

__17. Return to the browser that was showing the page and refresh the page. You should see that the page is now styled with some basic styles that should make it easy to read no matter what format the browser displays it as.

__18. Leave the browser open to the page as you will refresh it later.

Part 3 - Enable IE "Fix"

Although it is not a good idea to try and adjust for every type of browser, Internet Explorer is often different enough from the other browser types that it may warrant this treatment so that you can support users who use it. Also, versions of Internet Explorer before IE 9 did not have support for HTML5 but there is a JavaScript "trick" we can use to add support so the HTML5 tags can at least be recognized.

__1. Return to the editor that was editing the 'index.html' file.

__2. Add the following bold code after the existing style sheets but again before the end of the </head> section. This will be viewed as a comment by all browsers except for Internet Explorer. For IE 8 and below it will load a JavaScript file that will trick Internet Explorer into recognizing HTML5 tags it would not normally recognize.

```
<link type="text/css" rel="stylesheet" media="all"
          href="cssreset-ericmeyer.css" />
<link rel="stylesheet" media="screen" href="default.css" />

<!--[if lte IE 8]>
    <script type="text/javascript" src="../script/HTML5forIE.js">
    </script>
<![endif]-->

</head>
```

Note: This page may not have as many troubles in Internet Explorer 8 and earlier before this fix because even though there are HTML5 tags there are also normal <p> tags for the main text. Other pages that use HTML5 may have major elements missing completely unless you use this trick to let IE 8 and earlier recognize the HTML5 tags.

__3. After the first IE conditional block, add the second IE conditional block as shown in bold below. Again make sure it is before the end </head> tag. The style sheet that is imported for IE will make sure many of the HTML5 tags have the 'display: block;' property set so the default text flow will be consistent with other browsers.

```
<!--[if lte IE 8]>
          <script type="text/javascript"
                    src="../script/HTML5forIE.js"></script>
<![endif]-->

<!--[if IE ]>
    <style type="text/css" media="all">
              @import url("ie-fix.css");
    </style>
<![endif]-->

</head>
```

__4. Save the file but leave it open.

__5. If you are not already using it, open Internet Explorer to the following location. Refresh the page if you were already using Internet Explorer.

http://localhost/css3/media/index.html

__6. The only change you will likely see is that the main <h1> tag changes in color from purple to pink. This was added to the 'ie-fix.css' file just to prove that the style sheet is loaded by Internet Explorer.

__7. If you want you can open or refresh the page in Firefox or Chrome and notice there is no change from before.

Part 4 - Use Media Queries

The key element to a "responsive design" is being able to adapt the same content to various ways that it could be displayed. This is most often based on elements like device width, ratio, etc. To determine when some conditions might indicate you should display a page differently you can use "media queries". These were added in CSS3 and are becoming increasingly important since a wider variety of devices, particularly mobile and tablet devices are expanding the range of how users will view a web site.

__1. Return to the editor that was editing the 'index.html' file.

__2. Add the following bold code before the <title> tag but still inside the starting <head> tag. This will add the "viewport" meta tag to the page.

```
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Alice&#8217;s Adventures in Wonderland</title>
```

Note: You could also set values for the 'minimum-scale', 'maximum-scale', and 'user-scalable' properties but this might limit the ability of a user on a small device to zoom in or out.

__3. Add the following bold code after the IE conditional style but before the end </head> tag. You can split the media query across multiple lines as long as the entire value is surrounded by a single set of quotes.

```
<!--[if IE ]>
    <style type="text/css" media="all">
              @import url("ie-fix.css");
    </style>
<![endif]-->

<link rel="stylesheet" media="screen and (min-width: 740px)
          and (min-device-width: 740px), (max-device-width: 800px)
          and (min-width: 740px) and (orientation:landscape)"
          href="medium.css" />

</head>
```

Note: In the code above there are actually two different media query expressions separated by a comma. This is the same effect as having two separate <link> tags with one expression each both linking to the same style sheet.

The 'screen' value is only part of the first expression so the second expression mainly looks for width and landscape orientation, most likely for a tablet.

__4. Save the file but leave it open.

__5. If you are not already using it, open Firefox or Chrome to the following location. Refresh the page if you were already using Firefox or Chrome.

```
http://localhost/css3/media/index.html
```

__6. You should notice some different colors and some additional space between the text and the image.

Note: Since media queries were added with CSS 3, earlier versions of Internet Explorer will not support them. It is important that the "default" styles applied to a page will make the content presentable no matter what. The page can then be "enhanced" with media queries for the browsers that will support the features being added.

__7. Leave the browser open to the page as you will refresh it later.

__8. Return to the editor that was editing the 'index.html' file.

__9. Add the following bold code after the first media query <link> tag but again before the end </head> tag. You can split the media query across multiple lines as long as the entire value is surrounded by a single set of quotes.

```
<link rel="stylesheet" media="screen and (min-width: 740px)
          and (min-device-width: 740px), (max-device-width: 800px)
          and (min-width: 740px) and (orientation:landscape)"
          href="medium.css" />
<link rel="stylesheet" media="screen and (min-width: 980px)
          and (min-device-width: 980px)" href="large.css" />
</head>
```

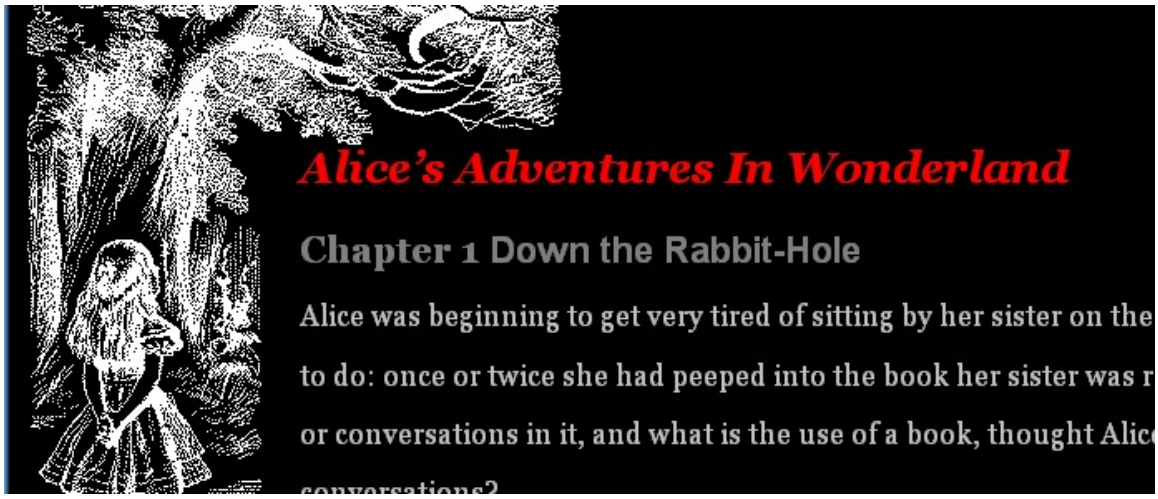
Note: Remember that the order of how styles are defined can matter when two style sheets or <style> tags have the same property. This follows a "last one wins" rule so you want the styles that may set default properties or be overridden to come first and other styles that may override these style properties to come later. In this case anything in the 'large.css' file will override anything that sets a matching property before it which is what we want.

__10. Save the file but leave it open.

__11. Return to the Firefox or Chrome window you were using and refresh the page. Open the following location with Firefox or Chrome if you had closed it. Maximize the browser.

```
http://localhost/css3/media/index.html
```

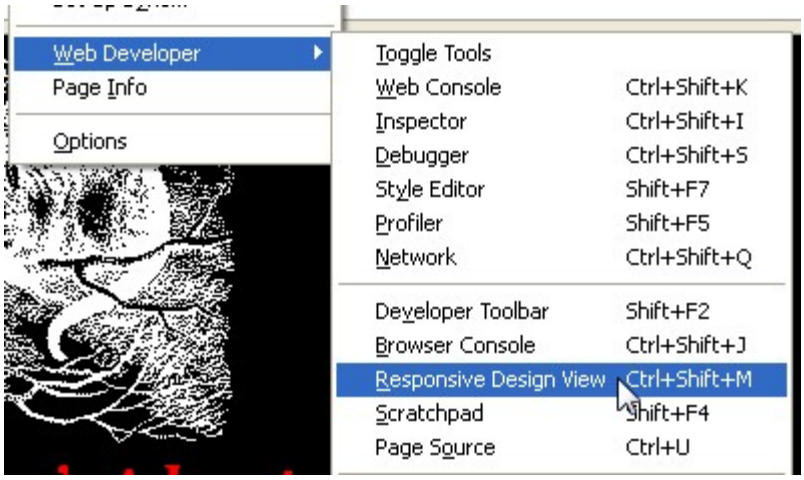

__12. You should now see a very different layout and style for the page. Make sure your window is maximized and check your screen resolution if you do not see the change. This is what would be shown perhaps on "normal" web browsers on a computer desktop.



Note: You might not normally make such drastic changes with colors and images between the different layouts for different size screens. This was done just to make it more obvious when changes were taking place. Normally you might change things like where navigation links are placed, how text flows on the page, or how many elements appear side by side on the page.

__13. Open the page in Firefox if you do not already have it open.

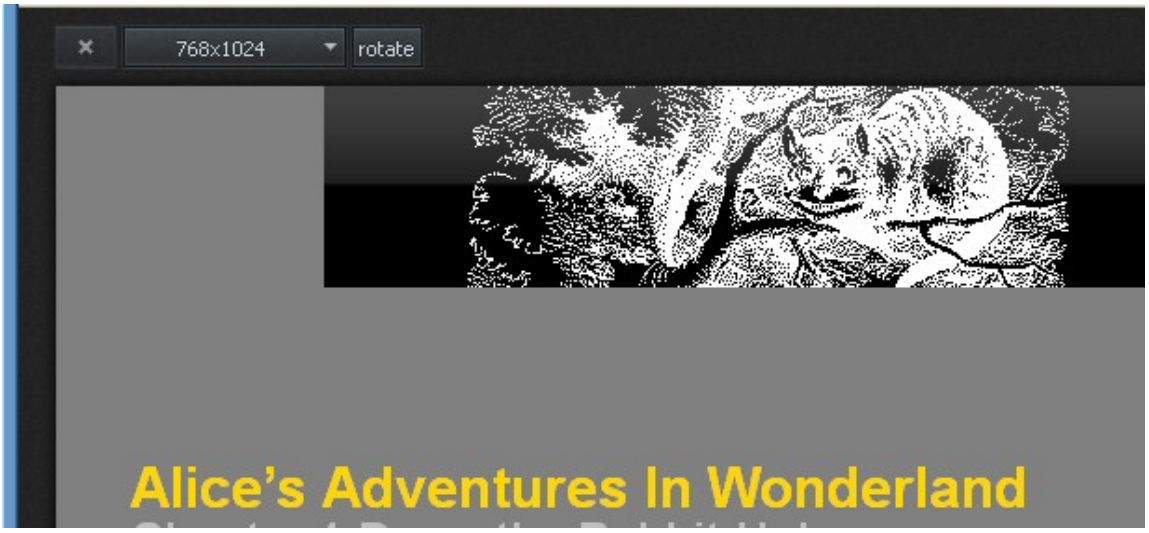
__14. In Firefox, select '**Tools → Web Developer → Responsive Design View**'. You may also need to click the 'Firefox' menu in the upper left to see the options below. Still another option could be just pressing the **Ctrl-Shift-M** short-cut.



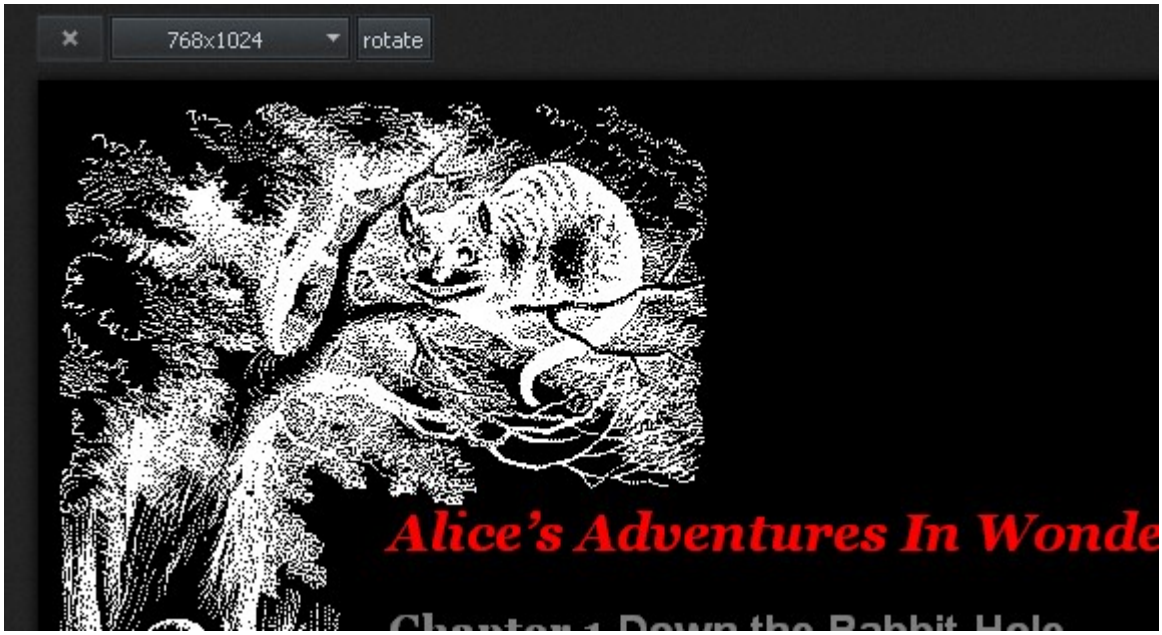
Note: You can also do similar testing in other browsers simply by resizing the window but this tool in Firefox makes it very easy and you can also test exact screen sizes.

__15. Use the drop-down at the top of the window to select the '**360x640**' option. The page should be displayed with the default styles as both media queries will now be false. This is what the page might look like on a phone.

__16. Use the drop-down to select the preset of '**768x1024**'. You should see the page displayed with the "medium" styles.



__17. Click the '**rotate**' button next to the drop-down list. The value won't change but the page should be wider and it should now be displayed with the "large" styles.



__18. Click the **X** at the top left of the window next to the drop-down for screen sizes. This will close the responsive design view.

Part 5 - Style for Print (optional)

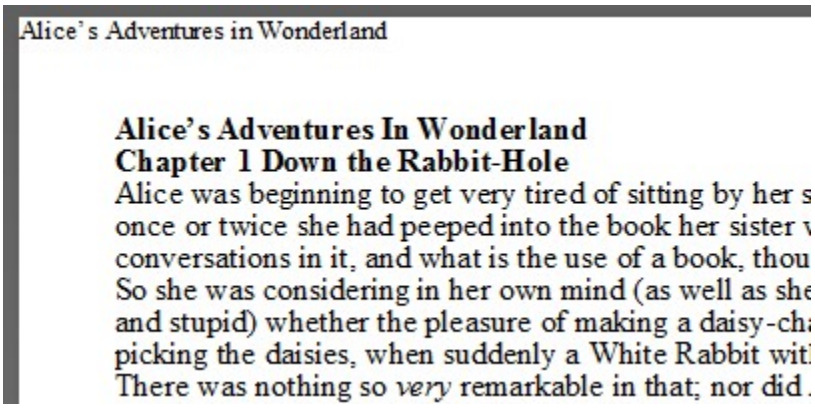
You can also add styles for how a page will display when printed. You will not be able to do this part if the computer you are using doesn't have at least one printer configured.

__1. Return to the browser that was showing the page and refresh the page. Open a new one to the following URL if you closed it:

```
http://localhost/css3/media/index.html
```

__2. Depending on the browser you are using open the 'Print Preview' of the page. Ask other students or the instructor if you are not sure how to do this.

__3. What you should see is that the page is displayed with the "reset" styles only being applied. This is because all of the other styles were applied to the 'screen' media which is the visual display of the page.



__4. Close the print preview view but leave the browser open.

__5. Return to the editor that was editing the 'index.html' file.

__6. Add the following bold code after the two media query <link> tags but again before the end </head> tag.

```
<link rel="stylesheet" media="screen and (min-width: 980px)
          and (min-device-width: 980px)" href="large.css" />
<link rel="stylesheet" media="print" href="print.css">
</head>
```

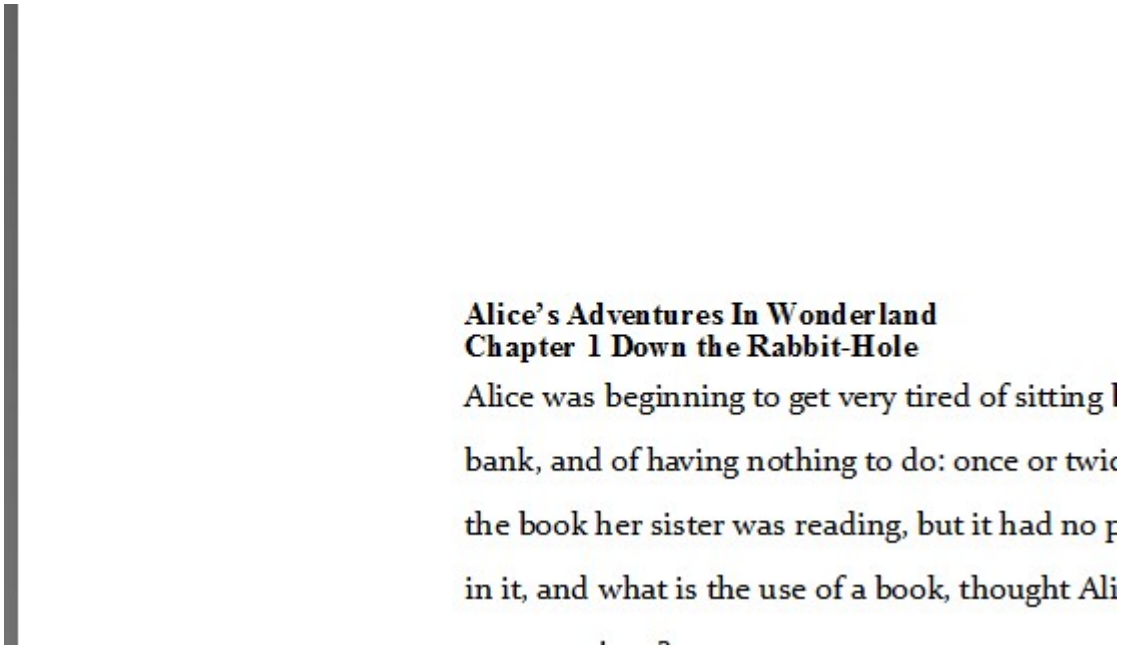
__7. Save the file but leave it open.

__8. Return to the browser that was showing the page and refresh the page.

__9. What you should see is that the page doesn't change because the styles added only apply to the '**print**' media and not the 'screen' which is the visible window.

__10. Again open the print preview view.

__11. You may or may not see an image in the upper left but you should at least be able to tell that the styles are different for how the page would be printed.



__12. Close all open browsers.

__13. If you are using Notepad++ then close all open files and then close Notepad++.

Part 6 - Review

The main goal of a "responsive design" is to present the same content but adapt it to how the user is viewing it. Besides providing a better user experience, it allows for the different "versions" of the web site to be more consistent with each other without having to maintain more than one source of the content. By applying different style properties and using media queries to decide how they should be applied the site can adapt to different viewing conditions.

Lab 23 - Responsive Layout

In this lab, we will work through a simple an effective technique that will help us build a responsive layout of a web page. This technique is based on the CSS float property which has been around for quite a long time and recently has become a popular choice of web designers tailoring their web pages for devices with small screens.

Part 1 - Get Started

__1. Copy **responsive_layout.html** from **C:\LabFiles** to **APACHE_DIR/htdocs/lab**:

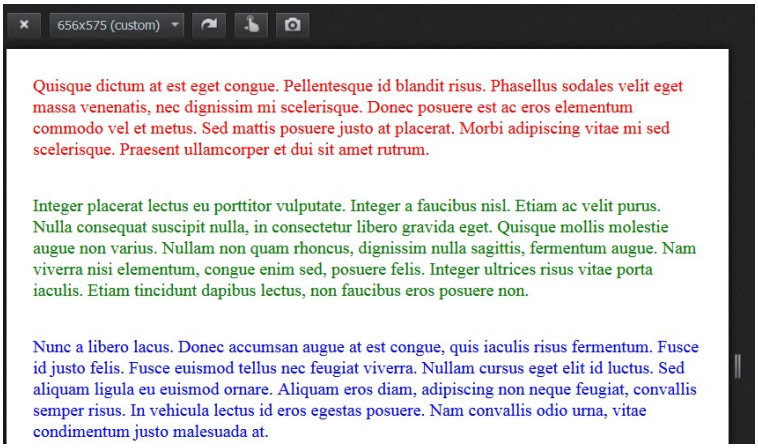
Part 2 - The First Run

__1. Open an instance of Firefox and navigate to:

```
http://localhost/lab/responsive_layout.html
```

__2. In Firefox, press **Ctr-Shift-M** to activate the Responsive Web Design View of the loaded page.

You should see the following page content:



Resizing the window (by dragging the two little parallel bars located just off the middle of the right side of the view) has no effect on the layout of the page - all three sections shown in three colors (red, green and blue) are invariably shown stacked.

We would like to have a layout that would present the three sections as separate in-line columns on wider screens, dropping a right-most column when the screen becomes narrower.

Part 3 - Make the Page Responsive

__1. Open the **APACHE_DIR/htdocs/lab/responsive_layout.html** page in your editor.

As you can see, the page is as simple as it gets. Nothing exciting about it - just random text generated by the Lorem Ipsum web site placed in the standard **div** elements.

Let's "columnize" the page layout (if there is such a word at all for creating columns on the page).

__2. Enter the following code just above the </style> tag:

```
.floatLeft {
          max-width: 330px;
          float: left;
}
```

This CSS style rules set effectively says that each element styled with it should have a maximum width of 330 pixels and whenever it is not possible to accommodate it on the screen, drop and flush it to the left of the screen.

Note: It would be better to convert the absolute dimensions expressed in pixels into relative **em**'s, but here we are trying to keep it simple for seeing the dynamic screen widths in Firefox's Responsive Design View which allows only for pixel measurement units.

__3. Update each div section's class by adding **floatLeft** as the second style class after the primary one (named **col**_X, where X takes 1, 2, or 3) so that your div sections will look as follows:

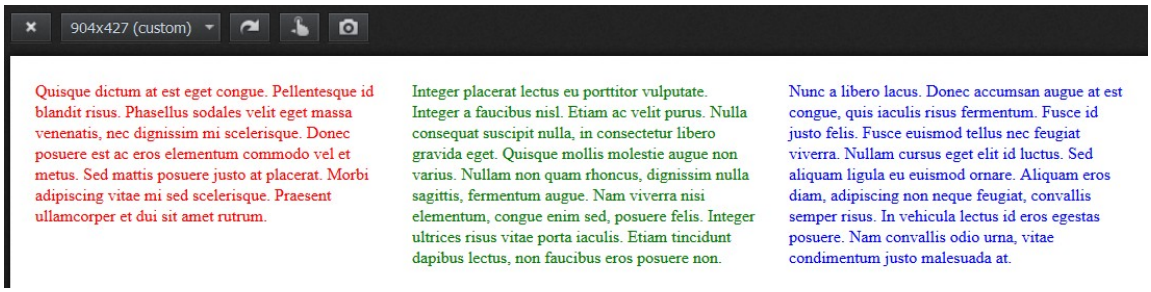
```
… Content removed for space ...

<div class="col_1 floatLeft">

… Content removed for space ...

<div class="col_2 floatLeft">

… Content removed for space ...

<div class="col_3 floatLeft">

… Content removed for space ...
```

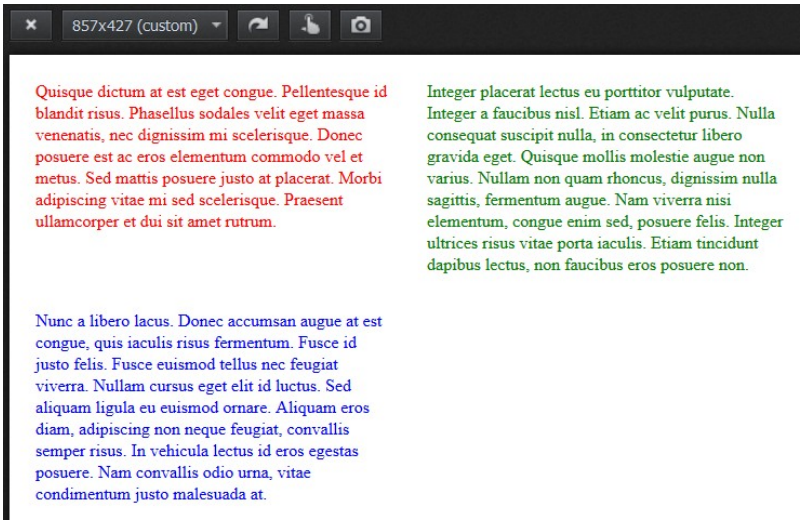
__4. Save the page in your editor.

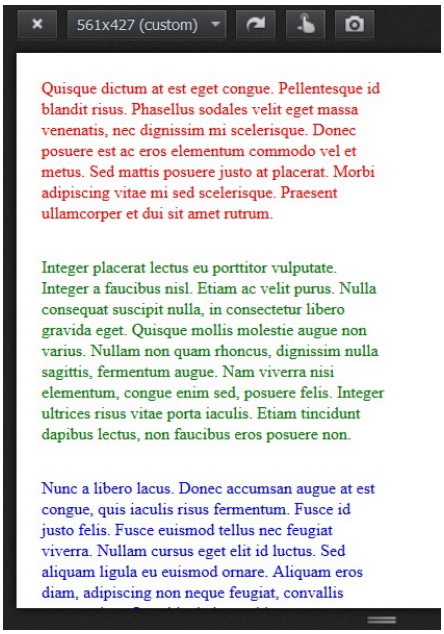
Keep the editor open on the **responsive_layout.html** file.

__5. Switch to Firefox and reload the page.

Now, when you resize the page, you should see the expected behavior of the page that fluidly accommodates the columns as the screen width changes.







Looks like we got what we were asking for.

Not quite yet.

Let's now run our page in a device emulator that is built-in with latest Chrome desktop

browsers (our tests were run in version 35).

Part 4 - View Page with Chrome's Mobile Browser Emulator

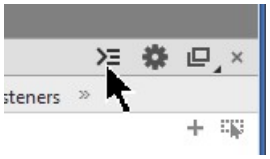
We are going to switch out browsers from Firebug to Chrome as the later supports emulation of mobile device browsers.

__1. Start Chrome browser and navigate to:

```
http://localhost/lab/responsive_layout.html
```

__2. Press **Ctrl-Shift-I** to bring up the *Developer Tools View*

__3. In the top right-hand corner of the *Developer Tools View*, click the *Show drawer* icon.

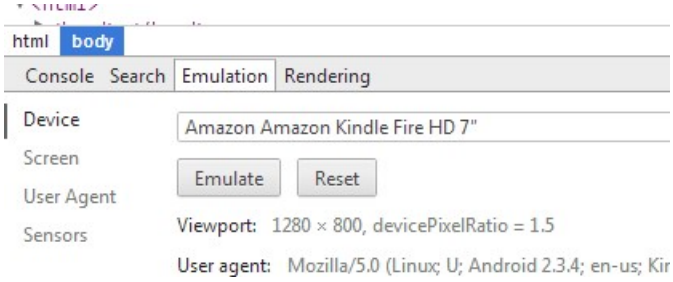


A new window opens at the bottom of Chrome.

In Chrome, this window is referred to as the *Drawer*.

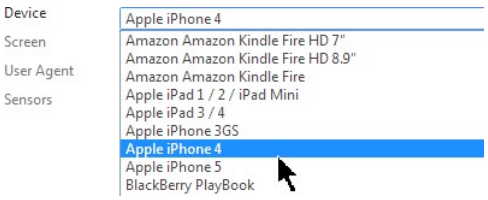
__4. Click the **Emulation** window tab.

You should see the following panel.



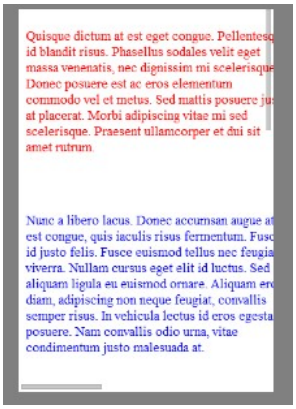
Note: The device currently displayed in the *Emulation* panel of your browser may differ.

__5. In the **Device** drop-down box, select **Apple iPhone 4**.



__6. Click the **Emulate** button just below the *Device* drop-down box.

You should see that our *responsive_layout.html* page gets re-rendered as if it were inside the iPhone's embedded AppleWebKit browser with its KHTML layout manager.



Chrome's mobile device emulator view automatically turns on the touch screen emulation so that we can use the mouse pointer to simulate the swipe gesture; the touch point is represented by a transparent gray circle at the mouse positioning point on the emulated window.

So, as you can see, not everything is good with our page - it did not respond to the emulator window we selected and the green (middle) column is only available for viewing when the user scrolls to the right.

So, back to the drawing board (some folks may heave a sigh at this point; bear with me, it is not going to be long).

Keep Chrome browser with our "unresponsive" page open.

Part 5 - Improve Page Responsiveness with the Viewport Meta Tag

As it turns out, we can fix the problem of not so responsive behaviour of our web page with just one line. In fact, it is not so much unresponsiveness on the part of our page as it is on the part of the target browser that has a virtual viewport larger than the visual viewport.

__1. Switch to the editor window open on **responsive_layout.html**.

__2. Enter the following code just below **<title>Responsive Layout</title>**:

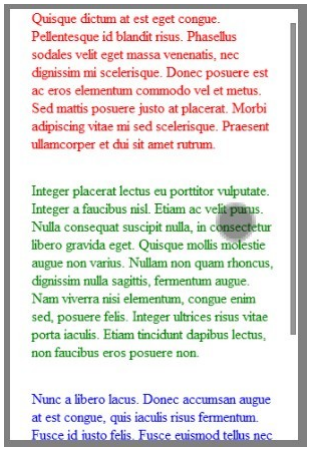
```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This line inserts the viewport meta tag into our page which will instruct the mobile device browser's layout manager to adjust the viewable width according to the system's screen width.

__3. Save the file.

__4. Switch back to Chrome and reload the page.

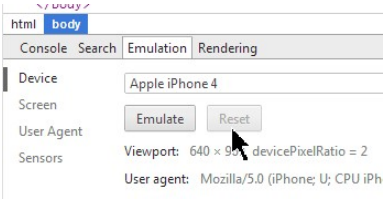
You should see the properly stacked view of our page as shown in the screen-shot below.



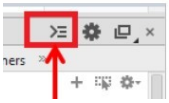
The horizontal scroll-bar has magically disappeared and now all is *honky dory* ("*all right*") and you can view the full page content by scrolling it up and down.

Part 6 - Lab Session Clean Up

__1. In Chrome browser, in the **Emulation** panel, click the **Reset** button to stop the Emulation mode.



__2. Click the *Drawer* icon to hide the *Drawer*.



__3. Close Chrome.

__4. Switch to Firefox, and press **Ctrl-Shift-M** to close the Responsive Design View window.

__5. Close Firefox.

__6. Close the editor.

Part 7 - Review

In this lab we worked through steps needed to make our web page responsive to different device windows. We used Firefox's Responsive Design View window to test our first cut changes using the CSS **float** property. Then we finished off our job by adding the

viewport meta tag instructions to the page and tested changes in Chrome's embedded device emulator.

Lab 24 - Orientation Responsiveness

In this lab, we will learn how to identify mobile device orientation and how to react to it. Techniques covered in this lab will help you improve user experience of your clients and may well pave your way to creating a number of simple yet interesting and porentially useful applications.

There will be two main parts in this lab:

- Identifying the Portrait / Landscape orientation using media query rules
- Identifying the 3D device orientation by interfacing with the Accelerometer sensor

Part 1 - Get Started

__1. Copy **responsive_orientation.html** from **C:\LabFiles** to **APACHE_DIR/htdocs/lab**:

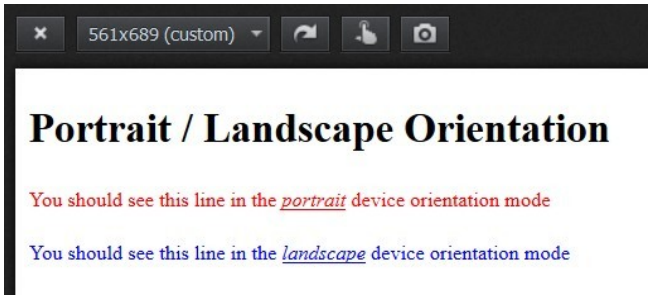
Part 2 - The First Run

__1. Open an instance of Firefox and navigate to:

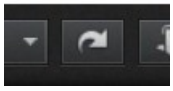
```
http://localhost/lab/responsive_orientation.html
```

__2. In Firefox, press **Ctr-Shift-M** to activate the Responsive Web Design View of the loaded page.

You should see the following page content:



__3. Click the **Rotate** button to simulate the change of the device orientation.



At the moment, the page won't react to your actions and will continue to display the same content in both orientation views as, currently, the page does not have a way to identify its orientation.

Part 3 - Make the Page Responsive to Portrait and Landscape Views

__1. Open the **APACHE_DIR/htdocs/lab/responsive_orientation.html** page in your editor.

__2. Locate and uncomment the following code fragment (remove the '/*' and '*/' comment anchors).

```
/*
@media all and (orientation:portrait){
    .portrait{color:red;font-size:1.25em; }
    .landscape{display:none;}
}
@media all and (orientation:landscape){
    .landscape {color:blue;font-size:1.25em; }
    .portrait{display:none;}
}
 */
```

This code uses media query rules to toggle styles depending on whether the device's screen is in the portrait or landscape view orientation.

The element styled with the **landscape** class is hidden

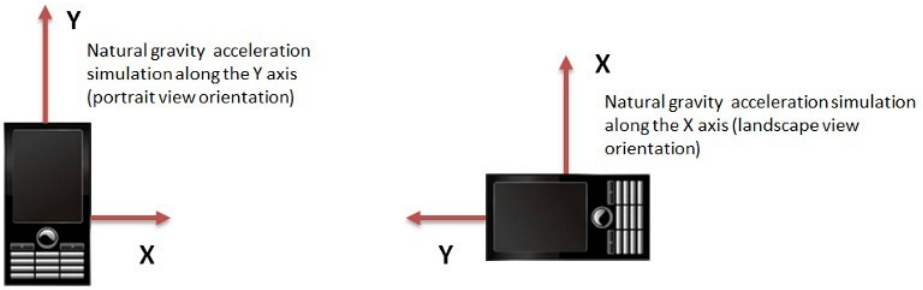
```
.landscape{display:none;}
```

when the browser identifies that it is in the *portrait* orientation (its screen height is pixel-wise bigger then its width), which is wrapped up in the

```
orientation:portrait
```

media query rule.

The CSS engine may receive orientation information by getting and processing the readings of the embedded accelerometer sensor as shown in the picture below.



Note: In some coordinate systems, the **Y** coordinate is represented by **γ** and **X** by **β** (and **Z** by **α**).

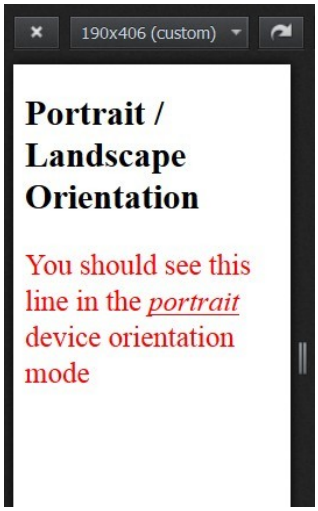
Conversely, when the device is in the *landscape* position, the element styled with the *portrait* style is hidden.

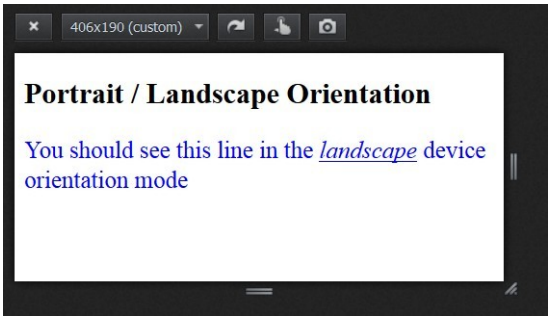
__3. Save the page in your text editor.

Keep the editor open on the **responsive_orientation.html** file as we will need it later in the lab.

__4. Switch to Firefox and reload the page.

Now, when you change page orientation, you should see the expected responsive behavior of the page.





Now, let's do something a little bit more technically involved, namely identifying the 3D device orientation.

__5. Switch back to the editor open on the **responsive_orientation.html** file.

__6. Locate the following code fragment:

```
.section1 {
```



```
}

.section2 {
    display: none;
}
```

__7. Swap the visibility of **section1** and **section2** CSS classes by cutting and pasting the **display:none;** rule from **section2** over to **section1**.

The resulting code of your updates should look as follows:

```
.section1 {
  display: none;
}

.section2 {

}
```

This swap will allow us concentrate on the second part of our lab, which, again, is

- Identifying the 3D device orientation

Part 4 - Chrome's Mobile Browser Emulator

We are going to switch out browsers from Firefox to Chrome as the later supports emulation of mobile device browsers and some of the device sensors.

__1. Start Chrome browser and navigate to:

http://localhost/lab/responsive_orientation.html

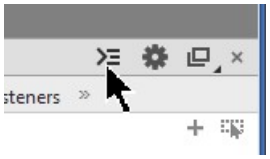
You should see the unhidden section of the page that we will use in our subsequent steps.

Responding to 3D Orientation

Event Supported	DeviceOrientation
Direction [alpha]	0
Tilt Front/Back [beta]	0
Tilt Left/Right [gamma]	0

__2. Press **Ctrl-Shift-I** to activate the *Developer Tools View.*

__3. In the top right-hand corner of the *Developer Tools View*, click the *Show drawer* icon.

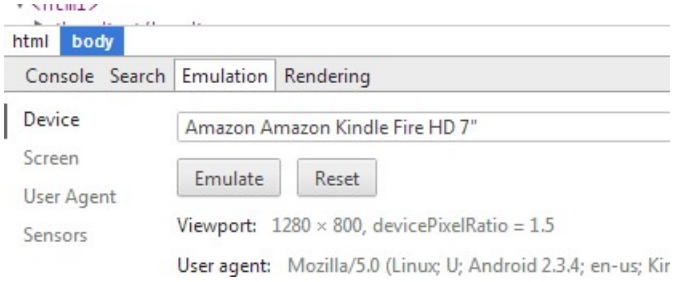


A new window opens at the bottom of Chrome.

In Chrome, this window is referred to as the *Drawer*.

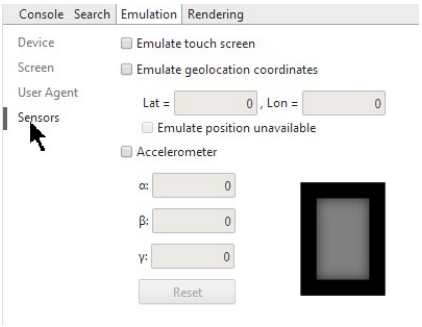
__4. Click the **Emulation** window tab.

You should see the following panel.



Note: The device currently displayed in the *Emulation* panel of your browser may differ.

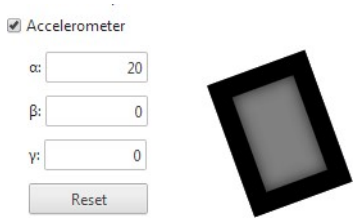
__5. Click the **Sensors** navigation link.



__6. In the *Sensors* panel that gets activated, click the **Accelerometer** checkbox.

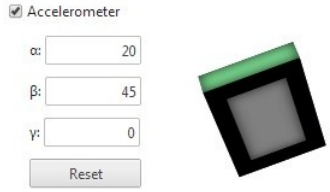
__7. Enter **20** in the **α** input box (values are in degrees, so we entered a 20°).

Observe the visual change in the orientation of the device image (displayed as an unimaginative picture frame to the right of the accelerometer 3D orientation parameters).



__8. Enter **45** in the **β** input box.

The device image changes further now tilting towards us at a 45 degree.



Now, if you take a look at our page loaded in Chrome, you will see that it properly records those simulated accelerometer readings as shown in a screen-shot below.

Responding to 3D Orientation

Event Supported	DeviceOrientation
Direction [alpha]	20
Tilt Front/Back [beta]	45
Tilt Left/Right [gamma]	0

Note: You can also change the device orientation by moving your mouse with the left button down.

OK, now it is time to see how our page intercepts device orientation in the 3D world.

Keep Chrome browser with our page loaded open.

Part 5 - Review the *DeviceOrientation* JavaScript Event

__1. Switch to the editor window open on **responsive_orientation.html**.

__2. Let's review the JavaScript code located at the bottom of the page.

Note: The code is an adaption of some samples available in the public domain of using

this experimental technology.

The core of its functionality is based on the availability of the *DeviceOrientationEvent* JavaScript global event which we try to intercept by using our anonymous event listener hooked on to this event as follows:

```
window.addEventListener('deviceorientation', function(eventData) {
```

After capturing the event's properties, the event listener passes on those properties to the *deviceOrientationHandler()* function for reporting purposes:

```
window.addEventListener('deviceorientation', function(eventData) {
   var tiltLR = eventData.gamma;
   var tiltFB = eventData.beta;
   var dir = eventData.alpha

   // call our orientation event handler
   deviceOrientationHandler(tiltLR, tiltFB, dir);
}, ...);
```

Our *deviceOrientationHandler()* simply reports the received orientation information in the respective HTML elements:

```
function deviceOrientationHandler(tiltLR, tiltFB, dir) {
  document.getElementById("doTiltLR").innerHTML = Math.round(tiltLR);
  document.getElementById("doTiltFB").innerHTML = Math.round(tiltFB);
  document.getElementById("doDirection").innerHTML = Math.round(dir);
}
```

Be aware that this technology is still regarded as experimental and even its specification has not yet been stabilized, so don't rush to start using it in production as the API (Application Programming Interface) and behavior may change in future versions of browsers that implement this spec.

For more information, visit the W3C site who sponsors specification for this project (http://w3c.github.io/deviceorientation/spec-source-orientation.html).

Currently, only Chrome and Firefox offer support for this technology; the support for this event is checked in this piece of code:

```
if (window.DeviceOrientationEvent)  ...
```

For you reference, the *DeviceOrientationEvent,* when fully constructed and passed on to your event listener's call-back method, has the following properties:

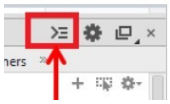
Property Name	Description

absolute	A boolean that indicates whether or not the device is providing orientation data absolutely.
alpha	A number representing the motion of the device around the z axis, express in degrees with values ranging from 0 to 360
beta	A number representing the motion of the device around the x axis, express in degrees with values ranging from -180 to 180. This represents a front to back motion of the device.
gamma	A number representing the motion of the device around the y axis, express in degrees with values ranging from -90 to 90. This represents a left to right motion of the device.

Part 6 - Lab Session Clean Up

__1. In Chrome browser, uncheck the Accelerometer checkbox to stop this sensor simulation.

__2. Click the *Drawer* icon to hide the *Drawer*.



__3. Close Chrome.

__4. Switch to Firefox, and press **Ctrl-Shift-M** to close the Responsive Design View window.

__5. Close Firefox.

__6. Close the text editor.

Part 7 - Review

In this lab, we familiarized ourselves with techniques related to making our web page

responsive to device orientation. We took advantage of the media query-based orientation rules (*orientation:portrait* and *orientation:landscape*) as well as the emerging API (Application Programming Interface) for interfacing with the Accelerometer sensor.

Lab 25 - Responsive Images with Media Queries

In this lab, we will show how to make images responsive to changes in the client device's screen size.

We will need the latest version of Firefox with its built-in Responsive Design View plug-in (which is now part and parcel of the Firefox browser platform).

Part 1 - Get Started

__1. Copy these files from **C:\LabFiles** to **APACHE_DIR/htdocs/lab**:

- **responsive-images-via-media-query.html**
- **toronto-city-hall-satellite-view.png**

Part 2 - The First Run

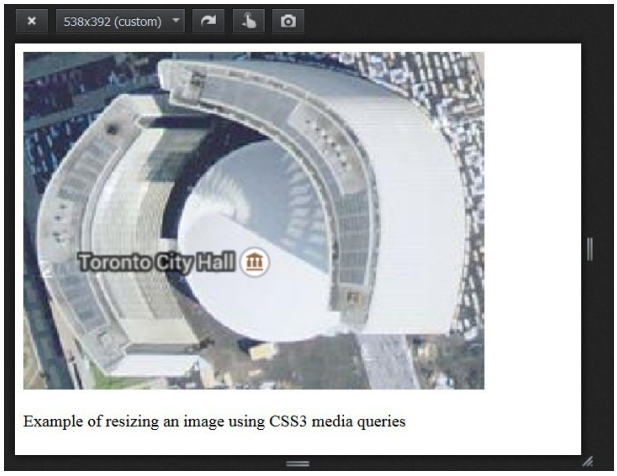
__1. Open an instance of Firefox, maximize its window and navigate to:

```
http://localhost/lab/responsive-images-via-media-query.html
```

You should see a satellite picture of the Toronto's City Hall building taken on a beautiful summer day.

__2. Activate the *Responsive Design View* in Firefox by pressing **Ctr-Shift-M**

You should see the following picture (the *View* sizes may differ in your case).



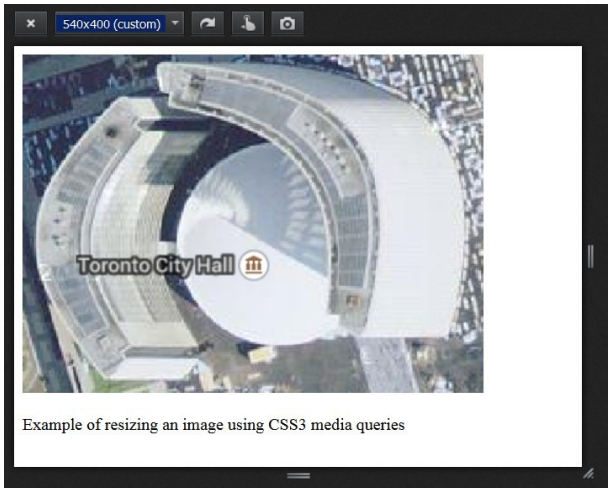
__3. Try to re-size the view by dragging the control handle at the bottom right-hand corner of the page.

Note: For better positioning precision, hold the *Control* key down as you do the dragging.

Currently, the size of the image stays unchanged regardless of the changes to the *Mobile View* window - it is not responsive.

We are going to fix it in the next lab part.

__4. Resize the *Mobile View* window to about **540 x 400 px** (keep the *Control* key down for finer dragging prevision).



That action will set the stage for viewing the changes we are going to make.

__5. Keep the browser window open.

Part 3 - Make the Image Responsive

__1. Start your text editor and open **responsive-images-via-media-query.html**

__2. Locate the *<style></style>* section and enter the following code between the <style> and </style> tags:

```
@media screen and ( max-width: 300px ) {
   img.responsive_image { width: 100px; }
}
@media screen and ( min-width:301px ) and ( max-width: 500px ) {
   img.responsive_image { width: 200px; }
}
@media screen and ( min-width: 501px ) {
   img.responsive_image { width: 300px; }
}
```

__3. Locate the *<img ...>* element and add the following attribute to it:

```
class="responsive_image"
```


which matches the CSS class references you added in the **style** section of your page.

After updates, the page code should look as follows (the added lines are shown in *bold*):

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8"/>
   <title>Responsive Image Example</title>
<style>
    @media screen and ( max-width: 300px ) {
       img.responsive_image { width: 100px; }
    }
    @media screen and ( min-width:301px ) and ( max-width: 500px ) {
       img.responsive_image { width: 200px; }
    }
    @media screen and ( min-width: 501px ) {
       img.responsive_image { width: 430px; }
    }
</style>
</head>

<body>
<img class="responsive_image" alt="View of the Toronto City Hall from Above"
src="toronto-city-hall-satellite-view.png">
<p>
Example of resizing an image using CSS3 media queries
</p>
</html>
```

__4. Save the file.

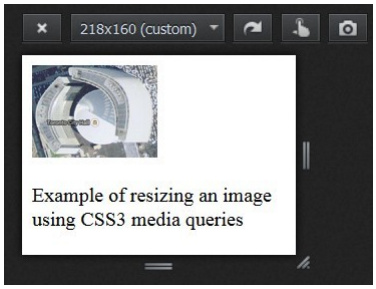
Part 4 - The Second Run

__1. Switch to Firefox and refresh the browser window to reload our **responsive-images-via-media-query.html** page.

Now we are ready to test our changes.

__2. Begin minimizing the *Mobile View* by dragging the bottom right-hand corner handle as you did previously.

Notice the re-rendering of the image in a smaller size - from **430px** down to **200px** when a width of the view of **501px** is passed, and then from **200px** down to 1**00px** triggered by the passing of the **301px** threshold width.



Consult the media queries you added in the styling section of the page to see how image

down-sizing is configured in the media queries.

Part 5 - Review

In this lab we demonstrated how to use the CSS3 media query facility to make an image responsive to the client device's view width.

Lab 26 - Responsive Images with Picturefill

In this lab, we will show how to make images responsive to changes in screen sizes using the Picturefill JavaScript library (http://responsivedesign.is/resources/images/picture-fill).

We will need the latest version of Firefox with its built-in *Responsive Design View* plug-in.

Part 1 - Get Started

__1. Copy all files from **C:\LabFiles\picturefill** to **APACHE_DIR/htdocs/lab**

These files are:

- **210px.png**
- **320px.png**
- **640px.png**
- **picturefill.js**
- **responsive_images.html**

Part 2 - Get to Know the Code

__1. Open the **APACHE_DIR/htdocs/lab/responsive_images.html** file in your text editor.

This file is ready to be used AS-IS.

```
<!doctype html>
<html>
<head>
<title>Responsive Images</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
img.fluid {
    max-width: 100%;
}

</style>
<script async type="text/javascript" src="picturefill.js"></script>
</head>
<body>

<!--srcset HTML5 property -->
<img class="fluid" srcset="640px.png 640w, 320px.png 320w, 210px.png 210w"
sizes="80vw" alt="The selective image loading demo" />

</body>
</html>
```

The page includes the Picturefill JavaScript library in this line:


```
<script async type="text/javascript" src="picturefill.js"></script>
```

which will be doing some dynamic population of the *img* tag's source depending on the screen size (as per the *"640px.png 640w, 320px.png 320w, 210px.png 210w"* rule).

We set the *sizes="80vw"* attribute to have a better alignment with the *Responsive Design View* reported width.

__2. Keep the editor open on the file.

Part 3 - Set Up the Browser Environment

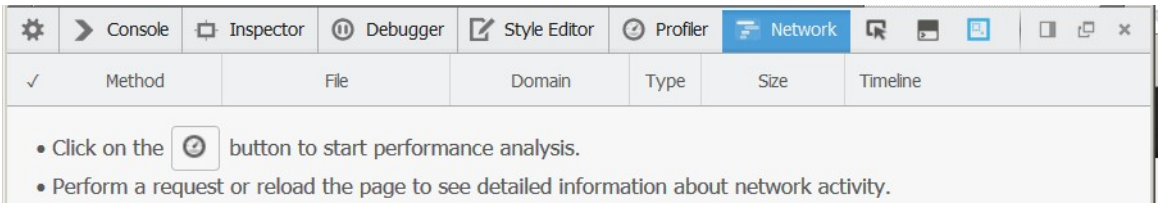
__1. Open an instance of Firefox and maximize its window.

__2. Activate the *Responsive Design View* in Firefox by pressing **Ctr-Shift-M**

__3. Make the width of the *Responsive Design View* wider than 800px by holding and dragging the two little gray parallel bars on the right side of the view window.

__4. Press **Ctrl-Shift-Q** to open the *Network* panel.

When it opens, it is placed at the bottom of your browser window.



__5. Click **All** button at the bottom left corner

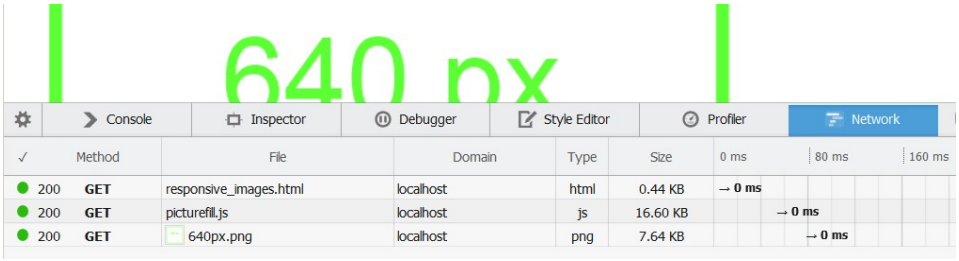
Note: If you don't see the bottom menu with the *All* button, increase the height of the **Network** panel:

__6. Navigate to:

```
http://localhost/lab/responsive_images.html
```

The target page gets loaded, and the *Network* panel at the bottom of the browser window

gets populated with the list of loaded assets.



What is interesting to observe here is that only the *640px.png* image file is loaded.

__7. Start slowly decreasing the width of the *Responsive Design View* by dragging the two parallel bars to the left; keep the **Shift** button down to get it done in 10px change increments.

At about 320px window size, you will see the *320px.png* image on the page; the loading of the image is also reported in the *Network* panel.

200	GET	responsive_images.html	localhost	html	0.44 KB
200	GET	picturefill.js	localhost	js	16.60 KB
200	GET	640px.png	localhost	png	7.64 KB
200	GET	320px.png	localhost	png	4.69 KB

Notice that the loading of the image happened when the *srcset* rules was met.

__8. Continue to reduce the width of the *Responsive Design View* until you get the 210px.png image loaded in the browser.

The sequence of the image loading is shown in the chart below; there are two breakpoints involved: 210 px and 320 px at which a new image HTTP request is made to the server.

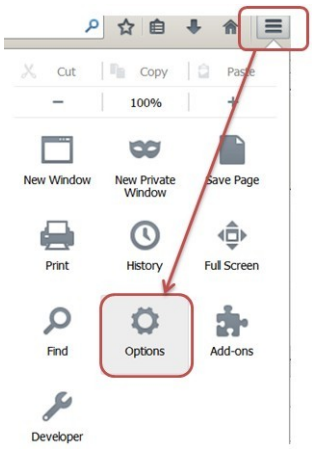
```
-------------------------------------------------------------------
      210px ->|          320px ->|                  ::Screen resolution
-------------------------------------------------------------------
210px.png     |      320px.png    |     640px.png     ::Image file loaded
-------------------------------------------------------------------
```

Now, if you started to increase the width of the page, you would see the changing images which happens without loading them from the server but, rather, serving them from the browser's cache. In order to see the reverse "slide show" happening, we need to purge the browser cache.

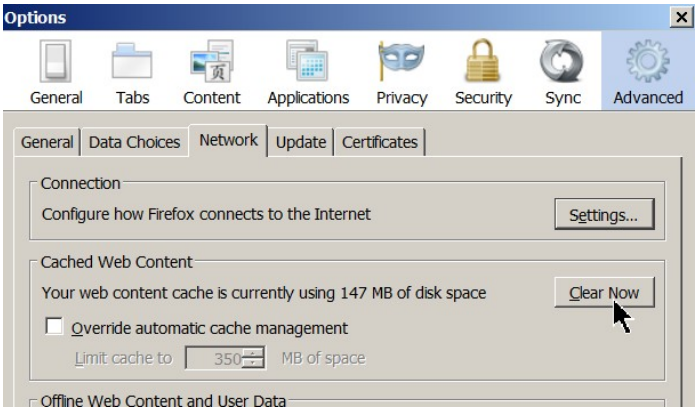
__9. Keep the *Responsive Design View* screen at the width where you have the *210px.png* image loaded.

Part 4 - Purge the Firefox Cache

__1. Open the Firefox **Options** panel by using the Firefox's Options menu.



__2. In the **Options** panel that opens, click the **Network** tab and clear the cached web content by clicking **Clear Now** button, then click **OK** to close the Options panel.

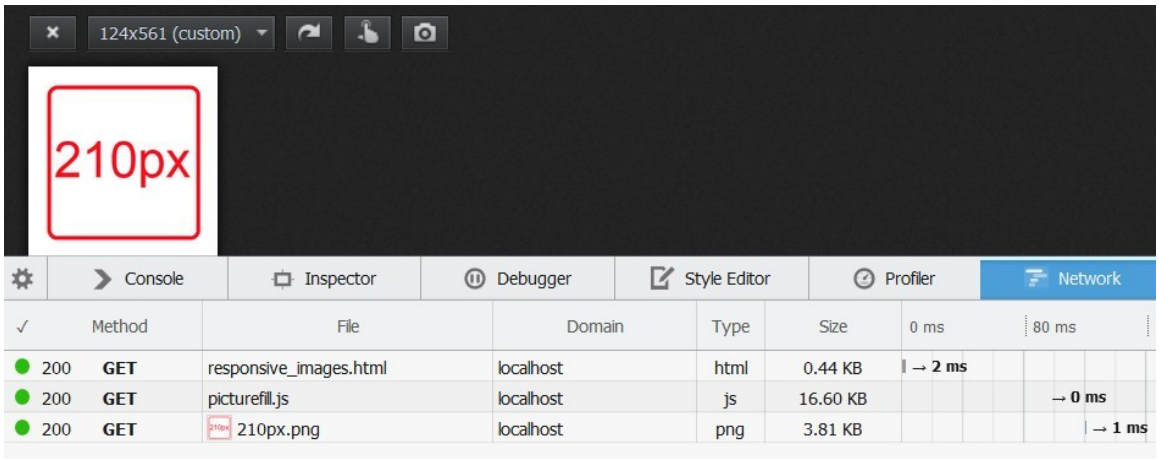


__3. Back in the browser, in the bottom right corner of the *Network* panel, click the **Clear** button to clear the recorded HTTP transactions (adjust the height of the panel if you don't see the Clear button.)

Part 5 - Start from the Smallest Screen Width

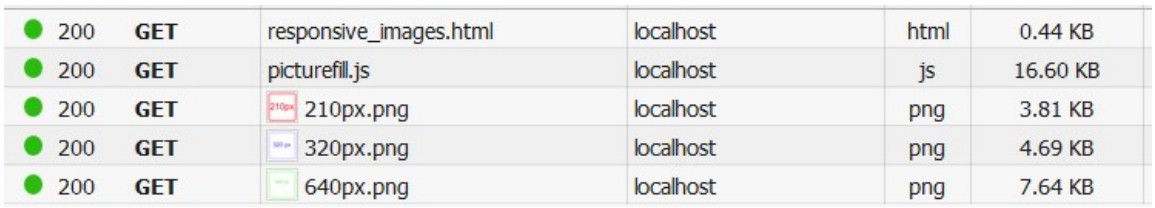
__1. Refresh the browser window by pressing **Ctrl-R**

You should see a set of new HTTP requests sent by the browser to the server, also fetching the 210px.png image.



__2. Start slowly increasing the width of the *Responsive Design View* by dragging the two parallel bars (adjust the height of the *Network* panel, if you don't see them).

Notice that the loading of the images is now happening in the reverse order.



Part 6 - Disable the Picturefill Script

__1. Switch back to the **APACHE_DIR/htdocs/lab/responsive_images.html** file open in your text editor.

__2. Delete the line:

```
<script async type="text/javascript" src="picturefill.js"></script>
```

__3. Save the file.

__4. Switch back to Firefox.

__5. Purge the browser cache using the steps described earlier in the Lab.

__6. Open a new browsing Tab by pressing **Ctrl-T**

__7. In the new tab, press **Ctrl-Shift-M** to open the *Responsive Design View*.

__8. Enter **http://localhost/lab/responsive_images.html** in the navigation window and press **Enter**.

Instead of an image, you should see the following message on the screen:


```
The selective image loading demo
```

that comes from the *alt* attribute of our *img* tag as we don't have the *src* attribute set.

As you see, disabling the Picturefill script ended our mesmerizing "slide show".

Note: If you actually see the images and everything works fine, that would mean that the latest Firefox version you run natively supports the *srcset* rules!

__9. Switch back to your text editor.

__10. Recover the deleted line by re-doing the previous line deletion:

```
<script async type="text/javascript" src="picturefill.js"></script>
```

__11. Save the file.

__12. Switch back to Firefox and reload the page.

You should be back in business.

Part 7 - Lab Session Clean Up

__1. Close Firefox browser, confirming the closing of tabs when prompted.

__2. Close the text editor.

Part 8 - Review

In this lab we demonstrated how to use the Picturefill JavaScript library to enable the dynamic loading of images configured in the media query rules set in the *srcset* attribute.

Lab 27 - Getting Started With Bootstrap

In this lab, you will load the bootstrap files into Apache's document root directory. Then you will configure a simple HTML file to use Bootstrap. .

Part 1 - Copy Bootstrap Distribution

In this part, you will copy the bootstrap files into Apache.

__1. Copy **C:\LabFiles\bootstrap** directory to **APACHE_DIR/htdocs/**.

Part 2 - Create the Bootstrap template HTML file

__1. Open **APACHE_DIR/htdocs/bootstrap/index.html** in a text editor.

__2. This is a simple HTML5 file. Examine the file:

```
<!DOCTYPE html>
<head>
  <title>Bootstrap Labs</title>

</head>
<body>

</body>
```

__3. First thing add the <meta> tag inside <head> tag specifying the viewport parameters. This tag will be added in the head section below the title

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

__4. Next add a link to the Bootstrap CSS file right below the <meta> tag:

```
<link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
```

The head section should look like:

```
<head>
  <title>Bootstrap Labs</title>
  <meta name="viewport"
        content="width=device-width, initial-scale=1.0">
  <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
</head>
```


__5. Finally, add the script references to the end of the body section (before the </body> tag):

```
  <script src="js/jquery-1.9.1.js"></script>
  <script src="js/bootstrap.min.js"></script>
```

The completed template file should look like:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bootstrap Labs</title>
    <meta name="viewport"
                content="width=device-width, initial-scale=1.0">
    <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
  </head>
  <body>
    <script src="js/jquery-1.9.1.js"></script>
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

__6. Save the file.

__7. Save a copy of the file as poetry.html. (You will use it in the next lab)

Part 3 - Add some Bootstrap content

In this part, you will add some bootstrap code to the template file to make sure everything is working by using a **jumbotron**. [Note: the jumbotron component uses bootstrap CSS but not JavaScript.]

__1. Open **index.html** in a text editor.

__2. Add the following div element after the **<body>** and before the first **<script>**:

```
    <div class="container">
      <div class="jumbotron">
        <h1>Hello, JUMBO world!</h1>
        <p>
          This is an example of a Bootstrap Jumbotron.
          This component allows you to highlight content in a
          resizeable, responsive manner.
        </p>
      </div>
```



```
<div class="row">
  <div class="col-sm-12">
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
    Bootstrap is really cool!
  </div>
</div>

</div>
```

__3. Save the file.

Part 4 - Test

__1. In a browser connect to **http://localhost/bootstrap**

You should see:

__2. Try resizing the browser window to see the responsive behavior.

__3. Close all open files and browsers.

Part 5 - Review

In this lab, you added support for Bootstrap to a file and then configured a simple component, the Jumbotron.

Lab 28 - Simple Components

In this lab, you will load the pre built Bootstrap template and a grid for layout and navigation.

Part 1 - Create the Navigation

In this part, you will add navigation to the template.

__1. Open **APACHE_DIR/htdocs/bootstrap/poetry.html** in a text editor.

__2. Add the following div element after the **<body>** and before the first **<script>**:

```
<nav class="navbar navbar-default" role="navigation">



</nav>
```

__3. Next add the following inside the **<nav>** elements.

```
  <div class="navbar-header">
    <button type="button" class="navbar-toggle" data-toggle="collapse"
            data-target="#bs-example-navbar-collapse-1">
    <span class="sr-only">Toggle navigation</span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
    </button>
  </div>
```


__4. Next add the following code after the ** ** from the previous step.

```
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
  <ul class="nav navbar-nav">
    <li class="active">
      <a href="#">
        <span class="glyphicon glyphicon-home"></span>
        Home
      </a>
    </li>
    <li><a href="#">Collections</a></li>
    <li class="dropdown">
      <a href="#" class="dropdown-toggle" data-toggle="dropdown">Poets
        <b class="caret"></b>
      </a>
      <ul class="dropdown-menu">
        <li><a href="#">Edgar Allen Poe</a></li>
        <li><a href="#">Robert Frost</a></li>
        <li><a href="#">Maya Angelou</a></li>
      </ul>
    </li>
  </ul>
  <ul class="nav navbar-nav navbar-right">
    <li><a href="#">About</a></li>
  </ul>
</div>
```

__5. Next add the following code after the ** ** from the previous step.

Note: the poems are in **C:\LabFiles\poetry.txt**. To get the full effect of the lab, copy the poems or at least enter multiple paragraphs of text in place of '**...**'

```
<div class="container">
  <div class="row">
    <div class="col-sm-12">
      <h1>Robert Frost - Selected Poems</h1>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-3">
      <h3>Fire and Ice</h3>
      Some say the world will end in fire,<br>
      ...
    </div>
    <div class="col-sm-3">
      <h3>The Plowman</h3>
      I hear men say to plow the snow.<br>
      ...
    </div>
```



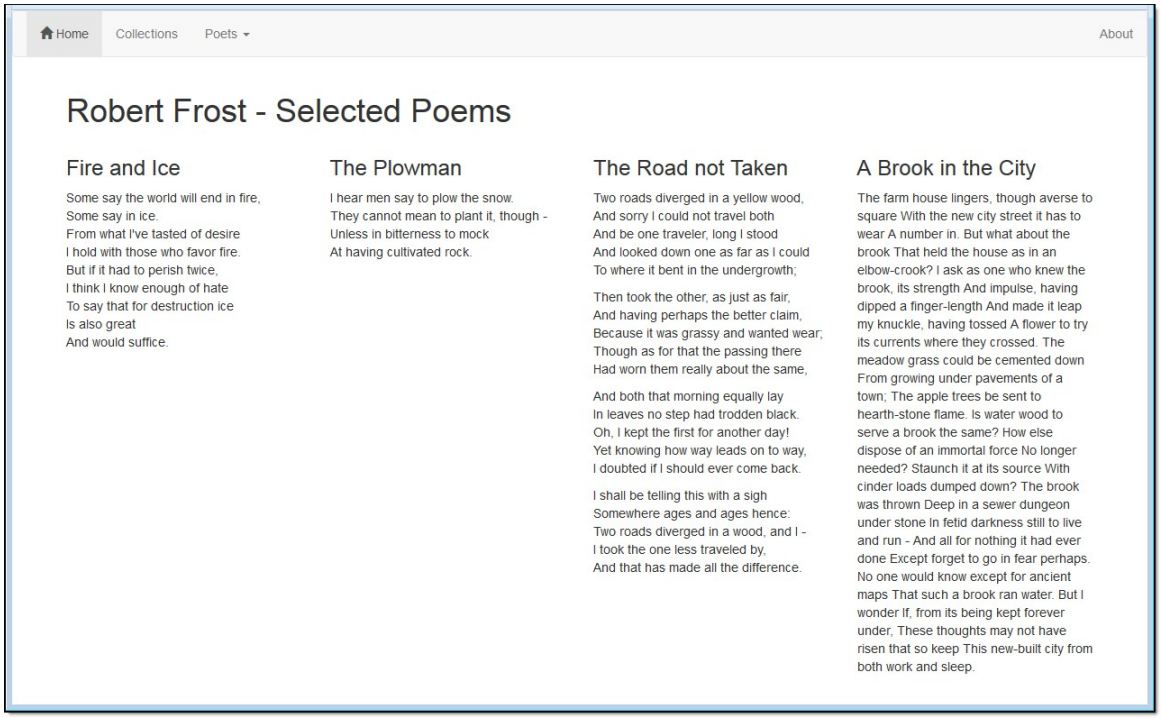
```
    <div class="col-sm-3">
      <h3>The Road not Taken</h3>
        Two roads diverged in a yellow wood,<br>
        ...
      </div>
      <div class="col-sm-3">
        <h3>A Brook in the City</h3>
        The farm house lingers, though averse to square<br>
        ...
      </div>
    </div>
</div>
```

__6. Save the file.

Part 2 - Test

__1. In a browser connect to **http://localhost/bootstrap/poetry.html**

You should see:



__2. Try resizing the browser window to see the responsible behavior. Note, some IE version won't show the expected behavior.

__3. Close all open files and browsers.

Part 3 - Review

In this lab, you used navigation and a grid layout to create a responsive web page.

Lab 29 - Integrating jQuery with Bootstrap Components

Most Bootstrap components are static HTML elements wired by stylesheets to make them responsive for various screen sizes and target browsers' capabilities. This lack of inherent dynamics can be easily compensated with the power of jQuery.

In this lab, we will explore some of the ways of augmenting Bootstrap components with jQuery.

This Lab depends on work done in one of the previous Labs, namely the copying of the **C:\LabFiles\bootstrap** directory to **APACHE_DIR/htdocs/**.

Part 1 - Reviewing the Page with JavaScript Functions' Stubs

Let's review the skeleton web page created to minimize the typing on your part.

__1. Open **APACHE_DIR/htdocs/bootstrap/Bootstrap_with_jQuery.html** in a text editor (e.g. Notepad++).

__2. You should see the following content:

```
<!DOCTYPE html>
<head>
<title>jQuery with Bootstrap Components Labs</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link href="css/bootstrap.min.css" rel="stylesheet" media="screen">

<script type="text/javascript">
   function hide(id) {
   }

   function show(id) {
   }

   function addAGlyphicon(glyphiconClass) {
   }

   function removeAGlyphicon(glyphiconClass) {
   }
</script>
</head>
<body>
    <div class="container" style="width: 80%">

        <div id="danger" class="alert alert-danger" style="display: none;">Alert!</div>

        <div class="btn-toolbar" role="toolbar">
            <button type="button" class="btn btn-default btn-sm"
onclick="show('danger');">Show the message ...</button>
            <button type="button" class="btn btn-default btn-sm"
onclick="hide('danger');">Hide the message ...</button>
            <button type="button" class="btn btn-default btn-sm"
onclick="addAGlyphicon('glyphicon glyphicon-star');">
                <span id="starId"></span> Assign me a star rating!...
            </button>
            <button type="button" class="btn btn-default btn-sm"
onclick="removeAGlyphicon('glyphicon glyphicon-star');">Assign him
                a starless rating ...</button>
```



```
        </div>

        <script src="js/jquery-1.9.1.js"></script>
        <script src="js/bootstrap.min.js"></script>
</body>
```

As you can see, there are already JavaScript functions' stubs created for you to fill out with code we are going to write.

__3. Open a web browser and navigate to the following URL:

http://localhost/bootstrap/Bootstrap_with_jQuery.html

You should see four in-line buttons grouped together by the * * element. Each button is reduced in size from the default one by applying the *.btn-sm* Bootstrap CSS class.

__4. Leave the browser open to the page.

__5. Switch back to the text editor.

In the next Lab section, our plan is to write code for the functions invoked from the *Show / Hide the message ...* buttons (the two left-most ones) to show and hide the alert message.

The alert message HTML element has the *"danger"* id attribute to make it accessible from jQuery:

```
        <div id="danger" class="alert alert-danger" style="display: none;">Alert!</div>
```

At page load time, the message is hidden (using the *"display: none;"* style).

The other two buttons (the two right-most ones) will toggle the Star glyphicon to be displayed on the *Assign me a star rating!* button. We will work on those later in the Lab.

So that's our plan.

While you can also write code for these JavaScript functions using plain-vanilla JavaScript, leveraging jQuery aids in your productivity and ensures your code works the same way across different browsers. Doing so will also reinforce your newly acquired knowledge of jQuery.

Part 2 - Adding jQuery Code for Component Hiding and Showing

__1. Switch to your text editor that has the **Bootstrap_with_jQuery.html** open and locate the following code:

```
function show(id) {
}
```

__2. Enter the following code between the {} curly braces:

```
   var ref = $("#" + id);
   alert("About to show the message");
   var showTimeMs = 5000;
   ref.show(showTimeMs);
```

This jQuery code takes the id of the hidden alert message ("*danger*"), formats it by prefixing it with '#' and, using it, gets the reference to the jQuery object: *var ref = $("#" + id);*

Then it displays a diagnostic message about the action to be performed and shows the message in slow motion (5 seconds).

__3. Locate the following code:

```
function hide(id) {
}
```

__4. Enter the following code between the {} curly braces:

```
   var ref = $("#" + id);
   var text = ref.html();
   alert("About to hide the '" + text + "' message");
   ref.hide("slow");
```

This code uses jQuery to hide the message box in slow motion.

Your updates to the function stubs should look as follows (we don't do any updates to the other two functions dealing with the star glyphicon manipulation).

```
function hide(id) {
   var ref = $("#" + id);
   var text = ref.html();
   alert("About to hide the '" + text + "' message");
   ref.hide("slow");
}
```



```
function show(id) {
   var ref = $("#" + id);
   alert("About to show the message");
   var showTimeMs = 5000;
   ref.show(showTimeMs);
}
```

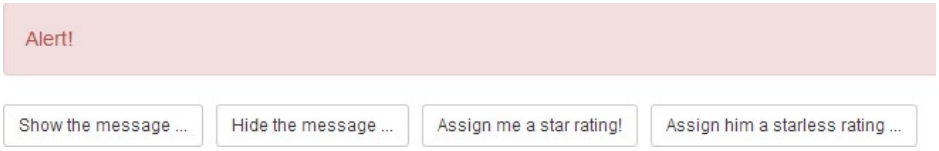
__5. Save the file (**Ctrl-S**).

__6. Switch to your web browser and refresh the page:

http://localhost/bootstrap/Bootstrap_with_jQuery.html

__7. Click the **Show the message …** button and discard the "**About to show the message**" pop-up by clicking **OK**.

You should see a gradually appearing Alert message.



__8. Click the **Hide the message …** button and discard the "**About to hide the 'Alert!' message**" pop-up by clicking **OK**.

You should see the gradually disappearing alert message.

Part 3 - Adding jQuery Code for Attaching and Removing a Glyphicon

__1. Switch to your text editor with **Bootstrap_with_jQuery.html** and locate the following code:

```
function addAGlyphicon(glyphiconClass) {
}
```

__2. Enter the following code between the {} curly braces:

```
    $("#starId").addClass(glyphiconClass);
```

This jQuery code dynamically attaches the *glyphicon glyphicon-star* Bootstrap class to the *span* tag nested in the *Assign me a star rating!* button. The *span* tag is identified with the "*starId"* id attribute used by jQuery to locate the element in the page DOM.

__3. Locate the following code:

```
function removeAGlyphicon(glyphiconClass) {
}
```

__4. Enter the following code between the {} curly braces:

```
    $("#starId").removeClass(glyphiconClass);
```

This jQuery code dynamically removes the *glyphicon glyphicon-star* Bootstrap class.

Your updates to the two functions should look as follows:

```
function addAGlyphicon(glyphiconClass) {
   $("#starId").addClass(glyphiconClass);
}

function removeAGlyphicon(glyphiconClass) {
   $("#starId").removeClass(glyphiconClass);
}
```

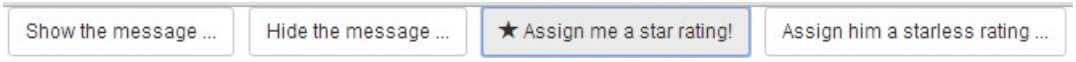
__5. Save the file (**Ctrl-S**).

__6. Switch to your web browser and refresh the page:

http://localhost/bootstrap/Bootstrap_with_jQuery.html

__7. Click the **Assign me a star rating!** button**.**

You should see a star glyphicon added to the button you just clicked added to the left of the button text (see if you can change the position of the star icon so that it is placed to the right of the button text).



Now, it is about time to bring the guy down a peg ...

__8. Click the **Assign him a starless rating ...** button.

The star should be removed from the **Assign me a star rating!** button all right.

Part 4 - Lab Clean-up

__1. Close your browser.

__2. Close your text editor.

Part 5 - Review

In this lab we were able to achieve a great deal of UI dynamics through the courtesy of jQuery with minimum amount of code and effort.

Lab 30 - Mobile Web Testing With Chrome

Mobile web applications are pretty much the same regular web application developed using standard web technologies. They are deployed on remote web servers where they can be accessed from a variety of mobile browsers running on handheld devices such as smartphones or tablets. Communication between the browser and the remote web server is done either over a mobile or wireless network.

In this lab, we will look at the basic tooling support for mobile web UI testing offered by Google's Chrome browser which is also useful for testing web pages created using Responsive Web Design techniques.

Using Chrome's free Device Emulation add-on can help testers complete a bulk of UI testing of mobile web apps using mobile browser emulators for a variety of mobile devices.

Part 1 - Emulation of Mobile Browsers in Chrome

The Google Chrome browser comes with a developer toolbox (a.k.a. DevTools) the value of which in mobile web testing cannot be overestimated. It is powerful and very easy to use.

Let's see it in action.

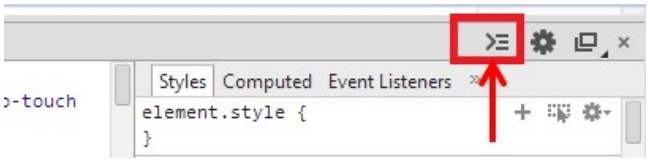
__1. Start Chrome browser and navigate to:

```
http://www.yahoo.com
```

__2. In the browser, enter **Ctrl-Shift-I**.

The *DevTools* panel opens at the bottom of the browser window.

__3. Click the **Show Console** icon as indicated below.

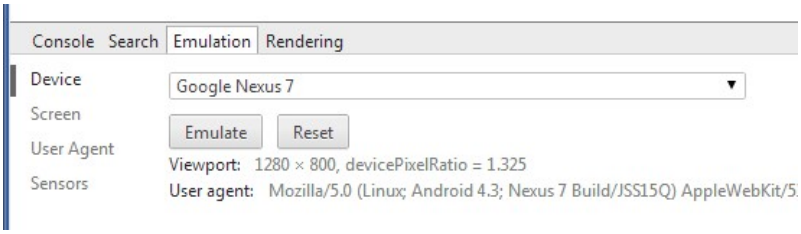


The console window opens below the DevTools panel.

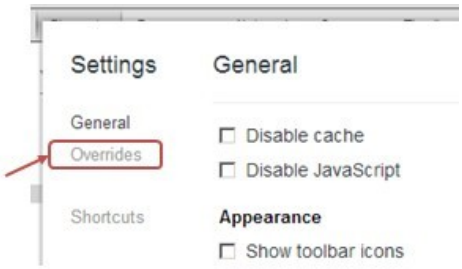
Note: Don't confuse this window with the JavaScript console (located on the *DevTools* panel). The console window we are going to work with is also referred to as the **Drawer**.

__4. In the Console (Drawer) tabbed menu, click the **Emulation** tab.

You should get the following *Emulation* screen of the Device Emulation add-on:



Note: Chrome team completely overhauled this panel in recent browser versions. In previous versions, the device emulation settings were under the *Overrides* tab on the Settings menu (invoked by clicking the black cog menu icon in the *DevTools* panel):



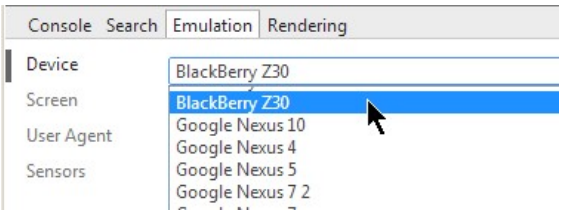
The instruction steps below cover device settings under the Emulation tab on the Console (Drawer) as found in more recent Chrome versions.

As you can see on the Emulation screen, there are a number of options that can be overridden in its appropriate sections.

For example, in the Sensors section, you can find options for emulating device's geolocation (latitude and longitude), device orientation along three axis (x, y and z); in the Screen section, you can specify the CSS media type, e.g. print, screen, speech, etc.

All those settings can really help with testing the above features, if needed.

__5. In the *Device* section, select the **BlackBerry Z30** option from the device drop-down.



The *Viewport* details and *User Agent* string are automatically updated for the type of the device selected.

In case of BlackBerry Z30 we just selected, the User Agent string is:

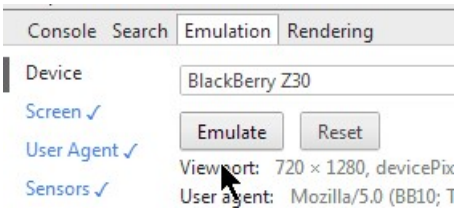
```
Mozilla/5.0 (BB10; Touch) AppleWebKit/537.10+ (KHTML, like Gecko)
Version/10.0.9.2372 Mobile Safari/537.10+
```

As you see, BlackBerry Z30 uses the AppleWebKit web browser engine (Apple's Safari and Google's Chrome also use the WebKit engine).

Note: KHTML is the name of the HTML layout engine.

__6. Click the **Emulate** button.

You should see the following updates on the *Emulation* screen.



Note: On Windows XP you may see boxes instead check marks.

The emulated device details are applied across all emulated features (as indicated by the blue check marks set next to all the sections).

In the browser you can see a squeezed view of the web page rendered such as to emulate the viewport metrics.

But we still see the original version of the web page created for desktop browsers.

You will need to reload the web page for proper User Agent spoofing and viewport rendering in the browser.

__7. Refresh the browser (**Ctrl-R**) that still shows the *www.yahoo.com* web site.

Now you should see a mobile version of the Yahoo landing page rendered in Chrome as you have been redirected to the *https://**m**.yahoo.com/* mobile site on the GET request triggered by the browser window refresh.

The Yahoo web site identifies the submitted User Agent string as belonging to a mobile device and redirects the emulated browser to the mobile version of the site via the 302 HTTP response code listing *https://**m**.yahoo.com/* in the Location response header (you can see the interaction between the browser and the web site in the *Network* section of the DevTools).

Note: Chrome DevTools' Emulation add-on does not provide a direct support for the *Opera* mobile browser as Opera Software (company behind it) has developed their own very good Opera mobile emulator that should be used for testing Opera mobile browsers.

Part 2 - Direct Spoofing of the User Agent String

You can also spoof the User Agent string directly by setting it to something you want in the User Agent section.

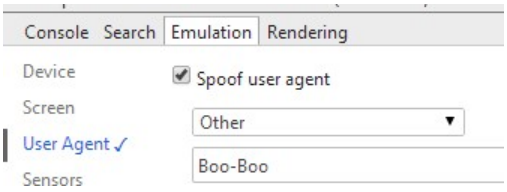
When a web site cannot interpret a User Agent string submitted in the *User-Agent* HTTP request header, it will normally assume the least capable browser (with least features) and provide a simple default view of the requested page. This makes a good test case for testing the responsive design of your pages.

__1. In the *Device* section, click the **Reset** button.

__2. Click the **User Agent** menu option.

__3. In the *User Agent* section, check the **Spoof user agent** checkbox.

__4. Select *Other* in the device drop-down (it should be pre-selected by default), and type in **Boo-Boo** for the User Agent string.



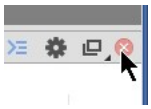
__5. Refresh the browser window (**Ctrl-R**).

Notice a simpler layout of the newly rendered page which was generated by the Yahoo mobile site in response to the unknown User Agent string *Boo-Boo*.

__6. Uncheck **Spoof user agent**.

Part 3 - Lab Clean-up

__1. Close the DevTools panel by clicking the X round icon in its upper right-hand corner.



__2. Close all browsers.

__3. Close your text editor.

Part 4 - Review

In this lab, we reviewed mobile browser emulation support offered by Chrome. Using Chrome's free DevTools with its Device Emulation add-on greatly aids in the testing of mobile web sites from the desktop computer without the need to test a variety of mobile browsers running on a wide range of devices.

