Extra Exercises

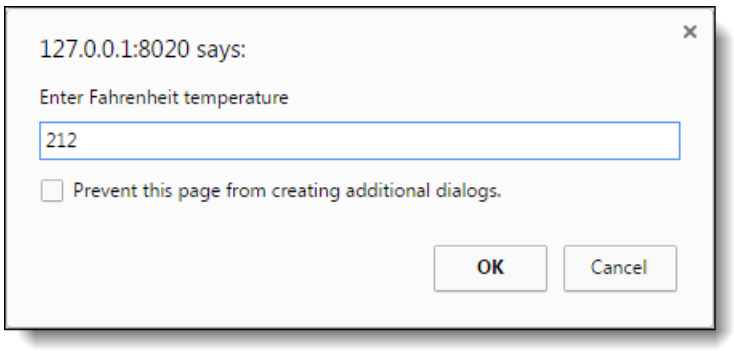
The extra exercises are designed to give you experience with all of the critical JavaScript skills. As a result, some of these exercises will take longer than an hour to complete. In general, the more exercises you do and the more time you spend doing them, the more competent you will become.

Guidelines for doing the extra exercises

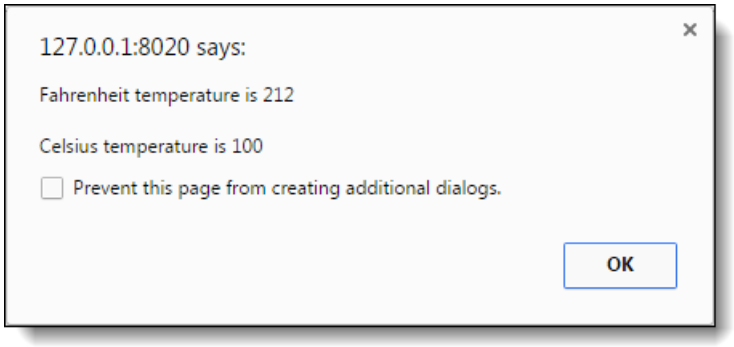
- For all the extra exercises, you will start with the HTML and CSS for the user interface. Then, you supply the JavaScript or jQuery that's required to get the desired results.
- Unless an exercise specifies that you need to modify the HTML or CSS, you won't have to do that.
- Do the exercise steps in sequence. That way, you will work from the most important tasks to the least important.
- If you are doing an exercise in class with a time limit set by your instructor, do as much as you can in the time limit.
- Feel free to copy and paste code from the book applications or exercises that you've already done.
- Use your book as a guide to coding.

Extra 2-1 Convert Fahrenheit to Celsius

In this exercise, you'll create an application that converts Fahrenheit temperatures to Celsius temperatures by using the prompt() and alert() methods. The prompt dialog box should look like this:



The alert dialog box should look like this:



To convert Fahrenheit to Celsius, first subtract 32 from the Fahrenheit temperature. Then, multiply that result by 5/9.

1. Open this file:

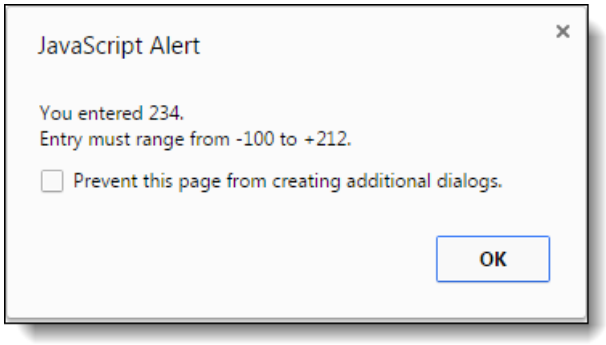
   `exercises_extra\ch02\convert_temps.html`

2. Review the script element in the head section and note that it's empty. You'll write the code for this application within this element.

3. Develop this application.

Extra 3-1 Enhance the Fahrenheit to Celsius application

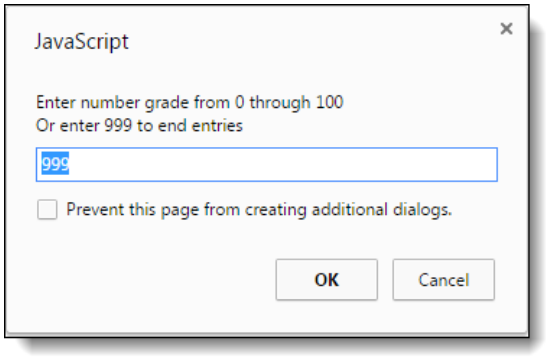
In this exercise, you'll add data validation to the application you created in extra exercise 2-1. You'll also let the user do multiple conversions before ending the application. This is the dialog box for an invalid entry:



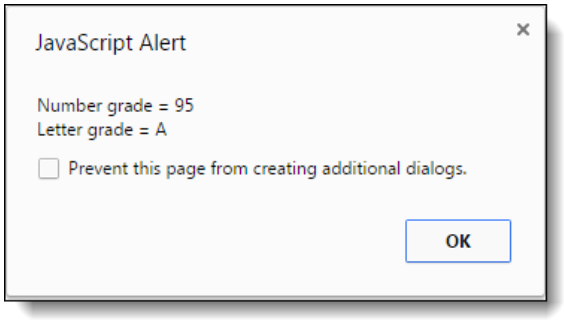
1. If you didn't already do extra exercise 2-1, do it now.
2. Add data validation to the application so it won't do the conversion until the user enters a Fahrenheit temperature between -100 and 212. If the entry is invalid, a dialog box like the one above should be displayed.
3. Add a loop to the code so the user can do a series of calculations without restarting the application. To end the application, the user must enter 999 as the temperature.

Extra 3-2 Convert number grades to letter grades

This exercise will give you some practice using if statements. To start, this application should display a prompt dialog box like the one below that gets a number grade from 0 through 100:



Then, it should display an alert dialog box like the one below that displays the letter grade for that number:



To derive the letter grade, you should use this table:

```
A   88-100
B   80-87
C   68-79
D   60-67
F   < 60
```

1. Open this HTML file:

   exercises_extra\ch03\letter_grade.html

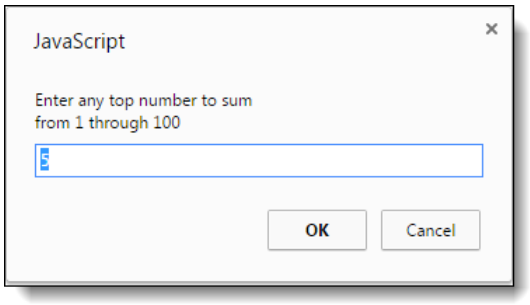
2. In the script element, add the JavaScript code for getting the user's entry while the entry amount isn't 999. This should provide for multiple entries and conversions.

3. Add the JavaScript code for deriving the letter grade from the table above and displaying it in an alert dialog box.

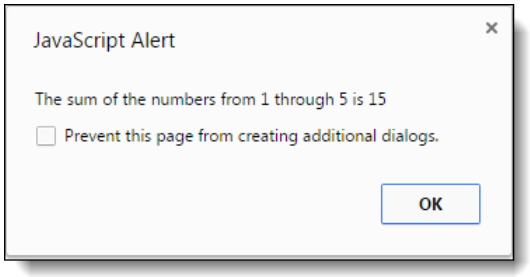
4. If you haven't already done so, add data validation to make sure the entry is a valid number from 0 through 100. If the entry is invalid, the application should just display the starting prompt dialog box. It doesn't need to display a special error message.

Extra 3-3 Create a Sum of Numbers application

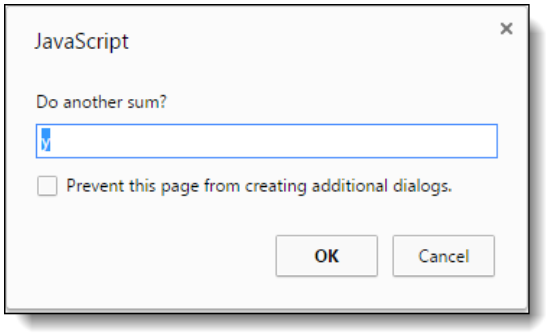
This application will give you a chance to use a for loop. It asks the user to enter a number from 1 through 100 with this prompt dialog box:



Then, it adds all the numbers from one to the user's entry and displays the sum of the numbers in an alert dialog box like this:



Then, to give the user a chance to do multiple entries, this dialog box is displayed:



1. Open this HTML file:

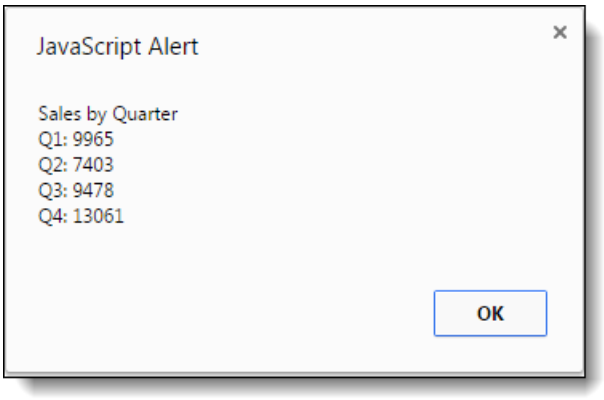
   `exercises_extra\ch03\sum_numbers.html`

2. In the script element, add a do-while loop that prompts the user for an entry from 1 through 100. If the entry is invalid, display an alert box with this message: "Please enter a number between 1 and 100". Then, continue the loop until the entry is valid.
3. After the do-while loop, code a for loop that sums the numbers, and then display the second dialog box above. For instance, the sum for an entry of four is 1 + 2 + 3 + 4.
4. Add a do-while loop around all of the code that uses the third dialog box above to determine whether the first dialog box should be displayed again so the user can enter another number. The application should end for any entry other than "y".

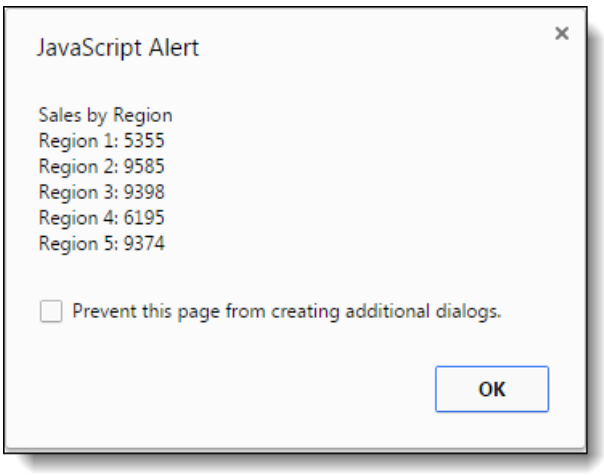
Extra 3-4 Use a Sales array

In this exercise, you'll start with five arrays that represent sales regions, and each array contains four values that represent the quarterly sales for the region. Then, you'll summarize the data in these arrays in three successive alert dialog boxes.

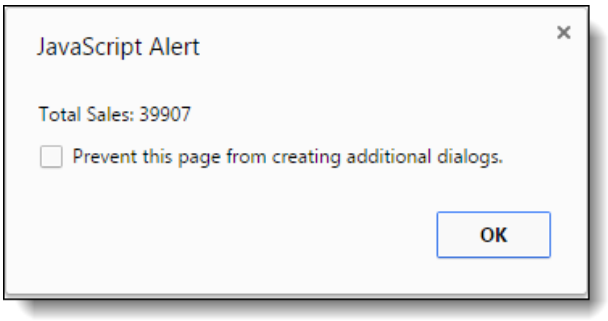
The first one displays sales by quarter:



The second one displays sales by region:



The third one displays total sales for all four quarters and all five regions:



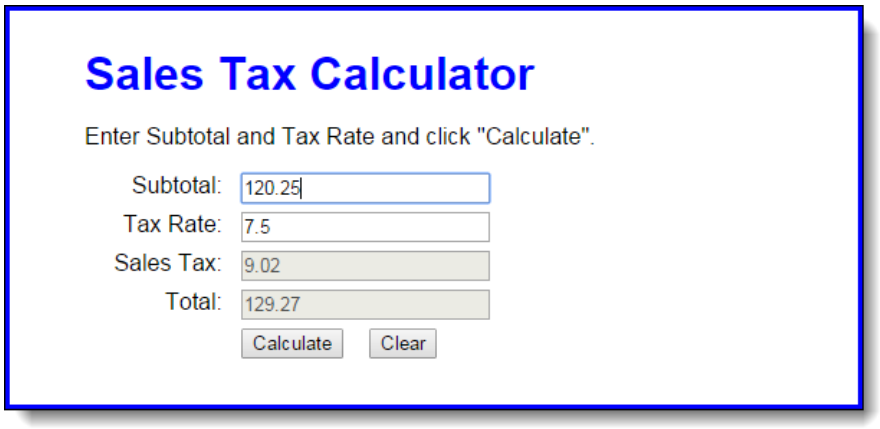
1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch03\sales_array\`

2. In the HTML file, note that the script element refers to the JavaScript file. In the JavaScript file, note that five arrays are declared with four values in each. Each of these arrays represents one sales region, and each of the values in an array represents one sales quarter. For instance, the sales for the third quarter in region 3 were 2710.
3. Write the code for summing the quarterly sales for each the five regions and displaying the first dialog box above.
4. Write the code for getting the data and displaying the second dialog box above.
5. Write the code for getting the data and displaying the third dialog box above.

Extra 4-1 Develop the Sales Tax Calculator

In this exercise, you'll develop an application that calculates the sales tax and invoice total after the user enters the subtotal and tax rate.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch04\sales_tax\`

   Then, run the application to see the user interface shown above, although that interface won't do anything until you develop the JavaScript for it.

2. In the JavaScript file, note that the $ function has been coded for you. It gets the object for the HTML element that's specified by the id attribute.

3. Code an event handler (function) named processEntries() that gets the user entries, calculates the sales tax and total, and displays those results in the text boxes.

4. Code an onload event handler that attaches the processEntries() function to the click event of the Calculate button. Then, test what you have so far.

5. Add data validation to the processEntries() function. The subtotal entry should be a valid, positive number that's less than 10,000. The tax rate should be a valid, positive number that's less than 12. The error messages should be displayed in alert dialog boxes, and the error messages should be:

   Subtotal must be > 0 and < 10000
   Tax Rate must be > 0 and < 12

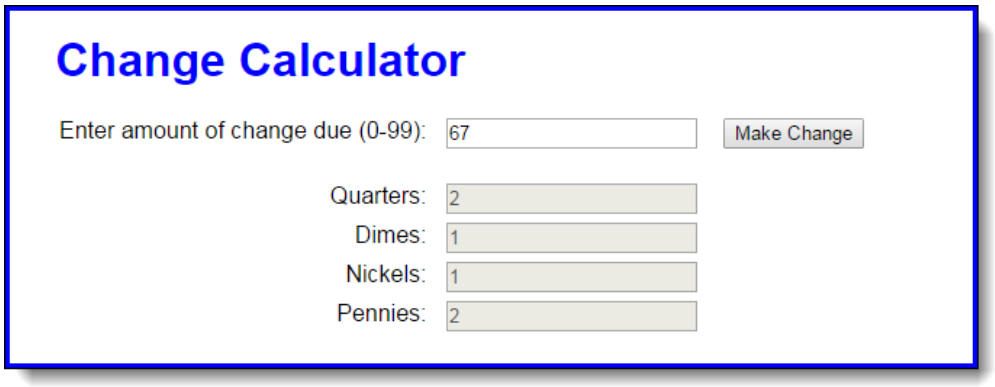
6. Add JavaScript that moves the cursor to the Subtotal field when the application starts and when the user clicks on the Calculate button.

7. Add the JavaScript event handler for the click event of the Clear button. This should clear all text boxes and move the cursor to the Subtotal field.

8. Add JavaScript event handlers for the click events of the Subtotal and Tax Rate text boxes. Each handler should clear the data from the text box.

Extra 4-2 Develop the Change Calculator

In this exercise, you'll develop an application that tells how many quarters, dimes, nickels, and pennies are needed to make change for any amount of change from 0 through 99 cents. One way to get the results is to use the divide and modulus operators along with the parseInt() method for truncating the results so they are whole numbers.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch04\change_maker\`

   Then, run the application to see the user interface shown above, although that interface won't do anything until you develop the JavaScript for it.

2. In the JavaScript file, note that the $ function has already been coded.

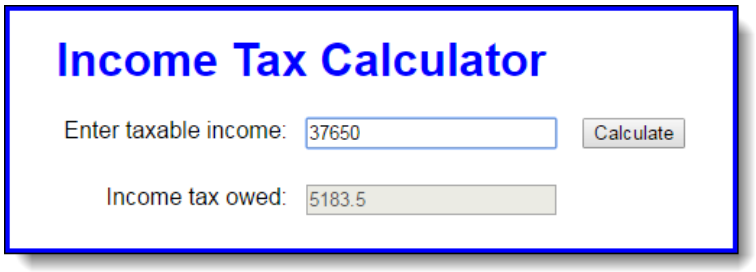
3. Code an event handler named processEntry() that gets the user's entry and checks to make sure that it is a number between 0 and 99. If it isn't, display an alert dialog box for the error. If it is valid, call a function named makeChange() and pass it the user's entry.

4. Code the makeChange() function, which should have one parameter that accepts the user's entry. This function shouldn't return anything, but it should display the results in the text boxes for Quarters, Dimes, Nickels, and Pennies.

5. Code an onload event handler that attaches the processEntry() event handler to the click event of the Make Change button. Then, test this application.

Extra 4-3 Develop the Income Tax Calculator

In this exercise, you'll use nested if statements and arithmetic expressions to calculate the federal income tax that is owed for a taxable income amount.



This is the 2017 table for the federal income tax on individuals that you should use for calculating the tax:

Taxable income		Income tax	
Over…	**But not over…**		**Of excess over…**
$0	$9,275	$0 plus 10%	$0
$9,275	$37,650	$927.50 plus 15%	$9,275
$37,650	$91,150	$5,183.75 plus 25%	$37,650
$91,150	$190,150	$18,558.75 plus 28%	$91,150
$190,150	$413,350	$46,278.75 plus 33%	$190,150
$413,350	$415,050	$119,934.75 plus 35%	$413,350
$415,050		$120,529.75 plus 39.6%	$415,050

1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch04\income_tax\`

   Note that the JavaScript file has some starting JavaScript code for this application, including the $ function and an onload event handler that attaches a function named processEntry() to the click event of the Calculate button and moves the focus to the first text box.

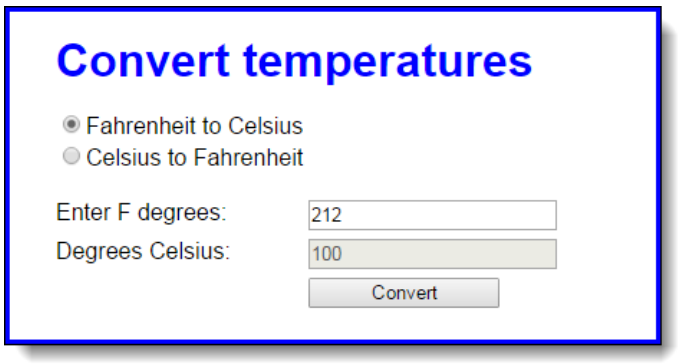
2. Code the processEntry() function. It should get the user's entry and make sure it's a valid number greater than zero. If it isn't, it should display an error message. If it is valid, it should pass the value to a function named calculateTax(), which should return the tax amount. That amount should then be displayed in the second text box. The focus should be moved to the first text box whether or not the entry is valid.

3. Code the calculateTax() function, but to start, just write the code for calculating the tax for any amount within the first two brackets in the table above. The user's entry should be converted to an integer, and the tax should be rounded to two decimal places. To test this, use income values of 9275 and 37650, which should display taxable amounts of 927 and 5183.50.

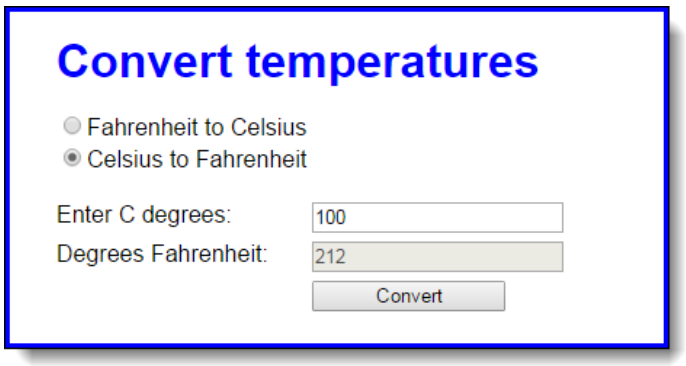
4. Add the JavaScript code for the next tax bracket. Then, if you have the time, add the JavaScript code for the remaining tax brackets.

Extra 6-1 Develop the Temperature Converter

In this exercise, you'll use radio buttons to determine whether the conversion is from Fahrenheit to Celsius or vice versa. You'll also modify the DOM so the labels change when a radio button is clicked. When the application starts, it will look like this:



When the user clicks on the second radio button, the labels will change so the interface will look like this:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch06\convert_temps\`

2. Note that the JavaScript file has some starting JavaScript code, including the $ function, a clearTextBoxes() function, and an onload event handler that attaches three event handlers named convertTemp(), toCelsius(), and toFahrenheit().

3. Code the toFahrenheit() function that is executed when the user clicks on the second radio button. It should change the text in the labels for the text boxes so they read as in the second interface above. It should also call the clearTextBoxes() function to clear the text boxes.

4. Code the toCelsuis() function that is executed when the user clicks on the first radio button. It should change the text in the labels for the text boxes so they read as in the first interface above. It should also call the clearTextBoxes() function to clear the text boxes.

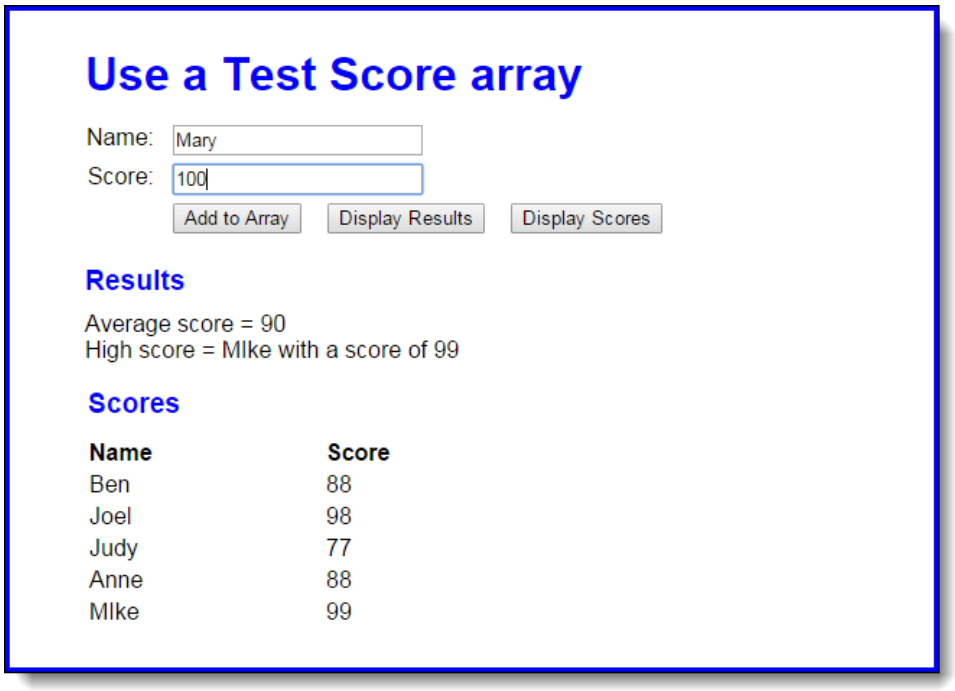
5. Code the convertTemp() function without any data validation. It should calculate the temperature based on which button is checked. To convert

Fahrenheit to Celsius, first subtract 32 from the Fahrenheit temperature, and then multiply that result by 5/9. To convert Celsius to Fahrenheit, first multiply Celsius by 9/5, and then add 32. The result in either case should be rounded to zero decimal places.

6. Add data validation to the convertTemp() function. The only test is whether the entry is a valid number. If it isn't, this message should be displayed in a dialog box: "You must enter a valid number for degrees."
7. Add any finishing touches to the application like moving the focus to the first text box whenever that's appropriate.

Extra 6-2 Use a Test Score array

In this exercise, you'll work with an array and you'll add nodes to the DOM to display the Results and the Scores.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch06\test_scores\`

   Then, run the application to see the user interface shown above, although that interface won't do anything until you develop the JavaScript for it.

2. At the start of the JavaScript file, you'll see the declarations for two arrays: one for names and one for scores, and each array contains four elements. You'll also see the code for the $ function as well as an onload event handler that attaches three functions named addScore(), displayResults(), and displayScores() to the click events of the buttons.

3. Write the displayResults function(). It should derive the average score and the highest score from the arrays and then display the results in the div element with "results" as its id, as shown above. To display the results, you need to add nodes to the DOM with the heading as an h2 element and the average and highest scores as <p> elements. The easiest way to do that is to use the innerHTML property as shown in figure 6-13.

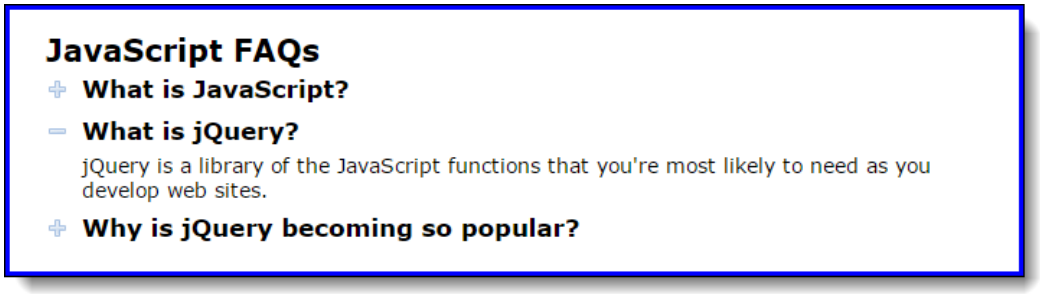
4. Write the displayScores() function. It should get the names and scores from the arrays and display them as rows in the HTML table element with "scores_table" as its id, as shown above.

5. Write the addScore() function. It should add a name and score to the two arrays. To test whether this works, you can click the Display Scores button and see if the new name and score have been added to the table.

6. If you haven't already done it, add data validation to addScore() function. The Name entry must not be empty and the Score entry must be a positive number from 0 through 100. If either entry is invalid, use the alert() method to display this error message: "You must enter a name and a valid score".
7. Make sure that your application moves the cursor to the Name field when the application starts and after a name and score have been added to the array.

Extra 6-3 Modify the FAQs application

This exercise has you make a minor modification to the FAQs application. When you're done, this application should work the same as before, except that only one answer can be displayed at a time. In other words, when the user clicks on a heading to display the answer, the other answers must be hidden.



1. Open the HTML and JavaScript files in this folder:

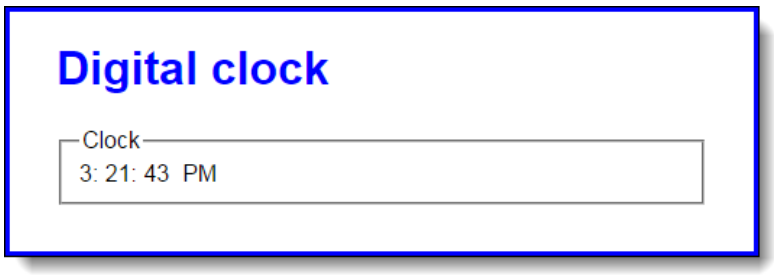
   `exercises_extra\ch06\faqs\`

   Then, run the application to refresh your memory about how it works.

2. Add code to the toggle() function so only one answer can be displayed at a time. To do that, create an array of the h2 elements. Then, use a for loop to go through the h2 elements in the array and remove the class attribute for all h2 elements that aren't the one that has been clicked. You also need to remove the class attributes for all of the div siblings of the h2 elements that weren't clicked.

Extra 7-1 Develop the Clock application

In this exercise, you'll create an application that displays the current time in hours, minutes, and seconds. The display should use a 12-hour clock and indicate whether it's AM or PM. The application looks like this:



To convert the computer's time from a 24-hour clock to a 12-hour clock, first check to see if the hours value is greater than 12. If so, subtract 12 from the hours value and set the AM/PM value to "PM". Also, be aware that the hours value for midnight is 0.

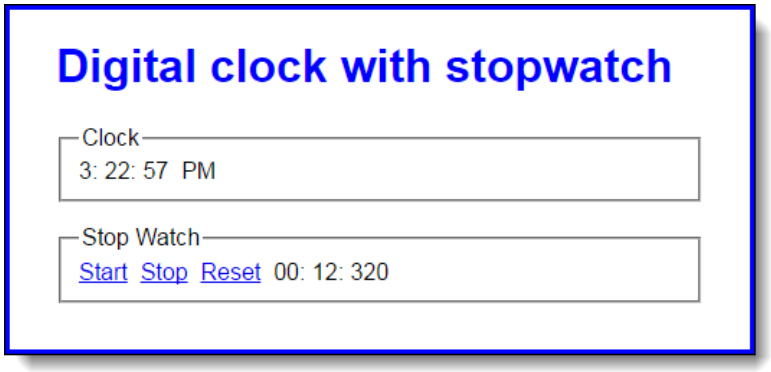
1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch07\clock\`

2. In the JavaScript file, note that four functions are supplied. The $ function. The start of a displayCurrentTime() function. The padSingleDigit() function that adds a leading zero to single digits. And the start of an onload event handler.
3. In the displayCurrentTime() function, add code that uses the Date object to determine the current hour, minute, and second. Convert these values to a 12-hour clock, determine the AM/PM value, and display these values in the appropriate span tags.
4. In the onload event handler, code a timer that calls the displayCurrentTime() function at 1 second intervals. Also, make sure that the current time shows as soon as the page loads.

Extra 7-2 Add a stopwatch to the Clock application

In this exercise, you'll add a stopwatch feature to the application you created in extra exercise 7-1. The stopwatch will display elapsed minutes, seconds, and milliseconds. The enhanced application looks like this:



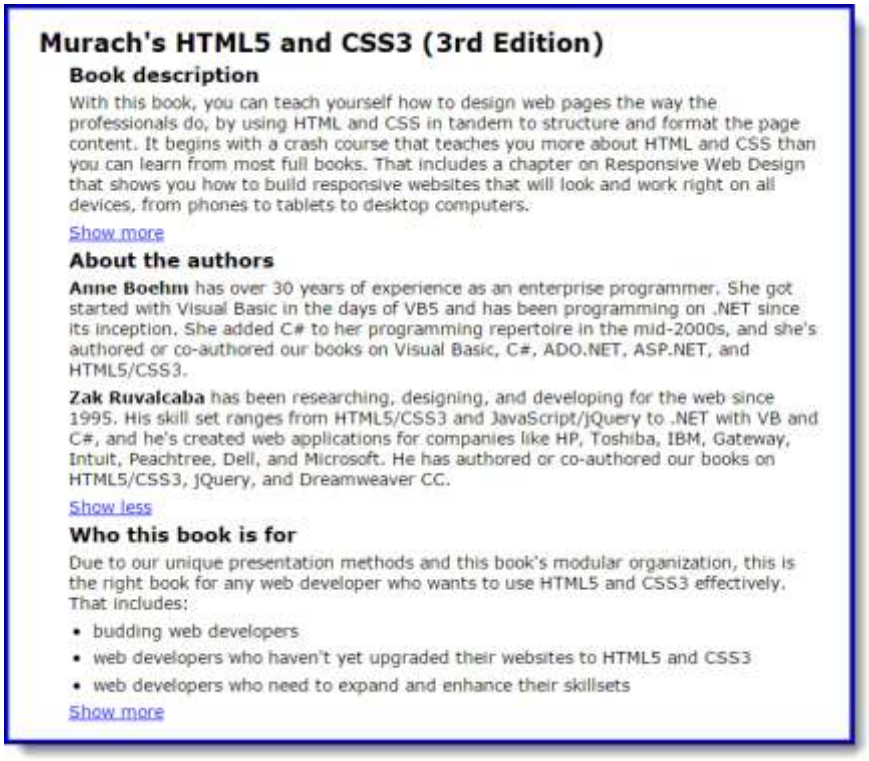
1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch07\clock_stopwatch\`

2. In the JavaScript file, note the $, displayCurrentTime(), padSingleDigit(), and onload event handler functions from the Clock application. In addition, global stopwatchTimer, elapsedMinutes, elapsedSeconds, and elapsedMilliseconds variables and starts for the tickStopwatch(), startStopwatch(), stopStopwatch(), and resetStopwatch() functions are supplied.
3. In the tickStopwatch() function, add code that adds 10 milliseconds to the elapsedMilliseconds variable and then adjusts the elapsedMinutes and elapsedSeconds variables accordingly. Then, add code that displays the result in the appropriate span tags in the page.
4. In the startStopwatch() function, add code that starts the stopwatch. Be sure to cancel the default action of the link too, but don't worry about providing for cross-browser compatibility.
5. In the stopStopwatch() and resetStopwatch() functions, add code that stops the stopwatch. Also, in the resetStopwatch() function, reset the elapsed time and the page display. Be sure to cancel the default action of the links too.
6. In the onload event handler, attach the stopwatch event handlers to the appropriate links.

Extra 8-1 Develop an Expand/Collapse application

In this exercise, you'll develop an application that displays the first paragraph of text for three topics and then lets the user click a link to expand or collapse the text for each topic.



1. Open the HTML, CSS, and JavaScript files in this folder:

   `exercises_extra\ch08\expand_collapse\`

   Then, run the application to see that the first paragraph of text is displayed for each topic, along with a link that lets you display additional text. Note, however, that the links don't work.

2. Review the HTML to see that each topic consists of two div elements followed by an <a> element. Notice that a class named "hide" is assigned to the second div element of each topic. Then, review the style rule for this class.

3. In the JavaScript file, add an event handler for the ready() event method.

4. Within the function for the ready event handler, code an event handler for the click() event method of the <a> elements. This event handler should start by using the toggleClass() method to add or remove the "hide" class from the div element above the link element that's clicked depending on whether that class is present.

5. Complete the click event handler by testing if the div element above the current link element has the "hide" class. If it doesn't, change the text for the link to "Show less". If it does, change it back to "Show more".

Extra 8-2 Develop an Image Gallery application

In this exercise, you'll develop an Image Gallery application that displays different images when the user clicks on the links at the top of the page. This works like the Image Swap application of figure 8-14.

1. You'll find the HTML, CSS, and image files for this application in this folder:

   `exercises_extra\ch08\image_gallery\`

   You'll also find an empty JavaScript file named image_gallery.js. You can add your code to this file.

2. In the JavaScript file, add an event handler for the ready() event method.

3. Use the each() method to run a function for each <a> element in the unordered list of items. Then, add jQuery code that gets the URL and caption for each image and preloads the image. You can get the URL from the href attribute of the <a> element, and you can get the caption from the title attribute.

4. Add an event handler for the click event of each link. The function for this event handler should accept a parameter named evt. The jQuery code for this event handler should display the image and caption for the link that was clicked. In addition, it should use the evt parameter to cancel the default action of the link.

5. Add a jQuery statement that moves the focus to the first link on the page when the page is loaded.

Extra 9-1 Modify an Image Swap application

In this exercise, you'll modify another Image Swap application so it uses effects to display and hide the images.

1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch09\image_swap\`

   Review the code in the JavaScript file to see that it's identical to the code for the Image Swap application in chapter 8. Now, run this application to see how it works.

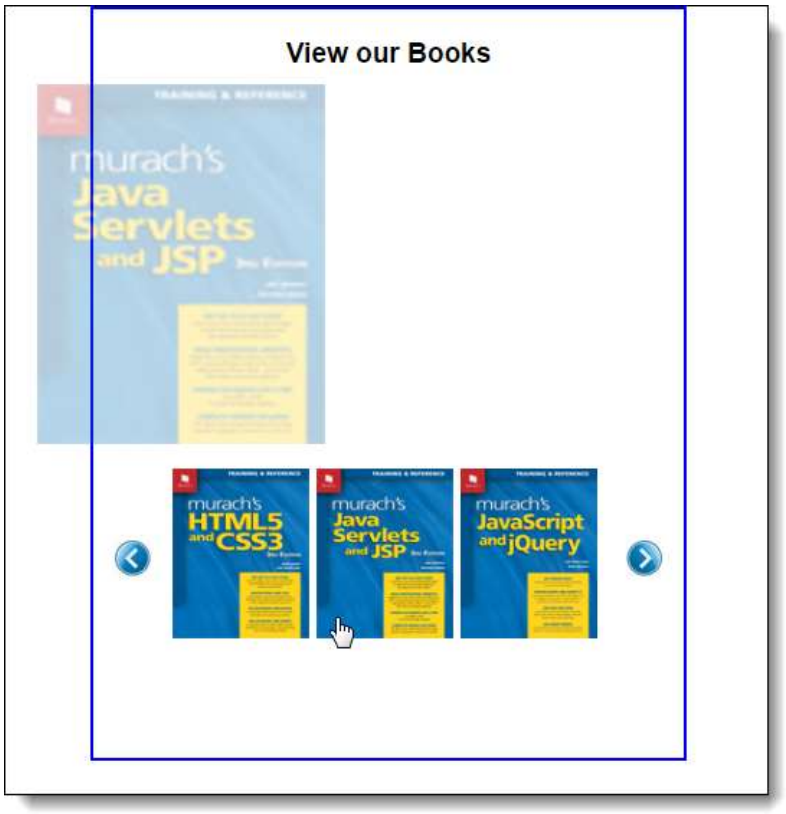
2. Add statements to the JavaScript file that fade the caption and image out over a duration of one second.

3. Modify the statements that display the new caption and image so the caption and image are faded in over a duration of one second. Then, run the application to see that this doesn't work the way you might expect. Instead, the new caption and image are displayed and then faded out and back in.

4. Add a callback function to the statement that fades out the image. Then, move the statements that display the new caption and image within this function. Now, the old caption and image should fade out and the new caption and image should fade in.

Extra 9-2 Modify a carousel to display an enlarged image using animation

In this exercise, you'll modify a carousel application so that when an image in the carousel is clicked, an enlarged image is displayed using animation.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch09\carousel\`

   Then, run the application and notice that an enlarged image of the first book in the carousel is displayed.

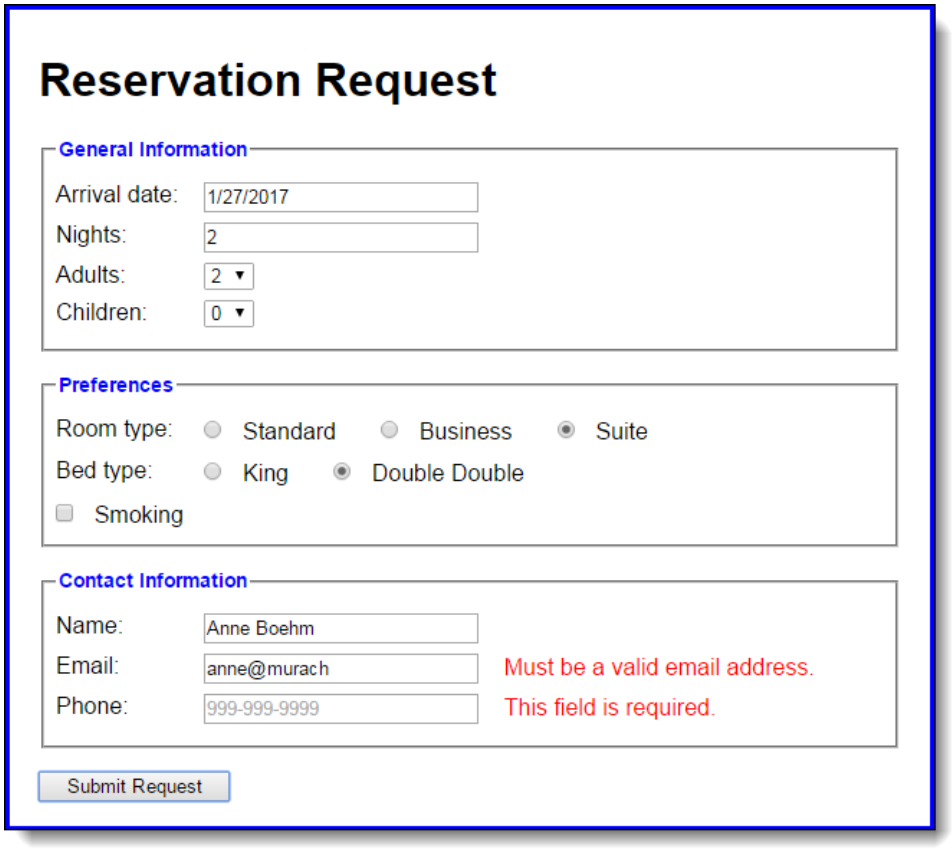
2. Review the HTML for the application, and notice that it contains an img element with an id of "image" following the heading. Also notice that the href attributes of the <a> elements in the carousel are set to the URL of the enlarged image to be displayed when the associated carousel image is clicked.

3. Code an event handler for the click event of the <a> elements in the list. This event handler should start by getting the URL for the image to be displayed. Then, it should assign this URL to the enlarged image.

4. Add animation to the click event handler so the opacity of the current image is set to 0 and 205 is subtracted from the left margin of the image over a period of 1 second. Use a callback function to reverse this animation. This function should also contain the statement that sets the URL for the enlarged image. The effect will be for the current image to fade out as it slides to the left, and then for the new image to fade in as it slides to the right.

Extra 10-1 Use JavaScript to validate a form

In this exercise, you'll use JavaScript to validate a reservation request form.



1. Open the index.html and reservation.js files in this folder:

   `exercises_extra\ch10\reservation\`

   Then, run the application and click the Submit Request button to see the page that's displayed when the form is submitted to the server.

2. In the JavaScript file, notice that the ready event handler contains the declaration for a variable named emailPattern that contains the pattern that will be used to validate the email address.

3. Code a statement that moves the focus to the Arrival date text box.

4. Code an event handler for the submit event of the form. This event handler should validate the user entries and cancel the submission of the form if any of the entries are invalid. The validation is as follows:

   A value must be entered into each text box.

   The number of nights must be numeric.

   The email address must match the pattern that's provided.

   Be sure to trim the entries and put them back into the controls regardless of whether the entries are valid.

Extra 11-1 Modify the Carousel application

In this exercise, you'll modify the way the bxSlider plugin is used with the Carousel application. To do that, you may have to refer to the options for this plugin that are described at http://bxslider.com.

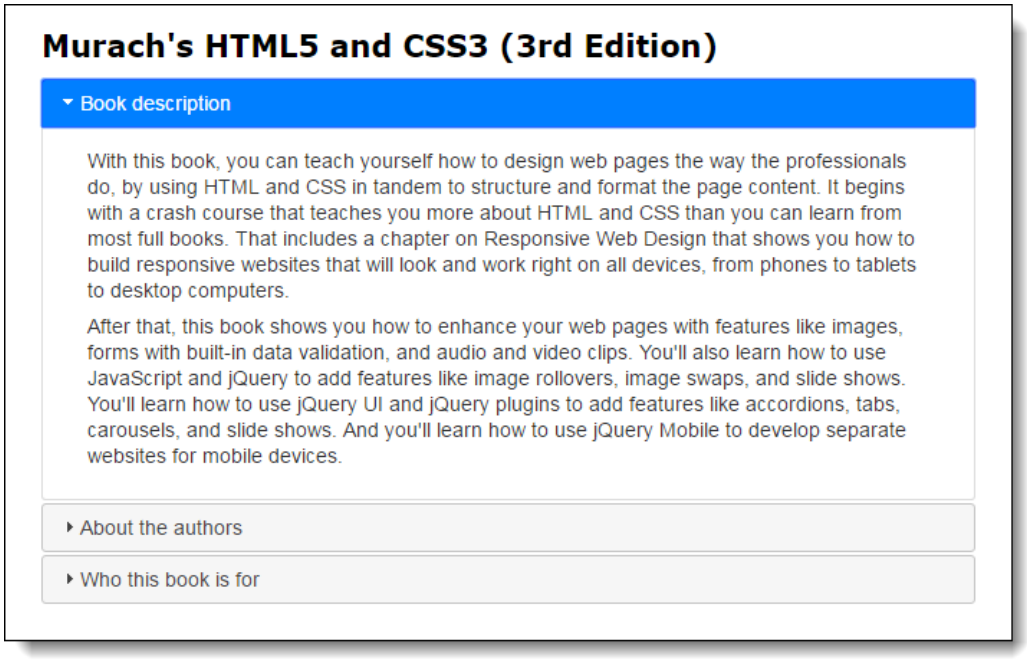
1. Open the HTML file in this folder and review its code:

   `exercises_extra\ch11\carousel\`

2. Modify the jQuery code that calls the bxSlider plugin so the first image that's displayed is selected randomly. To do that, you can use the randomStart option.
3. Add an option to the code that calls the bxSlider plugin so the carousel moves one slide at a time.
4. Modify the code for the bxSlider plugin so that only one image is displayed at a time.
5. Add an option to the code that sets the time between the automatic transitions to 3 seconds.
6. Add options to the code so a pager is displayed in the format shown above. This pager should be displayed in the paragraph with the id of "pager" that's below the list of images.

Extra 11-2 Replace a Tabs widget with an Accordion widget

In this exercise, you'll modify an application that uses a Tabs widget to display the content of three panels to an application that uses an Accordion widget to display those panels.



1. Open the HTML file in this folder:

   `exercises_extra\ch11\accordion\`

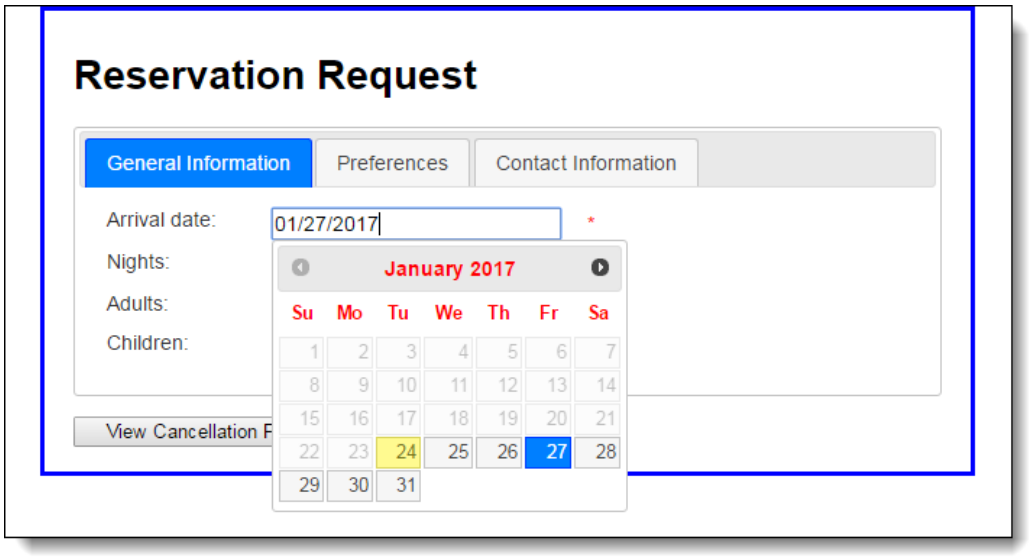
   Now, run this application to refresh your memory about how it works.

2. Review the HTML, and then modify it so it can be used with the Accordion widget. To do that, you'll need to omit the list of tabs and add an h3 element for each panel. Then, make any additional changes as appropriate.

3. Modify the jQuery code so it uses an Accordion widget. A panel should be opened when the user clicks on it. If the panel is already opened, it should be closed.

Extra 11-3 Use widgets in a form

In this exercise, you'll modify a Reservation application so it uses Tabs, Datepicker, and Dialog widgets.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch11\reservation\`

   Now, run this application to see what the user interface looks like.

2. Modify the HTML so the contents of the three fieldset elements can be implemented as three tabs of a Tabs widget. When you do that, you can delete the fieldset and legend elements, and you can set the headings for the tabs to the content of the legend elements.

3. Add the jQuery code that implements the tabs.

4. Add the jQuery code that implements the Datepicker widget for the arrival date. The date can be from the current date to 90 days after the current date.

5. Code an event handler for the click event of the View Cancellation Policies button. This event handler should display the div element with an id of "dialog" as a modal Dialog widget.

Extra 12-1 Convert an Ajax application from XML to JSON

In this exercise, you'll modify the Ajax application in figure 12-11 so it uses a JSON file instead of an XML file. The web page for this application should be the same as it is when the XML file is used:

1. You'll find the HTML, CSS, and JavaScript files for the XML version of this application in this folder:

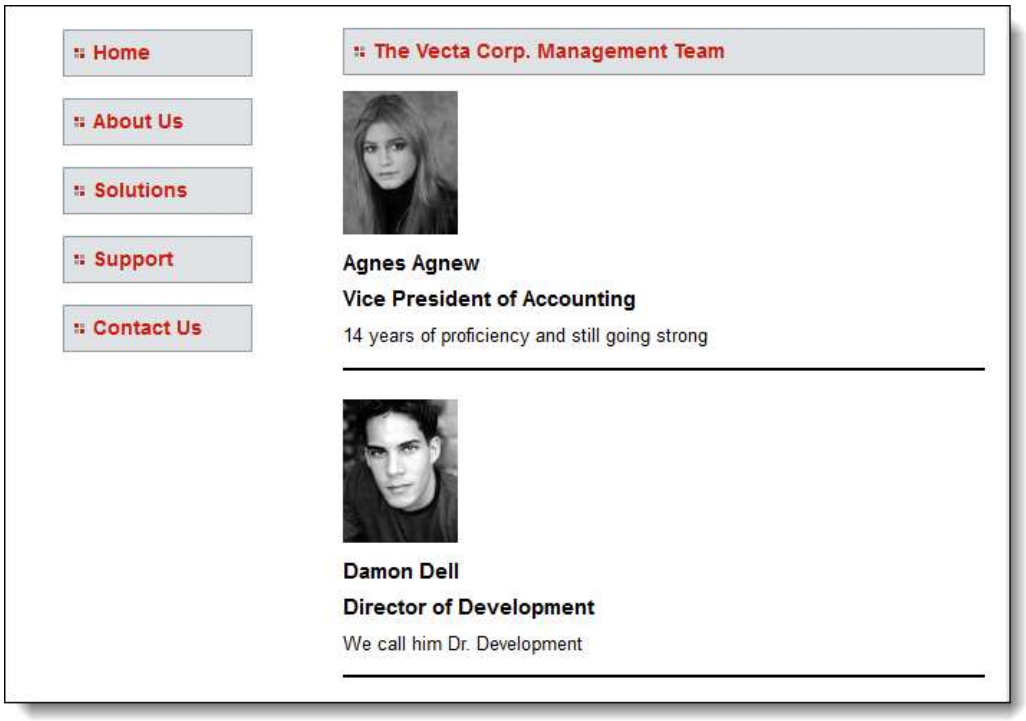
   `exercises_extra\ch12\xml_to_json\`

   You'll also find a file named team.json that you should use for the Ajax request instead of the file named team.xml.

2. Modify the $.ajax() method that gets the data from the XML file so it gets the data from the JSON file.

Extra 12-2 Enhance an Ajax application so it uses an expanded JSON file

In this exercise, you'll modify the JSON application in figure 12-8 so it uses an expanded JSON file. The resulting page should look like this:



1. You'll find the HTML, CSS, JavaScript, JSON, and image files for this application in this folder:

   `exercises_extra\ch12\get_json\`

   Now, run this application to refresh your memory about how it works.

2. Review the data in the team.json file. Then, review the image files in the staff folder to see the images that are referred to by image fields in the JSON file.

3. Modify the JavaScript so it displays the image, full name, title, and tag line for each manager, but not the biography. These four data items should be in img, h3, h3, and <p> elements respectively. If you do this right, you shouldn't have to modify the CSS for this application because the CSS formats the h3 headings and applies a bottom border to the<p> element.

Extra 12-3 Load speakers as they're requested

In this exercise, you'll write the JavaScript code for loading data from JSON files whenever the links in the sidebar are clicked. If, for example, the user clicks on the third link in the sidebar, the Ajax request will return the data from the JSON file named chua.json. This will show how seamlessly Ajax can add new data to a page without reloading the page in the browser.

1. You'll find the HTML, CSS, JavaScript, JSON, and image files for this application in this folder:

   `exercises_extra\ch12\speakers\`

2. Run this application to see that it starts with the information for the first speaker displayed, but the links in the sidebar don't work. Then, look at the HTML for this application to see that the starting HTML provides the content for the first speaker.

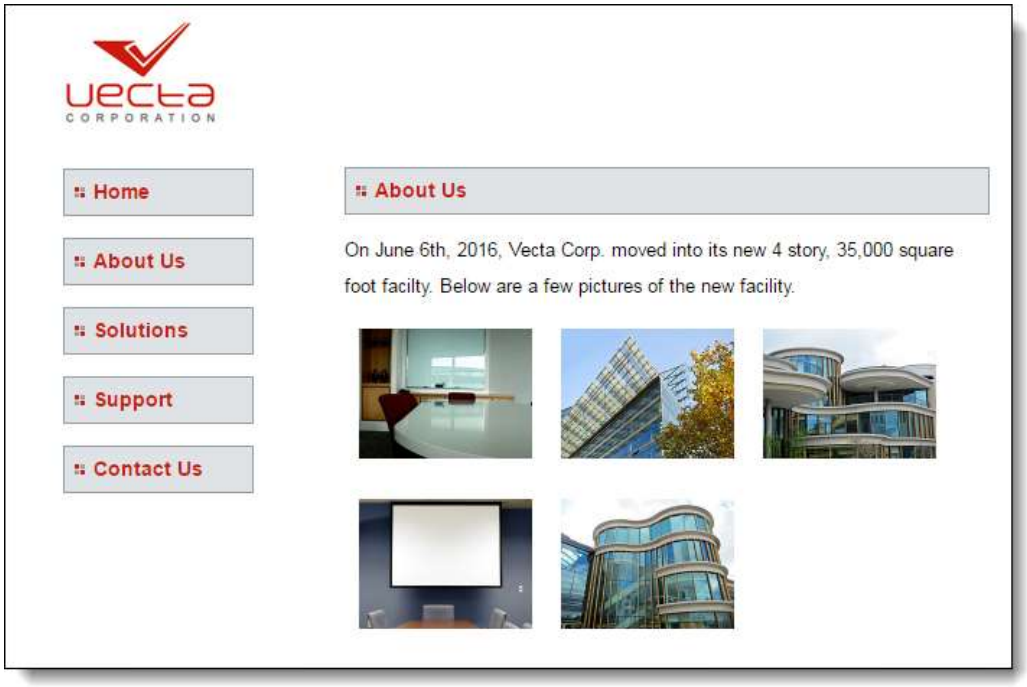
3. Review the data in the files in the json_files folder. There, you can see that there is one file for each speaker.

4. In the speakers.js file, write the JavaScript code for this application. Within the click event handlers for the <a> elements in the sidebar, use the title attribute for each link to build the name of the JSON file that needs to be retrieved. Then, use that name to get the data for the speaker, enclose that data in HTML elements that are just like the ones that are used in the starting HTML for the first speaker, and put those elements in the main element in the HTML. That way, the CSS will work the same for the Ajax data as it did for the starting HTML.

5. Don't forget to clear the elements from the main element before you put the new Ajax data in that element.

Extra 12-4 Display a gallery of Flickr photos

In this exercise, you'll create an application that displays the photos with a specific tag for a specific user. You'll also use the Lightbox plugin to display a larger version of a photo when it's clicked. When you're done, the page should look like this:



1. You'll find the HTML, CSS, and JavaScript files for this application in this folder:

   `exercises_extra\ch12\flickr_gallery\`

2. Review the HTML file to see that it includes a link element for the Lightbox CSS file and a script element for the Lightbox JavaScript file. It also includes a div element with the id "new_building" where the photo gallery will be displayed.

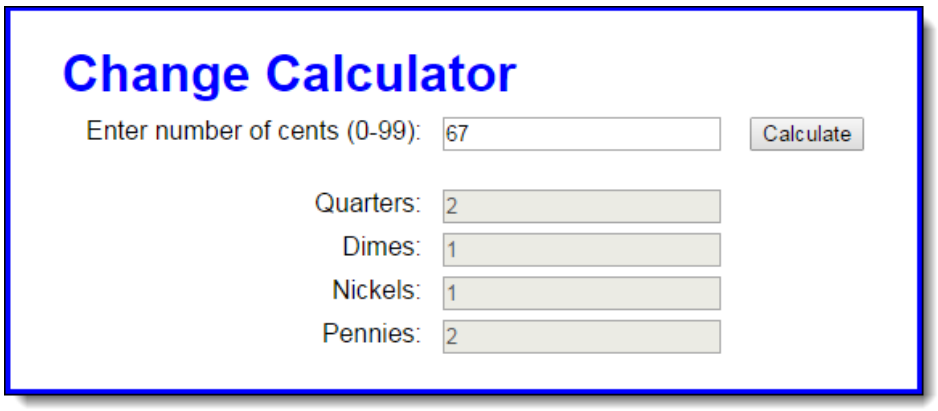
3. Within the ready event handler for the document, declare a variable to store the URL for the Flickr feed. The URL should be for the user with the id "82407828@N07" and the tag "vectacorpbuilding", and the return format should be set to JSON.

4. Add code to get the data from the feed. For each item that's returned, add an img element for the photo to the gallery.

5. Enhance the code for the photo gallery so each photo is wrapped in an <a> element that provides for displaying the photo in a Lightbox. All of the photos should be treated as a group.

Extra 13-1 Develop the Change Calculator

In this exercise, you'll create an application that displays the minimum number of quarters, dimes, nickels, and pennies that make up the number of cents specified by the user. The application interface looks like this:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch13\change_calculator\`

2. In the JavaScript file, note the jQuery ready event handler, as well as the start of an event handler for the click event of the Calculate button. Also note that the ready event handler sets the focus on the cents text box.

3. In the event handler for the Calculate button, get the value entered by the user and make sure it's an integer that's between 0 and 99. If it isn't, display an alert dialog box with this message: "Please enter a valid number between 0 and 99".

4. If the number entered by the user is valid, write code to calculate the number of coins needed for the cents entered by the user. Start with the quarters and work your way down to the pennies. Use the Math.floor() method to round your results to the next lower integer whenever needed. And use the number of cents remaining from the last calculation as the starting point for the next calculation.

5. Display the number for each coin in the corresponding text box. Be sure to display whole numbers. Finally, set the focus on the cents text box for the next calculation.

Extra 13-2 Develop the Calendar application

In this exercise, you'll create an application that displays a calendar for the current month:

Note: To build this calendar, you're going to need the getDay() method of a Date object. This method returns the number of the day of the week (0 for Sunday, 1 for Monday, etc.).

1. Open the HTML, CSS, and JavaScript files in this folder:

   `exercises_extra\ch13\calendar\`

2. In the HTML file, note the span element within the h1 element that will display the month name and year. Note also the table element that contains one row. To build the calendar, you need to add rows to this table after the row that it already contains.

3. In the CSS file, note the style rule for the td elements of the table. The rules in this set will format the calendar as shown above.

4. In the JavaScript file, note the ready event handler with two functions. A getMonthText() function that accepts the number for a month and returns the month name in text. And the start of a getLastDayofMonth() function.

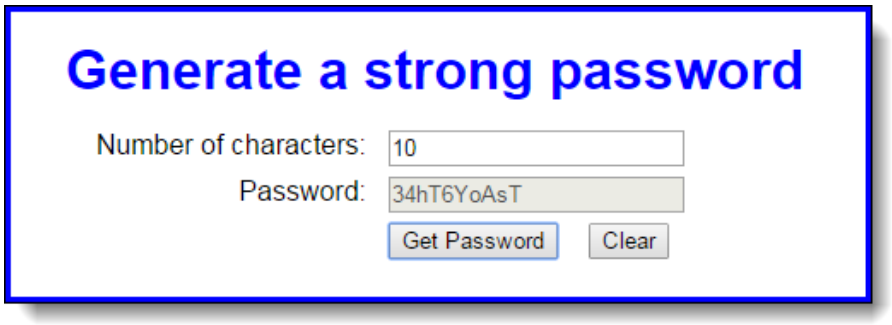
5. Write the code for the getLastDayofMonth() function. It should use the number passed in the currentMonth parameter to calculate and return the last day of the current month. See figure 13-10 for ideas on how to code this.

6. In the ready event handler, write the code that gets and displays the name of the current month and the current year above the month table.

7. In the ready event handler, write the code that loops through the days of the month to create the rows for the calendar. Remember to deal with the blank dates that can occur at the beginning of the first week and the end of the last week of the month. Use a tr element for each new row and td elements within the rows for the days of the months. To display the rows, use the jQuery html() method of the calendar table, but remember that the new rows have to go after the row that's already in the HTML.

Extra 13-3 Develop a password generator

In this exercise, you'll develop an application that generates strong passwords of the length entered by the user. The interface looks like this:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch13\password\`

2. In the JavaScript file, note that the jQuery ready event handler contains three functions: the getRandomNumber() function, the handler for the click event of the Get Password button, and the handler for the click event of the Clear button.

   The handler for the Get Password button clears the password text box and has a string variable that contains several characters, but it doesn't do anything else. The handler for the Clear button resets the text boxes and moves the focus to the first text box.

3. In the handler for the Get Password button, get the value entered by the user and make sure it's a number. If it isn't, display an alert dialog box with this message: "Please enter a valid number".

4. If the number entered by the user is valid, code a for loop that iterates that number of times. In each iteration of the loop, randomly select one of the characters from the chars variable and concatenate it to the password variable.

5. When the loop is finished, display the password in the password textbox.

Extra 14-1 Use a switch statement to enhance code that validates a date

In this exercise, you'll improve the code that validates a date in the Account Profile application. When you're done, the application will correctly validate dates like 2/30/2017 or 11/31/2017. The interface looks like this:

1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch14\profile\`

   Then, run the application to see the user interface shown above.

2. Enter values for all the values on the form. For the Date of Birth field, enter the invalid date 11/31/1980, and click Save. Note that the application accepts this date as valid.

3. Change the Date of Birth to 13/31/1980, and click Save. Note that this time the application doesn't accept the date.

4. In the JavaScript file, note that the ready event handler contains an isDate() function and a handler for the click event of the Save button that contains the validation code.

5. In the isDate() function, remove the if statement that makes sure the value of the day variable isn't greater than 31. Replace it with an else statement that contains a switch statement that evaluates the value of the month variable.

6. Code a case label for the value 2 that returns false if the value of the day variable is greater than 28. Note that this doesn't handle leap years, but that's OK for this exercise.

7. Code case labels for the values 4, 6, and 9 that fall through to the case label for 11. The code for case label 11 should return false if the value of day is greater than 30.

8. Finally, code a default case that returns false if the value of day is greater than 31.

9. Run the application and test it with the invalid dates from steps 2 and 3. The application should now correctly identify both of these dates as invalid.

Extra 14-2 Adjust a regular expression pattern

In this exercise, you'll adjust a regular expression pattern that validates phone numbers so it accepts more options. The application interface looks like this:

1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch14\regex\`

   Then, run the application to see the user interface shown above.

2. Click on the Validate button and note that the application correctly says the default phone number (123-456-7890) is in a valid format. Now, change the phone number to look like the one shown above and click Validate again. This time, the application says the phone number is invalid.

3. In the JavaScript file, note that the ready event handler contains a handler for the click event of the Validate button that contains the validation code.

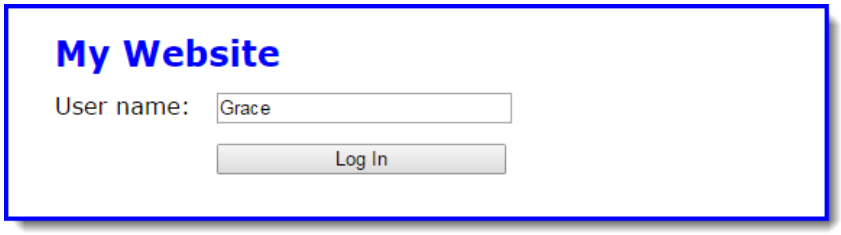
4. Change the regular expression pattern in the pattern variable so the phone number can contain an optional "1-" prefix. The best way to do this is to copy the pattern variable to a new line and then comment out the original. This way, you can refer to the original pattern as you adjust it.

5. When the validation in step 4 is working correctly, change the pattern so the phone number can also contain either dashes or periods. Again, it's best to make a copy so you can refer to what came before.

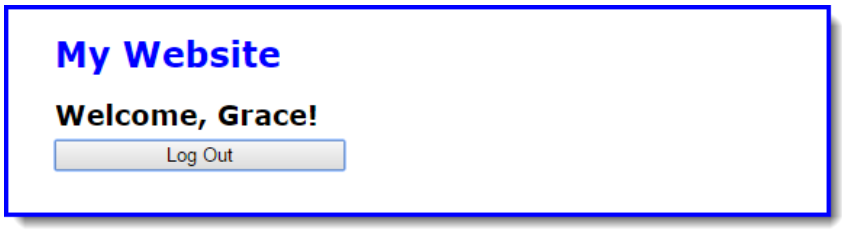
6. When the validation in step 5 is working correctly, change the pattern so the phone number can have optional parentheses around the area code. To accommodate this change, you'll want to allow blank spaces instead of dashes or periods after the optional "1" and after the area code.

Extra 15-1 Navigate between pages and save data in a cookie

In this exercise, you'll develop an application that allows you to log in, save the user data in a cookie, navigate to a new page, and log out. The interface looks like this initially:



And the interface looks like this after you've logged in:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch15\login\`

   Then, run the application to see the user interface shown above.

2. Review the code in the cookies.js file. As you can see, it contains starts for three functions, getCookieByName(), setCookie(), and deleteCookie(). The getCookieByName() function returns an empty string, while the other two contain no code.

3. Update each of these functions so they perform the tasks described by their names. Use the examples in figures 15-6 through 15-8 as a guide.

4. Display the index.html file and notice that its embedded JavaScript code uses the functions that you just updated. Then, find the two places in the code where you need to redirect to the login.html page. Use the location object to do that.

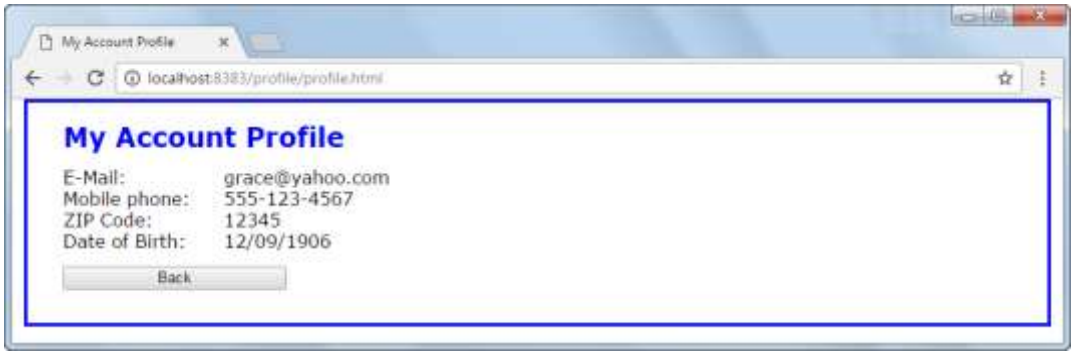
5. Display the login.html file and notice that its embedded JavaScript code uses the functions that you just updated. Then, find the place in the code where you need to redirect back to the index.html page. Use the location object to do that.

6. Run the application, enter a user name, and click Log In. When the login.html page displays, press F12 to open the developer tools and display the Application panel to view the cookies for this application.

7. Click on the Log Out button. When the index.html page is displayed, display the Application panel of the developer tools again and view the cookies for this application.

Extra 15-2 Navigate between pages and save data in session storage

In this exercise, you'll enhance the Account Profile application to save its data in session storage, navigate to a new page, and allow you to navigate back to the original page. When you click on the Save button, a new page gets the data from session storage and displays it like this:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch15\profile\`

2. In the JavaScript file (profile.js), find the code in the handler for the click event of the Save button that validates the user entries. Then, find the if statement that checks the value of the isValid variable.

3. Add code to the if statement that saves the values in the email, phone, zip, and dob variables to session storage. Then, add code that uses the location object to navigate to the profile.html file.

4. In the profile.html file, find the embedded JavaScript in the head element of the page. Note that it contains the jQuery ready event handler and a handler for the click event of the Back button.

5. In the ready event handler, add code that retrieves the profile information from session storage and displays it in the span elements whose id attributes are "email", "phone", "zip", and "dob". Use the jQuery text() method of the span elements to do this.

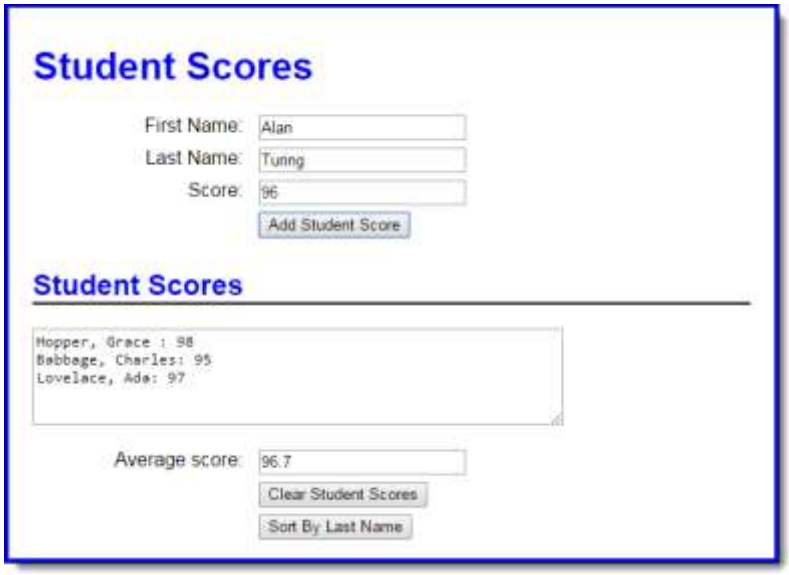
6. In the handler for the click event of the Back button, add code that uses the history object to go back to the previous page.

7. Run the application, enter valid data, and click Save. After you review the data that's displayed on the profile.html page, press F12 to open the developer tools and display the Application panel to view the data in session storage for this application.

8. Click on the Back button, make a change, and click Save. Then, display the Application panel of the developer tools again to see how the data in session storage has changed.

Extra 16-1 Develop the Student Scores application

In this exercise, you'll develop an application that tracks student's scores, tallies the average of the entered scores, and sorts the entered students by last name. The interface looks like this:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch16\scores\`

2. In the JavaScript file, note that the ready event handler contains the start of a displayScores() function and starts of the handlers for the click events of the Add Student Score, Clear Student Scores, and Sort By last Name buttons. Notice that the handler for the Add Student Score button ends by clearing the add form and setting the focus on its first field. Also, the handler for the Clear Student Scores button ends by clearing the display area and setting the focus on the first name field.

3. Code two arrays outside of these functions, one for score values and the other for strings that display the students' names and scores.

4. In the displayScores() function, add the code that calculates the average score of all the scores in the first array and displays it in the text box below the text area. Then, add the code that gets the students' names and scores in the second array and displays them in the text area.

5. In the click event handler for the Add Student Score button, use the push() method to save the score in the first array and to save the name and score string (as shown in the text area) in the second array. Then, call the displayScores() function to redisplay the updated data.

6. In the click event handler for the Clear Student Scores button, add code that clears both arrays.

7. In the click event handler for the Sort By Last Name button, add code that sorts the students by last name and then re-displays the score information.

Extra 16-2 Use an associative array with the Account Profile application

In this exercise, you'll adjust the Account Profile application to use an associative array. When you're done, the application will work the same as it did before. The interfaces of the two pages look the same as those shown in extra exercises 14-1 and 15-2:

1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch16\profile\`

2. In the JavaScript file, find the handler for the click event of the Save button. Then, find the code that gets the values entered by the user and stores them in four variables. Replace this with code that stores those values in an associative array. Make sure the indexes for the elements in the array are the same as the values of the id attributes of the text boxes.

3. Still in the handler for the click event of the Save button, find the code that saves the profile data to session storage. Replace this with code that loops through the associative array. For each element in the array, concatenate the index and value with an equal sign and add a pipe to the end, like this:

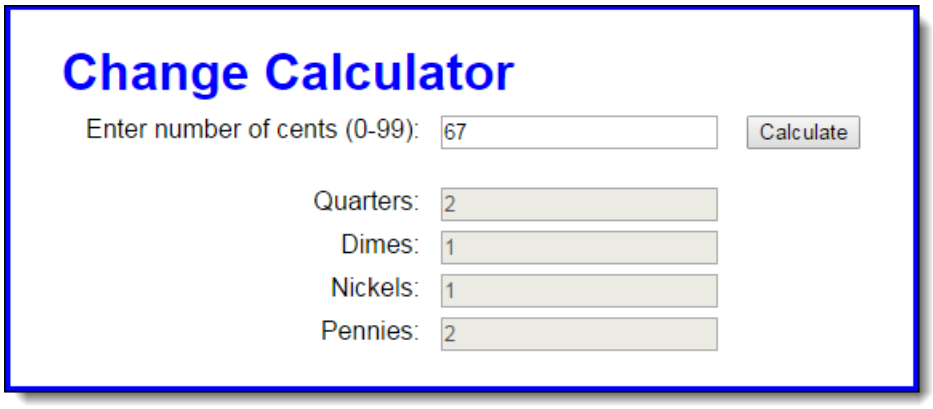
   ```
   email=grace@yahoo.com|
   ```

   Then concatenate this string to an item in session storage named "profile". Note: for this to work, you must create the profile item in session storage before the loop begins and set its initial value to an empty string.

4. In the embedded JavaScript in the profile.html file, find the code that gets the data from session storage and displays it in the span elements of the page. Replace this with code that splits the profile item from session storage on the pipe character. Then, loop through the resulting array and split each item on the equal sign character. Use the values in the resulting array to select the appropriate span element and set its text.

Extra 17-1 Use an object literal with the Change Calculator

In this exercise, you'll modify a Change Calculator application so it uses an object literal.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch17\change_literal\`

   Note that this application uses two JavaScript files: the main JavaScript file (calculate.js) and the start of a library file (library_coin.js).

2. In the calculate.js file, note that all of the code for the application is in the handler for the click event of the Calculate button.

3. In the library_coin.js file, note that just the strict mode declaration has been provided.

4. In the index.html file, add the script tag for the library file.

5. In the library file, code an object literal named coins that has a cents property and two methods:

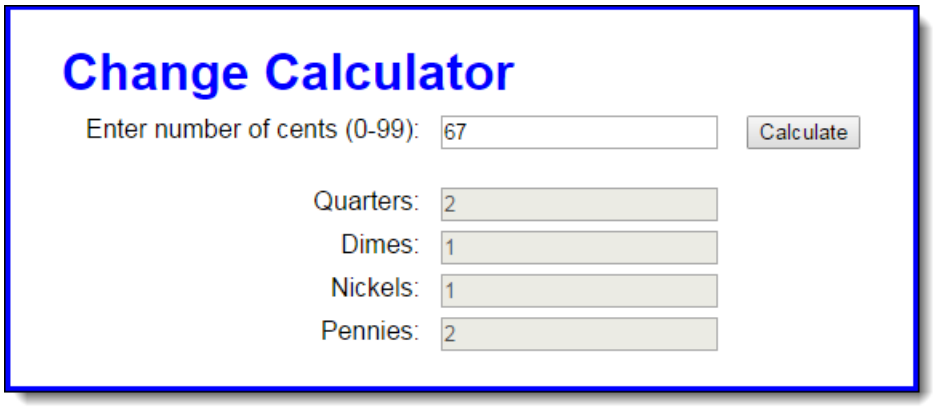
   The isValid() method should determine whether the cents property is valid.

   The getNumber() method should accept a divisor parameter (like 25 for quarters), calculate the number of coins of that type that are required, update the cents property with the remaining cents, and return the number of coins.

6. Change the code in the calculate.js file to use the object literal to get the cents entered by the user, validate the user's entry, and calculate the number of coins.

Extra 17-2 Use a constructor with the Change Calculator

In this exercise, you'll modify a Change Calculator application so it uses a constructor function.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch17\change_constructor\`

   Note that this application uses two JavaScript files: the main JavaScript file (calculate.js) and the start of a library file (library_coin.js).

2. In the calculate.js file, note that all of the code for the application is in the handler for the click event of the Calculate button.

3. In the library_coin.js file, note that just the strict mode declaration has been provided.

4. In the index.html file, add the script tag for the library file.

5. In the library file, code a constructor function named Coins() that accepts a parameter named cents, which is the number of cents entered by the user. Then, code a cents property and two methods:

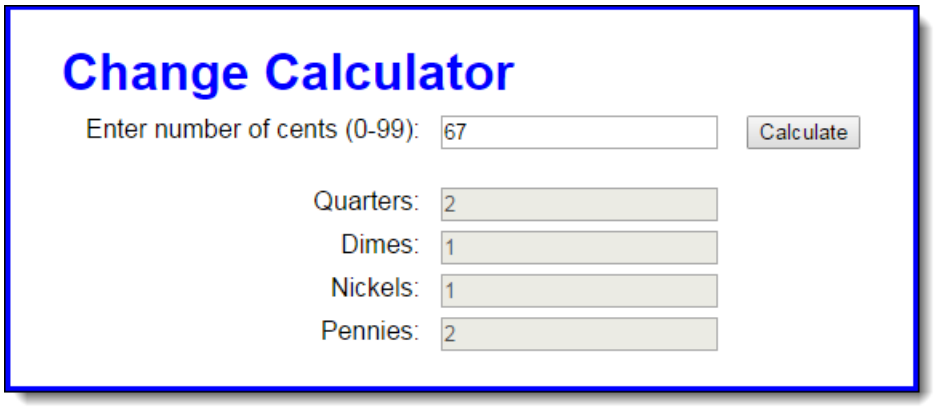
   The isValid() method should determine whether the cents property is valid.

   The getNumber() method should accept a divisor parameter (like 25 for quarters), calculate the number of coins of that type that are required, update the cents property with the remaining cents, and return the number of coins.

6. Change the code in the calculate.js file to create an instance of the Coins object type to validate the user's entry and calculate the number of coins.

Extra 17-3 Use a factory function with the Change Calculator

In this exercise, you'll modify a Change Calculator application so it uses a factory function.



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch17\change_factory\`

   Note that this application uses two JavaScript files: the main JavaScript file (calculate.js) and the start of a library file (library_coin.js).

2. In the calculate.js file, note that all of the code for the application is in the handler for the click event of the Calculate button.

3. In the library_coin.js file, note that just the strict mode declaration has been provided.

4. In the index.html file, add the script tag for the library file.

5. In the library file, code a factory function named calculateCoins() that accepts a parameter named cents, which is the number of cents entered by the user. Then, use the create() method of the Object object to create an object that has two methods:

   The isValid() method should determine whether the cents property is valid.

   The getNumber() method should accept a divisor parameter (like 25 for quarters), calculate the number of coins of that type that are required, update the cents property with the remaining cents, and return the number of coins.

6. Add a cents property to the object that the Object.create() method creates.

7. Change the code in the calculate.js file to use the object returned by the factory function to validate the user's entry and calculate the number of coins.

Extra 17-4 Convert the PIG app to objects

In this exercise, you'll finish an objects version of the PIG application of chapter 13. The interface looks like this:

To refresh your memory about the PIG game, you start by entering the names for the two players and clicking on the New Game button. Then, the messages below the names tell whose turn it is and announce the winner.

1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch17\pig\`

   Note that this application uses three JavaScript library files besides the main dice.js file.

2. Review the library_die.js file, and note that it contains a constructor function named Die with no properties and a single method that rolls a die.

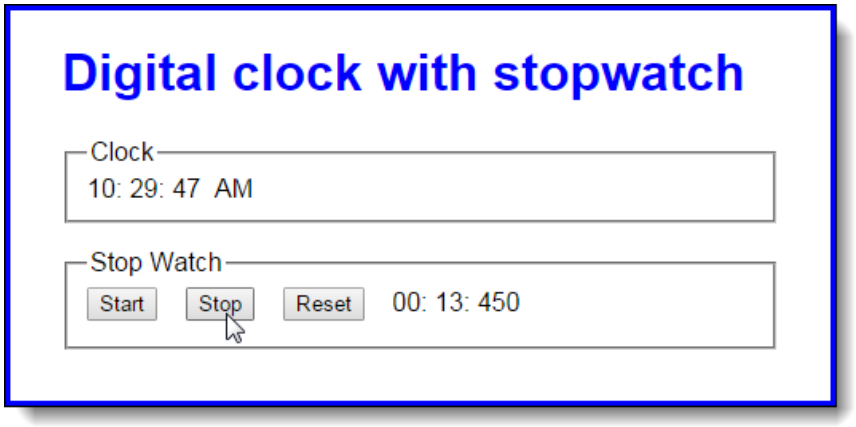
3. Review the library_pig.js file, and note that it contains a constructor named Pig with three properties. This constructor's prototype inherits the one method of the Die object type and provides three more methods: takeTurn(), hold(), and reset().

4. Review the library_game.js file, and note that it's an object literal with three properties and six methods. The start() and isValid() methods are already coded for you, and the remaining methods have comments indicating what they should do. Now, finish the code for those four methods and be sure to make them cascading methods whenever that's possible.

5. Review the dice.js file and note that the ready event handler and a handler for the click event of the New Game button are supplied. There are also starts for the click event handlers for the Roll and Hold buttons, which you need to finish. These handlers will use the methods of the game object in the library_game.js file.

Extra 18-1 Convert the Clock application to closures

In this exercise, you'll convert a Clock application like the one in extra exercise 7-2 to use closures. The enhanced application looks like this, but note that the application now uses buttons instead of links:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch18\clock_closures\`

2. Note that the folder contains two library files named library_clock.js and library_stopwatch.js. In the main JavaScript file, you'll see the variables, objects, and functions that provide all the functionality for the clock and stopwatch, but without using the libraries.

3. In the library_clock.js file, there's a start for a function named createClock(). Note that this function has parameters for the span tags that display the clock in the page.

4. In the library_stopwatch.js file, there's a start for a function named createStopwatch(). Note that this function has parameters for the span tags that display the stopwatch in the page.

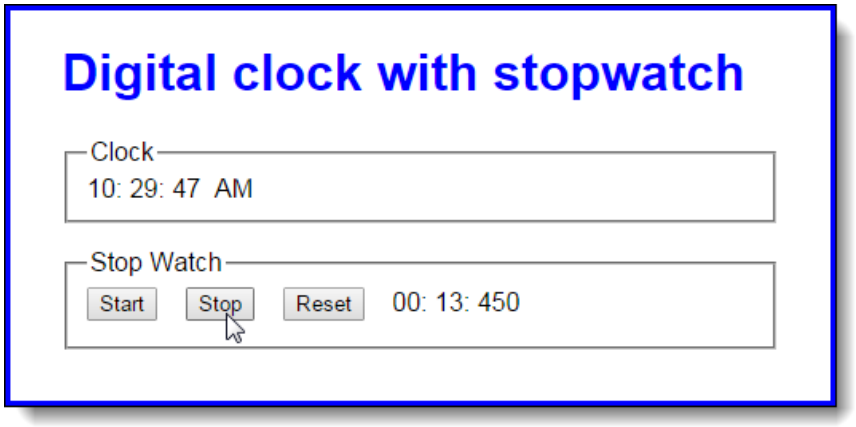
5. In the clock.js file, find the functions that run the clock and move them to the private state section of the library_clock.js file. Then, in the public methods section of the clock library file, code and return an object that contains a method named start(). This method should use the private state to start the clock. Adjust the code as needed to make this work.

6. In the clock.js file, find the variables, objects, and functions that run the stopwatch and move them to the private state section of the library_stopwatch.js file. Then, in the public methods section, code and return an object that contains methods named start(), stop(), and reset(). These methods should use the private state to start, stop, and reset the stopwatch. Adjust the code as needed to make this work.

7. Still in the clock.js file, rewrite the remaining code so the ready event handler calls the functions in the library files, passes them the span tags they need, and uses the returned objects to start the clock and attach the stopwatch event handlers.

Extra 18-2 Use namespaces and the module pattern with the Clock

In this exercise, you'll convert the Clock application from exercise 18-1 to use namespaces and the module pattern. The enhanced application looks the same as in the last exercise:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch18\clock_namespaces\`

   Notice that this application uses two additional library files: a namespace library and an empty utility library. In the HTML file, note that the script tags for these libraries come before the script tags for the clock and stopwatch libraries.

2. In the namespace library, review the code that creates the myapp namespace and the following nested namespaces:

   ```
   time
   utility
   time.clock
   time.stopwatch
   ```

3. Review the clock and stopwatch libraries, and note that they both contain the padSingleDigit() function. Move this function to the library_utility.js file, add it to the utility namespace, and adjust the clock and stopwatch libraries accordingly. Use an alias in those libraries to keep the code short.

4. Change the clock library so it uses the module pattern and adds the object it creates to the clock namespace.

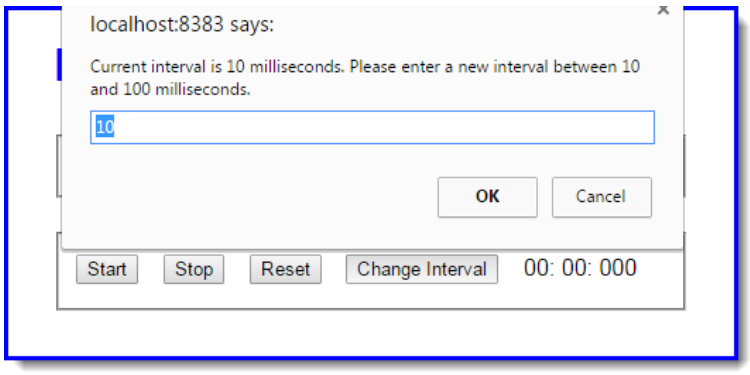
   NOTE: In this step and in step 5, you'll need to adjust the objects so the span elements are passed to the start() method and then stored in private state so the other functions can use them.

5. Change the stopwatch library so it uses the module pattern and adds the object it creates to the stopwatch namespace.

6. Change the code in the clock.js file to use the new namespaces and modules. Since you no longer create the clock and stopwatch objects, remove that code and pass the span elements to the start() methods instead. Use aliases for the namespaces, and make sure you don't add anything to the global namespace.

Extra 18-3 Enhance the Clock app's stopwatch

In this exercise, you'll enhance the stopwatch module from extra exercise 18-2 to allow the user to change the interval between ticks of the stopwatch. The enhanced application looks like this when the Change Interval button is clicked:



1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch18\clock_augment\`

   Notice that this application includes a new library file named library_stopwatch_augment.js. In the HTML file, note that the script tag for this new library comes after the script tag for the stopwatch library. Also note that there's a new Change Interval button.

2. In the library_stopwatch.js file, add a variable named increment to private state, and set its value to 10. Then, in the private tickStopWatch() function, change the first line of code so it uses the increment variable rather than the numeric literal of 10. Finally, in the public start() method, change the call to the setInterval() function so it uses the increment variable rather than the numeric literal of 10.

3. Create an object literal named properties that contains an accessor property named interval. This property should get the value of the increment variable, but only set the increment variable if the value it receives is a number between 10 and 100. Otherwise, it should reset the increment variable to its default value of 10.

4. Adjust the code so the stopwatch library uses the Object.create() method to create the object it returns. Pass the create() method a prototype object that contains the start(), stop(), and reset() methods, and the properties object.

5. In the library_stopwatch_augment.js file, add code that augments the stopwatch module with a new method named changeInterval(). This method should use the prompt() method to display the message shown above. Then, it should convert the value received from the user to an integer, set the interval property to this new value, and reset the stopwatch.

6. In the clock.js file, code a handler for the click event of the Change Interval button. Within the event handler, call the changeInterval() method of the stopwatch object.

7. Run the application and test the Change Interval button. See how it works when you enter values that aren't numbers, or are below 10 or above 100.

Extra 18-4 Create a rollover plugin

In this exercise, you'll create a plugin that displays a different image when the mouse is rolled over an image in a list of images.

1. Open the HTML and JavaScript files in this folder:

   `exercises_extra\ch18\image_rollover\`

   Now, run this application to refresh your memory about how it works.

2. Review the code in the jquery.rollover.js file to see that it contains code for the standard plugin structure with the method name set to changeImage().

3. Copy the code in the ready event handler of the rollover.js file into the each() method of the plugin. Then, modify the selector for the each() method in this copied code so its function is executed for each img element that's a descendant of the element that the changeImage() method is executed on.

4. Return to the rollover.js file, and replace the code in the ready event handler with a statement that calls the changeImage() method for the "image_rollovers" list.

5. Add a script element for the jquery.rollover.js file to the HTML file before the script element for the rollover.js file. The application should now run just like it did before.