

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	3
Extra 2-1 Convert Fahrenheit to Celsius	4
Extra 3-1 Enhance the Fahrenheit to Celsius application	5
Extra 3-2 Convert number grades to letter grades	6
Extra 3-3 Create a Sum of Numbers application	7
Extra 3-4 Use a Sales array	8
Extra 4-1 Develop the Sales Tax Calculator	9
Extra 4-2 Develop the Change Calculator	10
Extra 4-3 Develop the Income Tax Calculator	11
Extra 6-1 Develop the Temperature Converter	12
Extra 6-2 Use a Test Score array	14
Extra 6-3 Modify the FAQs application	16
Extra 7-1 Develop the Clock application	17
Extra 7-2 Add a stopwatch to the Clock application	18
Extra 8-1 Develop an Expand/Collapse application	19
Extra 8-2 Develop an Image Gallery application	20
Extra 9-1 Modify an Image Swap application	21
Extra 9-2 Modify a carousel to use animation	22
Extra 10-1 Use JavaScript to validate a form	23
Extra 11-1 Modify the Carousel application	24
Extra 11-2 Replace a Tabs widget with an Accordion widget	25
Extra 11-3 Use widgets in a form	26
Extra 12-1 Develop the Change Calculator	27
Extra 12-2 Develop the Calendar application	28
Extra 12-3 Develop a password generator	29
Extra 13-1 Use a switch statement validate a date	30
Extra 13-2 Adjust a regular expression pattern	31
Extra 14-1 Navigate between pages and use a cookie	32
Extra 14-2 Navigate between pages and use session storage	33
Extra 15-1 Develop the Student Scores application	34
Extra 15-2 Use an array of arrays with the Account Profile app	35
Extra 16-1 Use an object literal with the Change Calculator	36
Extra 16-2 Use a constructor with the Change Calculator	37
Extra 16-3 Convert the PIG app to objects	38
Extra 17-1 Convert the Clock application to closures	39
Extra 17-2 Use IIFEs with the Clock	40

2 Extra exercises for *Murach's JavaScript and jQuery (4th Edition)*

Extra 17-3	Review a Clock application that uses ES modules	41
Extra 18-1	Use data from the JSON Placeholder API	42
Extra 18-2	Use data from NASA's Mars Rover API	43
Extra 19-1	Create and run a script with parameters	44
Extra 19-2	Run a Clock application that uses ES modules	45

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.
- Make sure every application is coded in strict mode.
- 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()` method of the window object and the `write()` method of the document object. The prompt dialog box should look like this:



After you write the results, the page should look like this:



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

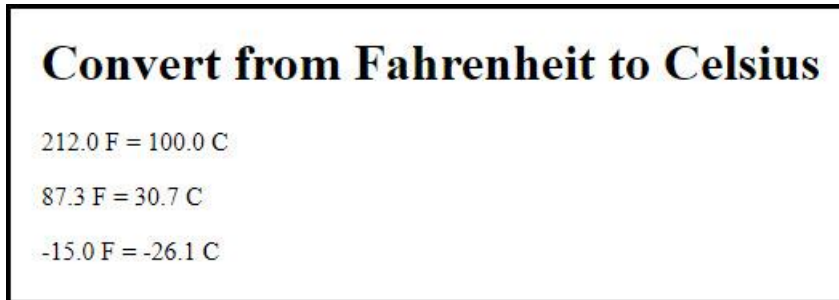
1. Open this file:
`exercises_extra\ch02\convert_temps.html`
2. Review the script element at the end of the body section and note that it's empty. You'll write the code for this application within this element.
3. Develop this application. Allow the user to enter decimal numbers, and display the Fahrenheit value entered by the user and the calculated Celsius value rounded to 1 digit.

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:



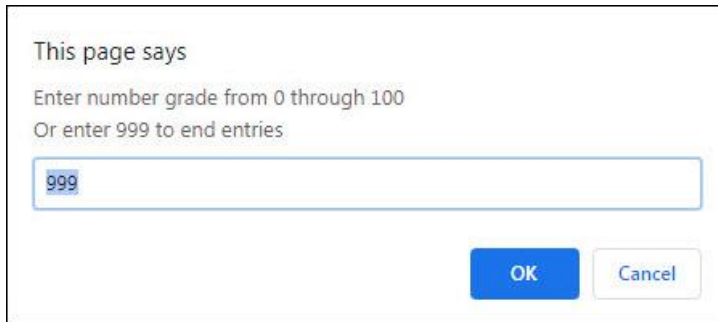
After you enter several temperatures to convert, the page should look like this:



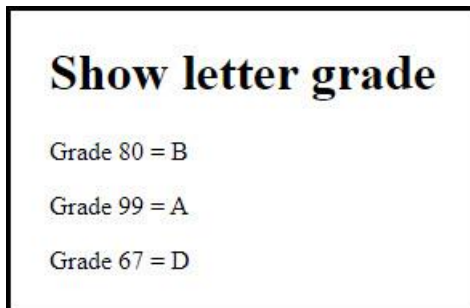
1. If you didn't already do extra exercise 2-1, do it now. Then, copy the `convert_temps.html` file into this folder:
`exercises_extra\ch03\`
2. Add data validation to the application so it won't do the conversion until the user enters a Fahrenheit temperature between -100 and 212. If the entry is invalid, 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 in the page letter grades for the numbers the user entered:



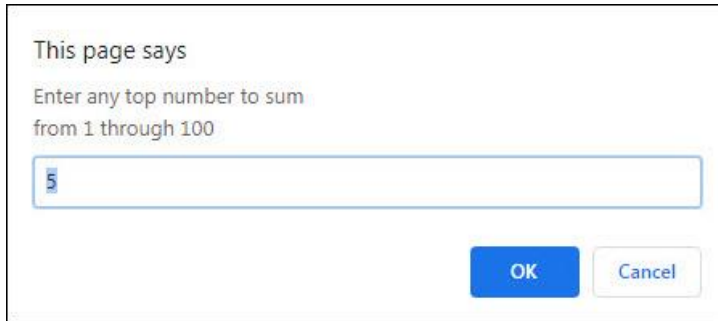
To derive the letter grade, you should use this table:

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 on the page.
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

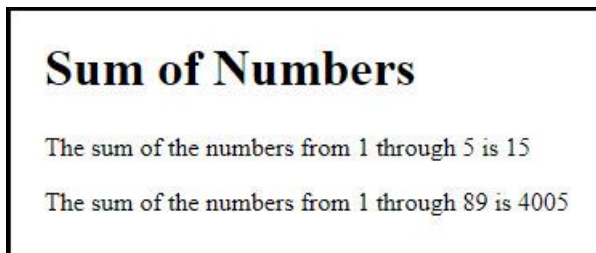
This application will give you a chance to use a for loop. It asks the user to enter a number from 1 through 100 with this prompt dialog box:



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



Finally, it adds all the numbers from each of the user's entries and displays the sum of the numbers in the page like this:



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 4 is $1 + 2 + 3 + 4$.
4. Add a do-while loop around all of the code that uses the second 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 the page, which should look like this:

Sales Data

Sales by Quarter

Q1: \$9965

Q2: \$7403

Q3: \$9478

Q4: \$13061

Sales by Region

Region 1: \$5355

Region 2: \$9585

Region 3: \$9398

Region 4: \$6195

Region 5: \$9374

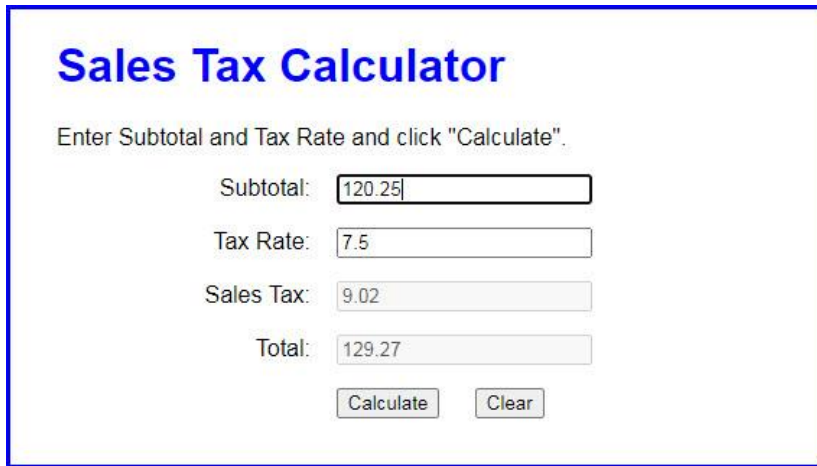
Total Sales

\$39907

1. Open the application in this folder:
`exercises_extra\ch03\sales_array\`
2. In the HTML file, note the link element that refers to the CSS file, and the script element refers to the JavaScript file.
3. 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.
4. Write the code for summing the quarterly sales for each the five regions and displaying them on the page with the `document.write()` method. To do that, use an `<h2>` tag for each header and a `
` tag for a line break at the end of each line of sales data.
5. Write the code for getting and displaying the regional sales data.
6. Write the code for getting and displaying the total sales data.

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.

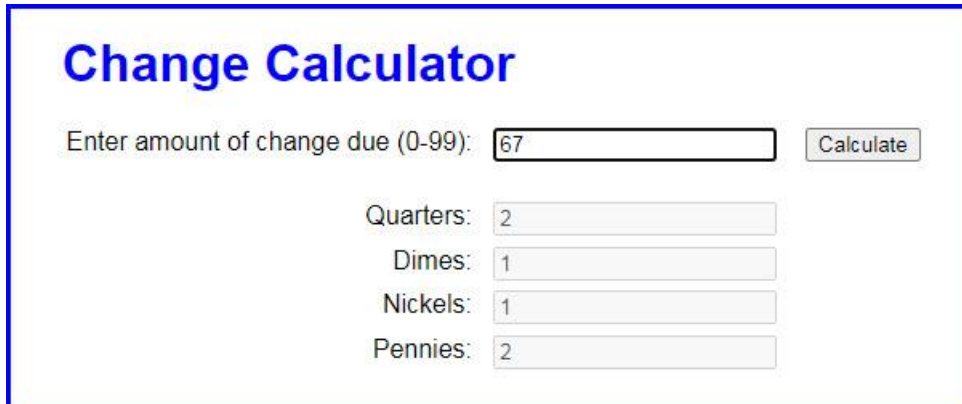


The screenshot shows a web application titled "Sales Tax Calculator" in blue text. Below the title is a prompt: "Enter Subtotal and Tax Rate and click 'Calculate'." There are four text input fields arranged vertically: "Subtotal:" with the value "120.25", "Tax Rate:" with the value "7.5", "Sales Tax:" with the value "9.02", and "Total:" with the value "129.27". At the bottom of the form are two buttons: "Calculate" and "Clear".

1. Open the application 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 CSS selector.
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 a `DOMContentLoaded` 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 event handler for the click event of the Clear button. This event handler should clear all text boxes and move the cursor to the Subtotal field.
8. Add event handlers for the click events of the Subtotal and Tax Rate text boxes. Each handler should clear the data from the text box. Update the event handler for the Clear button to call these functions as well (to reduce code duplication).

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.



The screenshot shows a web application titled "Change Calculator" in a blue serif font. Below the title is a text input field with the placeholder text "Enter amount of change due (0-99):" and the value "67". To the right of the input field is a button labeled "Calculate". Below the input field are four rows of labels and text input fields: "Quarters:" with value "2", "Dimes:" with value "1", "Nickels:" with value "1", and "Pennies:" with value "2". The entire interface is enclosed in a blue rectangular border.

1. Open the application 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 so, call a function named `makeChange()` and pass it the user's entry. Otherwise, display an alert dialog box for the error.
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 a `DOMContentLoaded` 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 2020 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	\$9,875	\$0 plus 10%	\$0
\$9,875	\$40,125	\$987.50 plus 12%	\$9,875
\$40,125	\$85,525	\$4,617.50 plus 22%	\$40,125
\$85,525	\$163,300	\$14,605.50 plus 24%	\$85,525
\$163,300	\$207,350	\$33,271.50 plus 32%	\$163,300
\$207,350	\$518,400	\$47,367.50 plus 35%	\$207,350
\$518,400		\$156,235.00 plus 37%	\$518,400

1. Open the application in this folder:

`exercises_extra\ch04\income_tax\`

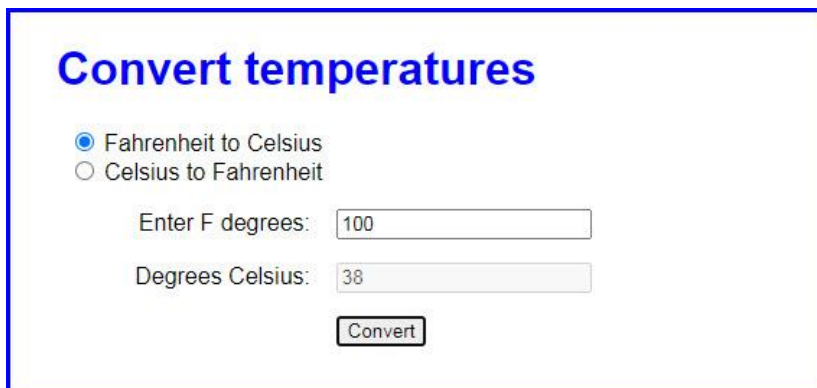
Note that the JavaScript file has some starting JavaScript code for this application, including the `$()` function and a `DOMContentLoaded` event handler that attaches a function named `processEntry()` to the click event of the Calculate button and moves the focus to the first text box.

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 valid, 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. 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 9875 and 40125, which should display taxable amounts of 987.50 and 4,617.50.
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

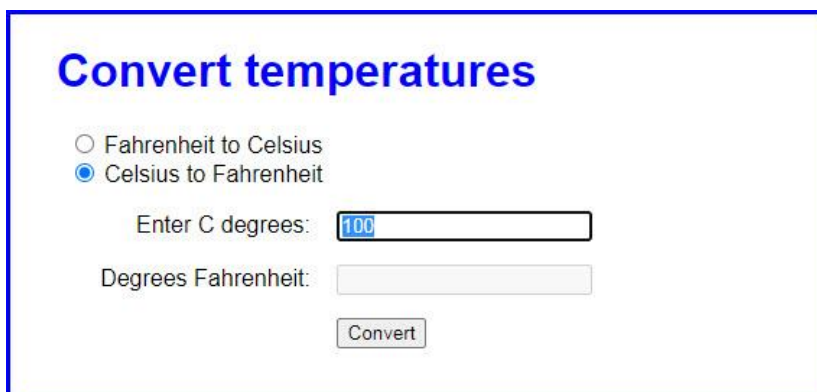
In this exercise, you'll use radio buttons to determine whether the conversion is from Fahrenheit to Celsius or vice versa. You'll also modify the DOM so the labels change when a radio button is clicked, and the page displays an error message when the user enters invalid data.

When the user does a conversion from Fahrenheit to Celsius, it looks like this:



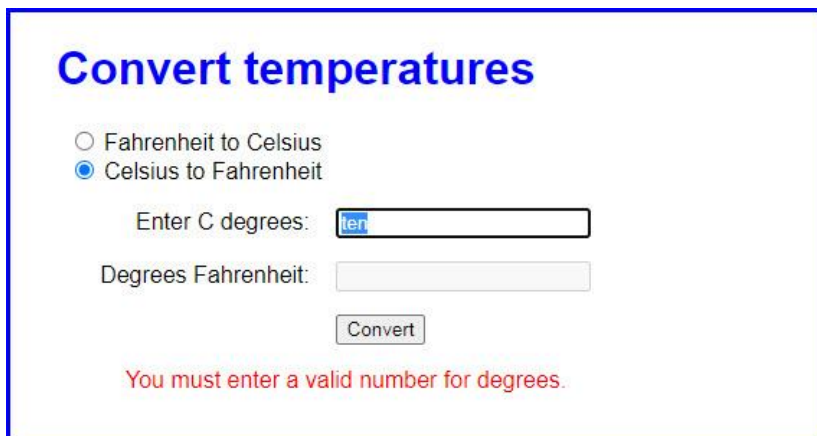
The screenshot shows a web form titled "Convert temperatures" in blue. It has two radio buttons: "Fahrenheit to Celsius" (selected) and "Celsius to Fahrenheit". Below the radio buttons, there are two text input fields. The first is labeled "Enter F degrees:" and contains the value "100". The second is labeled "Degrees Celsius:" and contains the value "38". At the bottom right of the form is a "Convert" button.

When the user clicks on the second radio button to do a conversion from Celsius to Fahrenheit, it looks like this:



The screenshot shows the same web form, but the "Celsius to Fahrenheit" radio button is now selected. The "Enter C degrees:" input field contains the value "100". The "Degrees Fahrenheit:" input field is empty. The "Convert" button remains at the bottom right.

And if the user enters invalid data, it looks like this:



The screenshot shows the web form with the "Celsius to Fahrenheit" radio button selected. The "Enter C degrees:" input field contains the text "ten". The "Degrees Fahrenheit:" input field is empty. The "Convert" button is present. Below the input fields, a red error message is displayed: "You must enter a valid number for degrees."

1. Open the application in this folder:

`exercises_extra\ch06\convert_temps\`

2. Note that the JavaScript file has some starting JavaScript code, including the `$()` function, three helper functions, three event handler functions, and a `DOMContentLoaded` event handler that attaches the three event handlers.
3. Review how the `toCelsius()` and `toFarhenheit()` event handler functions call the `toggleDisplay()` helper function and pass it strings to display. Also note that the `toggleDisplay()` helper function and the `convertTemp()` event handler function are incomplete.
4. Code the `toggleDisplay()` function so it changes the text in the labels for the text boxes to the values in the parameters that it receives. It should also clear the disabled text box where the computed temperature is displayed.
5. Code the `convertTemp()` function without any data validation. It should use the helper functions to calculate the temperature based on which radio button is checked. The value returned by the helper functions should be rounded to zero decimal places.
6. Add data validation to the `convertTemp()` function. If the entry is not a valid number, a validation message should be displayed in the element with the id of "message".
7. Add any finishing touches to the application whenever that's appropriate. This can be things like moving the focus to the first text box and selecting the text, clearing a previous error message, or clearing a previously computed temperature value.

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. The interface after clicking the Display Results and Display Scores buttons should look like this:

The screenshot shows a web application titled "Use a Test Score array". It features two input fields: "Name:" with the value "Mary" and "Score:" with the value "94". Below these are three buttons: "Add to Array", "Display Results", and "Display Scores". Under the "Display Results" button, the text "Results" is shown, followed by "Average score = 90" and "High score = Mike with a score of 99". Under the "Display Scores" button, the text "Scores" is shown, followed by a table of scores:

Ben	88
Joel	98
Judy	77
Anne	88
Mike	99

If the user enters invalid data as described below, it should look like this:

The screenshot shows the same web application as above, but with validation errors. The "Name:" input field is empty, and a red error message "Please enter a name" is displayed to its right. The "Score:" input field is also empty, and a red error message "Score must be between 0 and 100." is displayed to its right. The buttons and other UI elements remain the same.

1. Open the application 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.

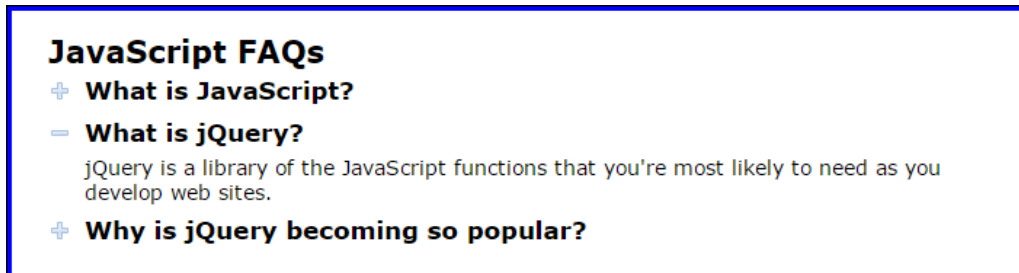
2. At the start of the JavaScript file, you'll see the declarations for two arrays: one for names and one for scores, and each array contains four elements. You'll also see the code for the `$()` function as well as a `DOMContentLoaded` event handler that attaches three functions named `addScore()`, `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, add nodes to the DOM

with the heading as an h2 element and the average and highest scores as <p> elements. If those nodes already exist, you'll need to replace them.

4. Write the displayScores() function. It should get the names and scores from the arrays and display them in the div element with "scores" as its id, as shown above. To display the results, add nodes to the DOM with the name and score as label elements, and a break element (
) to go to a new line. Before adding these nodes, you can clear any previous nodes by setting the text content of the div to an empty string.
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, display messages in the span elements after the input elements, as shown above.
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

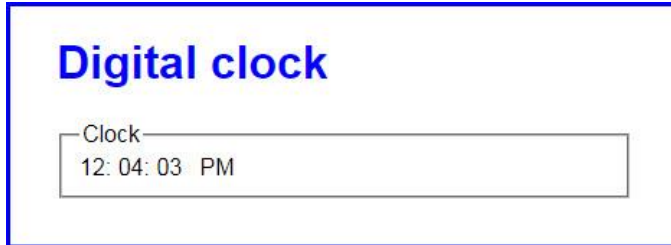
This exercise has you make a minor modification to the FAQs application. When you're done, this application should work the same as before, except that only one answer can be displayed at a time. In other words, when the user clicks on a heading to display the answer, the other answers must be hidden.



1. Open the application 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-of` 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:

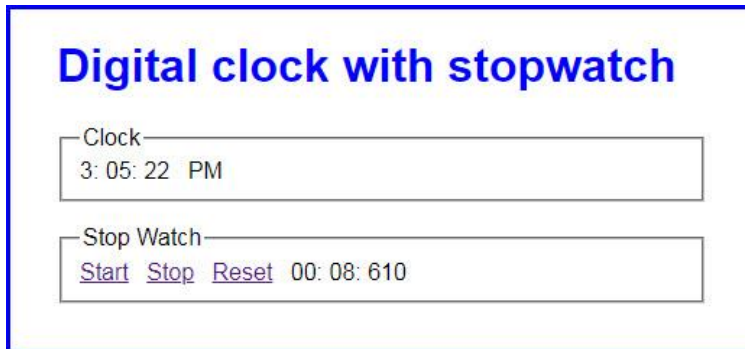


To convert the computer's time from a 24-hour clock to a 12-hour clock, first check to see if the hours value is greater than 12. If so, subtract 12 from the hours value and set the AM/PM value to "PM". Also, be aware that the hours value for midnight is 0.

1. Open the application in this folder:
`exercises_extra\ch07\clock\`
2. In the JavaScript file, note that four functions are supplied. The `$()` function that you can use to select elements. The `padSingleDigit()` function, that adds a leading zero to single digits using a string method that you'll learn about in chapter 12. The start of a `displayCurrentTime()` function. And the start of a `DOMContentLoaded` 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 `DOMContentLoaded` 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 application in this folder:
`exercises_extra\ch07\clock_stopwatch\`
2. In the JavaScript file, note the `$()`, `displayCurrentTime()`, `padSingleDigit()`, and `DOMContentLoaded` event handler functions from the Clock application. In addition, note the global variables and starting code for the `tickStopwatch()`, `startStopwatch()`, `stopStopwatch()`, and `resetStopwatch()` functions.
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.
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 `DOMContentLoaded` 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.

Murach's HTML5 and CSS3 (4th Edition)

Book description

Murach's HTML5 and CSS3 provides all HTML and CSS a professional needs, and it adds coverage of Flexible Box and Grid Layout, two new CSS3 ways to implement page layouts. So whether you're a web designer, a JavaScript programmer, a server-side programmer, or a rookie, this book delivers all the HTML and CSS skills that you need on the job. It begins with an 8-chapter hands-on course that teaches you HTML and CSS from scratch, including the latest HTML5 and CSS3 features.

After that, you'll learn how to use Flexible Box and Grid Layout, and how to work with forms and data validation. Then you'll learn how to enhance a site with video clips, CSS3 transitions, transforms, and animations. You'll learn how to design and deploy a website, as well as other professional skills like how to use JavaScript and jQuery and how to use development tools like Bootstrap, SASS, and Emmet. And after you've learned all the skills that you need, this book becomes the best on-the-job reference you've ever used.

[Show less](#)

About the authors

Anne Boehm has over 30 years of experience as an enterprise programmer. She got started with Visual Basic in the days of VB5 and has been programming on .NET since its inception. She added C# to her programming repertoire in the mid-2000s, and she's authored or co-authored our books on Visual Basic, C#, ADO.NET, ASP.NET, and HTML5/CSS3.

[Show more](#)

Who this book is for

Due to our unique presentation methods and this book's modular organization, this is

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. Note that each topic consists of two div elements followed by an <a> element. Also, note 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 ready() event method, code an event handler for the click() event method of the <a> elements. This event handler should start by preventing the default action of the link and storing the clicked link in a constant. Then, it should use 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

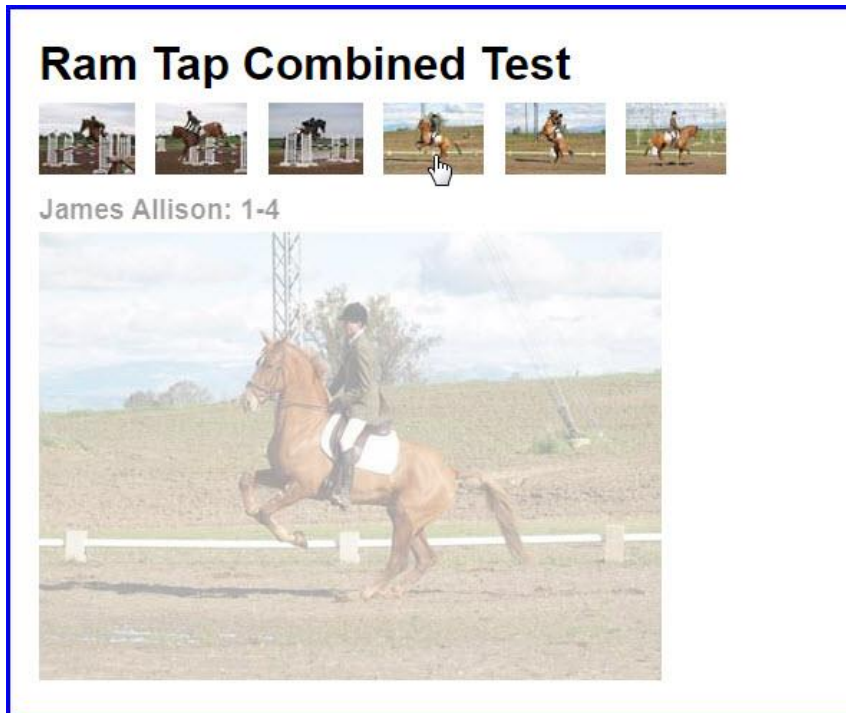
In this exercise, you'll develop an Image Gallery application that displays different images when the user clicks on the links at the top of the page. This works like the Image Swap application of figure 8-14.



1. Open the application in this folder:
`exercises_extra\ch08\image_gallery\`
This folder contains 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 method 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

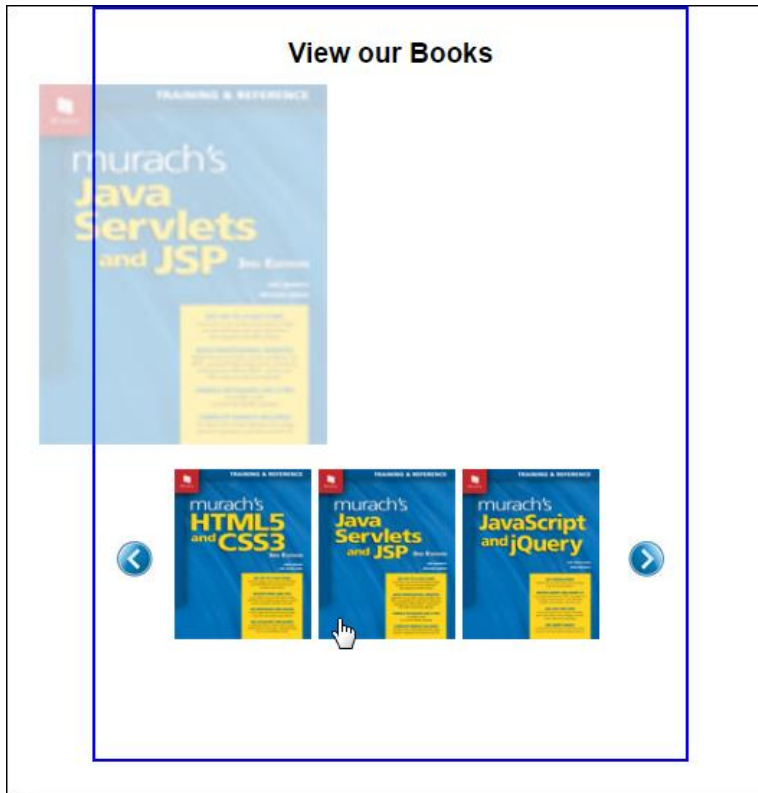
In this exercise, you'll modify another Image Swap application so it uses effects to display and hide the images.



1. Open the application in this folder:
`exercises_extra\ch09\image_swap\`
Review the script tags in the HTML file to see that this application doesn't use the slim version of the jQuery file. That way, the jQuery effects are available to this application.
2. 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.
3. Add statements to the JavaScript file that fade the caption and image out over a duration of one second.
4. 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.
5. 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 use 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 application 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.

Reservation Request

General Information

Arrival date:

Nights: Must be numeric.

Adults:

Children:

Preferences

Room type: ☒ Standard ☐ Business ☐ Suite

Bed type: ☒ King ☐ Double Double

☐ Smoking

Contact Information

Name:

Email: Must be a valid email address.

Phone: This field is required.

1. Open the application 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, note that the ready event handler contains the declaration for a constant 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. If any of the entries are invalid, the code should cancel the submission of the form. The validation is as follows:

A value must be entered into each text box.

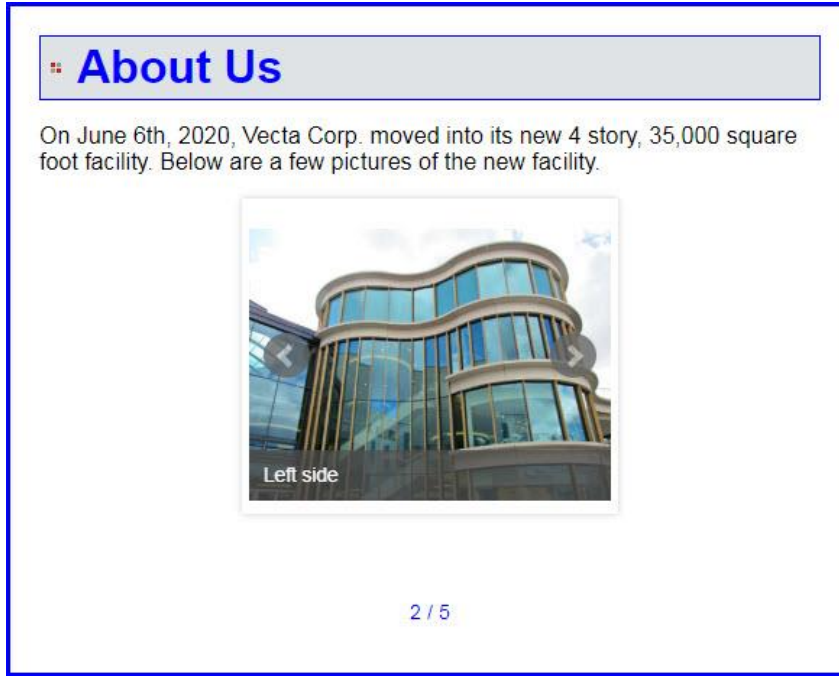
The number of nights must be numeric.

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

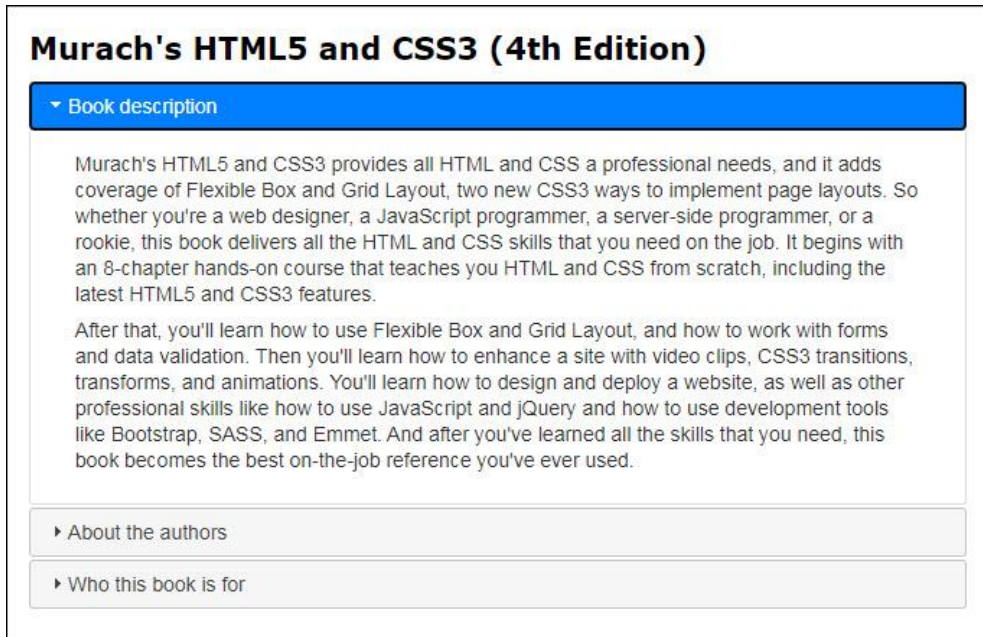
In this exercise, you'll modify the way the bxSlider plugin is used with the Carousel application. To do that, you may have to refer to the options for this plugin that are described at <http://bxslider.com>.



1. Open the application in this folder:
`exercises_extra\ch11\carousel\`
2. Review the code in the HTML file.
3. 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.
4. Add an option to the code that calls the bxSlider plugin so the carousel moves one slide at a time.
5. Modify the code for the bxSlider plugin so that only one image is displayed at a time.
6. Add an option to the code that sets the time between the automatic transitions to 3 seconds.
7. 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 application in this folder:
`exercises_extra\ch11\accordion\`
Now, run this application to refresh your memory about how it works.
2. Modify the HTML 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. In the HTML file, 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.

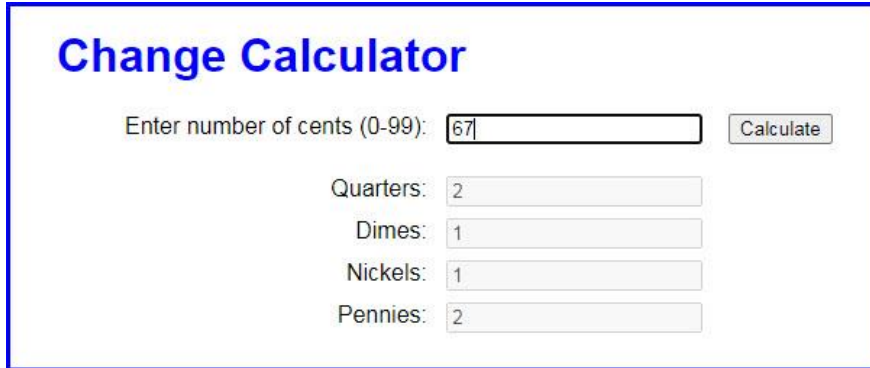
Extra 11-3 Use widgets in a form

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

1. Open the application 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. In the JavaScript file, 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 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 screenshot shows a web application titled "Change Calculator" in blue text. Below the title, there is a label "Enter number of cents (0-99):" followed by a text input field containing the number "67". To the right of the input field is a button labeled "Calculate". Below the input field, there are four rows of labels and text input fields: "Quarters:" with a field containing "2", "Dimes:" with a field containing "1", "Nickels:" with a field containing "1", and "Pennies:" with a field containing "2".

1. Open the application in this folder:
`exercises_extra\ch12\change_calculator\`
2. In the JavaScript file, note the jQuery `ready()` event method, and 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 a validation message.
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 12-2 Develop the Calendar application

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

November 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

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 application in this folder:

`exercises_extra\ch12\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 and 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 `currentMonth` parameter to calculate and return the last day of the current month. See figure 12-11 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.

Extra 12-3 Develop a password generator

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



Generate a strong password

Number of characters:

Password:

1. Open the application in this folder:
`exercises_extra\ch12\password\`
2. In the JavaScript file, note the getRandomNumber() function. Also, note that the ready event handler contains 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 constant named chars that contains some characters. 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 constant and concatenate it to the password variable.
5. When the loop is finished, display the password in the password textbox.

Extra 13-1 Use a switch statement validate 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 11/31/2020 or 2/30/2021.

My Account Profile

E-Mail: Please enter a valid email.

Mobile phone: Please enter a phone number in NNN-NNN-NNNN format.

ZIP Code: Please enter a valid zip code.

Date of Birth: Please enter a valid date in MM/DD/YYYY format.

1. Open the application in this folder:
`exercises_extra\ch13\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 the `isDate()` function and the ready event handler with 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 constant isn't greater than 31. Replace it with an else statement that contains a switch statement that evaluates the value of the month constant.
6. Code a case label for the value 2 that returns false if the value of the day constant is greater than 28 and returns true otherwise. 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 and true otherwise.
8. Code a default case that returns false if the value of day is greater than 31 and true otherwise.
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 13-2 Adjust a regular expression pattern

In this exercise, you'll adjust a regular expression pattern that validates phone numbers so it accepts more options.



Validate phone number

Phone Number:

Valid phone number

1. Open the application in this folder:
`exercises_extra\ch13\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 constant so the phone number can contain an optional “1-” prefix. The best way to do this is to copy the pattern constant 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 14-1 Navigate between pages and use 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:



The initial interface is a simple login form. It features a blue heading "My Website". Below the heading, there is a label "User name:" followed by a text input field containing the name "Grace". Below the input field is a "Log In" button.

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



The interface after login shows a blue heading "My Website" and a bold "Welcome, Grace!" message. Below the welcome message is a "Log Out" button.

NOTE: When you run this application from the file system, it doesn't work in the Chrome browser, but it should work in the Firefox browser.

1. Open the application in this folder:
`exercises_extra\ch14\login\`
Then, run the application to see the user interface shown above.
2. Review the code in the `login.js` file. Note that it contains starts for four functions: `getCookieByName()`, `setCookie()`, `deleteCookie()`, and `goToPage()`. The `getCookieByName()` function returns an empty string, while the others contain no code.
3. Review the `index.html` file and notice that its embedded JavaScript code uses the functions in the `login.js` file.
4. Review the `login.html` file and notice that its embedded JavaScript code uses the functions in the `login.js` file.
5. In the `login.js` file, update each of the functions so they perform the tasks described by their names. Use the examples in figures 14-5 through 14-7 as a guide for the first three, and figure 14-1 for the last one.
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 Storage panel (in FireFox) to view the cookies for this application. This should show the cookie for this application.
7. Click on the Log Out button. When the `index.html` page is displayed, use the Storage panel of the developer tools to view the cookies for this application. This shouldn't show any cookies.

Extra 14-2 Navigate between pages and use session storage

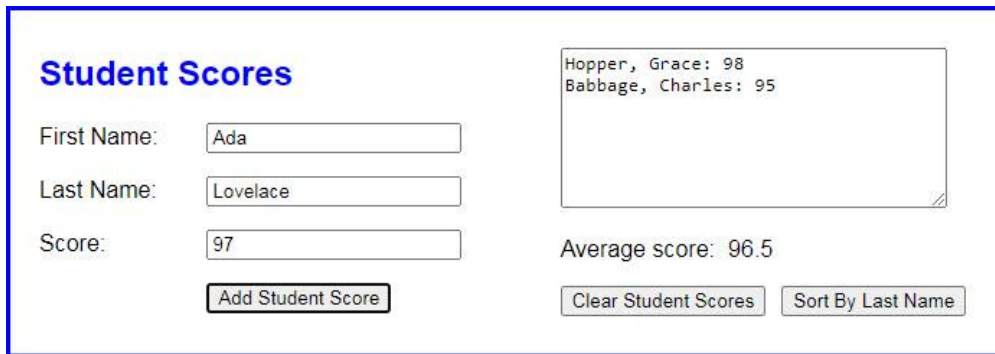
In this exercise, you'll enhance the Account Profile application to save its data in session storage, navigate to a new page, and allow you to navigate back to the original page. When you click on the Save button, a new page gets the data from session storage and displays it like this:

My Account Profile
E-Mail: grace@yahoo.com
Mobile phone: 555-123-4567
ZIP Code: 12345
Date of Birth: 12/09/1906

1. Open the application in this folder:
`exercises_extra\ch14\profile\`
2. In the `save_profile.js` file, 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 constants to session storage. Then, add code that uses the location object to navigate to the `profile.html` file.
4. Review the code in the `display_profile.js` file. 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 15-1 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 screenshot shows a web application titled "Student Scores". On the left, there is a form with three input fields: "First Name:" with the value "Ada", "Last Name:" with the value "Lovelace", and "Score:" with the value "97". Below these fields is a button labeled "Add Student Score". To the right of the form, there is a text area displaying the following text: "Hopper, Grace: 98" and "Babbage, Charles: 95". Below the text area, there is a label "Average score: 96.5". At the bottom right, there are two buttons: "Clear Student Scores" and "Sort By Last Name".

1. Open the application in this folder:
`exercises_extra\ch15\scores\`
2. In the JavaScript file, note the start of a `displayScores()` function. In the ready event handler, note the starts of the handlers for the click events of the Add Student Score, Clear Student Scores, and Sort By last Name buttons. Note that the handler for the Add Student Score button validates the data. For valid data, it clears the add form and sets 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 label with the id of "average_score". 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 15-2 Use an array of arrays with the Account Profile app

In this exercise, you'll adjust the Account Profile application to use an array of arrays. When you're done, the application will work the same as it did before.

My Account Profile

E-Mail: Please enter a valid email.

Mobile phone: Please enter a phone number in NNN-NNN-NNNN format.

ZIP Code: Please enter a valid zip code.

Date of Birth: Please enter a valid date in MM/DD/YYYY format.

My Account Profile

E-Mail: grace@yahoo.com

Mobile phone: 555-123-4567

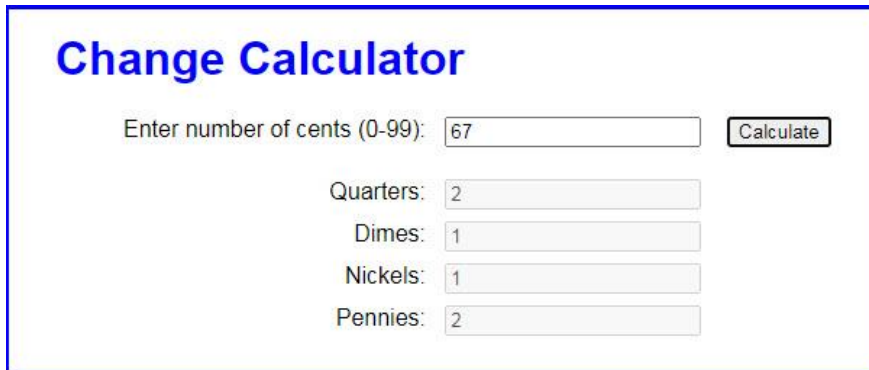
ZIP Code: 12345

Date of Birth: 12/09/1906

1. Open the application in this folder:
`exercises_extra\ch15\profile\`
2. In the `save_profile.js` file, find the handler for the click event of the Save button. Then, find the code that saves the profile data to session storage. Create an empty array. Then, for each profile value, add to that array an array that contains the value and the id attribute of the text box the value came from.
3. Change the session storage code to store a single value named `profile`. Then, use the JSON type to convert the array to a string and store that string in session storage.
4. In the `display_profile.js` file, find the code that gets the data from session storage and displays it in the span elements of the page. Update this with code that that uses the JSON type to convert the string from session storage to an array of arrays. Then, loop through the resulting array and set the value of the appropriate text box.

Extra 16-1 Use an object literal with the Change Calculator

In this exercise, you'll modify a Change Calculator application so it uses an object literal. When you're done, the application will work the same as it did before.



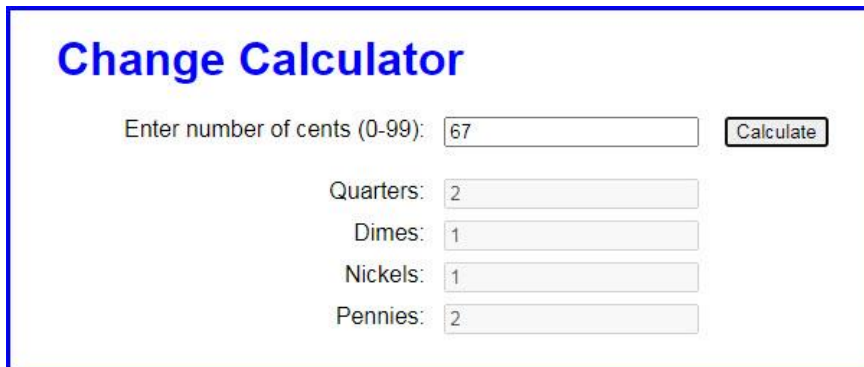
The screenshot shows a web form titled "Change Calculator" in blue text. Below the title, there is a label "Enter number of cents (0-99):" followed by a text input field containing the number "67". To the right of the input field is a button labeled "Calculate". Below this, there are four rows of labels and input fields: "Quarters:" with a field containing "2", "Dimes:" with a field containing "1", "Nickels:" with a field containing "1", and "Pennies:" with a field containing "2".

1. Open the application in this folder:
`exercises_extra\ch16\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. Modify the code in the library file so it defines an object literal named `coins`. This object should have properties for quarters, dimes, nickels, and cents. In addition, it should have a `makeChange()` method that accepts the number of cents.

If the number of cents is valid, the `makeChange()` method should calculate the number of coins for each type and update the quarters, dimes, nickels, and cents properties. Otherwise, it should throw an exception.
6. Modify the code in the `calculate.js` file to use the object literal named `coins` to calculate and display the number of coins for each type of coin. Note that this code should still validate the user entry. That way, this script won't pass an invalid amount to the `makeChange()` method, and that method won't throw an exception.

Extra 16-2 Use a class with the Change Calculator

In this exercise, you'll modify a Change Calculator application so it uses a class. When you're done, the application will work the same as it did before.



The screenshot shows a web form titled "Change Calculator" in blue text. Below the title, there is a label "Enter number of cents (0-99):" followed by a text input field containing the number "67". To the right of the input field is a button labeled "Calculate". Below this, there are four rows of labels and input fields: "Quarters:" with a field containing "2", "Dimes:" with a field containing "1", "Nickels:" with a field containing "1", and "Pennies:" with a field containing "2".

1. Open the application in this folder:
`exercises_extra\ch16\change_class\`
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. Modify the code in the library file so it defines a class named `Coins`. This class should have a constructor that defines properties for quarters, dimes, nickels, and cents and initializes them to 0. In addition, it should have a `makeChange()` method that accepts the number of cents.

If the number of cents is valid, the `makeChange()` method should calculate the number of coins for each type and update the quarters, dimes, nickels, and cents properties. Otherwise, it should throw an exception.
6. Modify the code in the `calculate.js` file to create an object from the `Coins` class. Then, use that object to calculate and display the number of coins for each type of coin. Note that this code should still validate the user entry. That way, this script won't pass an invalid amount to the `makeChange()` method, and that method won't throw an exception.

Extra 16-3 Convert the PIG app to objects

In this exercise, you'll finish converting the PIG application from chapter 12 so it uses objects. When you're done, the application will work the same as it did before, but the code will be organized differently.

The screenshot shows a web application titled "Let's Play PIG!". It features a "Rules" section with a list of game rules. Below the rules, there are input fields for "Player 1" (containing "Mary") and "Player 2" (containing "Mike"), each followed by a "Score" field (20 for Mary, 13 for Mike). A "New Game" button is located to the right of the Player 2 score. Below the player information, it says "Mike's turn" in red. At the bottom, there are buttons for "Roll" and "Hold", followed by "Die" (6) and "Total" (9) fields.

To start this game, you enter the names for the two players and click the New Game button. Then, the message below the names indicates whose turn it is. When the game is over, a dialog box announces the winner.

1. Open the application in this folder:
`exercises_extra\ch16\pig\`
Note that it uses three JavaScript library files besides the main `pig.js` file.
2. Review the `library_die.js` file. Note that it contains a class named `Die` with no properties and a single method that rolls a die.
3. Review the `library_pig.js` file. Note that it contains a class named `Pig` that has a property that's an object created from the `Die` class. The `Pig` class has a `constructor()` function with five data properties, a read-only accessor property, and three methods.
4. Review the `library_game.js` file. Note that it's an object literal with three data properties, one read-only accessor property, and five methods. The `start()` method and `isValid` read-only property are already coded for you, and the other methods have comments indicating what they should do. Now, finish the code for those four methods and make them cascading methods when you can.
5. Review the `pig.js` file and note that it supplies the ready event handler and the click event handler of the New Game button. There are also starts for the click event handlers for the Roll and Hold buttons, which you need to finish. These handlers should use the methods of the game object in the `library_game.js` file.

Extra 17-1 Convert the Clock application to closures

In this exercise, you'll convert a Clock application to use closures.



1. Open the application in this folder:
`exercises_extra\ch17\clock_closures\`
2. Note that the folder contains two library files named `library_clock.js` and `library_stopwatch.js`. In the main JavaScript file, note that the variables, objects, and functions 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 17-2 Use IIFEs with the Clock

In this exercise, you'll convert a Clock application to use immediately invoked function expressions (IIFEs).



1. Open the application in this folder:
`exercises_extra\ch17\clock_iifes\`
2. Change the clock library so it uses an IIFE.
NOTE: In this step and in step 3, 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.
3. Change the stopwatch library so it uses an IIFE.
4. Change the code in the clock.js file to use the objects created by the clock and stopwatch IIFEs. Since you no longer call code to create these objects, remove that code and pass the span elements to the start() methods instead.

Extra 17-3 Review a Clock application that uses ES modules

In this exercise, you'll review a Clock application that uses ES modules.

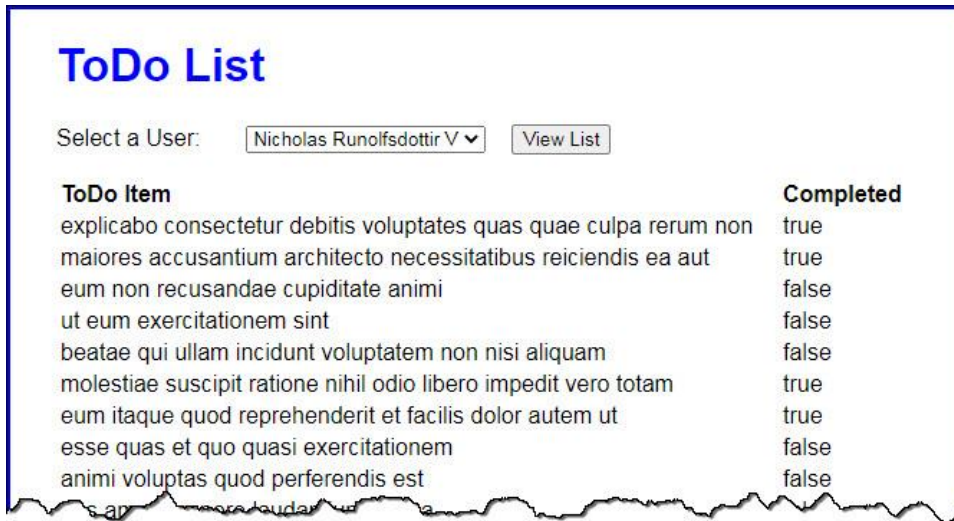
NOTE: Because ES modules can't be run from the file system, this exercise only has you review the code for this application. However, extra exercise 19-2 shows how to use Node.js to run this application on a web server.



1. Open the application in this folder:
`exercises_extra\ch17\clock_modules\`
In the HTML file, note that the script tags for the library files and the main JavaScript file set the type attribute to "module". Also, note a script tag for a library_utility.js that comes before the other library files.
2. Review the library_utility.js file. The export statement at the top of the file identifies the padSingleDigit() function as an item of the module.
3. Review the library_clock.js file. The export statement at the top of the file identifies the start() function as an item of the module. Additionally, the import statement imports the padSingleDigit() function.
4. Review the rest of the library_clock.js file. Note that it has an object and a function that are private. That is, they aren't exported. Also, note that the code in the private function calls the imported padSingleDigit() function.
5. Review the library_stopwatch.js file. The export statement at the top of the file identifies the start(), stop(), and reset() functions as items of the module. Additionally, the import statement imports the padSingleDigit() function.
6. Review the rest of the library_stopwatch file. Note the constants, variables, objects, and functions of its private state. Also, note that it uses the imported padSingleDigit() function.
7. Review the clock.js file. The import statements at the top of the file import all the items from the clock and stopwatch modules. Additionally, the import statements use aliases to make the imported functions easier to use.
8. Review the rest of the clock.js file. Note how it uses the aliases and the imported module items.

Extra 18-1 Use data from the JSON Placeholder API

In this exercise, use data from the JSON Placeholder API to create an application that displays a list of to-do items for a specified user. After selecting a user and clicking “View List”, the application should look like this:



1. Open the application in this folder:
`exercises_extra\ch18\todo_list\`
2. In the `todo_list.js` file, there's a const named `domain` that contains the URL for the JSON Placeholder API.
3. Add an asynchronous function named `displayUsers()` that makes an asynchronous request from the API for all ten users.
4. Display the users in the select element whose id is “users”. Make sure the option elements have a value attribute that contains the id value for each user.
5. Add an asynchronous ready event handler. It should call the `displayUsers()` function and add an asynchronous event handler for the click event of the button.
6. The click event handler should get the user id for the selected user from the select element and use it to make an asynchronous request from the API for all the to-do items associated with that user. To do that, you can use add a querystring to the API URL, like this:
`https://jsonplaceholder.typicode.com/todos/?userId=<id goes here>`
7. Display the to-do items in the div element whose id is “list”. Format the items as an HTML table.

Extra 18-2 Use data from NASA's Mars Rover Photos API

In this exercise, you'll modify an application that allows you to view photos from any of the rovers sent to Mars by NASA. After selecting a rover, a date, and camera and clicking the "View" button, the application should look like this:

1. Open the application in this folder:
`exercises_extra\ch18\mars_rover\`
2. Open the `rover.js` file. Note that it already contains all of the JavaScript for working with the user interface, except the asynchronous `getJSON()` function.
3. Add the function named `getJSON()` that makes an asynchronous request for the specified URL and returns an object that's created from the JSON that's returned by the request.
4. Run the application to make sure it works correctly. It should.
5. Add a `console.log()` statement to the `getJSON()` function that prints the request URL to the console.
6. Run the application. When you select a rover, note that the URL that's passed to the `getJSON()` function is:
`https://api.nasa.gov/mars-photos/api/v1/rovers?api_key=DEMO_KEY&page=1`
7. When you click the View button, note that the end of the URL that's passed to the `getJSON()` function uses a format like this:
`.../rovers/Spirit/photos/?api_key=DEMO_KEY&page=1&earth_date=2010-3-21`
8. To see all the options available for the Mars Rover Photos API, visit <https://api.nasa.gov/>, scroll down, and expand the Mars Rover Photos item.

Extra 19-1 Create and run a script with parameters

In this exercise, you'll finish the JavaScript for a server-side script named `test_scores` that calculates the average of the test scores that are passed to it. You should be able to pass one or more test scores to this script.

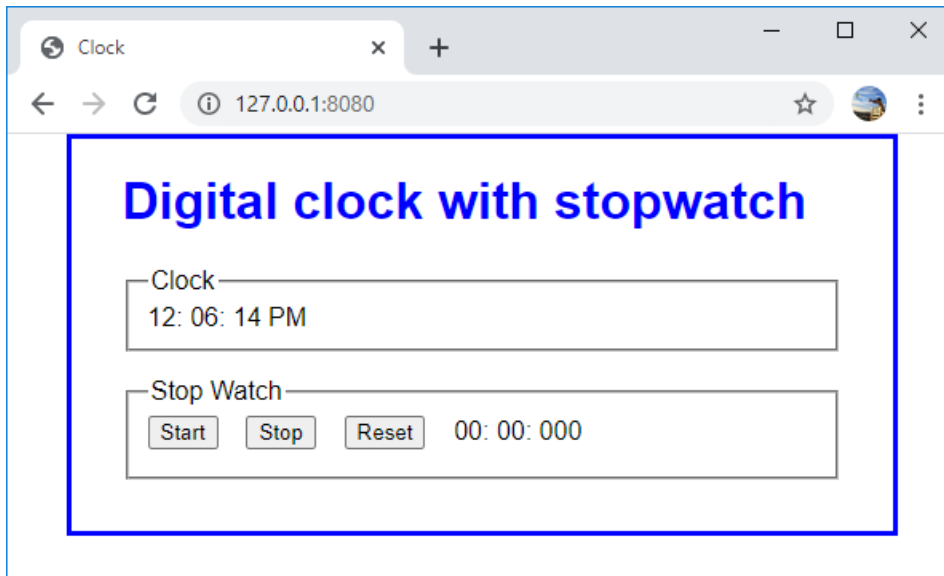
```
> cd /murach/javascript_jquery/extra_exercises/ch19

> node test_scores 89 100 92 93
All scores: 89, 100, 92, 93
Average score: 94
```

1. If necessary, install Node.js on your system.
2. Open the `index.js` file in this directory:
`exercises_extra\ch19\test_scores\`
3. The `index.js` file contains an empty array named `scores`, a function named `displayScores()` that converts the array to a comma-separated string, and a function named `calculateAverage()` that calculates the average of the scores stored in the `scores` array.
4. Add code to the JavaScript file that loops through the `process.argv` array and stores the values entered by the user in the `scores` array. Remember to skip the first two elements in the `process.argv` array.
5. Add code that displays all the scores and the average score in the console. Use the `displayScores()` and `calculateAverage()` functions to do that.
6. Start the command line for your operating system. Then, use the `cd` command to change the current directory to the `extra_exercises/ch19` directory that contains the `test_scores` directory.
7. Use the `node` command to run the `index.js` script. Test this script to make sure it can handle any number of test score parameters. For example, test it with 2 test scores, and test it with 5 test scores.

Extra 19-2 Run a Clock application that uses ES modules

In this exercise, you'll create a web server with node.js so you can run the Clock application you reviewed in extra exercise 17-3:



1. If necessary, install Node.js on your system.
2. Start a command line for your system.
3. If necessary, use the npm install command to install the http-server module globally.
4. Use a command like the following cd command to change to the ch17/clock_modules directory:

```
cd /murach/javascript_jquery/exercises_extra/ch17/clock_modules
```
5. Use the http-server command to start the web server. If your firewall displays a message asking whether you want to allow Node.js to communicate on private networks, click Allow. If successful, this should display a message that indicates that the server is running. Make a note of the URL in the message.
6. Open a web browser and enter the URL from the last step in the address bar. Be sure to include the port number that comes after the colon.
7. Test the application and make sure the clock and stopwatch work correctly. When you're done, close the browser.
8. Switch to the command line and press Ctrl-C to stop the server. This should display a message that indicates that the server was stopped.