

Short Exercises

Each of these exercises is designed to test just one or two JavaScript or jQuery skills, and each is designed so it can be done within 5 to 30 minutes. At the start of each exercise, you'll see an estimated time for the exercise.

Guidelines for doing the short exercises	2
Short 1-1 Test an application and find an error	3
Short 2-1 Modify the Test Scores application	4
Short 3-1 Enhance the Future Value application	5
Short 4-1 Enhance the MPG application	6
Short 4-2 Use arrow functions with the Test Scores application	7
Short 5-1 Debug the MPG application	8
Short 6-1 Upgrade the MPG application	9
Short 6-2 Display Test Score arrays	10
Short 7-1 Preload images and use a timer	11
Short 8-1 Redo a Future Value application with jQuery	12
Short 8-2 Create a FAQs Rollover application	13
Short 9-1 Add effects to an Image Gallery application	14
Short 9-2 Debug a Slide Show application	15
Short 11-1 Convert the FAQs app to an Accordion widget	16
Short 11-2 Create a Progressbar widget that uses a timer	17
Short 12-1 Improve the validation of the Countdown application	18
Short 12-2 Add dates to the Invoice application	19
Short 13-1 Add a default invoice date to the Invoice application	20
Short 13-2 Add exception handling to the Countdown application	21
Short 14-1 View the query string of a URL	22
Short 15-1 Allow multiple task entries in the Task List application	23
Short 16-1 Use a class instead of an object literal	24
Short 16-2 Use an object literal instead of a class	25
Short 17-1 Use the module pattern to create private state	26
Short 18-1 Enhance the Ajax for an application	27
Short 19-1 Run and modify a server-side script	28

Guidelines for doing the short exercises

- For all the short exercises, you will start with the HTML, CSS, and JavaScript or jQuery for the application. Then, you will modify the JavaScript or jQuery as directed by the exercise.
- 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.

Short 1-1 Test an application and find an error

In this exercise, you'll run a version of the Email List application and discover that it stops running due to a coding error. Then, you'll use Chrome to identify the statement that caused the error. Estimated time: 5 to 10 minutes.

Please join our email list

Email Address:

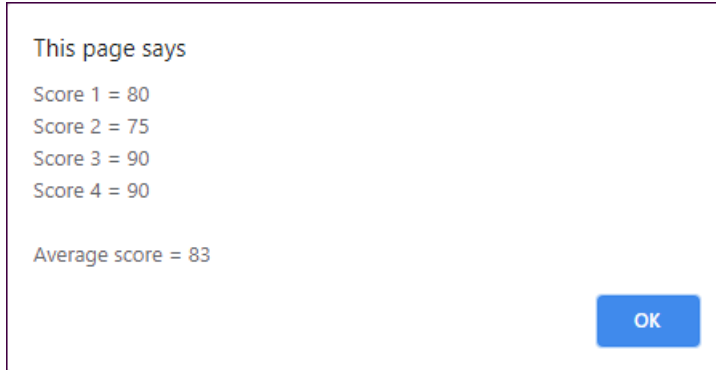
Confirm Email Address: **Emails must match.**

First Name: **First name is required.**

1. Open the application in this folder:
`exercises_short\ch01\email_list`
2. Start the application, enter an email address in the first text box, and click the Join List button. Note the error messages that are displayed to the right of the other two text boxes.
3. Enter a different email address in the second text box, and enter your name in the third text box. Then, click the Join List button to see what error messages are displayed.
4. Enter valid data in all three text boxes and click the Join List button. Then, note that nothing happens.
5. Use Chrome's developer tools to locate the statement that caused the error.
6. Use your editor or IDE to fix the error (change *submitt* to *submit*). Then, save your files, and test the application again with valid data. This time, a new page should be displayed when you click the Join List button.

Short 2-1 Modify the Test Scores application

In this exercise, you'll modify the Test Scores application so it provides for four test scores and displays the results in a dialog box like the one that follows. Estimated time: 10 to 20 minutes.



1. Open the application in this folder:
`exercises_short\ch02\test_scores`
2. Run the application, and note that it works like the one in the book and that it writes the results to the browser page. Then, review the code in the JavaScript file, and note that it's slightly different than the code in the book, although it gets the same results.
3. Modify the application so it provides for a fourth test score.
4. Modify the application so it displays the results in a dialog box like the one above, as well as in the browser page after the dialog box is closed.

Short 3-1 Enhance the Future Value application

In this exercise, you'll make a quick enhancement that shows not only the future value when interest is compounded yearly, but also when it's compounded monthly. As a result, the display in the browser should look like this:

The Future Value Calculator	
Future Value with Yearly Interest	
Investment amount:	10000
Interest rate:	7.5
Years:	10
Future Value:	20610.32
Future Value with Monthly Interest	
Investment amount:	10000
Interest rate:	7.5
Years:	10
Future Value:	21120.65

Estimated time: 15 to 20 minutes.

1. Open application in this folder:
`exercises_short\ch03\future_value`
2. Run the application to make sure it works correctly.
3. Review its code. Note that it is just like the code in the book and that the interest is calculated each year.
4. Add the code for calculating the future value when interest is calculated each month. Then, add the code for displaying the results, as shown above. Be sure to add the heading that identifies each result.

Short 4-1 Enhance the MPG application

In this exercise, you'll make a couple of quick enhancements to the Miles Per Gallon application, like clearing the two entries if the user double-clicks in the Miles Per Gallon text box. Estimated time: 10 to 15 minutes.

The Miles Per Gallon Calculator

Miles Driven:

Gallons of Gas Used:

Miles Per Gallon:

1. Open the application in this folder:
`exercises_short\ch04\mpg`
2. Run the application to see that it works just like the one in the book. Then, in the JavaScript file, note that there's a `clearEntries()` function that isn't used.
3. Enhance the application so the entries are cleared when the user double-clicks in the Miles Per Gallon text box. (Incidentally, this won't work if the text box is disabled.)
4. Enhance the application so the Miles Driven text box is cleared when it receives the focus. Then, do the same for the Gallons of Gas Used text box.
5. Enhance the application so the calculation is done when the focus leaves the Gallons of Gas Used text box. To do the calculation, you just need to run the `processEntries()` function when that event occurs.

Short 4-2 Use arrow functions with the Test Scores application

In this exercise, you'll modify a version of the Test Scores application so it uses arrow functions instead of function expressions. This version displays the average score after each score is entered as shown below. You'll also use an anonymous function for the DOMContentLoaded event handler. Estimated time: 10 to 15 minutes.

The Test Scores App

Score:

Average:

1. Open the application in this folder:
`exercises_short\ch04\test_scores`
2. Run the application and add two or more scores to see that the new average is displayed each time another score is added.
3. Review the code in the JavaScript file and note that all of the functions are written as function expressions.
4. Modify the `$()`, `addScore()`, and `calculateAverage()` functions so they use arrow functions instead of function expressions.
5. Modify the `document.addEventListener()` method so it uses an anonymous function for the event handler instead of the `processDOM()` function.

Short 5-1 Debug the MPG application

This exercise gives you a chance to use your debugging skills to find the cause of an error. Estimated time: 5 to 15 minutes.

The Miles Per Gallon Calculator

Miles Driven:

Gallons of Gas Used:

Miles Per Gallon:

1. Open the application in this folder:
`exercises_short\ch05\mpg`
2. Start the application, enter the values shown above, and click on the Calculate MPG button. Nothing happens.
3. Use Chrome's developer tools to find the cause or causes of the problem. Then, use your IDE or text editor to fix the code.
4. Start the application again, enter the values shown above, and click on the Calculate MPG button. This time, 27 will be displayed in the Miles Per Gallon text box.
5. Fix the application so the miles per gallon will be displayed with one decimal place.

Short 6-1 Upgrade the MPG application

In this exercise, you'll upgrade a version of the MPG application so the error messages are displayed in span elements to the right of the text boxes. Estimated time: 10 to 15 minutes.

The Miles Per Gallon Calculator

Miles Driven: Must be a valid number greater than zero.

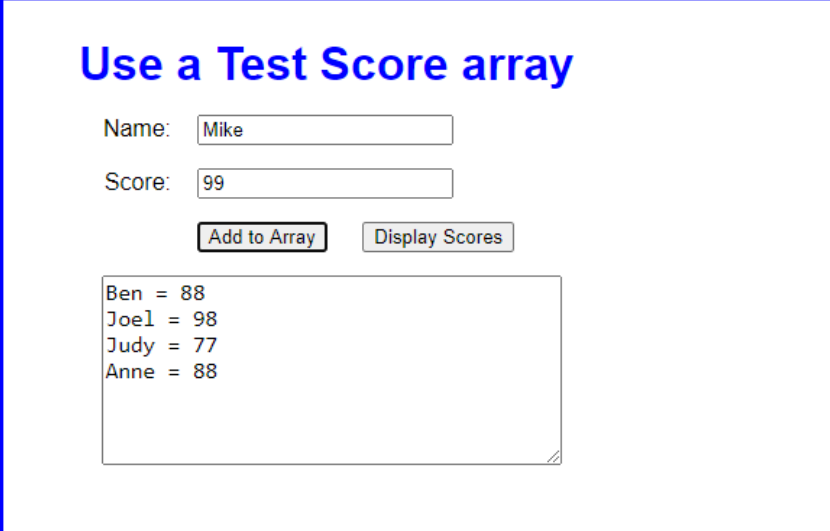
Gallons of Gas Used: Must be a valid number greater than zero.

Miles Per Gallon:

1. Open the application in this folder:
`exercises_short\ch06\mpg`
Then, run the application and click the Calculate MPG button to see that an error message is displayed in an alert dialog box for each of the two input fields.
2. In the HTML file, add a span element after the input element for the first two text boxes and set the initial content to an asterisk.
3. Enhance the data validation so it displays the error messages in the span elements instead of in alert dialog boxes.
4. If the user entries are valid, clear the contents of the span elements.
5. If the user clicks the Clear Entries button, set the contents of each span element to an asterisk.

Short 6-2 Display Test Score arrays

In this exercise, you'll start with the declarations for two arrays, a names array and a scores array. Then, you'll add an event handler that displays the names and scores in the text area of the interface. Estimated time: 20 to 30 minutes.



Use a Test Score array

Name:

Score:

```
Ben = 88
Joel = 98
Judy = 77
Anne = 88
```

1. Open the application in this folder:
exercises_short\ch06\test_scores
If you run the application, you can enter a name and score and then click the Add to Array button to add names and scores to the arrays, but you can't use the other button to display the scores.
2. Look at the JavaScript code to see that it starts with the declarations for two arrays that contain four elements each. The first is an array of names. The second is an array of scores. Note too that the JavaScript code includes the `$()` function, a `DOMContentLoaded` event handler, and the `addScore()` function for adding a name and score to the arrays.
3. Add an event handler for the Display Scores button that displays the names and scores in the arrays, as shown above. You will also need to use the `DOMContentLoaded` event handler to attach this function to the click event of the Display Scores button. After you test this with the starting array values that are shown above, add a name and score to the arrays and test the display again.
4. Modify the event handler for the Add to Array button so the data in the text area is cleared after a name and score are added to the arrays.

Short 7-1 Preload images and use a timer

In this exercise, you'll modify an Image Rollover application so it preloads the images that are displayed when the original images are rolled over. In addition, you'll create timers that cause the rollover images to be displayed when the page is first loaded. Estimated time: 10 to 15 minutes.



1. Open the application in this folder:
`exercises_short\ch07\rollover`
2. Run the application and move the mouse pointer over each of the two images to see that the original image is replaced with another image when the mouse is in the image.
3. Add JavaScript code to preload the rollover images. These are the images that are specified by the id attributes of the img elements.
4. Add a timer that causes the rollover images to be displayed one second after the page is loaded.
5. Add another timer that causes the original images to be displayed again two seconds after the page is loaded.

Short 8-1 Redo a Future Value application with jQuery

In this exercise, you'll rewrite the code for a Future Value application that uses JavaScript so it uses jQuery. That will show you how jQuery can simplify an application, but also that JavaScript is still needed with jQuery applications. Estimated time: 10 to 15 minutes.

Future Value Calculator

Investment Amount:

Annual Interest Rate:

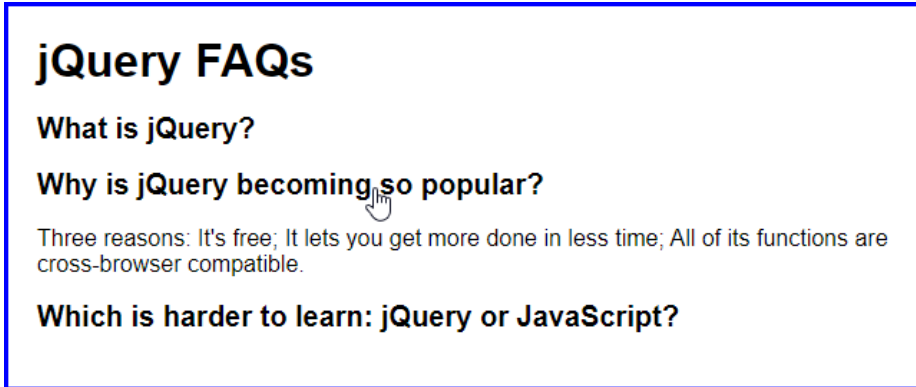
Number of Years:

Future Value:

1. Open the application in this folder:
`exercises_short\ch08\future_value`
Then, run the application to refresh your memory about how it works.
2. Add a script element to the HTML file that gives you access to the jQuery library.
3. Rewrite the code in the JavaScript file so it uses as much jQuery as possible. That includes the code for the `ready()` and `click()` event methods.
4. When you have the application working right, add jQuery code to move the focus to the Investment Amount text box each time the Calculate button is clicked.

Short 8-2 Create a FAQs Rollover application

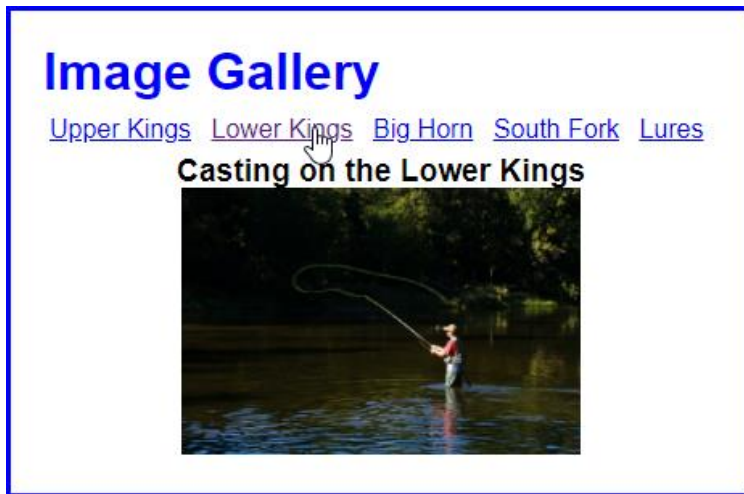
In this exercise, you'll write the jQuery code for an application that displays the answer to a question in a list of questions when the user hovers the mouse over that question. Estimated time: 10 to 15 minutes.



1. Open the application in this folder:
`exercises_short\ch08\faqs_rollover`
2. Review the HTML to see that it includes an unordered list with list items that consist of an `h2` element and a `<p>` element. The `h2` element contains a question, and the `<p>` element, which is hidden, contains the answer to the question.
3. Add jQuery code that runs when the user moves the mouse pointer into or out of a question in the list. When the mouse pointer moves into the question, the answer should be displayed. When the mouse pointer moves out of the question, the answer should be hidden.

Short 9-1 Add effects to an Image Gallery application

In this exercise, you'll use effects to change the way the images are displayed and hidden. Estimated time: 5 to 10 minutes.



1. Open the application in this folder:
`exercises_short\ch09\image_gallery`
Then, run the application to see that when you click on one of the links, a different caption and image is displayed.
2. Modify the jQuery so that when a link is clicked, the current caption and image are hidden using a sliding motion that takes two seconds.
3. Modify the jQuery so that after the caption and URL are set for the new image, that caption and image are displayed using a sliding motion that takes two seconds.

Short 9-2 Debug a Slide Show application

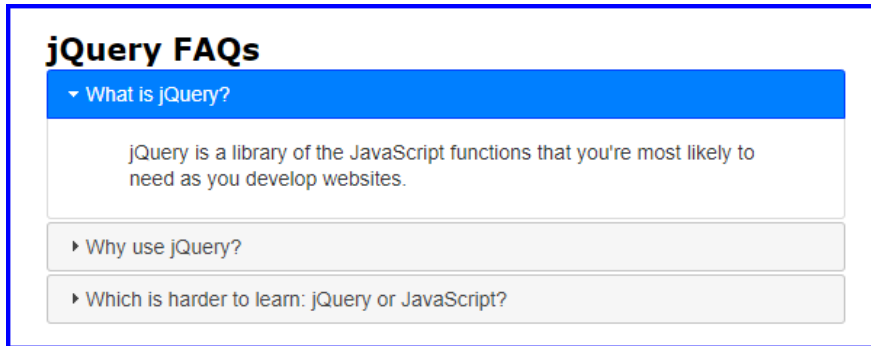
This Slide Show application is like the one in the book, but it has a bug in it. Instead of displaying the slides in sequence, it displays every other slide. Your task is to debug this application. Estimated time: 5 to 10 minutes if you can find the bug.



1. Open the application in this folder:
exercises_short\ch09\slide_show
2. Review the HTML to see how the images for the slides are structured. Then, run the application and note that it displays every other slide. Because there are only five slides in the HTML, this means that it displays slides 1, 3, 5, 2, 4, and so on.
3. Debug this application.

Short 11-1 Convert the FAQs app to an Accordion widget

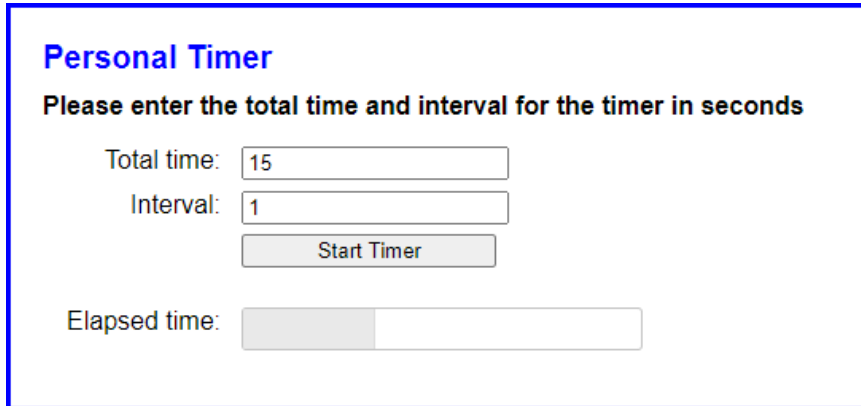
This exercise has you change the FAQs application from JavaScript and jQuery code to a jQuery UI Accordion widget. Estimated time: 15 to 20 minutes.



1. Open the application in this folder:
exercises_short\ch11\faqs
2. In the HTML file, note that the link and script tags that you need for jQuery UI have been coded for you. Then, rewrite the HTML so it's consistent with the HTML that the Accordion widget needs. Here's what the jQuery UI website says about the Accordion widget:
"The underlying HTML markup is a series of headers (h3 tags) and content divs so the content is usable without JavaScript."
3. In the JavaScript file, comment out everything within the ready() event handler. Then, write the code for using the Accordion widget so all the panels can close and the panel height is based on the content height.
4. In the CSS file, comment out everything that's no longer needed for this application.

Short 11-2 Create a Progressbar widget that uses a timer

In this exercise, you'll modify an application that uses a timer to display the elapsed minutes and seconds in a text box so it displays the elapsed time in a Progressbar widget. Estimated time: 10 to 15 minutes to research the widget and another 10 to 15 minutes to implement the changes.



Personal Timer

Please enter the total time and interval for the timer in seconds

Total time:

Interval:

Elapsed time:

1. Open the application in this folder:
`exercises_short\ch11\personal_timer`
2. Run this application and test it. When you enter a total time and interval and click the Start Timer button, note that the elapsed minutes and seconds are displayed in a text box and are updated each time the interval passes.
3. Go to the jQuery UI website at <https://jqueryui.com>, and display the documentation for the Progressbar widget. Review the code for one or more of the examples, then review the API documentation.
4. Modify the HTML for the application so it uses a Progressbar widget to display the elapsed time instead of a text box. Be sure to give the widget an id of "progressbar" so the CSS for the application works correctly.
5. Add a statement to the JavaScript file that activates the widget with a starting value of 0.
6. Modify the code that's executed if the two entries are valid so it uses the widget. When you do that, you can delete the code that calculates and displays the elapsed minutes and seconds since it's no longer needed. Then, you can add code that calculates and changes the value of the Progressbar widget.

Short 12-1 Improve the validation of the Countdown application

In this exercise, you'll improve the validation for the date entered by the user in the Countdown application in chapter 12. When you're done, this application should display specific error messages for dates with invalid months or days. Estimated time: 15 to 25 minutes.

Countdown To...

Event Name:

Event Date:

Please enter the date in MM/DD/YYYY format.

1. Open the application in this folder:
`exercises_short\ch12\countdown`
2. Start the application, enter an invalid date like the one shown above, and note that the error message doesn't accurately describe the error.
3. In the JavaScript file, find the if statement that checks whether the date string has a four-digit year. Following this code, add code that gets the month and day from the dateParts array that was created earlier in the code.
4. Add code that checks that a valid day was entered depending on the month that was entered. For example, April can have the days 1 through 30, and May can have the days 1 through 31. Be sure to account for leap years, which are years that are evenly divisible by 4. If the days are invalid, display an appropriate error message and return.
5. Add code that displays an error message and returns if a month other than 1 through 12 are entered.

Short 12-2 Add dates to the Invoice application

In this exercise, you'll modify an Invoice application so it gets the invoice date for each invoice entered by the user and calculates the due date. Estimated time: 15 to 20 minutes.

Invoice Total Calculator

Customer Type:

Invoice Subtotal:

Invoice Date:

Discount Percent: %

Discount Amount:

Invoice Total:

Due Date:

1. Open the application in this folder:
`exercises_short\c12\invoice`
2. Start the application and click the Calculate button without entering a subtotal or invoice date. An error message will be displayed indicating that the subtotal must be a number greater than zero.
3. Enter a valid subtotal and click the Calculate button again. This time, the discount and invoice total will be calculated, but no invoice date or due date will be displayed.
4. Code a function that formats the Date object that's passed to it in MM/DD/YYYY format and then returns the date string.
5. Add code to the click() event handler for the Calculate button that gets the invoice date and creates a Date object from it.
6. Add code that checks whether the invoice date is not equal to an empty string and whether the Date object is not a valid date. If so, display an error message, clear the controls, move the focus to the Invoice Date text box, and return.
7. Add an if statement that checks whether the invoice date is equal to an empty string. If so, use the current date as the default date. To do that, you'll need to get the current date and format it using the function you coded in step 4.
8. Add code that calculates the due date as 30 days after the invoice date. Then, format that date.
9. Add code that sets the values of the Invoice Date and Due Date fields.

Short 13-1 Add a default invoice date to the Invoice application

In this exercise, you'll modify the Invoice application so it provides a default value for the Invoice Date field. Estimated time: 5 to 10 minutes.

Invoice Total Calculator

Enter the two values that follow and click "Calculate".

Customer Type:

Invoice Subtotal:

Invoice Date:

Discount Percent: %

Discount Amount:

Invoice Total:

1. Open the application in this folder:
`exercises_short\ch13\invoice`
2. Open the HTML file and notice that an Invoice Date field has been added. Then, open the HTML file and notice that it includes a `getFormattedDate()` function that formats the Date object that's passed to it in MM/DD/YYYY format and then returns the date string.
3. Start the application, enter a subtotal, and click the Calculate button. Note that nothing is displayed in the Invoice Date field.
4. In the `click()` event handler for the Calculate button, add code that gets the current date and formats it using the `getFormattedDate()` function. Then, add code that provides a default value of the formatted current date for the invoice date. Note that if an invoice date is entered, that date isn't validated.
5. Add code that sets the value of the Invoice Date field. If you've done this right, the application should display the current date when you click the Calculate button without entering anything in the Invoice Date field.

Short 13-2 Add exception handling to the Countdown application

In this exercise, you'll add exception handling to the Countdown application to make sure an argument is passed to a function. When you're done, the Countdown application will display a custom error message if the argument isn't sent (see below). Estimated time: 20 to 30 minutes.

Countdown To...

Event Name:

Event Date:

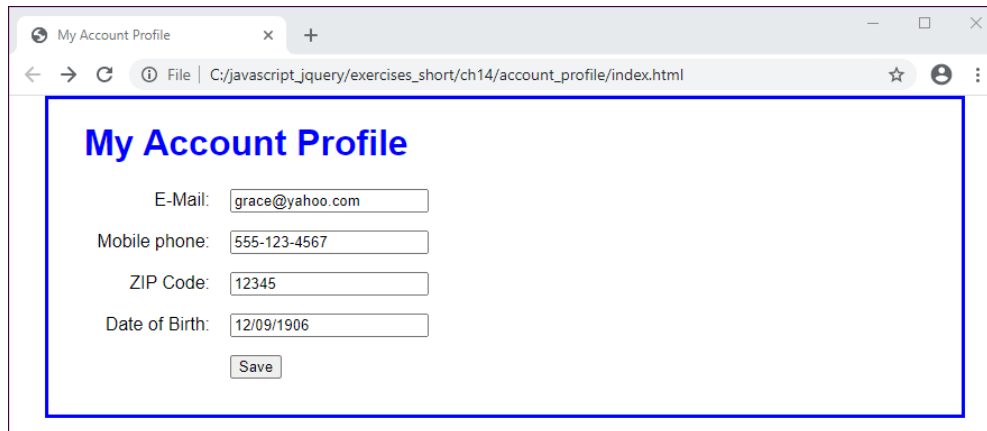
ReferenceError: The calculateDays function requires a Date parameter.

1. Open the application in this folder:
`exercises_short\ch13\countdown`
2. Review the JavaScript code, and notice that the code that calculates the number of days to the event date has been stored in a function named `calculateDays()`.
3. Comment out the call to the `calculateDays()` function. Then, code a new call that passes no arguments to the function. Now, run the application and see that nothing happens.
4. Open the Console panel of the developer tools and view the error message there - "Cannot read property 'getTime' of undefined". Note that this error message gives you information about the problem, but you'll still need to do some work to track down what's wrong.
5. In the JavaScript file, modify the `calculateDays()` function so it makes sure the "date" parameter is a Date object. If it is, the function should be done. Otherwise, this function should create and throw an error object with the custom message shown above.
6. Run the application again and then view the error message in the Console panel. This time, the error message explains exactly what is wrong.
7. Put the call to the `calculateDays()` function and the if statements that follow it inside a try-catch statement. In the catch block of the statement, display the custom error message in the span tag whose id is "message". Run the application and view the error message in the page.
8. Fix the application so the call to the `calculateDays()` function passes the date as an argument like it originally did.

Short 14-1 View the query string of a URL

In this exercise, you'll adjust the Account Profile application so it displays the data that is in the query string of the URL. Estimated time: 5 to 10 minutes.

Here is the enhanced application, with account profile information about to be saved:

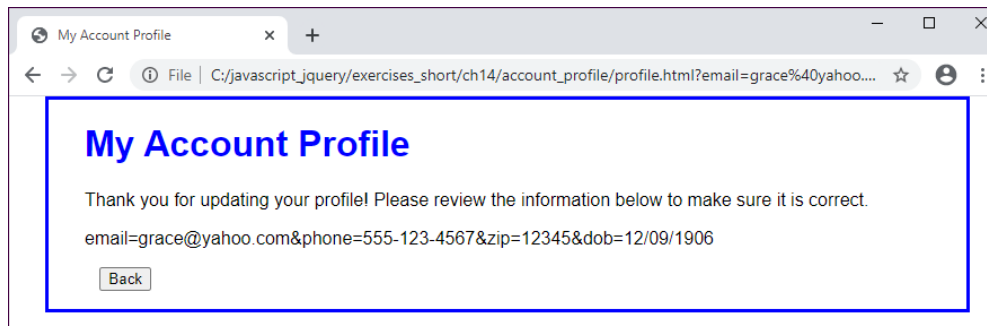


The screenshot shows a web browser window titled "My Account Profile". The address bar shows the file path: `C:/javascript_jquery/exercises_short/ch14/account_profile/index.html`. The page content is titled "My Account Profile" in blue. Below the title is a form with the following fields and values:

- E-Mail:
- Mobile phone:
- ZIP Code:
- Date of Birth:

At the bottom of the form is a "Save" button.

And here is the second page of the application after the profile is saved. Notice that the profile data is in the URL, and special characters like "@" and "/" are encoded:



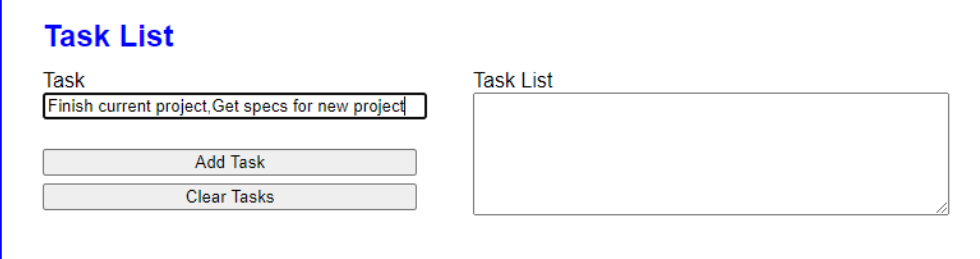
The screenshot shows the same web browser window, but the page content has changed. The title is still "My Account Profile". The main content area says: "Thank you for updating your profile! Please review the information below to make sure it is correct." Below this is the encoded query string: `email=grace%40yahoo.com&phone=555-123-4567&zip=12345&dob=12/09/1906`. At the bottom is a "Back" button.

1. Open the application in this folder:
exercises_short\ch14\account_profile
2. Review the `index.html` file, and notice that the elements on the page are coded within a form element with its method attribute set to "get" and its action attribute set to "profile.html". Then, review the JavaScript file, and notice that when the validation checks pass, the form is submitted. This is how the form data gets in the query string.
3. In the embedded JavaScript for the `profile.html` file, use the location object to retrieve the query string value. Remove the question mark from that value and then display the value in the span element whose id is "profile". Use the `decodeURIComponent()` function so the special characters display correctly.

Short 15-1 Allow multiple task entries in the Task List application

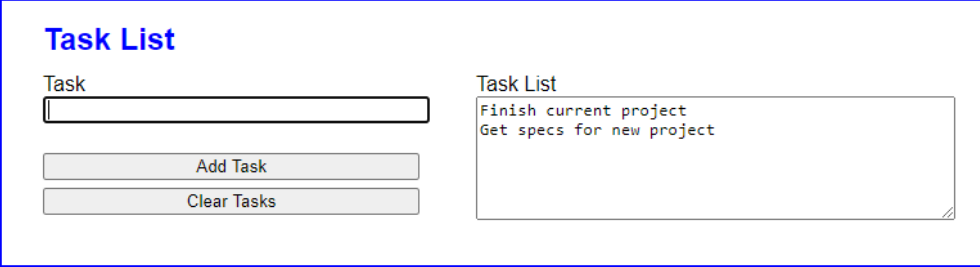
In this application, you'll make an enhancement that allows you to enter multiple tasks separated by commas in a single entry. Estimated time: 10 to 20 minutes.

Here is the enhanced application, with multiple tasks about to be entered:



The screenshot shows a web application titled "Task List". On the left, there is a "Task" input field containing the text "Finish current project, Get specs for new project". Below this input field are two buttons: "Add Task" and "Clear Tasks". On the right, there is a "Task List" container, which is currently empty.

And here is the application after the multiple tasks have been entered:



The screenshot shows the same "Task List" application. The "Task" input field is now empty. The "Add Task" button has been clicked, and the tasks "Finish current project" and "Get specs for new project" have been added to the "Task List" container, each on a new line.

1. Open the application in this folder:
`exercises_short\ch15\task_list\`
2. Run the application and add two tasks, separated by a comma. Note that the tasks are stored as one task, exactly as you entered it.
3. In the JavaScript file, find the `click()` event method of the Add Task button. Then, find the code that adds the task entered by the user to the tasks array. Replace that code with code that works for one or more tasks in an entry.

To do that, you can use the `split()` method of the String object to convert the user's entry into an array. Then, you can loop through that array to add the new tasks to the tasks array.

Short 16-1 Use a class instead of an object literal

In this exercise, you'll update the Miles Per Gallon application so it uses a class named `Trip` instead of an object literal named `mpg`. When you're done, the application should work the same as it did before. Estimated time: 10 to 20 minutes.

Calculate Miles Per Gallon

Miles Driven:

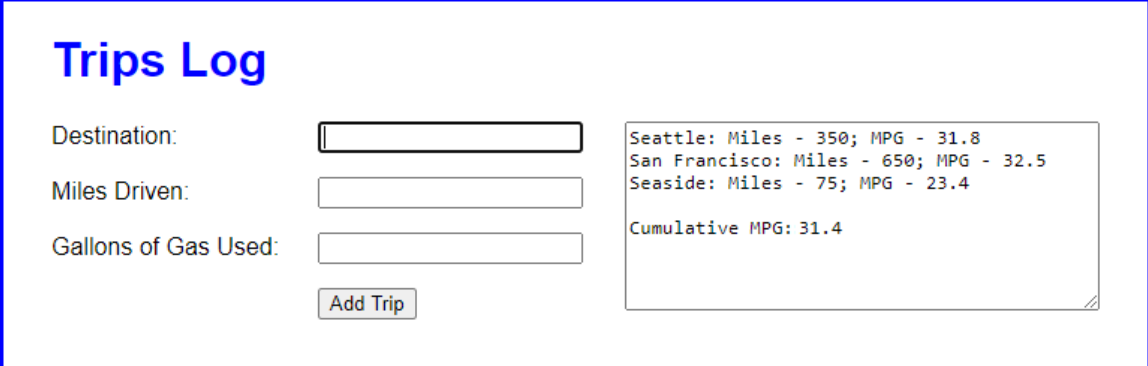
Gallons of Gas Used:

Miles Per Gallon

1. Open the application in this folder:
`exercises_short\ch16\mpg\`
Run and test the application.
2. Review the library file (`lib_mpg.js`). Note that it creates an object literal named `mpg` that contains some properties and a `calculate()` method.
3. Modify the library file, so it defines a class named `Trip` that has the same properties and methods as the `mpg` object. However, use a constructor to set the miles and gallons properties, and rename the `calculate()` method to `calculateMPG()`.
4. Open the main file (`mpg.js`) and modify it so it uses an object created from the `Trip` class instead of the `mpg` object literal. To do that, you need to pass the miles and gallons properties to the constructor of the `Trip` class.
5. Run and test the application again to make sure it still works as expected.

Short 16-2 Use an object literal instead of a class

In this exercise, you'll update the Trips application so it uses an object literal to store trips instead of using an object created from the Trips class. When you're done, the application should work the same as it did before. Estimated time: 10 to 20 minutes.

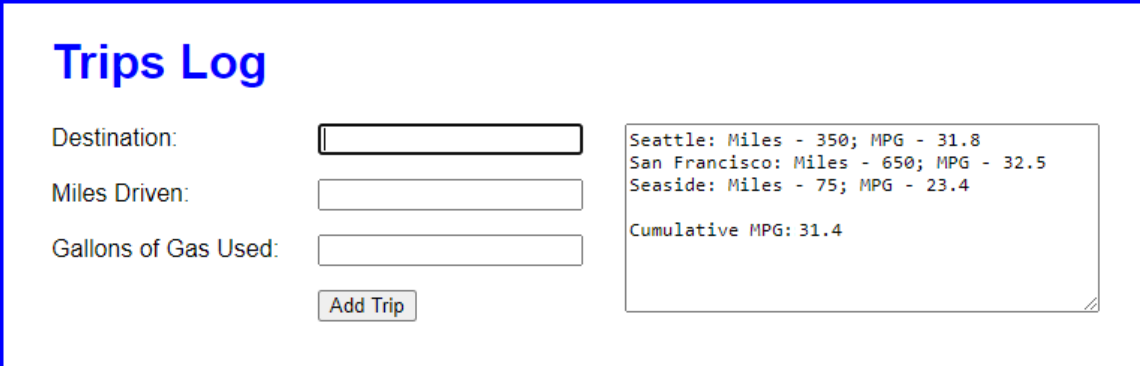


The screenshot shows a web application titled "Trips Log" in blue text. On the left, there are three input fields labeled "Destination:", "Miles Driven:", and "Gallons of Gas Used:". Below these fields is a button labeled "Add Trip". On the right, there is a text area displaying the following information: "Seattle: Miles - 350; MPG - 31.8", "San Francisco: Miles - 650; MPG - 32.5", "Seaside: Miles - 75; MPG - 23.4", and "Cumulative MPG: 31.4".

1. Open the application in this folder:
`exercises_short\ch16\trips\`
Run and test the application.
2. Review the library file (`lib_trips.js`). Note that it provides a class named `Trip` that you can use to create trip objects and a class named `Trips` that you can use to create trips objects.
3. Modify the library file, so it uses an object literal named `trips` instead of using the `Trips` class. This object literal should provide the same properties and methods as an object created from the `Trips` class.
4. Open the main file (`trips.js`) and modify it so it uses the `trips` object defined by the library instead of using the `Trips` class to create that object. Note how this makes the code shorter.
5. Run and test the application again to make sure it still works as expected.

Short 17-1 Use the module pattern to create private state

In this exercise, you'll update the Trips application so it uses the module pattern to create private state for the object that stores the array of trips. When you're done, the application should work the same as it did before. Estimated time: 15 to 30 minutes.



The screenshot shows a web application titled "Trips Log" in blue text. On the left, there are three input fields labeled "Destination:", "Miles Driven:", and "Gallons of Gas Used:". Below these fields is a button labeled "Add Trip". On the right, there is a text area displaying the following information: "Seattle: Miles - 350; MPG - 31.8", "San Francisco: Miles - 650; MPG - 32.5", "Seaside: Miles - 75; MPG - 23.4", and "Cumulative MPG: 31.4". The text area has a small icon in the bottom right corner, possibly for clearing or editing the text.

1. Open the application in this folder:
`exercises_short\ch17\trips\`
Run and test the application.
2. Review the library file (`lib_trips.js`). Note that it provides a class named `Trip` that you can use to create trip objects and a class named `Trips` that you can use to create trips objects.
3. Modify the library file, so it uses the module pattern to create a trips object that uses a private constant to store the array of `Trip` objects but provides public properties and methods named `push()`, `totalMpg`, and `toString()`.
4. Open the main file (`trips.js`) and modify it so it uses the trips object defined by the library instead of using the `Trips` class to create an object. Note how this makes the code shorter.
5. Run and test the application again to make sure it still works as expected.

Short 18-1 Enhance the Ajax for an application

In this exercise, you'll update a Blog application to change the way the blog posts are displayed. When you're done, the application should look and work the same as it did before, but it should display different blog posts. Estimated time: 15 to 30 minutes.

My Latin Blog

sunt aut facere repellat provident occaecati excepturi optio reprehenderit

quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

Posted by Bret [View comments](#)

et ea vero quia laudantium autem

delectus reiciendis molestiae occaecati non minima eveniet qui voluptatibus accusamus in eum beatae sit vel qui neque voluptates ut commodi qui incidunt ut animi commodi

Posted by Antonette [View comments](#)

asperiores ea ipsam voluptatibus modi minima quia sint

repellat aliquid praesentium dolorem quo sed totam minus non itaque nihil labore molestiae sunt dolor eveniet hic recusandae veniam tempora et tenetur expedita sunt

Posted by Samantha [View comments](#)

1. Open the application in this folder:
`exercises_short\ch18\blog\`
Run and test the application.
Note that it displays five blog posts. Note that each post includes the name of the user (in this case, all posts were made by Bret). And note that a “View comments” link allows you to view the comments for the post.
2. Review the JavaScript file. Note that it uses the `async` and `await` keywords to define asynchronous methods and to call them.
3. In the `ready()` event method, note that the code begins by getting an Array object that contains all posts. Then, it discards all posts except the first five.
4. In the `ready()` event method, modify the code so it only gets the posts with ids of 1, 11, 21, 31, and 41. To do that, you can create an array of Promise objects where each promise gets an object by requesting a URL that ends with `posts/1`, `posts/11`, `posts/21`, and so on. Then, you can use the `Promise.all()` method to execute the asynchronous requests.
5. Run and test the application again to make sure it works as expected. This time, each post should be from a different user.

Short 19-1 Run and modify a server-side script

In this exercise, you'll use Node.js to run an existing server-side script that reads a text file. Then, you'll modify this script so it works better. Estimated time: 15 to 30 minutes.

```
Script: C:\murach\javascript_jquery\exercises_short\ch19\email\read
Filename: email_list2.txt
=====
mike@murach.com (Mike Murach)
judy@murach.com (Judy Taylor)
```

1. If necessary, install Node.js on your system.
2. Open the application in this directory:
`exercises_short\ch19\email_list\`
3. Review the text files named `email_list.txt` and `email_list2.txt`. Note that the first contains three lines of text and the second contains two lines of text.
4. Review the JavaScript file named `read.js`. Note that it uses the `fs.promises` API to read the `email_list.txt` file.
5. Start a command line, change to the `exercises_short\ch19\email` directory, and use the `node` command to run the `read.js` file. It should display the contents of the text file named `email_list.txt` on the command line.
6. In the `read.js` file, edit the code so it gets the filename from an argument that's passed to the script.
7. Run the `read` script again but pass it an argument for the filename that you want to read. Now, you should be able to use the `read` script to read either text file.
8. In the `read.js` file, edit the code so it displays the path to the script that's being executed as well as the filename of the text file that's being read as shown above.
9. Run the `read` script again and make sure that it works correctly.