

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.

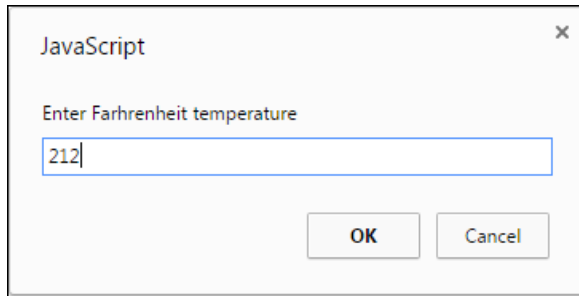
Guidelines for doing the extra exercises	2
Extra 2-1 Convert Fahrenheit to Celsius	3
Extra 3-1 Enhance the Fahrenheit to Celsius application	4
Extra 3-2 Convert number grades to letter grades	5
Extra 3-3 Create a Sum of Numbers application	6
Extra 3-4 Use a Sales array	7
Extra 4-1 Develop the Sales Tax Calculator	9
Extra 4-2 Develop the Change Calculator	10
Extra 4-3 Develop the Income Tax Calculator	11
Extra 5-1 Develop the Temperature Converter	12
Extra 5-2 Use a Test Score array	14
Extra 5-3 Modify the FAQs app	16
Extra 7-1 Develop the Change Calculator application	17
Extra 7-2 Develop the Calendar application	18
Extra 8-1 Develop an Odds and Evens game	19
Extra 8-2 Develop a password generator	21
Extra 9-1 Save a reservation in session storage	22
Extra 9-2 Develop the Student Scores application	24
Extra 10-1 Convert the Countdown application to functions	25
Extra 10-2 Develop the Monthly Balance Calculator	26
Extra 11-1 Use an object literal with the Change Calculator	28
Extra 11-2 Use a constructor with the Change Calculator	29
Extra 11-3 Use a factory function with the Change Calculator	30
Extra 11-4 Convert the PIG app to objects	31
Extra 12-1 Adjust a regular expression pattern	33
Extra 12-2 Add validation to the Reservation application	34
Extra 13-1 Develop the Clock application	35
Extra 13-2 Add a stopwatch to the Clock app	36
Extra 14-1 Convert the Clock app to closures	37
Extra 14-2 Convert the Clock app to callbacks	38
Extra 15-1 Use namespaces and module pattern with the Clock	39
Extra 15-2 Enhance the Clock app's stopwatch	40
Extra 16-1 Develop a Contact List app that uses JSON	41
Extra 16-2 Use replacers and revivers with the Contact List	42
Extra 17-1 Develop a jQuery plugin that provides table styles	43
Extra 17-2 Enhance the jQuery plugin with options	44

Guidelines for doing the extra exercises

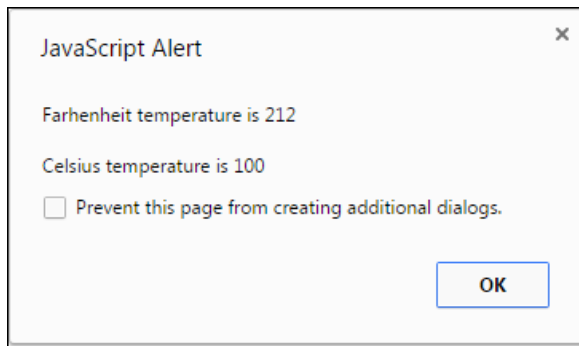
- For all the extra exercises, you will start with the HTML and CSS for the user interface. Then, you supply the JavaScript that's required to get the desired results.
- Unless an exercise specifies that you need to modify the HTML or CSS, you won't have to do that.
- Do the exercise steps in sequence.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

Extra 2-1 Convert Fahrenheit to Celsius

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



The alert dialog box should look like this:

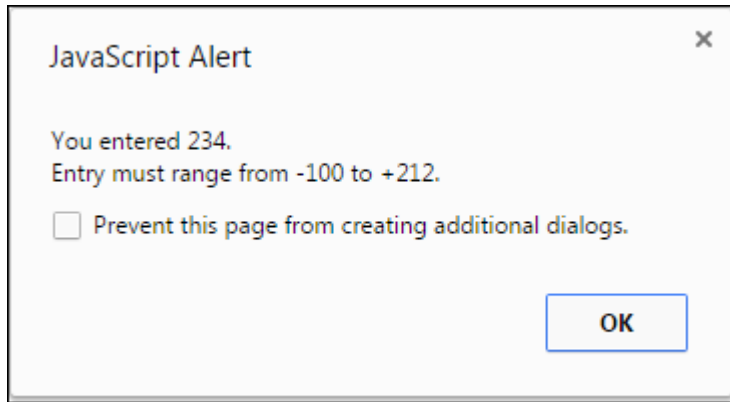


To convert Fahrenheit to Celsius, first subtract 32 from the Fahrenheit temperature. Then, multiply that result by 5/9.

1. Open this file:
`exercises_extra\ch02\convert_temps.html`
2. Review the script element in the head section and note that it's empty. You'll write the code for this application within this element.
3. Develop this application.

Extra 3-1 Enhance the Fahrenheit to Celsius application

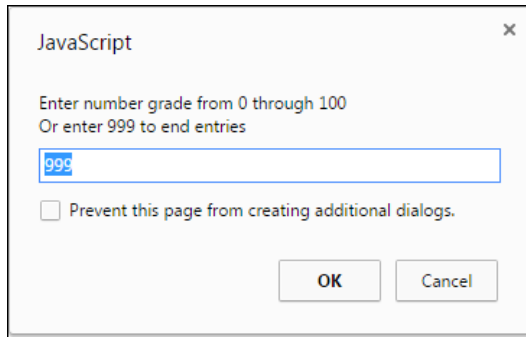
In this exercise, you'll add data validation to the application you created in extra exercise 2-1. You'll also let the user do multiple conversions before ending the application. This is the dialog box for an invalid entry:



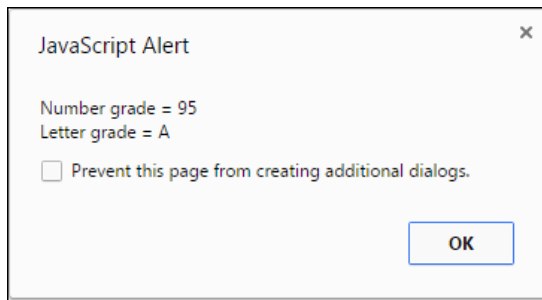
1. If you didn't already do extra exercise 2-1, do it now.
2. Add data validation to the application so it won't do the conversion until the user enters a Fahrenheit temperature between -100 and 212. If the entry is invalid, a dialog box like the one above should be displayed.
3. Add a loop to the code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

Extra 3-2 Convert number grades to letter grades

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



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



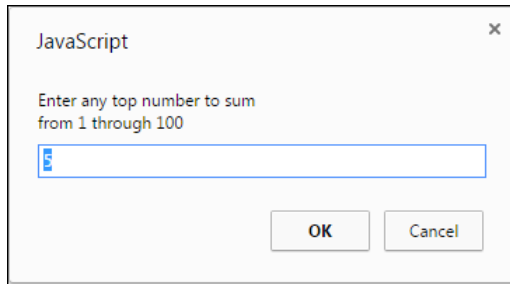
To derive the letter grade, you should use this table:

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

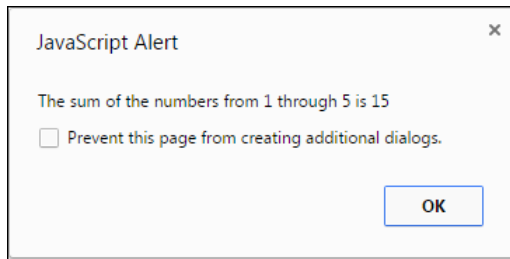
1. Open this HTML file:
`exercises_extra\ch03\letter_grade.html`
2. In the script element, add the JavaScript code for getting the user's entry while the entry amount isn't 999. This should provide for multiple entries and conversions.
3. Add the JavaScript code for deriving the letter grade from the table above and displaying it in an alert dialog box.
4. If you haven't already done so, add data validation to make sure the entry is a valid number from 0 through 100. If the entry is invalid, the application should just display the starting prompt dialog box. It doesn't need to display a special error message.

Extra 3-3 Create a Sum of Numbers application

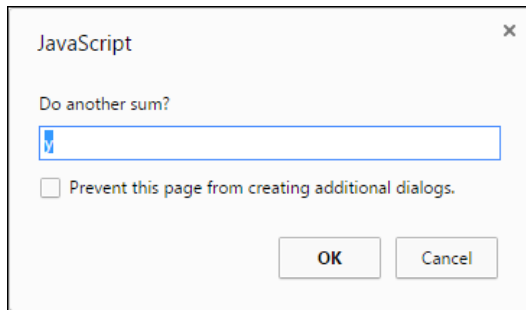
This application will give you a chance to use a for loop. It asks the user to enter a number from 1 through 100 with this prompt dialog box:



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



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

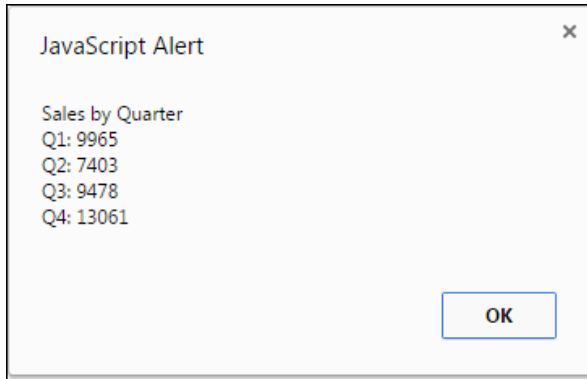


1. Open this HTML file:
`exercises_extra\ch03\sum_numbers.html`
2. In the script element, add a do-while loop that prompts the user for an entry from 1 through 100. If the entry is invalid, display an alert box with this message: "Please enter a number between 1 and 100". Then, continue the loop until the entry is valid.
3. After the do-while loop, code a for loop that sums the numbers, and then display the second dialog box above. For instance, the sum for an entry of four $1 + 2 + 3 + 4$.
4. Add a do-while loop around all of the code that uses the third dialog box above to determine whether the first dialog box should be displayed again so the user can enter another number. The application should end for any entry other than "y".

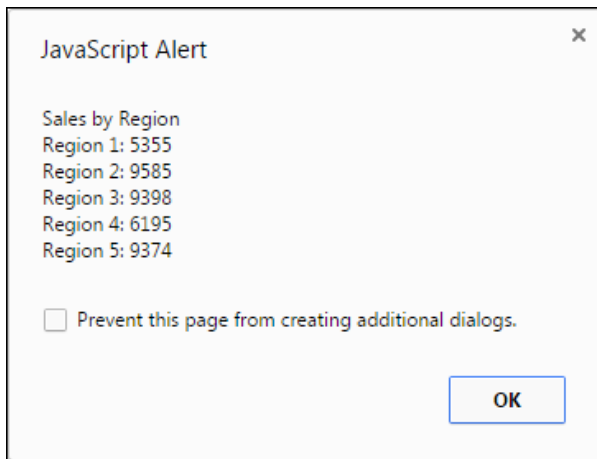
Extra 3-4 Use a Sales array

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

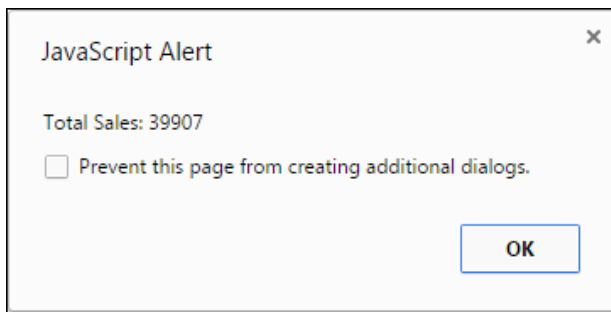
The first one displays sales by quarter:



The second one displays sales by region:



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



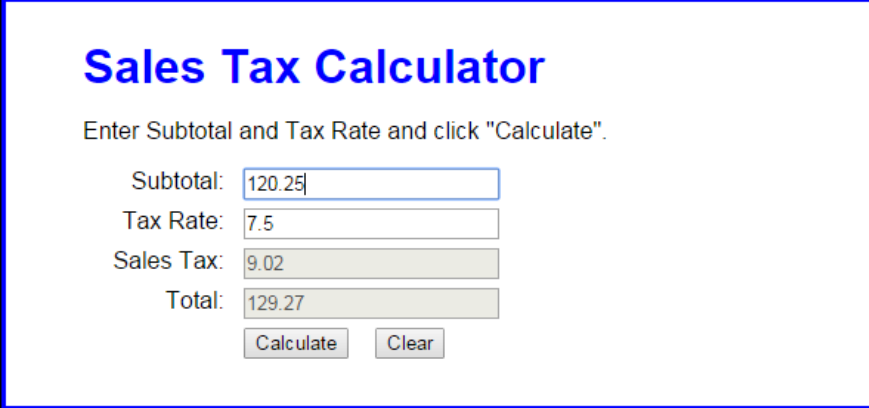
1. Open the HTML and JavaScript files in this folder:

`exercises_extra\ch03\sales_array\`

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

Extra 4-1 Develop the Sales Tax Calculator

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



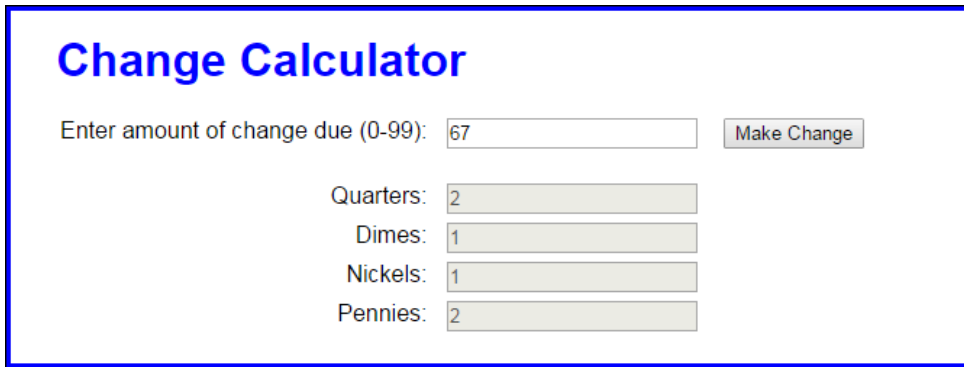
The screenshot shows a web form titled "Sales Tax Calculator" in blue text. Below the title is a instruction: "Enter Subtotal and Tax Rate and click 'Calculate'." The form contains four input fields: "Subtotal:" with the value "120.25", "Tax Rate:" with the value "7.5", "Sales Tax:" with the value "9.02",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

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch04\sales_tax\`
Then, run the application to see the user interface shown above, although that interface won't do anything until you develop the JavaScript for it.
2. In the JavaScript file, note that the `$` function has been coded for you. It gets the object for the HTML element that's specified by the `id` attribute.
3. Code an event handler (function) named `processEntries` that gets the user entries, calculates the sales tax and total, and displays those results in the text boxes.
4. Code an `onload` event handler that attaches the `processEntries` function to the click event of the Calculate button. Then, test what you have so far.
5. Add data validation to the `processEntries` function. The subtotal entry should be a valid, positive number that's less than 10,000. The tax rate should be a valid, positive number that's less than 12. The error messages should be displayed in the `span` elements to the right of the input text boxes, and the error messages should be:

`Must be > 0 and < 10000`
`Must be > 0 and < 12`
6. Add JavaScript that moves the cursor to the Subtotal field when the application starts and when the user clicks on the Calculate button.
7. Add the JavaScript event handler for the click event of the Clear button. This should clear all text boxes, restore the starting messages shown above to the right of the input text boxes, and move the cursor to the Subtotal field.
8. Add JavaScript event handlers for the click events of the Subtotal and Tax Rate text boxes. Each handler should clear the data from the text box.

Extra 4-2 Develop the Change Calculator

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

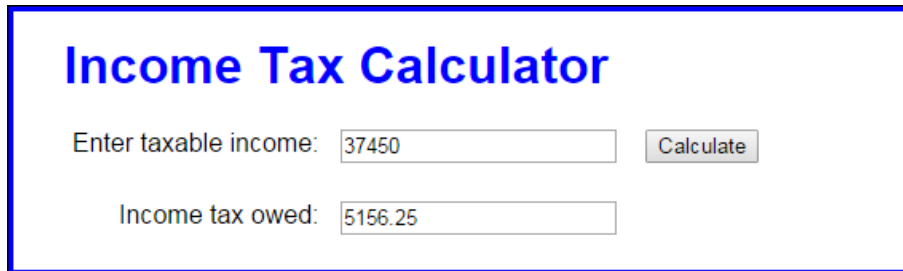


The screenshot shows a web form titled "Change Calculator" in blue text. Below the title, there is a label "Enter amount of change due (0-99):" followed by a text input field containing the number "67". To the right of the input field is a button labeled "Make Change". Below this, there are four rows 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 HTML and JavaScript files in this folder:
`exercises_extra\ch04\change_maker\`
Then, run the application to see the user interface shown above, although that interface won't do anything until you develop the JavaScript for it.
2. In the JavaScript file, note that the `$` function has already been coded.
3. Code an event handler named `processEntries` that gets the user's entry and checks to make sure that it is a number between 0 and 99. If it isn't, display an alert dialog box for the error. If it is valid, call a function named `makeChange` and pass it the user's entry.
4. Code the `makeChange` function, which should have one parameter that accepts the user's entry. This function shouldn't return anything, but it should display the results in the text boxes for Quarters, Dimes, Nickels, and Pennies.
5. Code an `onload` event handler that attaches the `processEntries` 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 income tax that is owed for a taxable income amount. When a user enters his taxable income and clicks the Calculate button, this application will display the federal income tax owed.



This is the 2015 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	\$9,225	\$0 plus 10%	\$0
\$9,225	\$37,450	\$922.50 plus 15%	\$9,225
\$37,450	\$90,750	\$5,156.25 plus 25%	\$37,450
\$90,750	\$189,300	\$18,481.25 plus 28%	\$90,750
\$189,300	\$411,500	\$46,075.25 plus 33%	\$189,300
\$411,500	\$413,200	\$119,401.25 plus 35%	\$411,500
\$413,200		\$119,996.25 plus 39.6%	\$413,200

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

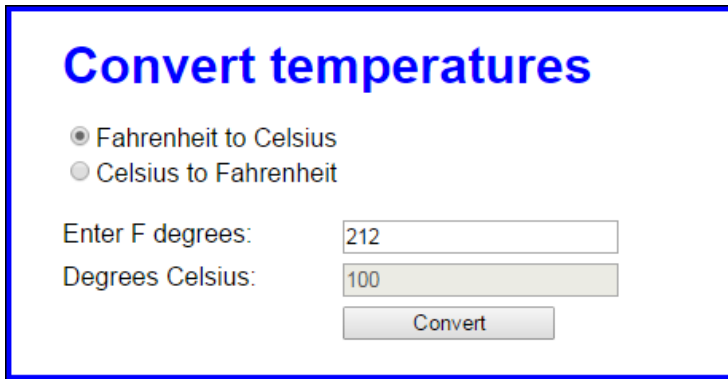
`exercises_extra\ch04\income_tax\`

Note that the JavaScript file has some starting JavaScript code for this application, including the \$ function and an onload event handler that attaches a function named processEntry to the click event of the Calculate button.

2. Code the processEntry function. It should get the user's entry and make sure it's a valid number. If it isn't, it should display an error message. If it is valid, it should pass the value to a function named calculateTax, which should return the tax amount. That amount should then be displayed in the second text box.
3. Code the calculateTax function, but to start just write the code for calculating the tax for any amount within the first two brackets in the table above. The tax should be rounded to two decimal places, and it should be returned to the calling function. To test this, use income values of 9225 and 37450, which should display taxable amounts of 922.50 and 5156.25.
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.

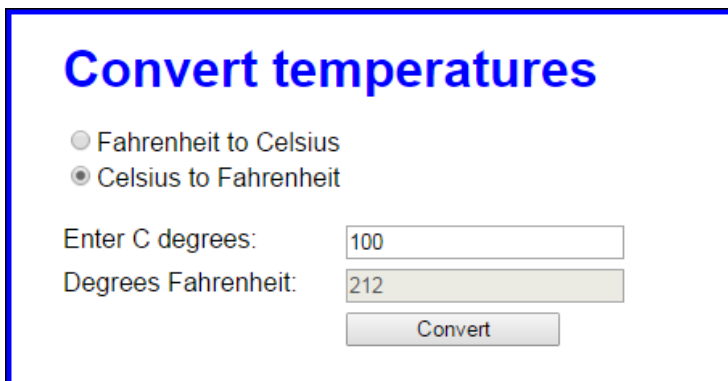
Extra 5-1 Develop the Temperature Converter

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



The screenshot shows a web form titled "Convert temperatures" in blue. It contains two radio buttons: "Fahrenheit to Celsius" (selected) and "Celsius to Fahrenheit". Below the radio buttons are two text input fields. The first field is labeled "Enter F degrees:" and contains the value "212". The second field is labeled "Degrees Celsius:" and contains the value "100". A "Convert" button is located below the second field.

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



The screenshot shows the same web form titled "Convert temperatures". The radio buttons are now "Celsius to Fahrenheit" (selected) and "Fahrenheit to Celsius". The first text input field is now labeled "Enter C degrees:" and contains the value "100". The second text input field is now labeled "Degrees Fahrenheit:" and contains the value "212". The "Convert" button remains below the second field.

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch05\convert_temps\`
2. Note that the JavaScript file has some starting JavaScript code, including the `$` function, a `clearTextBoxes` function, and an `onload` event handler that attaches three event handlers named `convertTemp`, `toCelsius`, and `toFahrenheit`.
3. Code the `toFahrenheit` function that is executed when the user clicks on the second radio button. It should change the text in the labels for the text boxes so they read as in the second interface above. It should also call the `clearTextBoxes` function to clear the text boxes.
4. Code the `toCelsius` function that is executed when the user clicks on the first radio button. It should change the text in the labels for the text boxes so they read as in the first interface above. It should also call the `clearTextBoxes` function to clear the text boxes.

13 Extra exercises for *Murach's JavaScript (2nd Edition)*

5. Code the `convertTemp` function without any data validation. It should calculate the temperature based on which button is checked. To convert Fahrenheit to Celsius, first subtract 32 from the Fahrenheit temperature, and then multiply that result by 5/9. To convert Celsius to Fahrenheit, first multiply Celsius by 9/5, and then add 32. The result in either case should be rounded to zero decimal places.
6. Add data validation to the `convertTemp` function. The only test is whether the entry is a valid number. If it isn't, this message should be displayed in a dialog box: "You must enter a valid number for degrees."
7. Add any finishing touches to the application like moving the focus to the first text box whenever that's appropriate.

Extra 5-2 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.

Use a Test Score array

Name:

Score:

Results

Average score = 90
High score = Mike with a score of 99

Scores

Name	Score
Ben	88
Joel	98
Judy	77
Anne	88
Mike	99

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

`exercises_extra\ch05\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.

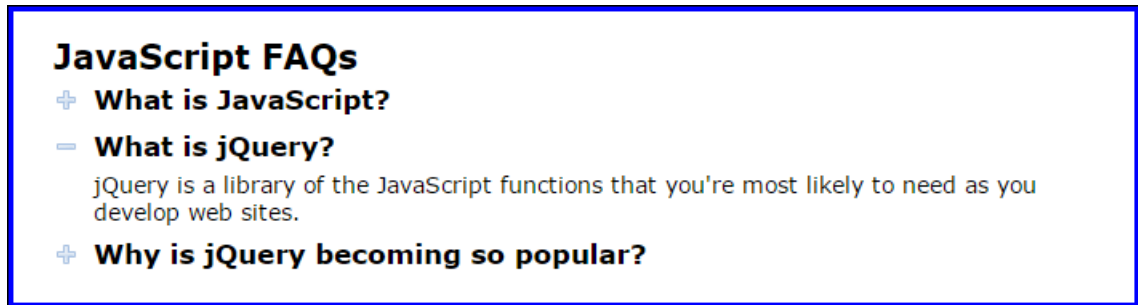
2. As the start of the JavaScript file, you'll see the declarations for two arrays: one for names and one for scores, and each array contains four elements. You'll also see the code for the `$` function as well as an `onload` event handler that attaches three functions `addScore`, `displayResults`, and `displayScores` to the click events of the buttons.
3. Write the `displayResults` function. It should derive the average score and the highest score from the arrays and then display the results in the `div` element with "results" as its `id`, as shown above. To display the results, you need to add nodes to the DOM with the heading as an `h2` element and the average and highest scores as `<p>` elements. The easiest way to do that is to use the `innerHTML` property as shown in figure 5-14.
4. Write the `displayScores` function. It should get the names and scores from the arrays and display them as rows in the HTML table element with "scores_table" as its `id`, as shown above.

15 Extra exercises for *Murach's JavaScript (2nd Edition)*

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

Extra 5-3 Modify the FAQs app

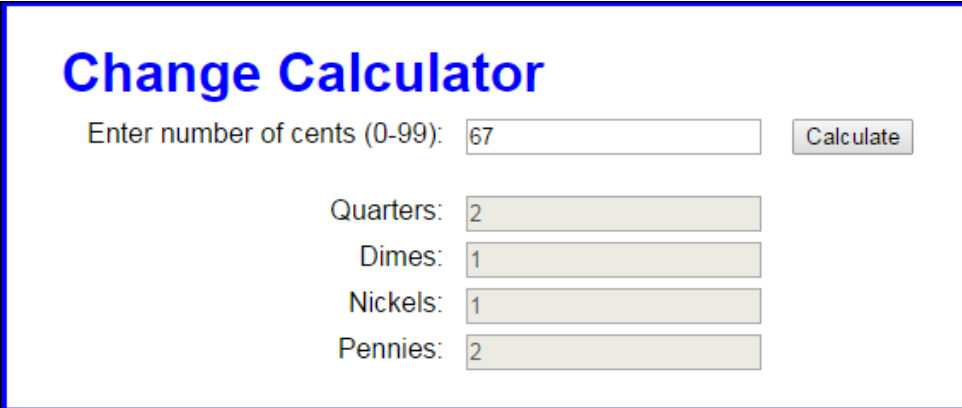
This exercise has you make a minor modification to the FAQs application. When you're done, this application should work the same as before, except that only one answer can be displayed at a time. In other words, when the user clicks on a heading to display the answer, the other answers must be hidden.



1. Open the HTML and JavaScript files in this folder:
`exercises_short\ch05\faqs\`
Then, run the application to refresh your memory about how it works.
2. Add code to the toggle function so only one answer can be displayed at a time. To do that, create an array of the h2 elements. Then, use a for loop to go through the h2 elements in the array and remove the class attribute for all h2 elements that aren't the one that has been clicked. You also need to remove the class attributes for all of the div siblings of the h2 elements that weren't clicked.

Extra 7-1 Develop the Change Calculator application

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



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 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 HTML and JavaScript files in this folder:
`exercises_extra\ch07\change_calculator\`
2. In the JavaScript file, note that three functions are supplied. The \$ function. The start of a calculateChange function. And an onload event handler that attaches the calculateChange function to the click event of the Calculate button.
3. In the calculateChange function, 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 box with this message: "Please enter a valid number between 0 and 99".
4. If the number entered by the user is valid, write code to calculate the number of coins needed for the cents entered by the user. Start with the quarters and work your way down to the pennies. Use the Math.floor method to round your results to the next lower integer whenever needed. And use the number of cents remaining from the last calculation as the starting point for the next calculation.
5. Display the number for each coin in the corresponding text box. Be sure to display whole numbers.

Extra 7-2 Develop the Calendar application

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



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 HTML, CSS, and JavaScript files in this folder:
`exercises_extra\ch07\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 rule set 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 that four functions are supplied. The `$` function. A `getMonthText` function that accepts the number for a month and returns the month name in text. The start of a `getLastDayOfMonth` function. And the start of the `onload` event handler.
5. Write the code for the `getLastDayOfMonth` function. It should use the number passed in the `currentMonth` parameter to calculate and return the last day of the current month. See figure 7-11 for ideas on how to code this.
6. In the `onload` 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 `onload` 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, add them to the `innerHTML` property of the calendar table, but remember that the new rows have to go after the row that's in the HTML.

Extra 8-1 Develop an Odds and Evens game

In this exercise, you'll develop a player-vs-computer game that tallies odd and even numbers until there's a winner. The interface looks like this:

Let's Play ODDS AND EVENS!

Rules

- The computer is even, you are odd (no offense).
- Each round, you and the computer "hold up" 1 to 5 fingers.
- The total number of fingers are tallied.
- Even totals go to the computer.
- Odd totals go to you.
- First to 50 wins.

Play

This round

You: Computer:

Totals

You: Computer:

After the Play button is clicked, the user will see a series of prompts, like this:

Enter a number between 1 and 5, or 999 to quit

OK Cancel

The game ends when either the player or the computer reaches 50 or when the user enters 999 to quit. In either case, the application should display an appropriate message like: "You WIN!", "You lose :(", or "You quit" in the span element to the right of the Play button.

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch08\odds_evens\`
2. In the JavaScript file, note that five functions are supplied. The \$ function. The getRandomNumber function. A resetFields function that resets the text boxes and message area. The start of a playGame function. And an onload event handler that attaches the playGame function to the click event of the Play button.
3. Code and test the playGame function. Within this function, use a while loop to get user entries until the player quits or either the player or computer has reached 50. Then, display the appropriate message to the right of the Play

button. Within the while loop, consider using a break statement to end the loop if the player quits.

4. Within the playGame function, call the resetFields function whenever the fields need to be reset.
5. If you haven't already done so, add data validation to make sure the entry is a valid number from 0 through 5.

Extra 8-2 Develop a password generator

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



Generate a strong password

Number of characters:

Password:

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch08\password\`
2. In the JavaScript file, note that five functions are supplied. The \$ function. The getRandomNumber function. The start of a generatePassword function. A clearFields function that resets the text boxes and moves the focus to the first text box. And an onload event handler that attaches the generatePassword and clearFields functions to the click events of the related buttons.
3. In the generatePassword function, get the value entered by the user and make sure it's a number. If it isn't, display an alert box with this message: "Please enter a valid number".
4. If the number entered by the user is valid, code a for loop that iterates that number of times. In each iteration of the loop, randomly select one of the characters from the chars variable and concatenate it to the password variable.
5. When the loop is finished, display the password in the password textbox.

Extra 9-1 Save a reservation in session storage

In this exercise, you'll develop an application that stores data in session storage. The interface looks like this:

Reservation Request

General Information
Arrival date:
Nights:
Adults:
Children:

Preferences
Room type: ☐ Standard ☒ Business ☐ Suite
Bed type: ☒ King ☐ Double Double
☐ Smoking

Contact Information
Name:
Email:
Phone:

Submit Reservation

Then, when you click on the Submit Reservation button, a new page gets the data from session storage and displays it like this:

The following reservation has been submitted

Name: Mary
Phone: 555-555-5555
Email: mary@yahoo.com
Arrival Date: 12/2/2015
Nights: 3
Adults: 2
Children: 0
Room Type: business
Bed Type: king
Smoking: no

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch09\reservation\`
2. In the index.html file, note the coding for the form element. It uses the get method to submit the data to the response.html page. Also, at the bottom of

the form element, a button element (not a submit button) is used for the button that submits the form.

3. In the response.html file, note that there is an embedded script tag within the main element in the body of the document, and it contains document.write methods that get the data that's submitted from session storage. This is the page that's called when the form is submitted.
4. In the JavaScript file, note that three functions are supplied. The \$ function. The start of a saveReservation function that ends by submitting the form. And an onload event handler that attaches the saveReservation function to the click event of the Submit Reservation button and sets the focus on the first textbox on the page.
5. Run this application without entering any data and click the submit button to see how the response.html file will look when no data has been saved to session storage.
6. In the saveReservation function of the JavaScript file, get the values from the controls on the page and store them in session storage using the same key names as the response.html file uses. But don't bother doing any data validation because that isn't the point of this exercise.

Extra 9-2 Develop the Student Scores application

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

The screenshot shows a web application titled "Student Scores". It features a form with three input fields: "First Name:" with the value "Alan", "Last Name:" with the value "Turing", and "Score:" with the value "96". Below these fields is a blue button labeled "Add Student Score".

Below the form is a section titled "Student Scores" with a horizontal line. Underneath is a text area containing the following text:

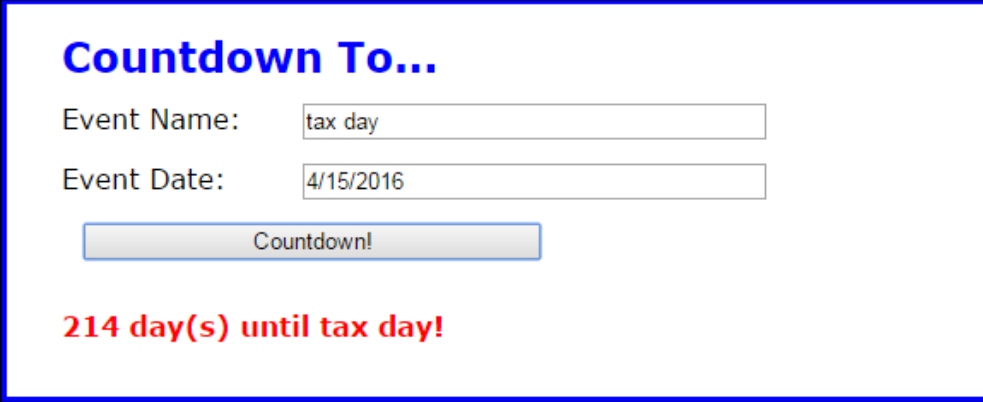
```
Hopper, Grace : 98
Babbage, Charles: 95
Lovelace, Ada: 97
```

Below the text area is a text box labeled "Average score:" containing the value "96.7". At the bottom are two buttons: "Clear Student Scores" and "Sort By Last Name".

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch09\scores\`
2. In the JavaScript file, note that six functions are supplied. The \$ function. The start of a displayScores function. The start of an addScore function that ends by clearing the add form and setting the focus on its first field. The start of a clearScores function that ends by clearing the display area and setting the focus on the first name field. The start of a sortScores function. And an onload event handler that attaches the addScore, clearScores, and sortScores functions to the click events of the appropriate buttons and sets the focus on the first name field.
3. To start, code two global arrays, one to hold the score values and the other to hold the 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 stores it in the text box below the text area. Then, add the code that gets the students' names and scores in the second array and displays it in the text area.
5. In the addScore function, 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 box) in the second array. Then, call the displayScores function to redisplay the updated data.
6. In the clearScores function, add code that clears both global arrays.
7. In the sortScores function, add code that sorts the students by last name and then re-displays the score information.

Extra 10-1 Convert the Countdown application to functions

In this exercise, you'll change the Countdown application of chapter 7 so it uses functions.



The screenshot shows a web form titled "Countdown To...". It contains two input fields: "Event Name:" with the value "tax day" and "Event Date:" with the value "4/15/2016". Below these fields is a button labeled "Countdown!". At the bottom of the form, a red message reads "214 day(s) until tax day!".

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch10\countdown\`
Note that there are two JavaScript files for this application: the main JavaScript file (countdown.js) and the start of a library file (library_countdown.js).
2. In the countdown.js file, note that three functions are supplied. The \$ function. The calculateDays function that contains all of the code for the application. And an onload event handler that attaches the calculateDays function to the click event of the Countdown button and sets the focus on the first field. This is the original code for this chapter 7 application.
3. In the library_countdown.js, note that two functions are supplied. The clearMessage function clears the message from the node that's passed to it. The hasNoError function returns true or false depending on whether the node that's passed to it is set to an empty space.
4. In the index.html file, add the script tag for this new library file. Be sure the script tags are in the correct order so the countdown.js file can use the functions in the library_countdown.js file.
5. Add the other functions that are needed for this application to the library file. To do that, move the code from the main JavaScript file to the library and adjust as needed. When you're through, the library should include separate functions for (1) making sure both entries have been made, (2) testing the validity of just the date entry, (3) calculating the number of days until the event, and (4) displaying the number of days until the event.
6. Modify the countdown.js file so it uses the functions in the library to get the correct results.

Extra 10-2 Develop the Monthly Balance Calculator

In this exercise, you'll develop an application that allows you to enter deposits and withdrawals for a bank account. The application displays all the transactions, along with a running balance. The interface looks like this:

Monthly Balance Calculator

Add Transaction

Date:

Type:

Amount:

Transactions

Date	Amount	Balance
		0
Dec 01 2015	1000	1000
Dec 05 2015	(100)	900
Dec 12 2015	(200)	700
Dec 15 2015	150	850

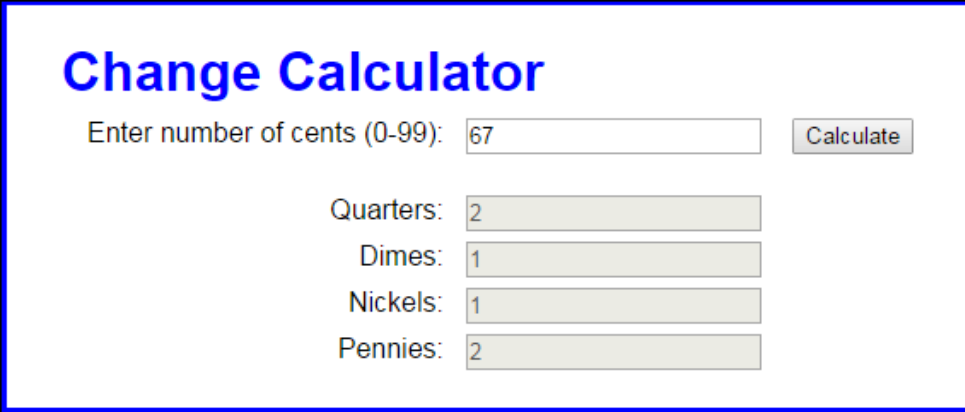
1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch10\balance\`
Notice that there are two JavaScript files, one with a prefix of "library_".
2. In the `balance.js` file, note that four functions are supplied. The `$` function. The `updateDisplay` and `add` functions that contain the code for the application. And an `onload` event handler that attaches the `add` function to the click event of the Add Transaction button and calls the `updateDisplay` function.
3. In the `library_balance.js` file, note that an empty array and the start of three functions are supplied.
4. Review the `updateDisplay` function in the `balance.js` file. Note that it calls the `getTransaction` function of the library file twice. In the first call, no argument is passed and the total number of transactions is returned. In the second call, the value of the loop's index is passed and a specific transaction is returned. Now, code the `getTransaction` function in the library file.
5. Note that the `updateDisplay` function also calls the `calculateBalance` function of the library file. This code passes the values it receives from the `getTransaction`

function, and the current balance is returned. Now, code the `calculateBalance` function in the library file.

6. Review the `add` function in the `balance.js` file. Note that it calls the `addTransaction` function of the library file, passes it values entered by the user, and doesn't receive any value in return. Also note that this function can accept variable numbers of arguments, depending on whether the user entered a value for the date. Now, code the `addTransaction` function in the library file. Assume that it will use today's date if it isn't passed a date.

Extra 11-1 Use an object literal with the Change Calculator

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



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". The entire form is enclosed in a blue border.

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

`exercises_extra\ch11\change_literal\`

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

2. In the `calculate.js` file, note that three functions are supplied. The `$` function. The `calculateChange` function that contains all of the code for the application. And an `onload` event handler that attaches this function to the click event of the Calculate button and sets the focus on the first field.
3. In the `library_coin.js`, note that just the strict declaration has been provided.
4. In the `index.html` file, add the script tag for the library file.
5. In the library file, code an object literal named `coins` that has a `cents` property and two methods:

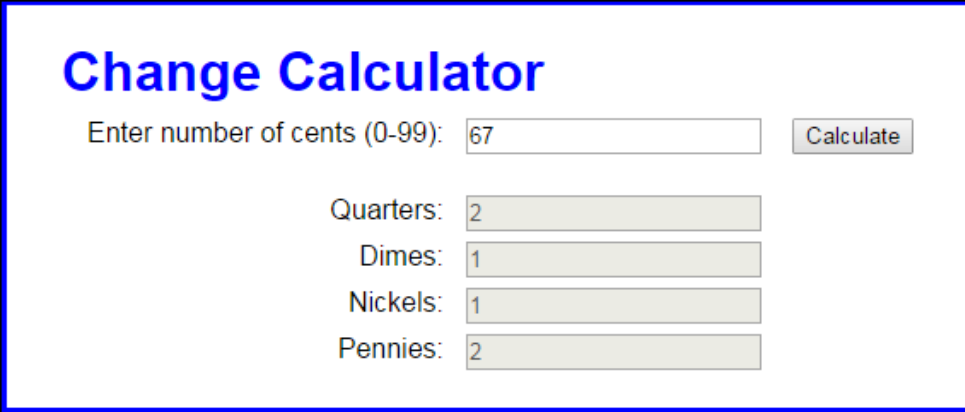
The `isValid` method should determine whether the `cents` property is valid.

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

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

Extra 11-2 Use a constructor with the Change Calculator

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



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". The entire form is enclosed in a blue border.

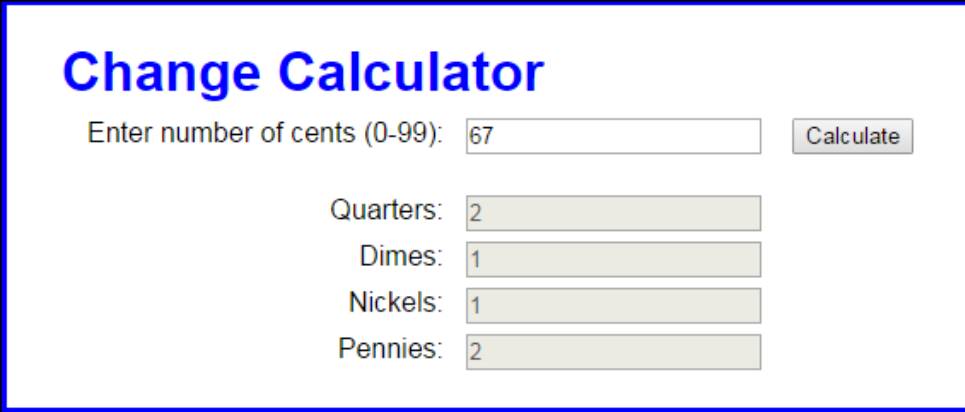
1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch11\change_constructor\`
Note that there are two JavaScript files for this application: the main JavaScript file (`calculate.js`) and the start of a library file (`library_coin.js`).
2. In the `calculate.js` file, note that three functions are supplied. The `$` function. The `calculateChange` function that contains all of the code for the application. And an `onload` event handler that attaches this function to the click event of the Calculate button and sets the focus on the first field.
3. In the `library_coin.js`, note that just the strict declaration has been provided.
4. In the `index.html` file, add the script tag for the library file.
5. In the library file, code a constructor function named `Coins` that accepts a parameter named `cents`, which is the number of cents entered by the user. Then, code a `cents` property and two methods:

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

The `getNumber` method should accept a divisor parameter (like 25 for quarters), calculate the number of coins of that type that are required, update the `cents` property with the remaining cents, and return the number of coins.
6. Change the code in the `calculate.js` file to create an instance of the `Coins` object type to validate the user's entry and calculate the number of coins.

Extra 11-3 Use a factory function with the Change Calculator

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



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". The entire form is enclosed in a blue border.

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

`exercises_extra\ch11\change_factory\`

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

2. In the `calculate.js` file, note that three functions are supplied. The `$` function. The `calculateChange` function that contains all of the code for the application. And an `onload` event handler that attaches this function to the click event of the Calculate button and sets the focus on the first field.
3. In the `library_coin.js`, note that just the strict declaration has been provided.
4. In the `index.html` file, add the script tag for the library file.
5. In the library file, code a factory function named `calculateCoins` that accepts a parameter named `cents`, which is the number of cents entered by the user. Then, use the `create` method of the `Object` object to create an object that has a `cents` property and two methods:

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

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

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

Extra 11-4 Convert the PIG app to objects

In this exercise, you'll finish an objects version of the PIG application of chapter 7.

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 (9 and 11 respectively). A "New Game" button is located to the right of the Player 2 score. Below this, a red text label indicates "Mary 's turn". At the bottom, there are "Roll" and "Hold" buttons, followed by "Die" (5) and "Total" (9) fields.

Let's Play PIG!

Rules

- First player to 100 wins.
- Players take turns rolling the die.
- Turn ends when player rolls a 1 or chooses to hold.
- If player rolls a 1, they lose all points earned during the turn.
- If player holds, points earned during the turn are added to their total.

Player 1 Score

Player 2 Score

Mary 's turn

Die Total

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

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch11\pig\`
Notice that there are three JavaScript library files besides the main `dice.js` file.
2. Review the `library_die.js` file, and note that it contains a constructor function named `Die` with no properties and a single method that rolls a die.
3. Review the `library_pig.js` file, and note that it contains a constructor named `Pig` with three properties. This constructor's prototype inherits the one method of the `Die` object in the `library_die.js` file. It also provides three methods: `takeTurn`, `hold`, and `reset`.
4. Review the `library_game.js` file, and note that it's an object literal with three properties and eight methods. The `load` and `isValid` methods are already coded for you, and the remaining methods have comments indicating what they should do. Now, finish the code for those six methods and be sure to make them cascading methods whenever that's possible.
5. Review the `dice.js` file and note that five functions are supplied. The `$` function and the `onload` event handler are already coded for you, but you need to provide

the code for the other functions. These functions will of course use the methods of the game object in the `library_game.js` file. Note that the last statement in the `onload` event handler calls the `load` method of the game object and passes it the HTML objects that it needs.

Extra 12-1 Adjust a regular expression pattern

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



Validate phone number

Phone Number:

Valid phone number

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch12\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 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 three functions are supplied. The \$ function. The validatePhone function that contains the validation code. And an onload event handler that attaches the validatePhone function to the click event of the Validate button.
4. Change the regular expression pattern in the pattern variable so the phone number can contain an optional “1-“ prefix. The best way to do this is to copy the pattern variable to a new line and then comment out the original. This way, you can refer to the original pattern as you adjust it.
5. When the validation in step 4 is working correctly, change the new pattern so that 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 new 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 12-2 Add validation to the Reservation application

In this exercise, you'll add data validation to a Reservation application. Here's how the interface looks when the submit button is clicked and there's bad data in the form:

The screenshot shows a web form titled "Reservation Request". It is divided into three sections: "General Information", "Preferences", and "Contact Information".

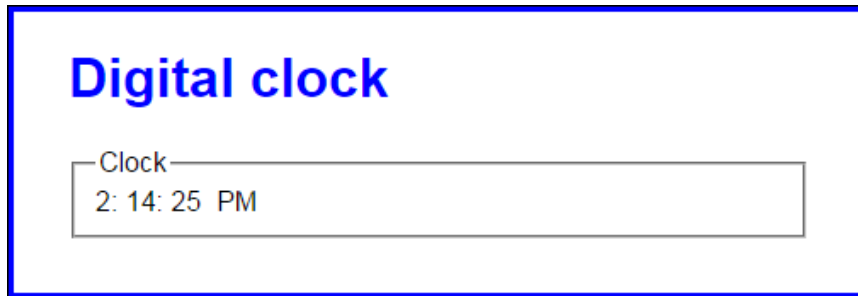
- General Information:** Contains four input fields. "Arrival date:" has the value "12/39/2016" and a red error message "Arrival date is invalid." to its right. "Nights:" has the value "three" and a red error message "Nights must be a number." to its right. "Adults:" is a dropdown menu with "1" selected. "Children:" is a dropdown menu with "0" selected.
- Preferences:** Contains three rows of radio buttons. "Room type:" has options "Standard" (selected), "Business", and "Suite". "Bed type:" has options "King" (selected) and "Double Double". There is also an unchecked checkbox for "Smoking".
- Contact Information:** Contains three input fields. "Name:" is empty and has a red error message "Name is required." to its right. "Email:" has the value "graceyahoo.com" and a red error message "Email address is invalid." to its right. "Phone:" has the value "123-456-789" and a red error message "Phone number is invalid." to its right.

At the bottom of the form are two buttons: "Submit Reservation" and "Clear".

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch12\reservation\`
Then, run the application to see the user interface shown above. Click on the Submit Reservation button and notice it lets you submit anything you want.
2. In the index.html file, notice that there are now span tags next to the text boxes, and the spans have the same name as the text boxes, with "_error" appended.
3. Also notice that there are script tags for three library files. Open each of these files to see that they all contain some starter code. Refer to the Register application in chapter 12 to complete the code for these files. Note that the date validation code in the Register application validates dates in "mm/yyyy" format, so you'll want to change that. Also, you might want to add checks for dates like 13/22/2016 or 12/39/2015.
4. In the main JavaScript file, add code to the saveReservation function that validates the form before storing and submitting the data.
5. Test the application to make sure the data validation is working correctly.

Extra 13-1 Develop the Clock application

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



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

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch13\clock\`
2. In the JavaScript file, note that four functions are supplied. The \$ function. The start of a `displayCurrentTime` function. The `padSingleDigit` function that adds a leading zero to single digits. And the start of an `onload` event handler.
3. In the `displayCurrentTime` function, add code that uses the `Date` object to determine the current hour, minute, and second. Convert these values to a 12 hour clock, determine the AM/PM value, and display these values in the appropriate span tags.
4. In the `onload` event handler, code a timer that calls the `displayCurrentTime` function at 1 second intervals. Also, make sure that the current time shows as soon as the page loads.

Extra 13-2 Add a stopwatch to the Clock app

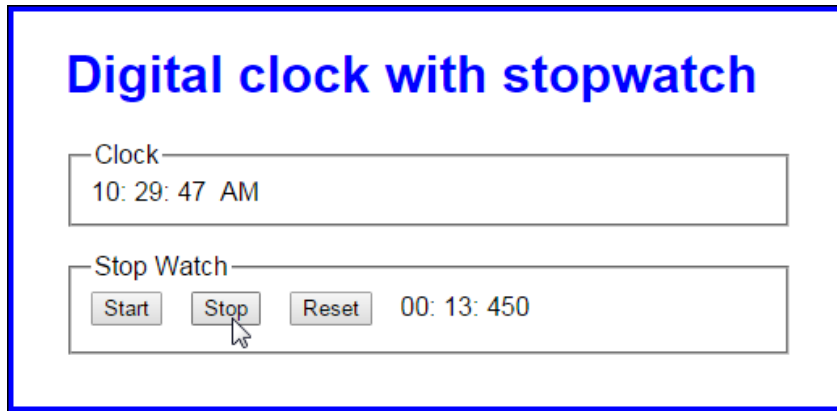
In this exercise, you'll add a stopwatch feature to the application you created in extra exercise 13-1. The stopwatch will display elapsed minutes, seconds, and milliseconds. The enhanced application looks like this:



1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch13\clock_stopwatch\`
2. Note that the folder contains an event library file with the start of an object named "evt". Using the event library from the FAQs application in chapter 13 as a guide, code the attach, detach, and preventDefault methods of the event library.
3. In the JavaScript file, note the \$, displayCurrentTime, padSingleDigit, and onload event handler functions from the Clock application. In addition, a global stopwatchTimer variable, a global elapsed object, and starts for the tickStopwatch, startStopwatch, stopStopwatch, and resetStopwatch functions are supplied.
4. In the tickStopwatch function, add code that increments the values in the elapsed object by 10 milliseconds. Then, add code that displays the result in the appropriate span tags in the page.
5. In the startStopwatch function, add code that starts the stopwatch. Be sure to cancel the default action of the link too.
6. 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 link too.
7. In the onload event handler, attach the stopwatch event handlers to the appropriate links.

Extra 14-1 Convert the Clock app to closures

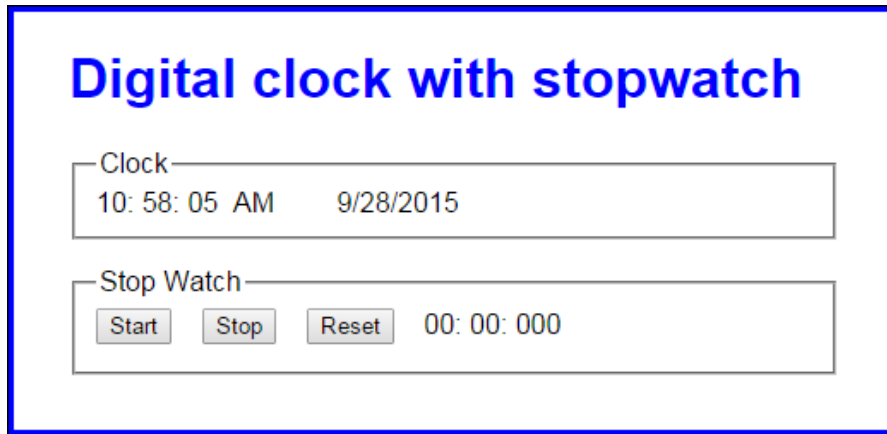
In this exercise, you'll convert the Clock application from extra exercise 13-2 to use closures. The enhanced application looks like this, but note that the application now uses buttons instead of links:



1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch14\clock_closures\`
2. Note that the folder contains two library files named `library_clock.js` and `library_stopwatch.js`. In the JavaScript file, you'll see the variables, objects, and functions that provide all the functionality for the clock and stopwatch, but without using the libraries. Also note that the event library isn't used in this version of the application.
3. In the `library_clock.js` file, there's a start for a function called `createClock`. Note that this function has parameters for the span tags that display the clock in the page. In the `library_stopwatch.js` file, there's a start for a function called `createStopwatch`. Note that this function has parameters for the span tags that display the stopwatch in the page.
4. 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 library file, code and return an object that contains a method named `start` that used the private state to start the clock. Adjust as needed to make this work.
5. In the `clock.js` file, find the variables, objects, and functions that run the stopwatch and move them to the private 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` that use the private state to start, stop, and reset the stopwatch. Adjust as needed to make this work.
6. Still in the `clock.js` file, re-write the remaining code so the `onload` 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 14-2 Convert the Clock app to callbacks

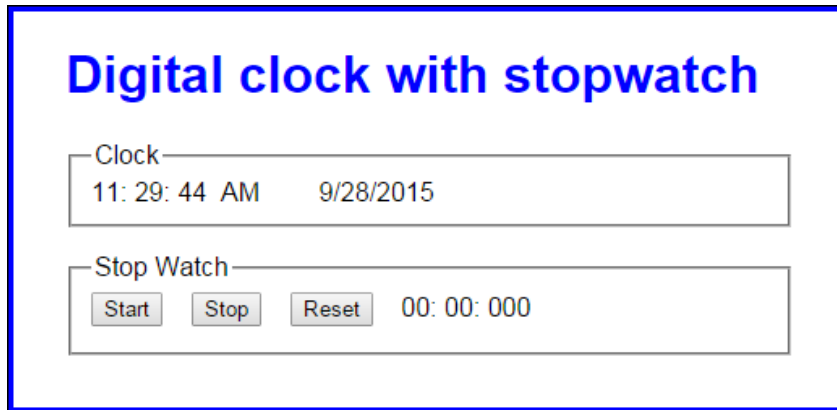
In this exercise, you'll convert the Clock application from extra exercise 14-1 to use callback functions, and add a feature that displays the date as well as the time. The enhanced application looks like this:



1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch14\clock_callbacks\`
2. In the `library_clock.js` file, note that the `createClock` function accepts a single callback function as a parameter instead of parameters for the span tags that display the clock in the page. Also note that the `displayCurrentTime` private function adds properties to the instance of the `Date` object it creates. In this private function, add code that uses the callback function to display the current time.
3. In the `library_stopwatch.js` file, note that the `createStopwatch` function accepts a single callback function as a parameter, instead of parameters for the span tags that display the stopwatch in the page. In the `tickStopwatch` private function, add code that uses the callback function to display the current elapsed time.
4. In the `library_stopwatch.js` file, note that the public `reset` method accepts a callback function. Add code to this method that uses the callback function to reset the stopwatch display.
5. In the `clock.js` file, note the `padSingleDigit` function and three callback functions named `displayTime`, `displayTick`, and `resetStopWatch`. In the `onload` event handler, pass the appropriate callback functions in the function or method calls that require them. Now, test this application.
6. In the `displayTime` callback function, add code that displays the current date in the span tag whose id is "date". Note how using a callback function lets you change the clock's display without changing any of the code in the clock library file.

Extra 15-1 Use namespaces and module pattern with the Clock

In this exercise, you'll convert the Clock application from extra exercise 14-2 to use namespaces and the module pattern. The enhanced application looks the same as in the last exercise:



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

exercises_extra\ch15\clock_namespaces

Notice that there are two more library files, a namespace library and a utility library. In the HTML file, note that the script tags for these libraries come before the script tags for the clock and stopwatch libraries.

2. In the namespace library, review the code that creates the namespace and adds a namespace creator method. Then, use that method to add the following namespaces:

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

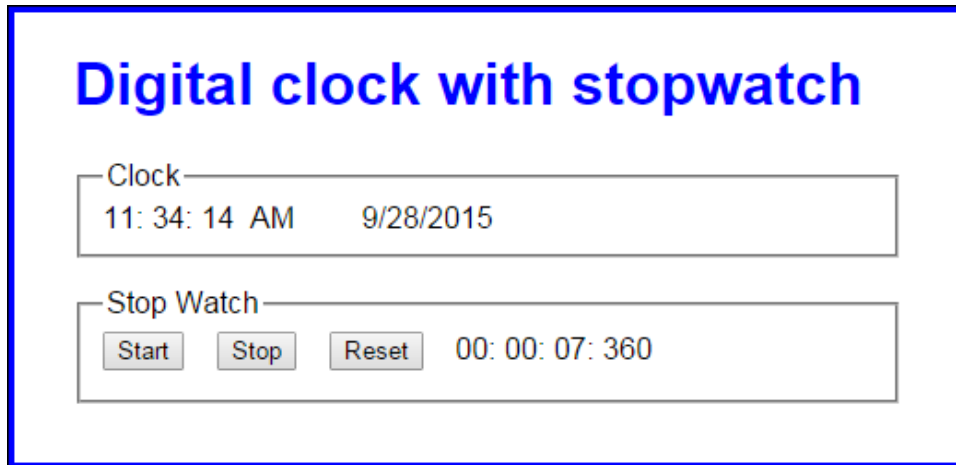
3. Move the padSingleDigit function and the \$ function from the clock.js file to the library_utility.js file, and add them to the utility namespace.
4. Change the library_clock.js file so it uses the module pattern and adds the object it creates to the clock namespace.

NOTE: In this step and in step 5, you'll need to adjust the objects so the callback function is passed to the start method and then stored so the function called by the timer can use it.

5. Change the library_stopwatch.js file so it uses the module pattern and adds the object it creates to the stopwatch namespace.
6. Change the code in the clock.js file to use the new namespaces and module objects. Since you no longer create the clock and stopwatch objects, remove that code and pass the callbacks to the start methods. Use aliases for the namespaces, and make sure you don't add anything to the global namespace.

Extra 15-2 Enhance the Clock app's stopwatch

In this exercise, you'll enhance the stopwatch module from extra exercise 15-1 to calculate the elapsed hours, and you'll change the Clock application to show the elapsed hours. The enhanced application looks like this:



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

`exercises_extra\ch15\clock_augment\`

Notice that there's a new library file named `library_stopwatch_augment.js`. In the HTML file, note that the script tag for this new library comes after the script tag for the stopwatch library. Also note that there's a new span tag for the hours value of the stopwatch with a default value of "00".

2. Change the `library_stopwatch.js` file so the module object has a read-only property that exposes the values of the private elapsed object. NOTE: Because elapsed is an object, if you return it directly, its properties can be changed even if the property that returns it is non-configurable. Thus, you'll need to return an object that contains copies of the values in the elapsed object's properties.
3. In the `library_stopwatch_augment.js` file, add code that augments the stopwatch module by adding a method named `getElapsedTimeWithHours`. Within this new method, get the object from the read-only property from step 2, add an hours property to it, calculate the number of elapsed hours, adjust the number of elapsed minutes, and return the adjusted object.
4. Change the code in the `clock.js` file so the `displayTick` callback function uses the `getElapsedTimeWithHours` function from step 3 to display the hours in the `s_hours` span tag. Be sure to reset this span tag in the `resetStopwatch` callback function.

Extra 16-1 Develop a Contact List app that uses JSON

In this exercise, you'll develop a Contact List application that uses the JSON object to convert objects to strings and back again for storage in local storage. The application looks like this:

And this is the JSON string that's stored in local storage:

```
[{"f":"Grace","l":"Hopper","o":"UNIVAC","p":"555-262-6500","e":""},
{"f":"Alan","l":"Turing","o":"","p":"","e":"alan@bletchleypark.uk"}]
```

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch16\contact_list\`
2. Review the code in the `contact_list.js` file. Note that it uses a storage object and Contact objects to add and display contacts.
3. In the `library_storage.js` file, add code to the `getContacts` method that converts a string to a JavaScript object and returns it, or returns an empty array if the string is null.
4. Still in the `library_storage.js` file, add code to the `setContacts` method that converts a JavaScript object to a string.
5. In the `library_contact.js` file, review the code for the Contact constructor function and its methods. Then, add code to the `toJSON` method that shortens the property names (you can refer to the JSON string above to see what the short names should be).
6. Still in the `library_contact.js` file, add code to the `loadJsonObject` method that uses the object with short names to populate the Contact object's properties.
7. Run and test the application. Notice that it allows you to store a phone number with dashes, slashes, plus signs, etc., but it displays the phone numbers exactly as you enter them.

Extra 16-2 Use replacers and revivers with the Contact List

In this exercise, you'll enhance the Contact List application from extra exercise 16-1 to use replacer and revive functions with the JSON object to control how the phone number value is stored and displayed. The application looks the same as before:

My Contacts

First Name:

Last Name:

Organization:

Phone:

Email:

Grace Hopper, UNIVAC
Phone: 555-262-6500
Email: _____

Alan Turing,
Phone: _____
Email: alan@bletchleypark.uk

But now the JSON string stores the phone number with number only, even though the application still displays it with dashes:

```
[{"f":"Grace","l":"Hopper","o":"UNIVAC","p":"5552626500","e":""},
{"f":"Alan","l":"Turing","o":"","p":"","e":"alan@bletchleypark.uk"}]
```

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch16\contact_list_enhanced\`
2. In the `library_storage.js` file, add code to the `getContacts` method that creates a function named `reviver`. Code the `reviver` method so it adds dashes back to regular phone numbers (7 characters), phone numbers with area codes (10 characters), and phone numbers with 1 + area code (11 characters). Then, use the `reviver` function with the `parse` method.
3. Still in the `library_storage.js` file, add code to the `setContacts` method that creates a function named `replacer`. Code the `replacer` function so it strips all non-numeric characters from the phone number value. Then, use the `replacer` function with the `stringify` method.
4. Run and test the application. It should strip all non-numeric characters from phone numbers for storage, and put dashes back in phone numbers that have an appropriate number of characters.

Extra 17-1 Develop a jQuery plugin that provides table styles

In this exercise, you'll create a jQuery plugin that provides styling for the header row and alternating rows of an HTML table using CSS classes. An HTML table with the CSS classes of the plugin applied to it looks like this:

Important People in Computer Science

First Name	Last Name	Date of Birth	Accomplishment
Charles	Babbage	12/26/1791	Originated the concept of a programmable computer, invented the first mechanical computer.
Ada	Lovelace	12/10/1815	First computer programmer. Wrote an algorithm for Babbage's mechanical computer.
Alan	Turing	6/23/1912	Invented the Turing Machine, a hypothetical device that's a model of a general purpose computer.
Grace	Hopper	12/9/1906	Invented the first compiler for a computer programming language, popularized the term "debugging" for fixing computer glitches.

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

`exercises_extra\ch17\alt_row\`

In the HTML file, note the script tags for the jQuery library and the plugin file (`jquery.altrow.js`).

2. In the `jquery.altrow.js` file, code a plugin that uses the `getElementsByName` method to get all of an element's "tr" child elements. Then, it should apply a class named "header" to the header row, and classes named "even" and "odd" on an alternating basis to the rest of the rows. Note: you can check whether a row contains "th" elements to see if it's a header row.
3. In the `altrow.js` file, add code to the `onload` event that calls the plugin for the table element. Run the application and see that the row styles are applied to the table.

Extra 17-2 Enhance the jQuery plugin with options

In this exercise, you'll enhance the jQuery plugin from extra exercise 17-1 to allow options. Specifically, you'll allow a user to specify the name of the CSS class to use for the header row, the even rows, and the odd rows. An HTML table with an alternate header CSS class applied to it looks like this:

First Name	Last Name	Date of Birth	Accomplishment
Charles	Babbage	12/26/1791	Originated the concept of a programmable computer, invented the first mechanical computer.
Ada	Lovelace	12/10/1815	First computer programmer. Wrote an algorithm for Babbage's mechanical computer.
Alan	Turing	6/23/1912	Invented the Turing Machine, a hypothetical device that's a model of a general purpose computer.
Grace	Hopper	12/9/1906	Invented the first compiler for a computer programming language, popularized the term "debugging" for fixing computer glitches.

1. Open the HTML and JavaScript files in this folder:
`exercises_extra\ch17\alt_row_options\`
2. In the `jquery.altrow.js` file, enhance the plugin so the user can specify the names of the CSS classes to be used for the header row and for the odd and even rows.
3. In the `altrow.css` file, note that rule sets have been added for classes named "alt_header", "alt_odd", and "alt_even".
4. In the `altrow.js` file, code the call to the jQuery plugin so it uses just one of the new CSS classes to modify the formatting of the table. The other rows should still use the default formatting. Then, test to make sure that all three options work.