A
Answers to Exercises

This appendix provides the answers to the questions you find at the end of each chapter in this book.

CHAPTER 2

Exercise 1 Question

Write a JavaScript program to convert degrees centigrade into degrees Fahrenheit, and to write the result to the page in a descriptive sentence. The JavaScript equation for Fahrenheit to centigrade is as follows:

```
degFahren = 9 / 5 * degCent + 32
```

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 2: Question 1</title>
</head>
<body>
    <script>
        var degCent = prompt("Enter the degrees in centigrade", 0);
        var degFahren = 9 / 5 * degCent + 32;

        document.write(degCent + " degrees centigrade is " + degFahren +
           " degrees Fahrenheit");
    </script>
</body>
</html>
```

Save this as `ch2_question1.html`.

Exercise 2 Question

The following code uses the `prompt()` function to get two numbers from the user. It then adds those two numbers and writes the result to the page:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 2, Question 2</title>
</head>
<body>

<script>
    var firstNumber = prompt("Enter the first number","");
    var secondNumber = prompt("Enter the second number","");
    var theTotal = firstNumber + secondNumber;

    document.write(firstNumber + " added to " + secondNumber +
        " equals " + theTotal);
</script>
</body>
</html>
```

However, if you try out the code, you'll discover that it doesn't work. Why not? Change the code so that it does work.

Exercise 2 Solution

The data that the `prompt()` actually obtains is a string. So both `firstNumber` and `secondNumber` contain text that happens to be number characters. When you use the + symbol to add the two variables together, JavaScript assumes that because it's string data, you must want to concatenate the two and not sum them.

To make it explicit to JavaScript that you want to add the numbers, you need to convert the data to numbers using the `parseFloat()` function:

```
var firstNumber = parseFloat(prompt("Enter the first number",""));
var secondNumber = parseFloat(prompt("Enter the second number",""));
var theTotal = firstNumber + secondNumber;

document.write(firstNumber + " added to " + secondNumber + " equals " +
   theTotal);
```

Save this as `ch2_question2.html`.

Now the data returned by the `prompt()` function is converted to a floating-point number before being stored in the `firstNumber` and `secondNumber` variables. Then, when you do the addition that is stored in `theTotal`, JavaScript makes the correct assumption that, because both the variables are numbers, you must mean to add them up and not concatenate them.

The general rule is that where you have expressions with only numerical data, the + operator means "do addition." If there is any string data, the + means concatenate.

CHAPTER 3

Exercise 1 Question

A junior programmer comes to you with some code that appears not to work. Can you spot where he went wrong? Give him a hand and correct the mistakes.

```
var userAge = prompt("Please enter your age");

if (userAge = 0) {;
    alert("So you're a baby!");
} else if ( userAge < 0 | userAge > 200)
    alert("I think you may be lying about your age");
else {
   alert("That's a good age");
}
```

Exercise 1 Solution

Oh dear, our junior programmer is having a bad day! There are two mistakes on the line:

```
if (userAge = 0) {;
```

First, he has only one equals sign instead of two in the `if`'s condition, which means `userAge` will be assigned the value of `0` rather than `userAge` being compared to `0`. The second fault is the semicolon at the end of the line—statements such as `if` and loops such as `for` and `while` don't require semicolons. The general rule is that if the statement has an associated block (that is, code in curly braces), no semicolon is needed. So the line should be:

```
if (userAge == 0) {
```

The next fault is with these lines:

```
else if ( userAge < 0 | userAge > 200)
   alert("I think you may be lying about your age");
else {
```

The junior programmer's condition is asking if `userAge` is less than `0` OR `userAge` is greater than `200`. The correct operator for a boolean OR is `||`, but the programmer has only used one `|`.

Exercise 2 Question

Using `document.write()`, write code that displays the results of the 12 times table. Its output should be the results of the calculations.

```
12 * 1 = 12
12 * 2 = 24
12 * 3 = 36
...
12 * 11 = 132
12 * 12 = 144
```

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 3: Question 2</title>
</head>
<body>
    <script>
        var timesTable = 12;

        for (var timesBy = 1; timesBy < 13; timesBy++) {
            document.write(timesTable + " * " +
                           timesBy + " = " +
                           timesBy * timesTable + "<br />");
        }
    </script>
</body>
</html>
```

Save this as `ch3_question2.html`.

You use a `for` loop to calculate from `1 * 12` up to `12 * 12`. The results are written to the page with `document.write()`. What's important to note here is the effect of the order of precedence; the concatenation operator (the `+`) has a lower order of precedence than the multiplication operator, `*`. This means that the `timesBy * timesTable` is done before the concatenation, which is the result you want. If this were not the case, you'd have to put the calculation in parentheses to raise its order of precedence.

CHAPTER 4

Exercise 1 Question

Change the code of Question 2 from Chapter 3 so that it's a function that takes as parameters the times table required and the values at which it should start and end. For example, you might try the four times table displayed starting with `4 * 4` and ending at `4 * 9`.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 4: Question 1</title>
</head>
<body>
    <script>
        function writeTimesTable(timesTable, timesByStart, timesByEnd) {
            for (; timesByStart <= timesByEnd; timesByStart++) {
                document.write(timesTable + " * " + timesByStart + " = " +
                    timesByStart * timesTable + "<br />");
            }
        }

        writeTimesTable(4, 4, 9);
    </script>
</body>
</html>
```

Save this as `ch4_question1.html`.

You've declared your function, calling it `writeTimesTable()`, and given it three parameters. The first is the times table you want to write, the second is the start point, and the third is the number it should go up to.

You've modified your `for` loop. First you don't need to initialize any variables, so the initialization part is left blank—you still need to put a semicolon in, but there's no code before it. The `for` loop continues while the `timesByStart` parameter is less than or equal to the `timesByEnd` parameter. You can see that, as with a variable, you can modify parameters—in this case, `timesByStart` is incremented by one for each iteration through the loop.

The code to display the times table is much the same. For the function's code to be executed, you now actually need to call it, which you do in the line:

```
writeTimesTable(4, 4, 9);
```

This will write the 4 times table starting at 4 times 4 and ending at 9 times 4.

Exercise 2 Question

Modify the code of Question 1 to request the times table to be displayed from the user; the code should continue to request and display times tables until the user enters `-1`. Additionally, do a check to make sure that the user is entering a valid number; if the number is not valid, ask the user to re-enter it.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 4: Question 2</title>
</head>
<body>
    <script>
        function writeTimesTable(timesTable, timesByStart, timesByEnd) {
            for (; timesByStart <= timesByEnd; timesByStart++) {
                document.write(timesTable + " * " + timesByStart + " = " +
                    timesByStart * timesTable + "<br />");
            }
        }

        var timesTable;

        while ((timesTable = prompt("Enter the times table", -1)) != -1) {
            while (isNaN(timesTable) == true) {
                timesTable = prompt(timesTable + " is not a " +
                                    "valid number, please retry", -1);
            }

            if (timesTable == -1) {
                break;
            }

            document.write("<br />The " + timesTable +
                           " times table<br />");
            writeTimesTable(timesTable, 1, 12);
        }
    </script>
</body>
</html>
```

Save this as `ch4_question2.html`.

The function remains the same, so let's look at the new code. The first change from Question 1 is that you declare a variable, `timesTable`, and then initialize it in the condition of the first `while` loop. This may seem like a strange thing to do at first, but it does work. The code in parentheses inside the `while` loop's condition:

```
(timesTable = prompt("Enter the times table",-1))
```

is executed first because its order of precedence has been raised by the parentheses. This will return a value, and it is this value that is compared to `-1`. If it's not `-1`, then the `while` condition is `true`, and the body of the loop executes. Otherwise it's skipped over, and nothing else happens in this page.

In a second `while` loop nested inside the first, you check to see that the value the user has entered is actually a number using the function `isNaN()`. If it's not, you prompt the user to try again, and this will continue until a valid number is entered.

If the user had entered an invalid value initially, then in the second `while` loop, that user may have entered `-1`, so following the `while` is an `if` statement that checks to see if `-1` has been entered. If it has, you break out of the `while` loop; otherwise the `writeTimesTable()` function is called.

CHAPTER 5

Exercise 1 Question

Using the `Date` type, calculate the date 12 months from now and write this into a web page.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 5: Question 1</title>
</head>
<body>
    <script>
        var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
                      "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"];

        var nowDate = new Date();

        nowDate.setMonth(nowDate.getMonth() + 12);
        document.write("Date 12 months ahead is " + nowDate.getDate());
        document.write(" " + months[nowDate.getMonth()]);
        document.write(" " + nowDate.getFullYear());
    </script>
</body>
</html>
```

Save this as `ch5_question1.html`.

Because the `getMonth()` method returns a number between `0` and `11` for the month rather than its name, an array called `months` has been created that stores the name of each month. You can use `getMonth()` to get the array index for the correct month name.

The variable `nowDate` is initialized to a new `Date` object. Because no initial value is specified, the new `Date` object will contain today's date.

To add 12 months to the current date you simply use `setMonth()`. You get the current month value with `getMonth()`, and then add 12 to it.

Finally you write the result out to the page.

Exercise 2 Question

Obtain a list of names from the user, storing each name entered in an array. Keep getting another name until the user enters nothing. Sort the names in ascending order and then write them out to the page, with each name on its own line.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 5: Question 2</title>
</head>
<body>
    <script>
```

```
        var inputName = "";
        var namesArray = [];

        while ((inputName = prompt("Enter a name", "")) != "") {
            namesArray[namesArray.length] = inputName;
        }

        namesArray.sort();

        var namesList = namesArray.join("<br/>");
        document.write(namesList);
    </script>
</body>
</html>
```

Save this as ch5_question2.html.

First you declare two variables: inputName, which holds the name entered by the user, and namesArray, which holds an Array object that stores each of the names entered.

You use a while loop to keep getting another name from the user as long as the user hasn't left the prompt box blank. Note that the use of parentheses in the while condition is essential. By placing the following code inside parentheses, you ensure that this is executed first and that a name is obtained from the user and stored in the inputName variable:

```
 (inputName = prompt("Enter a name",""))
```

Then you compare the value returned inside the parentheses—whatever was entered by the user—with an empty string (denoted by ""). If they are not equal—that is, if the user did enter a value, you loop around again.

Now, to sort the array into order, you use the sort() method of the Array object:

```
namesArray.sort();
```

Finally, to create a string containing all values contained in the array elements with each being on a new line, you use the HTML <br/> element and write the following:

```
var namesList = namesArray.join("<br/>")
document.write(namesList);
```

The code namesArray.join("<br/>") creates the string where a <br/> is between each element in the array. Finally, you write the string into the page with document.write().

Exercise 3 Question

Example 8 uses a function to create objects using literal notation. Modify this example to use the Person data type.

Exercise 3 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 5, Question 3</title>
</head>
<body>
    <script>
        function Person(firstName, lastName) {
            this.firstName = firstName;
            this.lastName = lastName;
        }

        Person.prototype.getFullName = function () {
            return this.firstName + " " + this.lastName;
        };

        Person.prototype.greet = function (person) {
            alert("Hello, " + person.getFullName() +
                  ". I'm " + this.getFullName());
        };

        var johnDoe = new Person("John", "Doe");
        var janeDoe = new Person("Jane", "Doe");

        johnDoe.greet(janeDoe);
    </script>
</body>
</html>
```

Save this as `ch5_question3.html`.

This is a simple matter of replacing the `createPerson()` function with the `Person` reference type you defined at the end of Chapter 5.

To create your `Person` objects, you use the `new` operator when calling the `Person` constructor function, passing in the first and last names of the people you want to represent.

CHAPTER 6

Exercise 1 Question

What problem does the following code solve?

```
var myString = "This sentence has has a fault and and we need to fix it."
var myRegExp = /(\b\w+\b) \1/g;
myString = myString.replace(myRegExp,"$1");
```

Now imagine that you change that code, so that you create the `RegExp` object like this:

```
var myRegExp = new RegExp("(\b\w+\b) \1");
```

Why would this not work, and how could you rectify the problem?

Exercise 1 Solution

The problem is that the sentence has "has has" and "and and" inside it, clearly a mistake. A lot of word processors have an autocorrect feature that fixes common mistakes like this, and what your regular expression does is mimic this feature.

So the erroneous `myString`:

"This sentence has has a fault and and we need to fix it."

will become:

"This sentence has a fault and we need to fix it."

Let's look at how the code works, starting with the regular expression:

```
/(\b\w+\b) \1/g;
```

By using parentheses, you have defined a group, so `(\b\w+\b)` is group `1`. This group matches the pattern of a word boundary followed by one or more alphanumeric characters, that is, a–z, A–Z, 0–9, and _, followed by a word boundary. Following the group you have a space then `\1`. What `\1` means is match exactly the same characters as were matched in pattern group 1. So, for example, if group 1 matched "has," then `\1` will match "has" as well. It's important to note that `\1` will match the exact previous match by group 1. So when group 1 then matches the "and," the `\1` now matches "and" and not the "has" that was previously matched.

You use the group again in your `replace()` method; this time the group is specified using the `$` symbol, so `$1` matches group 1. It's this that causes the two matched "has" and "and" to be replaced by just one.

Turning to the second part of the question, how do you need to change the following code so that it works?

```
var myRegExp = new RegExp("(\b\w+\b) \1");
```

Easy; now you are using a string passed to the `RegExp` object's constructor, and you need to use two slashes (\\) rather than one when you mean a regular expression syntax character, like this:

```
var myRegExp = new RegExp("(\\b\\w+\\b) \\1","g");
```

Notice you've also passed a `g` to the second parameter to make it a global match.

Exercise 2 Question

Write a regular expression that finds all of the occurrences of the word "a" in the following sentence and replaces them with "the":

"a dog walked in off a street and ordered a finest beer"

The sentence should become:

“the dog walked in off the street and ordered the finest beer”

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 6: Question 2</title>
</head>
<body>
    <script>
        var myString = "a dog walked in off a street and " +
                       "ordered a finest beer";
        var myRegExp = /\ba\b/gi;

        myString = myString.replace(myRegExp, "the");
        alert(myString);
    </script>
</body>
</html>
```

Save this as `ch6_question2.html`.

With regular expressions, it’s often not just what you want to match, but also what you don’t want to match that is a problem. Here you want to match the letter `a`, so why not just write:

```
var myRegExp = /a/gi;
```

Well, that would work, but it would also replace the “a” in “walked,” which you don’t want. You want to replace the letter “a” but only where it’s a word on its own and not inside another word. So when does a letter become a word? The answer is when it’s between two word boundaries. The word boundary is represented by the regular expression special character `\b` so the regular expression becomes:

```
var myRegExp = /\ba\b/gi;
```

The `gi` at the end ensures a global, case-insensitive search.

Now with your regular expression created, you can use it in the `replace()` method’s first parameter:

```
myString = myString.replace(myRegExp,"the");
```

Exercise 3 Question

Imagine you have a website with a message board. Write a regular expression that would remove barred words. (You can make up your own words!)

Exercise 3 Solution

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
    <title>Chapter 6: Question 3</title>
</head>
<body>
    <script>
        var myRegExp = /(sugar )?candy|choc(olate|oholic)?/gi;
        var myString = "Mmm, I love chocolate, I'm a chocoholic. " +
           "I love candy too, sweet, sugar candy";

        myString = myString.replace(myRegExp,"salad");
        alert(myString);
    </script>
</body>
</html>
```

Save this as `ch6_question3.html`.

For this example, pretend you're creating script for a board on a dieting site where text relating to candy is barred and will be replaced with a much healthier option, salad.

The barred words are

- chocolate
- choc
- chocoholic
- sugar candy
- candy

Let's examine the regular expression to remove the offending words:

1. Start with the two basic words, so to match "choc" or "candy," you use:

```
candy|choc
```

2. Add the matching for "sugar candy." Because the "sugar" bit is optional, you group it by placing it in parentheses and adding the "?" after it. This means match the group zero times or one time:

```
(sugar )?candy|choc
```

3. You need to add the optional "olate" and "oholic" end bits. You add these as a group after the "choc" word and again make the group optional. You can match either of the endings in the group by using the | character:

```
(sugar )?candy|choc(olate|oholic)?/gi
```

4. You then declare it as:

```
var myRegExp = /(sugar )?candy|choc(olate|oholic)?/gi
```

The `gi` at the end means the regular expression will find and replace words on a global, case-insensitive basis.

So, to sum up:

```
/(sugar )?candy|choc(olate|oholic)?/gi
```

Au: close up line space

reads as:

Either match zero or one occurrences of "sugar" followed by "candy." Or alternatively match "choc" followed by either one or zero occurrences of "olate" or match "choc" followed by zero or one occurrence of "oholic."

Finally, the following:

```
myString = myString.replace(myRegExp,"salad");
```

replaces the offending words with "salad" and sets `myString` to the new clean version:

```
"Mmm, I love salad, I'm a salad. I love salad too, sweet, salad."
```

CHAPTER 7

Exercise 1 Question

Create a page that gets the user's date of birth. Then, using that information, tell her on what day of the week she was born.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 7: Question 1</title>
</head>
<body>
    <script>
        var days = ["Sunday", "Monday", "Tuesday", "Wednesday",
                    "Thursday", "Friday", "Saturday"];

        var year = prompt("Enter the four digit year you were born.");
        var month = prompt("Enter your birth month (1 - 12).");
        var date = prompt("Enter the day you were born.");

        var birthDate = new Date(year, month - 1, date);

        alert(days[birthDate.getDay()]);
    </script>
</body>
</html>
```

Save this as `ch7_question1.html`.

The solution is rather simple. You create a new `Date` object based on the year, month, and day entered by the user. Then you get the day of the week using the `Date` object's `getDay()` method. This returns a number, but by defining an array of days of the week to match this number, you can use the value of `getDay()` as the index to your `days` array.

Exercise 2 Question

Create a web page similar to Example 4, but make it display only the hour, minutes, and seconds.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 7, Question 2</title>
</head>
<body>
    <div id="output"></div>
    <script>
        function updateTime() {
            var date = new Date();

            var value = date.getHours() + ":" +
                        date.getMinutes() + ":" +
                        date.getSeconds();

            document.getElementById("output").innerHTML = value;
        }

        setInterval(updateTime, 1000);
    </script>
</body>
</html>
```

Save this as `ch7_question2.html`.

Displaying only the hour, minutes, and seconds is an easy task; it just requires a little extra code. You modify the `updateTime()` function to first create a `Date` object to get the time information from.

```
var date = new Date();
```

Then you build a string in hh:mm:ss format:

```
var value = date.getHours() + ":" +
            date.getMinutes() + ":" +
            date.getSeconds();
```

Finally, you output that string to the page:

```
document.getElementById("output").innerHTML = value;
```

CHAPTER 8

Exercise 1 Question

Create two pages, one called `legacy.html` and the other called `modern.html`. Each page should have a heading telling you what page is loaded. For example:

```
<h2>Welcome to the Legacy page. You need to upgrade!</h2>
```

Using feature detection and the `location` object, send browsers that do not support geolocation to `legacy.html`; send browsers that do support geolocation to `modern.html`.

Exercise 1 Solution

The `modern.html` page is as follows:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 2: Question 1</title>
</head>
<body>
    <h2>Welcome to the Modern page!</h2>
    <script>
        if (!navigator.geolocation) {
            location.replace("legacy.html");
        }
    </script>
</body>
</html>
```

The `legacy.html` page is very similar:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 2: Question 1</title>
</head>
<body>
    <h2>Welcome to the Legacy page. You need to upgrade!</h2>
    <script>
        if (navigator.geolocation) {
            location.replace("modern.html");
        }
    </script>
</body>
</html>
```

These two pages are incredibly simple. Starting with the `legacy.html` page, you check if `navigator.geolocation` is a truthy value:

```
if (navigator.geolocation) {
    location.replace("modern.html");
}
```

If it is, you redirect the user to the `modern.html` page. Note that you use `replace()` rather than `href`, because you don't want the user to be able to click the browser's Back button. This way it's less easy to spot that a new page is being loaded.

The `modern.html` page is almost identical, except that in your `if` statement you check if `navigator.geolocation` is falsey:

```
if (!navigator.geolocation) {
    location.replace("legacy.html");
}
```

If so, you redirect to `legacy.html`.

Exercise 2 Question

Modify Example 3 to display one of the four images randomly. Hint: refer to Chapter 5 and the `Math.random()` method.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 8, Question 2</title>
</head>
<body>
    <img src="" width="200" height="150" alt="My Image" />
    <script>
        function getRandomNumber(min, max) {
            return Math.floor(Math.random() * max) + min;
        }

        var myImages = [
            "usa.gif",
            "canada.gif",
            "jamaica.gif",
            "mexico.gif"
        ];

        var random = getRandomNumber(0, myImages.length);

        document.images[0].src = myImages[random];
    </script>
</body>
</html>
```

Save this as `ch8_question2.html`.

The key to this solution is getting a random number between `0` and the length of the `myImages` array, and writing a function to generate a random number would greatly help with that. So, you write a function called `getRandomNumber()`:

```
function getRandomNumber(min, max) {
    return Math.floor(Math.random() * max) + min;
}
```

It generates a random number within the range of `min` and `max`. The algorithm was copied from Chapter 5.

Now you can use `getRandomNumber()` to generate a number for you, passing `0` as the min and the length of the array as the max:

```
var random = getRandomNumber(0, myImages.length);
```

You then use the random number to get the image:

```
        document.images[0].src = myImages[random];
```

CHAPTER 9

Exercise 1 Question

Here's some HTML code that creates a table. Re-create this table using only JavaScript and the core DOM objects to generate the HTML. Test your code in all browsers available to you to make sure it works in them. Hint: Comment each line as you write it to keep track of where you are in the tree structure, and create a new variable for every element on the page (for example, not just one for each of the TD cells but nine variables).

```
<table>
    <tr>
        <td>Car</td>
        <td>Top Speed</td>
        <td>Price</td>
    </tr>
    <tr>
        <td>Chevrolet</td>
        <td>120mph</td>
        <td>$10,000</td>
    </tr>
    <tr>
        <td>Pontiac</td>
        <td>140mph</td>
        <td>$20,000</td>
    </tr>
</table>
```

Exercise 1 Solution

It seems a rather daunting example, but rather than being difficult, it is just a conjunction of two areas, one building a tree structure and the other navigating the tree structure. You start at the `<body/>` element and create a `<table/>` element. Now you can navigate to the new `<table/>` element you've created and create a new `<tr/>` element and carry on from there. It's a lengthy, repetitious, and tedious process, so that's why it's a good idea to comment your code to keep track of where you are.

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 9: Question 1</title>
</head>
<body>
<script>
var tableElem = document.createElement("table");
var trElem1 = document.createElement("tr");
var trElem2 = document.createElement("tr");
var trElem3 = document.createElement("tr");
var tdElem1 = document.createElement("td");
var tdElem2 = document.createElement("td");
var tdElem3 = document.createElement("td");
var tdElem4 = document.createElement("td");
var tdElem5 = document.createElement("td");
var tdElem6 = document.createElement("td");
var tdElem7 = document.createElement("td");
var tdElem8 = document.createElement("td");
var tdElem9 = document.createElement("td");
var textNodeA1 = document.createTextNode("Car");
var textNodeA2 = document.createTextNode("Top Speed");
var textNodeA3 = document.createTextNode("Price");
var textNodeB1 = document.createTextNode("Chevrolet");
var textNodeB2 = document.createTextNode("120mph");
var textNodeB3 = document.createTextNode("$10,000");
var textNodeC1 = document.createTextNode("Pontiac");
var textNodeC2 = document.createTextNode("140mph");
var textNodeC3 = document.createTextNode("$14,000");

var docNavigate = document.body;  //Starts with body element

docNavigate.appendChild(tableElem);      //Adds the table element
docNavigate = docNavigate.lastChild;     //Moves to the table element
docNavigate.appendChild(trElem1);        //Adds the TR element
docNavigate = docNavigate.firstChild;    //Moves the TR element
docNavigate.appendChild(tdElem1);        //Adds the first TD element in the
                                            // heading
docNavigate.appendChild(tdElem2);        //Adds the second TD element in the
                                            // heading
docNavigate.appendChild(tdElem3);        //Adds the third TD element in the
                                            // heading
docNavigate = docNavigate.firstChild;    //Moves to the first TD element
```

```
docNavigate.appendChild(textNodeA1);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeA2);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeA3);     //Adds the third text node
docNavigate = docNavigate.parentNode;    //Moves back to the TR element
docNavigate = docNavigate.parentNode;    //Moves back to the table element

docNavigate.appendChild(trElem2);        //Adds the second TR element
docNavigate = docNavigate.lastChild;     //Moves to the second TR element
docNavigate.appendChild(tdElem4);        //Adds the TD element
docNavigate.appendChild(tdElem5);        //Adds the TD element
docNavigate.appendChild(tdElem6);        //Adds the TD element
docNavigate = docNavigate.firstChild;    //Moves to the first TD element
docNavigate.appendChild(textNodeB1);     //Adds the first text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeB2);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeB3);     //Adds the third text node
docNavigate = docNavigate.parentNode;    //Moves back to the TR element
docNavigate = docNavigate.parentNode;    //Moves back to the table element

docNavigate.appendChild(trElem3);        //Adds the TR element
docNavigate = docNavigate.lastChild;     //Moves to the TR element
docNavigate.appendChild(tdElem7);        //Adds the TD element
docNavigate.appendChild(tdElem8);        //Adds the TD element
docNavigate.appendChild(tdElem9);        //Adds the TD element
docNavigate = docNavigate.firstChild;    //Moves to the TD element
docNavigate.appendChild(textNodeC1);     //Adds the first text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeC2);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeC3);     //Adds the third text node
</script>

</body>
</html>
```

Save this as `ch9_question1.html`.

Exercise 2 Question

Modify Example 6 so that the amount of pixels moved in either direction is controlled by a global variable. Call it `direction`. Remove the `switchDirection` variable, and change the code to use the new `direction` variable to determine when the animation should change directions.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 9, Question 2</title>
```

```
        <style>
            #divAdvert {
                position: absolute;
                font: 12px Arial;
                top: 4px;
                left: 0px;
            }
        </style>
    </head>
    <body>
        <div id="divAdvert">
            Here is an advertisement.
        </div>

        <script>
            var direction = 2;

            function doAnimation() {
                var divAdvert = document.getElementById("divAdvert");
                var currentLeft = divAdvert.offsetLeft;

                if (currentLeft > 400 || currentLeft < 0) {
                    direction = -direction;
                }

                var newLocation = currentLeft + direction;

                divAdvert.style.left = newLocation + "px";
            }

            setInterval(doAnimation, 10);
        </script>
    </body>
    </html>
```

Save this as `ch9_question2.html`.

This modification sounds complex at first, but it actually simplifies the `doAnimation()` function because one variable is responsible for:

- the amount of pixels moved
- the direction the element is moved

First, you remove the `switchDirection` variable and create a new one called `direction`, initializing it with the value of `2`:

```
var direction = 2;
```

Then inside the `doAnimation()` function, you change the value of `direction` when the `<div/>` element reaches one of its bounds (0 pixels or 400 pixels):

```
if (currentLeft > 400 || currentLeft < 0) {
    direction = -direction;
}
```

The new `direction` value is simple; you simply make `direction` negative. So if `direction` is positive, it becomes negative. If `direction` is negative, it becomes positive (remember: a negative times a negative is a positive).

You then calculate the new left position and change the element's style:

```
var newLocation = currentLeft + direction;

divAdvert.style.left = newLocation + "px";
```

CHAPTER 10

Exercise 1 Question

Add a method to the event utility object called `isOldIE()` that returns a boolean value indicating whether or not the browser is old-IE.

Exercise 1 Solution

```
var evt = {
    addListener: function(obj, type, fn) {
        if (typeof obj.addEventListener != "undefined") {
            obj.addEventListener(type, fn);
        } else {
            obj.attachEvent("on" + type, fn);
        }
    },
    removeListener: function(obj, type, fn) {
        if (typeof obj.removeEventListener != "undefined") {
            obj.removeEventListener(type, fn);
        } else {
            obj.detachEvent("on" + type, fn);
        }
    },
    getTarget: function(e) {
        if (e.target) {
            return e.target;
        }

        return e.srcElement;
    },
    preventDefault : function(e) {
        if (e.preventDefault) {
            e.preventDefault();
        } else {
            e.returnValue = false;
        }
    },
    isOldIE: function() {
        return typeof document.addEventListener == "undefined";
    }
};
```

Save this as `ch10_question1.html`.

You have many ways to determine if the browser is an older version of Internet Explorer. Your author chose to check if `document.addEventListener()` is undefined. IE9+ supports the method, whereas IE8 and below do not.

Exercise 2 Question

Example 15 exhibits some behavior inconsistencies between standards-compliant browsers and old-IE. Remember that the event handlers execute in reverse order in old-IE. Modify this example to use the new `isOldIE()` method so that you can write specific code for old-IE and standards-compliant browsers (hint: you will call the `addListener()` method four times).

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 10: Question 2</title>
</head>
<body>
    <img id="img0" src="usa.gif" />
    <div id="status"></div>

    <script src="ch10_question1.js"></script>
    <script>
        var myImages = [
            "usa.gif",
            "canada.gif",
            "jamaica.gif",
            "mexico.gif"
        ];

        function changeImg(e) {
            var el = evt.getTarget(e);
            var newImgNumber = Math.round(Math.random() * 3);

            while (el.src.indexOf(myImages[newImgNumber]) != -1) {
                newImgNumber = Math.round(Math.random() * 3);
            }

            el.src = myImages[newImgNumber];
        }

        function updateStatus(e) {
            var el = evt.getTarget(e);
            var status = document.getElementById("status");

            status.innerHTML = "The image changed to " + el.src;

            if (el.src.indexOf("mexico") > -1) {
                evt.removeListener(el, "click", changeImg);
                evt.removeListener(el, "click", updateStatus);
            }
        }

        var imgObj = document.getElementById("img0");

        if (evt.isOldIE()) {
            evt.addListener(imgObj, "click", updateStatus);
            evt.addListener(imgObj, "click", changeImg);
        } else {
            evt.addListener(imgObj, "click", changeImg);
            evt.addListener(imgObj, "click", updateStatus);
        }
    </script>
</body>
</html>
```

Save this as `ch10_question2.html`.

The majority of the code is identical to Example 15. The only difference is how you register the event listeners. With your new `isOldIE()` method, you can register the click event listeners in the correct order (for old-IE). For standards-compliant browsers, you register the event listeners in the same order as Example 15.

Exercise 3 Question

Example 17 had you write a cross-browser tab script, but as you probably noticed, it behaves peculiarly. The basic idea is there, but the tabs remain active as you click another tab. Modify the script so that only one tab is active at a time.

Exercise 3 Solution

Example 17 is incomplete because the script doesn't keep track of which tab is active. Probably the simplest way to add state recognition to the script is to add a global variable that keeps track of the tab that was last clicked. This particular solution uses this idea. Changed lines of code are highlighted.

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 10: Question 3</title>
    <style>
        .tabStrip {
            background-color: #E4E2D5;
            padding: 3px;
            height: 22px;
        }

        .tabStrip div {
            float: left;
            font: 14px arial;
```

```
                cursor: pointer;
            }

            .tabStrip-tab {
                padding: 3px;
            }

            .tabStrip-tab-hover {
                border: 1px solid #316AC5;
                background-color: #C1D2EE;
                padding: 2px;
            }

            .tabStrip-tab-click {
                border: 1px solid #facc5a;
                background-color: #f9e391;
                padding: 2px;
            }
        </style>
    </head>
    <body>
        <div class="tabStrip">
            <div data-tab-number="1" class="tabStrip-tab">Tab 1</div>
            <div data-tab-number="2" class="tabStrip-tab">Tab 2</div>
            <div data-tab-number="3" class="tabStrip-tab">Tab 3</div>
        </div>
        <div id="descContainer"></div>

        <script src="ch10_question1.js"></script>
        <script>
            var activeTab = null;

            function handleEvent(e) {
                var target = evt.getTarget(e);

                switch (e.type) {
                case "mouseover":
                    if (target.className == "tabStrip-tab") {
                        target.className = "tabStrip-tab-hover";
                    }
                    break;
                case "mouseout":
                    if (target.className == "tabStrip-tab-hover") {
                        target.className = "tabStrip-tab";
                    }
                    break;
                case "click":
                    if (target.className == "tabStrip-tab-hover") {

                        if (activeTab) {
                            activeTab.className = "tabStrip-tab";
                        }

                        var num = target.getAttribute("data-tab-number");

                        target.className = "tabStrip-tab-click";
                        showDescription(num);
                        **activeTab = target;**
                    }
                    break;
                }
            }

            function showDescription(num) {
                var descContainer = document.getElementById("descContainer");

                var text = "Description for Tab " + num;

                descContainer.innerHTML = text;
            }

            evt.addListener(document, "mouseover", handleEvent);
            evt.addListener(document, "mouseout", handleEvent);
            evt.addListener(document, "click", handleEvent);
        </script>
    </body>
    </html>
```

Save this as ch10_question3.html.

This solution starts with a new global variable called activeTab. Its purpose is to contain a reference to the tab element that was last clicked, and you initialize it as null.

When you click a tab element, you first need to deactivate the currently active tab:

```
if (activeTab) {
    activeTab.className = "tabStrip-tab";
}
```

To do that, you first check if you have an active tab, and if so, you set its className property back to the original tabStrip-tab. Then after you activate the new tab, you assign it to the activeTab variable:

```
activeTab = target;
```

Simple, but effective.

CHAPTER 11

Exercise 1 Question

Using the code from the temperature converter example you saw in Chapter 2, create a user interface for it and connect it to the existing code so that the user can enter a value in degrees Fahrenheit and convert it to centigrade.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 11: Question 1</title>
</head>
<body>
    <form action="" name="form1">
        <p>
            <input type="text" name="txtCalcBox" value="0.0" />
        </p>
        <input type="button" value="Convert to centigrade"
               id="btnToCent" name="btnToCent" />
    </form>
    <script>
        function convertToCentigrade(degFahren) {
            var degCent = 5 / 9 * (degFahren - 32);

            return degCent;
        }

        function btnToCentClick() {
            var calcBox = document.form1.txtCalcBox;

            if (isNaN(calcBox.value) == true || calcBox.value == "") {
                calcBox.value = "Error Invalid Value";
            } else {
                calcBox.value = convertToCentigrade(calcBox.value);
            }
        }

        document.getElementById("btnToCent")
                .addEventListener("click", btnToCentClick);
    </script>
</body>
</html>
```

Save this as `ch11_question1.html`.

The interface part is simply a form containing a text box into which users enter the Fahrenheit value and a button they click to convert that value to centigrade. The button has a `click` event listener that executes `btnToCentClick()` when the event fires.

The first line of `btnToCentClick()` declares a variable and sets it to reference the object representing the text box:

```
var calcBox = document.form1.txtCalcBox;
```

Why do this? Well, in your code when you want to use `document.form1.txtCalcBox`, you can now just use the much shorter `calcBox`; it saves typing and keeps your code shorter and easier to read.

So:

```
alert(document.form1.txtCalcBox.value);
```

is the same as:

```
alert(calcBox.value);
```

In the remaining part of the function you do a sanity check—if what the user has entered is a number (that is, it is not `NotANumber`) and the text box does contain a value, you use the Fahrenheit-to-centigrade conversion function you saw in Chapter 2 to do the conversion, the results of which are used to set the text box's value.

Exercise 2 Question

Create a user interface that allows users to pick the computer system of their dreams, similar in principle to the e-commerce sites selling computers over the Internet. For example, they could be given a choice of processor type, speed, memory, and hard drive size, and the option to add additional components like a DVD-ROM drive, a sound card, and so on. As the users change their selections, the price of the system should update automatically and notify them of the cost of the system as they specified it, either by using an `alert` box or by updating the contents of a text box.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 11: Question 2</title>
</head>
<body>
    <form action="" name="form1">
        <p>
            Choose the components you want included on your computer
        </p>
        <p>
            <label for="cboProcessor">Processor</label>
            <select name="cboProcessor" id="cboProcessor">
                <option value="100">Dual-core 2GHz</option>
                <option value="101">Quad-core 2.4GHz</option>
                <option value="102">Eight-core 3GHz</option>
            </select>
        </p>
        <p>
            <label for="cboSsd">Solid-state Drive</label>
            <select name="cboSsd" id="cboSsd">
                <option value="200">250GB</option>
                <option value="201">512GB</option>
                <option value="202">1TB</option>
            </select>
        </p>
        <p>
            <label for="chkDVD">DVD-ROM</label>
            <input type="checkbox" id="chkDVD" name="chkDVD" value="300" />
        </p>
        <p>
            <label for="chkBluRay">Blu-ray</label>
            <input type="checkbox" id="chkBluRay" name="chkBluRay"
                   value="301" />
        </p>
        <fieldset>
            <legend>Case</legend>
            <p>
                <label for="desktop">Desktop</label>
                <input type="radio" id="desktop"
                       name="radCase" checked value="400" />
            </p>
            <p>
                <label for="minitower">Mini-tower</label>
                <input type="radio" id="minitower"
                       name="radCase" value="401" />
            </p>
            <p>
                <label for="fulltower">Full-tower</label>
                <input type="radio" id="fulltower"
                       name="radCase" value="402" />
            </p>
        </fieldset>

        <p>
            <input type="button" value="Update"
                   id="btnUpdate" name="btnUpdate" />
        </p>
        <p>
            <label for="txtOrder">Order Summary:</label>
        </p>
        <p>
            <textarea rows="20" cols="35" id="txtOrder"
                      name="txtOrder"></textarea>
        </p>
    </form>
    <script>
        var productDb = [];
        productDb[100] = 150;
        productDb[101] = 350;
        productDb[102] = 700;

        productDb[200] = 100;
        productDb[201] = 200;
        productDb[202] = 500;

        productDb[300] = 50;
        productDb[301] = 75;

        productDb[400] = 75;
        productDb[401] = 50;
```

```
productDb[402] = 100;

function getDropDownInfo(element) {
    var selected = element[element.selectedIndex];

    return {
        text: selected.text,
        price: productDb[selected.value]
    };
}

function getCheckboxInfo(element) {
    return {
        checked: element.checked,
        price: productDb[element.value]
    };
}

function getRadioInfo(elements) {
    for (var i = 0; i < elements.length; i++) {
        if (!elements[i].checked) {
            continue;
        }

        var selected = elements[i];

        var label = document.querySelector(
                    "[for=" + selected.id + "]");

        return {
            text: label.innerHTML,
            price: productDb[selected.value]
        };
    }
}

function btnUpdateClick() {
    var total = 0;
    var orderDetails = "";
    var theForm = document.form1;

    var selectedProcessor = getDropDownInfo(theForm.cboProcessor);
    total = selectedProcessor.price;
    orderDetails = "Processor : " + selectedProcessor.text;
    orderDetails = orderDetails + " $" +
                   selectedProcessor.price + "\n";

    var selectedSsd = getDropDownInfo(theForm.cboSsd);
    total = total + selectedSsd.price;
    orderDetails = orderDetails + "Solid-state Drive : " +
                   selectedSsd.text;
    orderDetails = orderDetails + " $" + selectedSsd.price + "\n";

    var dvdInfo = getCheckboxInfo(theForm.chkDVD);
    if (dvdInfo.checked) {
```

```
                    total = total + dvdInfo.price;

                    orderDetails = orderDetails + "DVD-ROM : $" +
                        dvdInfo.price + "\n";
                }

                var bluRayInfo = getCheckboxInfo(theForm.chkBluRay);
                if (bluRayInfo.checked) {
                    total = total + bluRayInfo.price;

                    orderDetails = orderDetails + "Blu-ray : $" +
                        bluRayInfo.price + "\n";
                }

                var caseInfo = getRadioInfo(theForm.radCase);
                total = total + caseInfo.price;
                orderDetails = orderDetails + caseInfo.text + " : $" +
                        caseInfo.price;

                orderDetails = orderDetails + "\n\nTotal Order Cost is " +
                               "$" + total;

                theForm.txtOrder.value = orderDetails;
            }

            document.getElementById("btnUpdate")
                    .addEventListener("click", btnUpdateClick);

            </script>




        </script>
    </body>
    </html>
```

Save this as `ch11_question2.html`.

This is just one of many ways to tackle this question—you may well have thought of a better way.

Here you are displaying the results of the user's selection as text in a `textarea` box, with each item and its cost displayed on separate lines and a final total at the end.

Each form element has a value set to hold a stock ID number. For example, a full tower case is stock ID 402. The actual cost of the item is held in an array called `productDb`. Why not just store the price in the `value` attribute of each form element? Well, this way is more flexible. Currently your array just holds price details for each item, but you could modify it that so it holds more data—for example price, description, number in stock, and so on. Also, if this form is posted to a server the values passed will be stock IDs, which you could then use for a lookup in a stock database. If the values were set to prices and the form was posted, you'd have no way of telling what the customer ordered—all you'd know is how much it all cost.

This solution includes an Update button which, when clicked, updates the order details in the `textarea` box. However, you may want to add event handlers to each form element and update when anything changes.

Turning to the function that actually displays the order summary, `btnUpdateClick()`, you can see that there is a lot of code, and although it looks complex, it's actually fairly simple. A lot of it is repeated with slight modification. It also relies upon several helper functions to pull various information from the selected form elements.

To save on typing and make the code a little more readable, this solution declares the `theForm` variable to contain the `Form` object After the variable's declaration, you then find out which processor has been selected and get its cost and text with the `getDropDownInfo()` function:

```
function getDropDownInfo(element) {
    var selected = element[element.selectedIndex];

    return {
        text: selected.text,
        price: productDb[selected.value]
    };
}
```

The `selectedIndex` property tells you which `Option` object inside the select control has been selected by the user. You return a new object that contains the selected `Option`'s text and the price from the product database.

So to get the processor information, you pass `theForm.cboProcessor` to `getDropDownInfo()` and assign the resulting object to `selectedProcessor`. You then calculate the total and update the order details:

```
var selectedProcessor = getDropDownInfo(theForm.cboProcessor);
total = selectedProcessor.price;
orderDetails = "Processor : " + selectedProcessor.text;
orderDetails = orderDetails + " $" + selectedProcessor.price + "\n";
```

The same principle applies when you find the selected solid-state drive, so let's turn next to the check boxes for the optional extra items, looking first at the DVD-ROM check box:

```
var dvdInfo = getCheckboxInfo(theForm.chkDVD);
if (dvdInfo.checked) {
    total = total + dvdInfo.price;

    orderDetails = orderDetails + "DVD-ROM : $" +
        dvdInfo.price + "\n";
}
```

Again, you use a helper function—this one's called `getCheckboxInfo()`—to retrieve the information about the given check box:

```
function getCheckboxInfo(element) {
    return {
        checked: element.checked,
        price: productDb[element.value]
    };
}
```

This returns a new object that tells you the price of the component, as well as if the check box is checked.

If the check box is checked, you add a DVD-ROM to the order details and update the running total. The same principle applies for the Blu-ray check box.

Finally, you have the computer's case. Because only one case type out of the options can be selected, you used a radio button group. Unfortunately, there is no `selectedIndex` for radio buttons as there is for check boxes, so you have to go through each radio button in turn and find out if it has been selected. The `getRadioInfo()` helper function does just that:

```
function getRadioInfo(elements) {
    for (var i = 0; i < elements.length; i++) {
        if (!elements[i].checked) {
            continue;
        }

        var selected = elements[i];

        var label = document.querySelector("[for=" + selected.id + "]");

        return {
            text: label.innerHTML,
            price: productDb[selected.value]
        };
    }
}
```

It loops through the radio button group and checks each radio button's `checked` property. If it's `false`, the loop iterates with the `continue` operator. But if the radio button is checked, you need to get the text associated with the label for the selected radio button and the component's price from the `productDb` array.

So, inside the `btnUpdateClick()` function, you can use this helper function to get everything you need to add the selected computer case to the total and description:

```
var caseInfo = getRadioInfo(theForm.radCase);
total = total + caseInfo.price;
orderDetails = orderDetails + caseInfo.text + " : $" +
        caseInfo.price;
```

Finally, set the `textarea` to the details of the system the user has selected:

```
orderDetails = orderDetails + "\n\nTotal Order Cost is " + total;
theForm.txtOrder.value = orderDetails;
```

CHAPTER 12

Exercise 1 Question

The code for alerting a single message in Example 1 isn't very exciting. Modify the code to display a random message from a set of three possible messages.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 12: Question 1</title>
    <style>
        [data-drop-target] {
            height: 400px;
            width: 200px;
            margin: 2px;
            background-color: gainsboro;
            float: left;
        }

        .drag-enter {
            border: 2px dashed #000;
        }

        .box {
            width: 200px;
            height: 200px;
        }

        .navy {
            background-color: navy;
        }

        .red {
            background-color: red;
        }
    </style>
</head>
<body>
    <div data-drop-target="true">
        <div id="box1" draggable="true" class="box navy"></div>
        <div id="box2" draggable="true" class="box red"></div>
    </div>
    <div data-drop-target="true"></div>

    <script>
        function getRandomMessage() {
            var messages = [
                "You moved an element!",
                "Moved and element, you have! Mmmmmmm?",
                "Element overboard!"
            ];

            return messages[Math.floor((Math.random() * 3) + 0)];
        }

        function handleDragStart(e) {
            var data = {
                elementId: this.id,
                message: getRandomMessage()
            };

            e.dataTransfer.setData("text", JSON.stringify(data));
        }

        function handleDragEnterLeave(e) {
            if (e.type == "dragenter") {
                this.className = "drag-enter";
            } else {
                this.className = "";
            }
        }

        function handleOverDrop(e) {
            e.preventDefault();

            if (e.type != "drop") {
                return;
            }

            var json = e.dataTransfer.getData("text");
            var data = JSON.parse(json);

            var draggedEl = document.getElementById(data.elementId);

            if (draggedEl.parentNode == this) {
                this.className = "";
                return;
            }

            draggedEl.parentNode.removeChild(draggedEl);

            this.appendChild(draggedEl);
            this.className = "";

            alert(data.message);
        }

        var draggable = document.querySelectorAll("[draggable]");
        var targets = document.querySelectorAll("[data-drop-target]");

        for (var i = 0; i < draggable.length; i++) {
            draggable[i].addEventListener("dragstart", handleDragStart);
```

```
            }

            for (i = 0; i < targets.length; i++) {
                targets[i].addEventListener("dragover", handleOverDrop);
                targets[i].addEventListener("drop", handleOverDrop);
                targets[i].addEventListener("dragenter", handleDragEnterLeave);
                targets[i].addEventListener("dragleave", handleDragEnterLeave);
            }
        </script>
    </body>
    </html>
```

Save this as `ch12_question1.html`.

This solution is rather simple. It introduces a new function called `getRandomMessage()`, which returns one of three messages:

```
    function getRandomMessage() {
        var messages = [
            "You moved an element!",
            "Moved and element, you have! Mmmmmmm?",
            "Element overboard!"
        ];

        return messages[Math.floor((Math.random() * 3) + 0)];
    }
```

And you use this function when you assign the `message` property to the data object in `handleDragStart()`:

```
    var data = {
        elementId: this.id,
        message: getRandomMessage()
    };
```

Sadly, this solution still doesn't add much excitement to the example. But some is better than none, right?

CHAPTER 13

Exercise 1 Question

Using local storage, create a page that keeps track of how many times the page has been visited by the user in the last month.

Exercise 1 Solution

```
    <!DOCTYPE html>

    <html lang="en">
    <head>
        <title>Chapter 13: Question 1</title>
    </head>
```

```
<body>
    <script>
        var pageViewCount = localStorage.getItem("pageViewCount");
        var pageFirstVisited = localStorage.getItem("pageFirstVisited");
        var now = new Date();

        if (pageViewCount == null) {
            pageViewCount = 0;
            pageFirstVisited = now.toUTCString();
        }

        var oneMonth = new Date(pageFirstVisited);
        oneMonth.setMonth(oneMonth.getMonth() + 1);

        if (now > oneMonth) {
            pageViewCount = 0;
            pageFirstVisited = now.toUTCString();
        }

        pageViewCount = parseInt(pageViewCount, 10) + 1;

        localStorage.setItem("pageViewCount", pageViewCount);
        localStorage.setItem("pageFirstVisited", pageFirstVisited);

        var output = "You've visited this page " + pageViewCount +
            " times since " + pageFirstVisited;

        document.write(output);
    </script>
</body>
</html>
```

Save this as `ch13_question1.html`.

The first two lines get two values from `localStorage` and store them in variables. The first holds the number of visits, the second the date the page was first visited. You also create a variable to contain the current date:

```
var pageViewCount = localStorage.getItem("pageViewCount");
var pageFirstVisited = localStorage.getItem("pageFirstVisited");
var now = new Date();
```

If the `pageViewCount` key does not exist in `localStorage`, the variable of the same name is `null`, and you'll need to initialize the `pageViewcount` and `pageFirstVisited` variables with `0` and the current date, respectively. Remember that `localStorage` contains only string data, so you use the `Date` object's `toUTCString()` method to convert the date to a string:

```
if (pageViewCount == null) {
    pageViewCount = 0;
    pageFirstVisited = now.toUTCString();
}
```

You're only tracking the number of visits within a month's time span. So, next you need a variable to contain a `Date` object one month from the first visit:

```
var oneMonth = new Date(pageFirstVisited);
oneMonth.setMonth(oneMonth.getMonth() + 1);
```

If the current date and time is later than `oneMonth`, it's time to reset the counter and visited variables:

```
if (now > oneMonth) {
    pageViewCount = 0;
    pageFirstVisited = now.toUTCString();
}
```

Then you increment the counter and store it and the first visit value in `localStorage`:

```
pageViewCount = parseInt(pageViewCount, 10) + 1;

localStorage.setItem("pageViewCount", pageViewCount);
localStorage.setItem("pageFirstVisited", pageFirstVisited);
```

Finally, write information to the page:

```
var output = "You've visited this page " + pageViewCount +
    " times since " + pageFirstVisited;

document.write(output);
```

Exercise 2 Question

Use cookies to load a different advertisement every time a user visits a web page.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 13: Question 2</title>
</head>
<body>
    <script>
        var ads = [
            "Buy Product A! You won't be sorry!",
            "You need Product B! Buy buy buy!",
            "Don't buy Product A or B! Product C is the only option for you!"
        ];

        function getRandomNumber(min, max) {
            return Math.floor((Math.random() * max) + min);
        }

        var lastAdNumber = localStorage.getItem("lastAdNumber");
        var nextNumber = getRandomNumber(0, ads.length);

        if (lastAdNumber == null) {
            lastAdNumber = nextNumber;
        } else {
            lastAdNumber = parseInt(lastAdNumber, 10);
            while (lastAdNumber == nextNumber) {
                nextNumber = getRandomNumber(0, ads.length);
            }
        }

        localStorage.setItem("lastAdNumber", nextNumber);

        document.write(ads[nextNumber]);
    </script>
</body>
</html>
```

Save this as ch13_question2.html.

This solution is loosely based on similar questions in previous chapters where you have displayed random images or messages. In this case you display a different message in the page each time the user visits it; you'll never see the same message displayed two times in a row in the same browser.

You store the last number of the previously displayed ad in localStorage with the key lastAdNumber. So, you retrieve that value and generate the next number with a getRandomNumber() helper function (you know this algorithm):

```
var lastAdNumber = localStorage.getItem("lastAdNumber");
var nextNumber = getRandomNumber(0, ads.length);
```

If lastAdNumber is null, you can use the value in nextNumber:

```
if (lastAdNumber == null) {
    lastAdNumber = nextNumber;
}
```

But if lastAdNumber is not null, you need to generate a random number that is not lastAdNumber. So first, you convert lastAdNumber to a number with the parseInt() function:

```
  else {
    lastAdNumber = parseInt(lastAdNumber, 10);
    while (lastAdNumber == nextNumber) {
        nextNumber = getRandomNumber(0, ads.length);
    }
}
```

Then you use a while loop to generate a unique random number. The loop iterates if lastAdNumber is equal to nextNumber, and it continues to do so until the next number is different than lastAdNumber.

Once you have a unique next number, you store it in localStorage and display the ad in the page:

```
localStorage.setItem("lastAdNumber", nextNumber);

document.write(ads[nextNumber]);
```

CHAPTER 14

Exercise 1 Question

Extend the `HttpRequest` module to include synchronous requests in addition to the asynchronous requests the module already makes. You'll have to make some adjustments to your code to incorporate this functionality. (Hint: Create an `async` property for the module.)

Exercise 1 Solution

```
function HttpRequest(url, callback) {
    this.url = url;
    this.callBack = callback;
    this.async = true;
    this.request = new XMLHttpRequest();
};

HttpRequest.prototype.send = function() {
    this.request.open("GET", this.url, this.async);

    if (this.async) {
        var tempRequest = this.request;
        var callback = this.callBack;

        function requestReadystatechange() {
            if (tempRequest.readyState == 4) {
                if (tempRequest.status == 200) {
                    callback(tempRequest.responseText);
                } else {
                    alert("An error occurred while attempting to " +
                        "contact the server.");
                }
            }
        }

        this.request.onreadystatechange = requestReadystatechange;
    }

    this.request.send(null);

    if (!this.async) {
        this.callBack(this.request.responseText);
    }
};
```

It's possible to add synchronous communication to your `HttpRequest` module in a variety of ways. The approach in this solution refactors the code to accommodate a new property called `async`, which contains either `true` or `false`. If it contains `true`, the underlying `XMLHttpRequest` object uses asynchronous communication to retrieve the file. If `false`, the module uses synchronous communication.

The first change made to the module is in the constructor itself. The original constructor initializes and readies the `XMLHttpRequest` object to send data. This will not do for this new version, however. Instead, the constructor merely initializes all the properties:

```
function HttpRequest(url, callback) {
    this.url = url;
    this.callBack = callback;
    this.async = true;
    this.request = new XMLHttpRequest();
};
```

You have three new properties here. The first, `url`, contains the URL that the `XMLHttpRequest` object should attempt to request from the server. The `callBack` property contains a reference to the callback function, and the `async` property determines the type of communication the `XMLHttpRequest` object uses. Setting `async` to `true` in the constructor gives the property a default value. Therefore, you can send the request in asynchronous mode without setting the property externally.

The new constructor and properties are actually desirable, because they enable you to reuse the same `HttpRequest` object for multiple requests. If you wanted to make a request to a different URL, all you would need to do is assign the `url` property a new value. The same can be said for the callback function as well.

The majority of changes to the module are in the `send()` method. It is here that the module decides whether to use asynchronous or synchronous communication. Both types of communication have very little in common when it comes to making a request; asynchronous communication uses the `onreadystatechange` event handler, and synchronous communication allows access to the `XMLHttpRequest` object's properties when the request is complete. Therefore, code branching is required:

```
HttpRequest.prototype.send = function() {
    this.request.open("GET", this.url, this.async);

    if (this.async) {
        //more code here
    }
    this.request.send(null);

    if (!this.async) {
        //more code here
    }
}
```

The first line of this method uses the `open()` method of the `XMLHttpRequest` object. The `async` property is used as the final parameter of the method. This determines whether or not the XHR object uses asynchronous communication. Next, an `if` statement tests to see if `this.async` is `true`; if it is, the asynchronous code will be placed in this `if` block. Next, the `XMLHttpRequest` object's `send()` method is called, sending the request to the server. The final `if` statement checks to see whether `this.async` is `false`. If it is, synchronous code is placed within the code block to execute.

```
HttpRequest.prototype.send = function() {
    this.request.open("GET", this.url, this.async);

    if (this.async) {
        var tempRequest = this.request;
        var callback = this.callBack;

        function requestReadystatechange() {
            if (tempRequest.readyState == 4) {
                if (tempRequest.status == 200) {
                    callback(tempRequest.responseText);
                } else {
                    alert("An error occurred while attempting to " +
                        "contact the server.");
                }
            }
        }

        this.request.onreadystatechange = requestReadystatechange;
    }

    this.request.send(null);

    if (!this.async) {
        this.callBack(this.request.responseText);
    }
};
```

This new code finishes off the method. Starting with the first `if` block, a new variable called `callback` is assigned the value of `this.callBack`. This is done for the same reasons as with the `tempRequest` variable—scoping issues—because `this` points to the `requestReadystatechange()` function instead of the `HttpRequest` object. Other than this change, the asynchronous code remains the same. The `requestReadystatechange()` function handles the `readystatechange` event and calls the callback function when the request is successful.

The second `if` block is much simpler. Because this code executes only if synchronous communication is desired, all you have to do is call the callback function and pass the `XMLHttpRequest` object's `responseText` property.

Using this newly refactored module is quite simple. The following code makes an asynchronous request for a fictitious text file called `test.txt`:

```
function requestCallback(responseText) {
    alert(responseText);
}

var http = new HttpRequest("test.txt", requestCallback);

http.send();
```

Nothing has really changed for asynchronous requests. This is the exact same code used earlier in the chapter. If you want to use synchronous communication, simply set `async` to `false`, like this:

```
function requestCallback(responseText) {
    alert(responseText);
}

var http = new HttpRequest("test.txt", requestCallback);

http.async = false;

http.send();
```

You now have an Ajax module that requests information in both asynchronous and synchronous communication!

Exercise 2 Question

It was mentioned earlier in the chapter that you could modify the smart forms to not use hyperlinks. Change the form that uses the `HttpRequest` module so that the Username and Email fields are checked when the user submits the form. Listen for the form's `submit` event and cancel the submission if a username or e-mail is taken.

Exercise 2 Solution

First, a disclaimer: Ideally, the service provided by `ch14_formvalidator.php` should allow you to check both the username and e-mail address with a single request. That would greatly simplify this solution. However, sometimes you need to make multiple requests, and each request is sometimes dependent upon the outcome of a previous request. This question and solution is meant to emulate that.

Additionally, issuing multiple (and linked) asynchronous operations is a rather complex ordeal—a condition referred to "callback hell." You'll get a taste of that in this solution.

```
 <!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 14: Question 2</title>
    <style>
        .fieldname {
            text-align: right;
        }

        .submit {
            text-align: right;
        }
    </style>
</head>
<body>
    <form name="theForm">
```

```
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
                <td>

                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
                <td>

                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>
                <td />
            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>
                <td />
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input id="btnSubmit" type="submit" value="Submit" />
                </td>
                <td />
            </tr>
        </table>
    </form>
    <script src="ch14_question1.js"></script>
    <script>
        function btnSubmitClick(e) {
            e.preventDefault();

            checkUsername();
```

```
}

function checkUsername() {
    var userValue = document.getElementById("username").value;

    if (!userValue) {
        alert("Please enter a user name to check!");
        return;
    }

    var url = "ch14_formvalidator.php?username=" + userValue;

    var request = new HttpRequest(url, handleUsernameResponse);
    request.send();
}

function checkEmail() {
    var emailValue = document.getElementById("email").value;

    if (!emailValue) {
        alert("Please enter an email address to check!");
        return;
    }

    var url = "ch14_formvalidator.php?email=" + emailValue;

    var request = new HttpRequest(url, handleEmailResponse);
    request.send();
}

function handleUsernameResponse(responseText) {
    var response = JSON.parse(responseText);

    if (!response.available) {
        alert("The username " + response.searchTerm +
              " is unavailable. Try another.");
        return;
    }

    checkEmail();
}

function handleEmailResponse(responseText) {
    var response = JSON.parse(responseText);

    if (!response.available) {
        alert("The email address " + response.searchTerm +
              " is unavailable. Try another.");
        return;
    }

    document.theForm.submit();
}

document.getElementById("btnSubmit")
        .addEventListener("click", btnSubmitClick);
        </script>
    </body>

    </html>
```

Save this as ch14_question2.html.

In the HTML, notice that the links for checking the username and e-mail address are gone. There is no need for them, because those values are checked when the user clicks the Submit button. The last statement of the JavaScript code registers that event listener:

```
document.getElementById("btnSubmit")
        .addEventListener("click", btnSubmitClick);
```

The function that handles the button's click event is called btnSubmitClick(). It's a simple function that kicks off the whole process:

```
function btnSubmitClick(e) {
    e.preventDefault();

    checkUsername();
}
```

Its first statement prevents the form from submitting. This is important because, due to the nature of asynchronous processes, btnSubmitClick() cannot be responsible for submitting the form. Therefore, another function will need to submit the form if both the username and e-mail address validate and are available.

The second statement calls checkUsername(), which is left mostly unchanged:

```
function checkUsername() {
    var userValue = document.getElementById("username").value;

    if (!userValue) {
        alert("Please enter a user name to check!");
        return;
    }

    var url = "ch14_formvalidator.php?username=" + userValue;

    var request = new HttpRequest(url, handleUsernameResponse);
    request.send();
}
```

In fact, the only change to this function is the callback passed to the HttpRequest constructor. It's a new callback function called handleUsernameResponse(), and it somewhat resembles the original handleResponse() function:

```
function handleUsernameResponse(responseText) {
    var response = JSON.parse(responseText);

    if (!response.available) {
        alert("The username " + response.searchTerm +
              " is unavailable. Try another.");
```

```
            return;
        }

        checkEmail();
    }
```

This function takes the response and parses it into a JavaScript object. If the username is not available, it displays the error message to the user and returns. Nothing else is processed when the username is unavailable, but if it is available, it calls `checkEmail()`:

```
    function checkEmail() {
        var emailValue = document.getElementById("email").value;

        if (!emailValue) {
            alert("Please enter an email address to check!");
            return;
        }

        var url = "ch14_formvalidator.php?email=" + emailValue;

        var request = new HttpRequest(url, handleEmailResponse);
        request.send();
    }
```

This function is also largely the same. The only difference is the callback function passed to the `HttpRequest` constructor; it's called `handleEmailResponse()`. It parses the request, and it is the last step in the process:

```
    function handleEmailResponse(responseText) {
        var response = JSON.parse(responseText);

        if (!response.available) {
            alert("The email address " + response.searchTerm +
                  " is unavailable. Try another.");
            return;
        }

        document.theForm.submit();
    }
```

Once again, if the e-mail address is not available, this function displays the error message to the user and returns. But if the e-mail address is available, the form is finally submitted.

CHAPTER 15

Exercise 1 Question

Being able to control playback is cool, but your custom UI needs to also control volume. Add an `<input type="range" />` element to Example 3 to control the volume. Remember that the range of volume supported by media elements is 0.0 to 1.0. Look back at Chapter 11 if you need a refresher of the range input type. This unfortunately will not work in IE.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 15: Question 1</title>
</head>
<body>
    <div>
        <button id="playbackController">Play</button>
        <button id="muteController">Mute</button>
        <input type="range" id="volumeController"
               min="0" max="1" step=".1" value="1"/>
    </div>
    <video id="bbbVideo">
        <source src="bbb.mp4" />
        <source src="bbb.webm" />
    </video>

    <script>
        function pauseHandler(e) {
            playButton.innerHTML = "Resume";
        }

        function playingHandler(e) {
            playButton.innerHTML = "Pause";
        }

        function volumechangeHandler(e) {
            muteButton.innerHTML = video.muted ? "Unmute" : "Mute";
        }

        function playbackClick(e) {
            video.paused ? video.play() : video.pause();
        }

        function muteClick(e) {
            video.muted = !video.muted;
        }

        function volumeInput(e) {
            video.volume = volumeSlider.value;
        }

        var video = document.getElementById("bbbVideo");
        var playButton = document.getElementById("playbackController");
        var muteButton = document.getElementById("muteController");
        var volumeSlider = document.getElementById("volumeController");

        video.addEventListener("pause", pauseHandler);
```

```
            video.addEventListener("playing", playingHandler);
            video.addEventListener("volumechange", volumechangeHandler);

            playButton.addEventListener("click", playbackClick);
            muteButton.addEventListener("click", muteClick);
            volumeSlider.addEventListener("input", volumeInput);
        </script>
    </body>
    </html>
```

Save this as ch15_question1.html.

This solution is built on Example 3. The additions are highlighted for your convenience.

The volume will be controlled by an `<input/>` element:

```
<input type="range" id="volumeController"
       min="0" max="1" step=".1" value="1"/>
```

It’s a range control, and it's set to a minimum value of `0`, a maximum value of `1`, and the step is `.1`. Its initial value is set to `1`, meaning full volume.

In the JavaScript code, you retrieve this element and store it in the `volumeSlider` variable:

```
var volumeSlider = document.getElementById("volumeController");
```

And you register an `input` event listener:

```
volumeSlider.addEventListener("input", volumeInput);
```

The `volumeInput()` function handles this event, and it is responsible for setting the media’s volume to the slider’s corresponding value:

```
function volumeInput(e) {
    video.volume = volumeSlider.value;
}
```

Exercise 2 Question

Add another range form control to Question 1’s answer, and program it to seek the media. It should also update as the media plays. Use the `durationchange` event to set the slider’s max value, and the `timeupdate` event to update the slider’s value.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 15: Question 2</title>
</head>
<body>
    <div>
        <button id="playbackController">Play</button>
        <button id="muteController">Mute</button>
        <input type="range" id="volumeController"
               min="0" max="1" step=".1" value="1"/>
    </div>
    <video id="bbbVideo">
        <source src="bbb.mp4" />
        <source src="bbb.webm" />
    </video>
    <div>
        <input type="range" id="seekController"
               min="0" step="1" value="0" />
    </div>
    <script>
        function pauseHandler(e) {
            playButton.innerHTML = "Resume";
        }

        function playingHandler(e) {
            playButton.innerHTML = "Pause";
        }

        function volumechangeHandler(e) {
            muteButton.innerHTML = video.muted ? "Unmute" : "Mute";
        }

        function durationchangeHandler(e) {
            seekSlider.max = video.duration;
        }

        function timeupdateHandler(e) {
            seekSlider.value = video.currentTime;
        }

        function playbackClick(e) {
            video.paused ? video.play() : video.pause();
        }

        function muteClick(e) {
            video.muted = !video.muted;
        }

        function volumeInput(e) {
            video.volume = volumeSlider.value;
        }

        function seekInput(e) {
            video.currentTime = seekSlider.value;
        }


        var video = document.getElementById("bbbVideo");
        var playButton = document.getElementById("playbackController");
        var muteButton = document.getElementById("muteController");
        var volumeSlider = document.getElementById("volumeController");
```

```
            var seekSlider = document.getElementById("seekController");

            video.addEventListener("pause", pauseHandler);
            video.addEventListener("playing", playingHandler);
            video.addEventListener("volumechange", volumechangeHandler);
            video.addEventListener("durationchange", durationchangeHandler);
            video.addEventListener("timeupdate", timeupdateHandler);


            playButton.addEventListener("click", playbackClick);
            muteButton.addEventListener("click", muteClick);
            volumeSlider.addEventListener("input", volumeInput);
            seekSlider.addEventListener("input", seekInput);

        </script>
    </body>
    </html>
```

Save this as ch15_question2.html.

Once again, the changes are highlighted. You add another range control to the page:

```
    <div>
        <input type="range" id="seekController"
               min="0" step="1" value="0" />
    </div>
```

This one is called seekController. It is set to a minimum value of 0, a step of 1, and an initial value of 0. A maximum value is not set because you do not yet know the duration of the video. You will, however, when the media's durationchange event fires. You register a listener for this event; it calls the durationchangeHandler() function when it fires.

```
    function durationchangeHandler(e) {
        seekSlider.max = video.duration;
    }
```

It simply sets the slider's max property to the media's duration.

You also register a listener for the media's timeupdate event with the timeupdateHandler() function. You use this to update the slider's value when the media's current time changes:

```
    function timeupdateHandler(e) {
        seekSlider.value = video.currentTime;
    }
```

But you also want to control the media's seek with the slider, so you listen for the range control's input event:

```
    function seekInput(e) {
        video.currentTime = seekSlider.value;
    }
```

And you set the media's currentTime property to the slider's value.

CHAPTER 16

Exercise 1 Question

Example 1 is based on Chapter 10's Example 17, and as you probably remember, you modified that example in response to one of Chapter 10's exercise questions. Modify this chapter's Example 1 so that only one tab is active at a time.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 16: Question 1</title>
    <style>
        .tabStrip {
            background-color: #E4E2D5;
            padding: 3px;
            height: 22px;
        }

            .tabStrip div {
                float: left;
                font: 14px arial;
                cursor: pointer;
            }

        .tabStrip-tab {
            padding: 3px;
        }

        .tabStrip-tab-hover {
            border: 1px solid #316AC5;
            background-color: #C1D2EE;
            padding: 2px;
        }

        .tabStrip-tab-click {
            border: 1px solid #facc5a;
            background-color: #f9e391;
            padding: 2px;
        }
    </style>
</head>
<body>
    <div class="tabStrip">
        <div data-tab-number="1" class="tabStrip-tab">Tab 1</div>
        <div data-tab-number="2" class="tabStrip-tab">Tab 2</div>
        <div data-tab-number="3" class="tabStrip-tab">Tab 3</div>
    </div>
```

```
        <div id="descContainer"></div>

        <script src="jquery-2.1.1.min.js"></script>
        <script>
            var activeTab = null;

            function handleEvent(e) {
                var target = $(e.target);
                var type = e.type;

                if (type == "mouseover" || type == "mouseout") {
                    target.toggleClass("tabStrip-tab-hover");
                } else if (type == "click") {

                    if (activeTab) {
                        activeTab.removeClass("tabStrip-tab-click");
                    }

                    target.addClass("tabStrip-tab-click");

                    var num = target.attr("data-tab-number");
                    showDescription(num);

                    activeTab = target;
                }
            }

            function showDescription(num) {
                var text = "Description for Tab " + num;

                $("#descContainer").text(text);
            }

            $(".tabStrip > div").on("mouseover mouseout click", handleEvent);
        </script>

    </body>
    </html>
```

Save this as `ch16_question1.html`.

The overall logic of this solution is identical to Chapter 10's Solution 3. You define a variable to track the active tab:

```
var activeTab = null;
```

Then when the user clicks one of the tabs, you remove the `tabStrip-tab-click` CSS class from the active tab (if one exists):

```
if (activeTab) {
    activeTab.removeClass("tabStrip-tab-click");
}
```

And then you set the current tab as the active tab:

```
activeTab = target;
```

Exercise 2 Question

There is some repetitive code in Example 2. Refactor the code to reduce the duplication. Additionally, add a function to handle any errors that may occur with the request.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 16: Question 2</title>
    <style>
        .fieldname {
            text-align: right;
        }

        .submit {
            text-align: right;
        }
    </style>
</head>
<body>
    <form>
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
                <td>
                    <a id="usernameAvailability" href="#">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
                <td>
                    <a id="emailAvailability" href="#">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>
                <td />
            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>
                <td />
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input type="submit" value="Submit" />
                </td>
                <td />
            </tr>
        </table>
    </form>
    <script src="jquery-2.1.1.min.js"></script>
    <script>
        function checkUsername(e) {
            e.preventDefault();

            var userValue = $("#username").val();

            if (!userValue) {
                alert("Please enter a user name to check!");
                return;
            }

            makeRequest({
                username: userValue
            });
        }

        function checkEmail(e) {
            e.preventDefault();

            var emailValue = $("#email").val();

            if (!emailValue) {
                alert("Please enter an email address to check!");
                return;
            }

            makeRequest({
                email: emailValue
            });
        }

        function makeRequest(parameters) {
            $.getJSON("ch14_formvalidator.php", parameters)
                .done(handleResponse)
                .fail(handleError);
        }

        function handleError() {
            alert("A network error occurred. Please try again " +
                  "in a few moments.");
        }

        function handleResponse(response) {
            if (response.available) {
                alert(response.searchTerm + " is available!");
            } else {
                alert("We're sorry, but " + response.searchTerm +
                      " is not available.");
            }
        }

        $("#usernameAvailability").on("click", checkUsername);
        $("#emailAvailability").on("click", checkEmail);
    </script>
</body>

</html>
```

Save this as `ch16_question2.html`.

The main source of duplication is the code that makes the actual request:

```
$.getJSON("ch14_formvalidator.php", parms).done(handleResponse);
```

Although it's a single line of code, it can be moved into a separate function to make it easier to maintain. Plus, when you add a function for handling Ajax errors, you have to visit only one function as opposed to two.

First write a function that displays a message to the user when the Ajax request fails. Call it `handleError()`:

```
function handleError() {
    alert("A network error occurred. Please try again " +
          "in a few moments.");
}
```

Now write a function that performs the Ajax request, and chain a `fail()` call to `done()`:

```
function makeRequest(parameters) {
    $.getJSON("ch14_formvalidator.php", parameters)
        .done(handleResponse)
        .fail(handleError);
}
```

You can now use this `makeRequest()` function inside the `checkUsername()` function:

```
function checkUsername(e) {
    e.preventDefault();

    var userValue = $("#username").val();

    if (!userValue) {
        alert("Please enter a user name to check!");
        return;
    }

    makeRequest({
        username: userValue
    });
}
```

And the `checkEmail()` function:

```
function checkEmail(e) {
    e.preventDefault();

    var emailValue = $("#email").val();

    if (!emailValue) {
        alert("Please enter an email address to check!");
        return;
    }

    makeRequest({
        email: emailValue
    });
}
```

You can also eliminate the `parms` variable in both of the functions, as shown in this solution. Just pass the object literal directly to `makeRequest()` to make your code more concise.

CHAPTER 17

Exercise 1 Question

Modify the answer to Chapter 14's Question 2 using Prototype. Also add error reporting for when an error occurs with the Ajax request.

Exercise 1 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 17: Question 1</title>
    <style>
        .fieldname {
            text-align: right;
        }

        .submit {
            text-align: right;
        }
    </style>
</head>
<body>
    <form name="theForm">
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
                <td></td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
                <td></td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>
                <td />
            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>
                <td />
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input id="btnSubmit" type="submit" value="Submit" />
                </td>
                <td />
            </tr>
        </table>
    </form>
    <script src="prototype.1.7.2.js"></script>
    <script>
        function btnSubmitClick(e) {
```

```
        e.preventDefault();

        checkUsername();
    }

    function checkUsername() {
        var userValue = $("username").value;

        if (!userValue) {
            alert("Please enter a user name to check!");
            return;
        }

        var options = {
            method: "get",
            onSuccess: handleUsernameResponse,
            onFailure: handleError,
            parameters: {
                username: userValue
            }
        };

        new Ajax.Request("ch14_formvalidator.php", options);
    }

    function checkEmail() {
        var emailValue = $("email").value;

        if (!emailValue) {
            alert("Please enter an email address to check!");
            return;
        }

        var options = {
            method: "get",
            onSuccess: handleEmailResponse,
            onFailure: handleError,
            parameters: {
                email: emailValue
            }
        };

        new Ajax.Request("ch14_formvalidator.php", options);
    }

    function handleUsernameResponse(transport) {
        var response = transport.responseJSON;

        if (!response.available) {
            alert("The username " + response.searchTerm +
                    " is unavailable. Try another.");
            return;
        }

        checkEmail();
        }

        function handleEmailResponse(transport) {
            var response = transport.responseJSON;

            if (!response.available) {
                alert("The email address " + response.searchTerm +
                        " is unavailable. Try another.");
                return;
            }

            document.theForm.submit();
        }

        function handleError() {
            alert("A network error occurred. Please try again " +
                  "in a few moments.");
        }

        $("btnSubmit").observe("click", btnSubmitClick);
    </script>
</body>

</html>
```

Save this as `ch17_question1.html`.

This solution is based on Chapter 14's Solution 2, but the code has changed to use Prototype's API. There's also a new function called `handleError()` for handling errors:

```
function handleError() {
    alert("A network error occurred. Please try " +
          "again in a few moments.");
}
```

This function is called `handleError()`, and it simply displays a message to the user. You'll assign this function to the `onFailure` option when you make your Ajax requests.

Making a request is many more lines of code due to Prototype's Ajax API:

```
var options = {
    method: "get",
    onSuccess: handleEmailResponse,
    onFailure: handleError,
    parameters: {
        email: emailValue
    }
};

new Ajax.Request("ch14_formvalidator.php", options);
```

This excerpt is taken from `checkUsername()`. You create your `options` object containing the `method`, `onSuccess`, `onFailure`, and `parameters` properties, and then you issue the request.

On a successful request, the `handleUsernameResponse()` and `handleEmailResponse()` functions execute. They no longer manually parse the JSON data into a JavaScript object; instead, they use the `responseJSON` property:

```
var response = transport.responseJSON;
```

And finally, you register the button's `click` event listener using the `observe()` method:

```
$("btnSubmit").observe("click", btnSubmitClick);
```

Exercise 2 Question

If you guessed that this question would be: "Change the answer to Chapter 14's Question 2 using MooTools, and add error reporting for when an error occurs with the Ajax request" then you won!! Your prize is…completing the exercise.

Exercise 2 Solution

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 17: Question 2</title>
    <style>
        .fieldname {
            text-align: right;
        }

        .submit {
            text-align: right;
        }
    </style>
</head>
<body>
    <form name="theForm">
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
                <td></td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
                <td></td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>
                <td />
            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>
                <td />
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input id="btnSubmit" type="submit" value="Submit" />
                </td>
                <td />
            </tr>
        </table>
    </form>
    <script src="mootools-core-1.5.1-compressed.js"></script>
    <script>
        function btnSubmitClick(e) {
            e.preventDefault();

            checkUsername();
        }

        function checkUsername() {
            var userValue = $("username").value;

            if (!userValue) {
                alert("Please enter a user name to check!");
                return;
            }

            var options = {
                url: "ch14_formvalidator.php",
                data: {
                    username: userValue
                },
                onSuccess: handleUsernameResponse,
                onFailure: handleError
            };

            new Request.JSON(options).get();


        }

        function checkEmail() {
            var emailValue = $("email").value;

            if (!emailValue) {
                alert("Please enter an email address to check!");
                return;
            }

            var options = {
                url: "ch14_formvalidator.php",
                data: {
                    email: emailValue
                },
                onSuccess: handleEmailResponse,
                onFailure: handleError
            };

            new Request.JSON(options).get();

        }

        function handleUsernameResponse(response) {
            if (!response.available) {
                alert("The username " + response.searchTerm +
                        " is unavailable. Try another.");
                return;
            }

            checkEmail();
        }

        function handleEmailResponse(response) {
            if (!response.available) {
                alert("The email address " + response.searchTerm +
                        " is unavailable. Try another.");
                return;
            }

            document.theForm.submit();
        }

        function handleError() {
            alert("A network error occurred. Please try again " +
                  "in a few moments.");
        }

        $("btnSubmit").addEvent("click", btnSubmitClick);
    </script>
```

```
</body>

</html>
```

CHAPTER 18

Exercise 1 Question

The example `ch18_example4.html` has a deliberate bug. For each times table it creates only multipliers with values from `1` to `11`.

Use the script debugger to work out why this is happening, and then correct the bug.

Exercise 1 Solution

The problem is with the code's logic rather than its syntax. Logic errors are much harder to spot and deal with because, unlike with syntax errors, the browser won't inform you that there's such and such error at line so and so but instead just fails to work as expected. The error is with this line:

```
for (var counter = 1; counter < 12; counter++)
```

You want the loop to go from `1` to `12` inclusive. Your `counter < 12` statement will be `true` up to and including `11` but will be `false` when the counter reaches `12`; hence `12` gets left off. To correct this, you could change the code to the following:

```
for (var counter = 1; counter <= 12; counter++)
```

Exercise 2 Question

The following code contains a number of common errors. See if you can spot them:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 18: Question 2</title>
</head>
<body>
    <form name="form1" action="">
        <input type="text" id="text1" name="text1" />
        <br />
        CheckBox 1<input type="checkbox" id="checkbox2" name="checkbox2" />
        <br />
        CheckBox 1<input type="checkbox" id="checkbox1" name="checkbox1" />
        <br />
        <input type="text" id="text2" name="text2" />
        <p>
            <input type="submit" value="Submit" id="submit1"
                   name="submit1" />
        </p>
    </form>

    <script>
        function checkForm(e) {
            var elementCount = 0;
            var theForm = document.form1;

            while(elementCount =<= theForm.length) {
                if (theForm.elements[elementcount].type == "text") {
                    if (theForm.elements[elementCount].value() = "")
                        alert("Please complete all form elements");
                    theForm.elements[elementCount].focus;
                    e.preventDefault();
                    break;
                }
            }
        }

        document.form1.addEventListener("submit", checkForm);
    </script>
</body>

</html>
```

Exercise 2 Solution

The bug-free version looks like this:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 18: Question 2</title>
</head>
<body>
    <form name="form1" action="">
        <input type="text" id="text1" name="text1" />
        <br />
        CheckBox 1<input type="checkbox" id="checkbox2" name="checkbox2" />
        <br />
        CheckBox 1<input type="checkbox" id="checkbox1" name="checkbox1" />
        <br />
        <input type="text" id="text2" name="text2" />
        <p>
            <input type="submit" value="Submit" id="submit1"
                   name="submit1" />
        </p>
    </form>

    <script>
        function checkForm(e) {
            var elementCount = 0;
            var theForm = document.form1;

            while(elementCount < theForm.length) {
                if (theForm.elements[elementCount].type == "text") {
                    if (theForm.elements[elementCount].value == "") {
                        alert("Please complete all form elements");
                        theForm.elements[elementCount].focus();
                        e.preventDefault();
                        break;
                    }
                }

                elementCount++;
            }
        }

        document.form1.addEventListener("submit", checkForm);
    </script>
</body>

</html>
```

Let's look at each error in turn. The first error is a logic error:

```
while(elementCount =< theForm.length)
```

Arrays start at `0` so the first `element` object is at index array `0`, the second at `1`, and so on. The last object has an index value of `4`. However, `theForm.length` will return `5` because there are five elements in the form. So the `while` loop will continue until `elementCount` is less than or equal to `5`, but because the last element has an index of `4`, this is one past the limit. You should write either this:

```
while(elementCount < theForm.length)
```

or this:

```
while(elementCount <= theForm.length - 1)
```

Either is fine, though the first is shorter.

You come to your second error in the following line:

```
if (theForm.elements[elementcount].type == "text")
```

On a quick glance it looks fine, but it's JavaScript's strictness on case sensitivity that has caused the downfall. The variable name is `elementCount`, not `elementcount` with a lowercase c. So this line should read as follows:

```
if (theForm.elements[elementCount].type == "text")
```

The next line with an error is this:

```
if (theForm.elements[elementCount].value() = "")
```

This has two errors. First, `value` is a property and not a method, so there is no need for parentheses after it. Second, you have the all-time classic error of one equals sign instead of two. Remember that one equals sign means "Make it equal to," and two equals signs mean "Check if it is equal to." So with the changes, the line is:

```
if (theForm.elements[elementCount].value == "")
```

The next error is the failure to put your block of `if` code in curly braces. Even though JavaScript won't throw an error because the syntax is fine, the logic is not so fine, and you won't get the results you expect. With the braces, the `if` statement should be as follows:

```
if (theForm.elements[elementCount].value == "") {
    alert("Please complete all form elements")
    theForm.elements[elementCount].focus;
    formValid = false;
    break;
}
```

The penultimate error is in this line:

```
theForm.elements[elementCount].focus;
```

This time you have a method but with no parentheses after it. Even methods that have no parameters must have the empty parentheses after them if you intend to execute that method. So, corrected, the line is as follows:

```
theForm.elements[elementCount].focus();
```

Now you're almost done; there is just one more error. This time it's not something wrong with what's there, but rather something very important that should be there but is missing. What is it? It's this:

```
elementCount++;
```

This line should be in your `while` loop, otherwise `elementCount` will never go above `0` and the `while` loop's condition will always be `true`, resulting in the loop continuing forever: a classic infinite loop.