Arrays

Simple variables, like the ones you learned about in the previous section, only hold one piece of information, such as a number or a string value. They're perfect when you only need to keep track of a single thing like a score, an age, or a total cost. However, if you need to keep track of a bunch of related items—like the names of all of the days in a week, or a list of all of the images on a Web page—simple variables aren't very convenient.

Figure 2-4: *The power of variables: this page customizes its message based on a visitor's response.*

For example, say you've created a JavaScript shopping cart system that tracks items a visitor intends to buy. If you wanted to keep track of all of the items the visitor adds to her cart using simple variables you'd have to write code like this:

```
var item1 = 'Xbox 360';
var item2 = 'Tennis shoes';
var item3 = 'Gift certificate';
```

But what if they wanted to add more items than that? You'd have to create more variables—item4, item5, and so on. And, because you don't know how many items the visitor might want to buy, you really don't know how many variables you'll have to create.

Fortunately, JavaScript provides a better method of tracking a list of items, called an *array*. An array is a way of storing more than one value in a single place. Think of an array like a shopping list. When you need to go to the grocery store, you sit down and write a list of items to buy. If you just went shopping a few days earlier, the list might only contain a few items; but if your cupboard is bare, your shopping list might be quite long. Regardless of how many items on the list, though, there's still just a single list.

Without an array, you have to create a new variable for each item in the list. Imagine, for example, that you couldn't make a list of groceries on a single sheet of paper, but had to carry around individual slips of paper—one for each item that you're shopping for. If you wanted to add another item to buy, you'd need a new slip of paper; then you'd need to keep track of each slip as you shopped (see Figure 2-5). That's how simple variables work. But with an array you can create a single list of items, and even add, remove, or change items at anytime.

Figure 2-5:
An array provides a simple, organized way to track a list of related items. Adding another item to the list is just like writing a new item at the bottom of the list.

Creating an Array

To create and store items in an array, you first declare the array's name (just as you would a variable) and then supply a list of comma separated values: each value represents one item in the list. As with variables, what you name your array is up to you, but you need to follow the same naming rules listed on page 44. To indicate an array, you put the list of items between opening and closing brackets—[ ]. For example, to create an array containing abbreviations for the seven days of the week, you could write this code:

```
var days = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'];
```

The brackets—[ ]—are very important; they tell the JavaScript interpreter that it's dealing with an array. You can create an empty array without any elements like this:

```
var playList = [];
```

Creating an empty array is the equivalent of declaring a variable as described on page 43. You'll create an empty array when you don't add items to the array until the program is running. For example, the above array might be used to track songs that someone selects from a list on a Web page—you don't know ahead of time which songs the person will choose, so you declare an empty array and later fill it with items as the person selects music. (Adding items to an array is described on page 61.)

Note: When looking through other people's JavaScript programs (or other JavaScript books), you may encounter another way to create an array using the *Array* keyword, like this:

```
var days = new Array('Mon', 'Tues', 'Wed');
```

This method is valid, but the method used in this book (called an *array literal*) requires less typing and less code.

You can store any mix of values in an array. In other words, numbers, strings, and Boolean values can all appear in the same array:

```
var prefs = [1, 223, 'www.oreilly.com', false];
```

Note: You can even store arrays and other objects as elements inside an array. This can help store complex data. You'll see an example of this advanced topic on page 108.

The array examples above show the array created on a single line. However, if you've got a lot of items to add, or the items are long strings, trying to type all of that on a single line can make your program difficult to read. Another option many programmers use is to create an array over several lines, like this:

```
var authors = [ 'Ernest Hemingway',
                'Charlotte Bronte',
                'Dante Alighieri',
                'Emily Dickinson'
              ];
```

As mentioned in the box on page 47, a JavaScript interpreter skips extra space and line breaks, so even though this code is displayed on five lines, it's still just a single statement, as indicated by the final semicolon on the last line.

Tip: To make the names line up as above, you'd type the first line – *var authors = ['Ernest Hemingway',*–hit Return, then press the space key as many times as it takes to line up the next value–*'Charlotte Bronte',*.

Accessing Items in an Array

You can access the contents of a simple variable just by using the variable's name. For example *alert(lastName)* opens an alert box with the value stored in the variable *lastName*. However, because an array can hold more than one value, you can't just use its name alone to access the items it contains. A unique number, called an *index*, indicates the position of each item in an array. To access a particular item in an array, you use that item's index number. For example, say you've created an array with abbreviations for the days of the week, and want to open an alert box that displayed the first item. You could write this:

```
var days = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'];
alert(days[0]);
```

This code opens an alert box with 'Mon' in it. Arrays are *zero-indexed*, meaning that the *first* item in an array has an index value of 0, and the *second* item has an index value of 1: in other words, subtract one from the item's spot in the list to get its index value—the fifth item's index is 5 – 1; that is 4. Zero-indexing is pretty confusing when you first get started with programming, so Table 2-4 shows how the array *days* (from the above example) is indexed, the values it contains and how to access each value.

Table 2-4. *Items in an array must be accessed using an index number that's the equivalent to their place in the list minus 1*

Index value	Item	To access item
0	Mon	`days[0]`
1	Tues	`days[1]`
2	Wed	`days[2]`
3	Thurs	`days[3]`
4	Fri	`days[4]`
5	Sat	`days[5]`
6	Sun	`days[6]`

You can change the value of an item in an array by assigning a new value to the index position. For example, to put a new value into the first item in the array *days*, you could write this:

```
days[0] = 'Monday';
```

Because the index number of the last item in an array is always one less than the total number of items in an array, you only need to know how many items are in an array to access the last item. Fortunately, this is an easy task since every array has a length property, which contains the total number of items in the array. To access the length property, add a period followed by *length* after the array's name: for example, *days.length* returns the number of items in the array named *days* (if you created a different array, *playList*, for example, you'd get its length like this: *playList.length*). So you can use this tricky bit of JavaScript to access the value stored in the last item in the array:

```
days[days.length-1]
```

This last snippet of code demonstrates that you don't have to supply a literal number for an index (for example, the 0 in *days[0]*). You can also supply an equation that returns a valid number: in this case *days.length – 1* is a short equation: it first retrieves the number of items in the *days* array (that's 7 in this example) and subtracts 1 from it. So, in this case, *days[days.length-1]* translates to *days[6]*.

You can also use a variable containing a number as the index:

```
var i = 0;
alert(days[i]);
```

The last line of code is the equivalent of *alert(days[0]);*. You'll find this technique particularly useful when working with loops as described in the next chapter (page 90).

Adding Items to an Array

Say you've created an array to track items that a user clicks on a Web page. Each time the user clicks the page, an item is added to the array. JavaScript supplies several ways to add contents to an array.

Adding an item to the end of an array

To add an item to the end of an array, you can use the index notation from page 59, using an index value that's one greater than the last item in the list. For example, say you've have created an array named *properties*:

```
var properties = ['red', '14px', 'Arial'];
```

At this point, the array has three items. Remember that the last item is accessed using an index that's one less than the total number of items, so in this case, the last item in this array is properties[2]. To add another item, you could do this:

```
properties[3] = 'bold';
```

This line of code inserts *'bold'* into the fourth spot in the array, which creates an array with four elements: *['red', '14px', 'Arial', 'bold']*. Notice that when you add the new item, you use an index value that's equal to the total number of elements currently in the array, so you can be sure you're always adding an item to the end of an array by using the array's *length* property as the index. For example, you can rewrite the last line of code like this:

```
properties[properties.length] = 'bold';
```

You can also use an array's *push( )* command, which adds whatever you supply between the parentheses to the end of the array. As with the *length* property, you apply *push( )* by adding a period to the array's name followed by *push( )*. For example, here's another way to add an item to the end of the *properties* array:

```
properties.push('bold');
```

Whatever you supply inside the parentheses (in this example the string *'bold'*) is added as a new item at the end of the array. You can use any type of value, like a string, number, Boolean, or even a variable.

One advantage of the *push( )* command is that it lets you add more than one item to the array. For example, say you want to add three values to the end of an array named *properties*, you could do that like this:

```
properties.push('bold', 'italic', 'underlined');
```

Note: *push( )*, *unshift( )*, and the other commands associated with arrays are technically called array *methods*. In this book, when you see the word *method*, you can just think of it as a command that accomplishes a task.

Adding an item to the beginning of an array

If you want to add an item to the beginning of an array, use the *unshift( )* command. Here's an example of adding the 'bold' value to the beginning of the *properties* array:

```
var properties = ['red', '14px', 'Arial'];
properties.unshift('bold');
```

After this code runs, the array *properties* contains four elements: *['bold', 'red', '14px', 'Arial']*. As with *push( )*, you can use *unshift( )* to insert multiple items at the beginning of an array:

```
properties.unshift('bold', 'italic', 'underlined');
```

Note: Make sure you use the *name* of the array followed by a period and the method you wish to use. In other words, *push('new item')* won't work. You must first use the array's name (whatever name you gave the array when you created it) followed by a period, then the method like this: *authors.push('Stephen King');*.

Choosing how to add items to an array

So far, this chapter has shown you three ways to add items to an array. Table 2-5 compares these techniques. Each of these commands accomplishes similar tasks, so the one you choose depends on the circumstances of your program. If the order that the items are stored in the array doesn't matter, then any of these methods work. For example, say you have a page of product pictures, and clicking one picture adds the product to a shopping cart. You use an array to store the cart items. The order the items appear in the cart (or the array) doesn't matter, so you can use any of these techniques.

However, if you create an array that keeps track of the order in which something happens, then the method you choose does matter. For example, say you've created a page that lets visitors create a playlist of songs by clicking song names on the page. Since a playlist lists songs in the order they should be played, the order is important. So if each time the visitor clicks a song, the song's name should go at the end of the playlist (so it will be the last song played), then use the *push( )* method.

Table 2-5. *Various ways of adding elements to an array*

Method	Original array	Example code	Resulting array	Explanation
.length property	`var p = [0,1,2,3]`	`p[p.length]=4`	[0,1,2,3,4]	Adds one value to the end of an array.
push( )	`var p = [0,1,2,3]`	`p.push(4,5,6)`	[0,1,2,3,4,5,6]	Adds one or more items to the end of an array.

Table 2-5. *Various ways of adding elements to an array (continued)*

Method	Original array	Example code	Resulting array	Explanation
unshift()	`var p = [0,1,2,3]`	`p.unshift(4,5)`	[4,5,0,1,2,3]	Adds one or more item to the beginning of an array.

The *push( )* and *unshift( )* commands return a value (see the Note on the page 55). To be specific, once *push( )* and *unshift( )* complete their tasks, they supply the number of items that are in the array. Here's an example:

```
var  p = [0,1,2,3];
var totalItems = p.push(4,5);
```

After this code runs, the value stored in *totalItems* is 6, because there are six items in the *p* array.

POWER USERS' CLINIC

Creating a Queue

The methods used to add items to an array–*push( )* and *unshift( )*–and the methods used to remove items from an array–*pop( )* and *shift( )*–are often used together to provide a way of accessing items in the order they were created. A classic example is a musical playlist. You create the list by adding songs to it; then, as you play each song, it's removed from the list. The songs are played in the order they appear in the list, so the first song is played and then removed from the list. This arrangement is similar to a line at the movies. When you arrive at the movie theater, you take your place at the end of the line; when the movie's about to begin, the doors open and the first person in line is the first to get in.

In programming circles, this concept is called FIFO for "First In, First Out." You can simulate this arrangement using arrays and the *push( )* and *shift( )* commands. For example say you had an array named *playlist*. To add a new song to the end of the list you'd use *push( )* like this:

```
playlist.push('Yellow Submarine');
```

To get the song that's supposed to play next, you get the first item in the list like this:

```
nowPlaying = playlist.shift( );
```

This code removes the first item from the array and stores it in a variable named *nowPlaying*. The FIFO concept is useful for creating and managing queues such as a playlist, a to-do list, or a slideshow of images.

Deleting Items from an Array

If you want to remove an item from the end or beginning of an array, use the *pop( )* or *shift( )* commands. Both commands remove one item from the array: the *pop( )* command removes the item from the end of the array, while *shift( )* removes one item from the beginning. Table 2-6 compares the two methods.

Table 2-6. *Two ways of removing an item from an array*

Method	Original array	Example code	Resulting array	Explanation
pop()	`var p = [0,1,2,3]`	p.pop()	`[0,1,2]`	Removes the last item from the array.
shift()	`var p = [0,1,2,3]`	p.shift()	`[1,2,3]`	Removes the first item from the array.

As with *push( )* and *unshift( )*, *pop( )* and *shift( )* return a value once they've completed their tasks of removing an item from an array. In fact, they return the value that they just removed. So, for example, this code removes a value and stores it in the variable *removedItem*:

```
var  p = [0,1,2,3];
var removedItem = p.pop( );
```

The value of *removedItem* after this code runs is 3 and the array *p* now contains [0,1,2].

Note: This chapter's files include a Web page that lets you interactively test out the different array commands. It's named *array_methods.html* and it's in the *tutorials → chapter02* folder. Open the file in a Web browser and click the various buttons on the Web page to see how the array methods work. (By the way, all the cool interactivity of that page is all thanks to JavaScript.)

Adding and Deleting with *splice( )*

The techniques in the previous sections for adding and removing array items only work for the beginning and end of arrays. What if you want to insert an item in the middle of an array, or remove the item that's in the third position in the array? For example, say you write a program that lets visitors create slideshows by selecting images from a Web page. You could store their selections (for example, information about each image such as the *src* attribute) in an array. However, the visitor may want to edit his selections—perhaps remove one of the pictures he previously selected.

JavaScript provides one command—*splice( )*—that lets you add items to an array and delete items from an array. It's a powerful command and a little hard to understand, so we'll explain it in stages.

Deleting items with splice( )

To remove items from an array, tell the *splice( )* command where it should begin deleting (the index number of the first item to remove) and how many items it should delete. For example, say you create an array named fruits like this:

```
var fruit=['apple','pear','kiwi','pomegranate'];
```

This code creates an array of four items. To remove 'pear' and 'kiwi' from the array, you need to tell *splice( )* to begin with the second item (which has an index of 1, remember) and delete two items like this:

```
fruit.splice(1,2);
```

The result as diagrammed in Figure 2-6, is an array with just two strings—'apple' and 'pomegranate'—left.

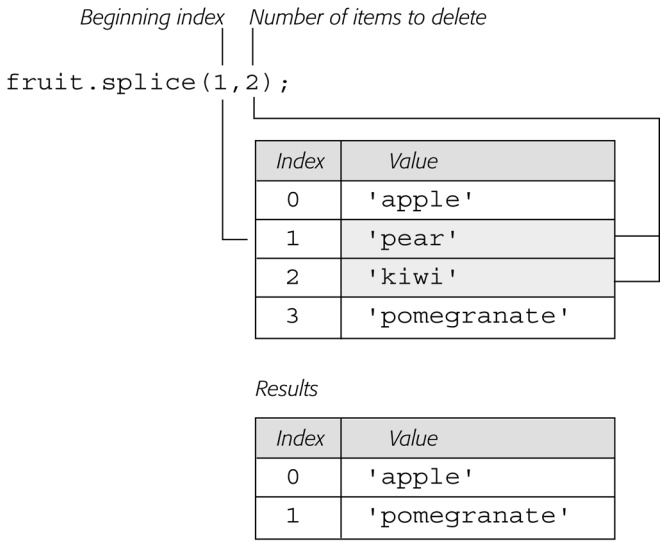


Figure 2-6: *The* splice( ) *method requires two pieces of information to delete elements from an array: the index value of where to start hacking away, and the number of elements to remove.*

Adding items with splice( )

The *splice( )* command does double duty: it can also add items in the middle of an array. To use *splice( )* in this way, provide the index value where the new items should be located, 0 to indicate that you don't want to delete any items, then the list of items to insert: one or more values separated by commas. For example, say you start out with the *fruit* array again:

```
var fruit=['apple','pear','kiwi','pomegranate'];
```

If you want to add two items in between 'pear' and 'kiwi' in this list, you can use *splice( )* like this:

```
fruit.splice(2,0,'grape','orange');
```

This code adds two strings—'grape' and 'orange'—starting at index 2. In other words, 'grape' becomes the third item in the list, 'orange' the fourth, and 'kiwi' and 'pomegranate' are moved to the end. You can see this diagrammed in Figure 2-7.

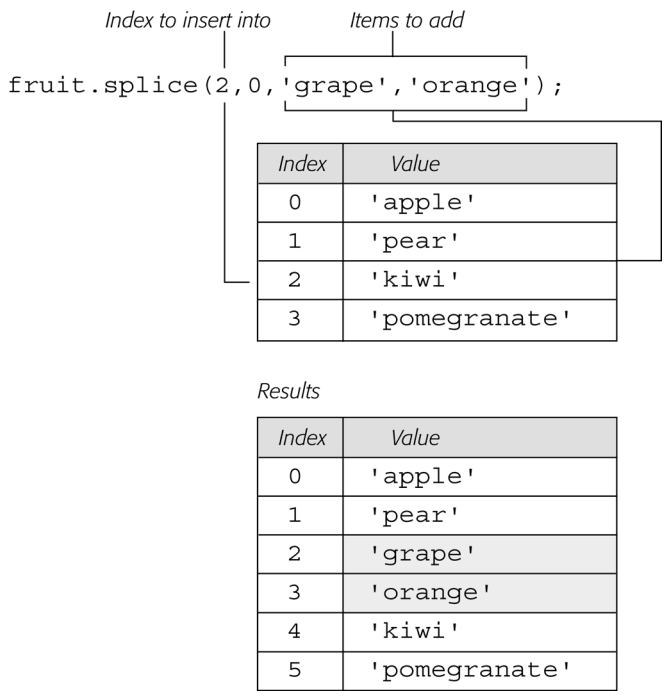


Figure 2-7:
Add items in the middle of an array using splice( ). *The first number you provide* splice( ) *represents the index position in the array where the new items will go. Make sure the second number is a 0; otherwise, you'll also delete elements from the array as you insert new items.*

Replacing items with splice( )

If you want to get really tricky, you can add and delete elements from an array in a single operation. This maneuver can come in handy when you want to replace one or more elements in an array with new items, for example, if someone wants to replace one song in a playlist with another song.

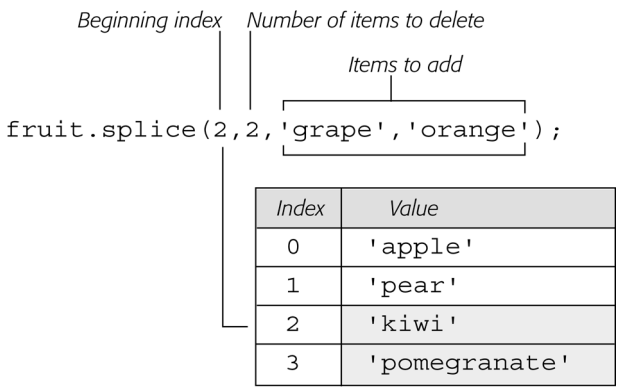
The process is the same as for adding an item, but instead of specifying 0 for the second piece of information that you supply *splice( )*, you indicate the number of items you wish to remove. So, if you start with the fruit array again:

```
var fruit=['apple','pear','kiwi','pomegranate'];
```

Say you want to replace both 'kiwi' and 'pomegranate' with 'grape' and 'orange', you can write this statement:

```
fruit.splice(2,2,'grape','orange');
```

In this case, the first 2 identifies which index position to start at, the second 2 specifies how many items to remove, and the other items indicate what should replace the deleted items. See Figure 2-8 for a clear picture of the process.



Index	Value
0	'apple'
1	'pear'
2	'kiwi'
3	'pomegranate'

Results

Index	Value
0	'apple'
1	'pear'
2	'grape'
3	'orange'

Figure 2-8:
When you want to replace items in an array with new items, you can turn to the splice( ) *method.*

Tutorial: Writing to a Web Page Using Arrays

You'll use arrays in many of the scripts in this book, but to get a quick taste of creating and using arrays, try this short tutorial.

Note: See the note on page 27 for information on how to download the tutorial files.

1. **In a text editor, open the file *2.3.html* in the *chapter02* folder.**

   You'll start by simply creating an array containing four strings. As with the previous tutorial, this file already contains <script> tags in both the head and body regions.

2. **Between the first set of <script> tags, type the bolded code:**

```
<script type="text/javascript">
var authors = [ 'Ernest Hemingway',
                'Charlotte Bronte',
                'Dante Alighieri',
                'Emily Dickinson'
              ];
</script>
```

This code comprises a single JavaScript statement, but it's broken over five lines. To create it, type the first line—*var authors = ['Ernest Hemingway',*—hit Return, then press the Space bar until you line up under the ' (about 16 spaces), and then type *'Charlotte Bronte',*.

Note: Most HTML editors use a *monospaced* font like Courier or Courier New for your HTML and JavaScript code. In a monospaced font, each character is the same width as every other character, so it's easy to line up columns (like all the author names in this example). If your text editor doesn't use Courier or something similar, you may not be able to line up the names perfectly.

As mentioned on page 59, when you create an array with lots of elements, you can make your code easier to read if you break it over several lines. You can tell it's a single statement since there's no semicolon until the end of line 5.

This line of code creates an array named *authors* and stores the names of 4 authors (4 string values) into the array. Next, you'll access an element of the array.

3. **Locate the second set of <script> tags, and add the code in bold:**

```
<script type="text/javascript">
document.write('<p>The first author is <strong>');
document.write(authors[0] + '</strong></p>');
</script>
```

The first line starts a new paragraph with some text and an opening tag—just plain HTML. The next line prints the value stored in the first item of the authors array and prints the closing and </p> tags to create a complete HTML paragraph. To access the first item in an array, you use a 0 as the index—*authors[0]*—instead of 1.

At this point, it's a good idea to save your file and preview it in a Web browser. You should see "The first author is **Ernest Hemingway**" printed on the screen. If you don't, you may have made a typo either when you created the array in step 2 or 3.

Note: Remember to use the Firefox Error Console described on page 34 to help you locate the source of any JavaScript errors.

---

4. **Return to your text editor and add the two lines of bolded code below to your script:**

```
document.write('<p>The last author is <strong>');
document.write(authors[4] + '</strong></p>');
```

This step is pretty much the same as the previous one, except that you're printing a different array item. Save the page and preview it in a browser. You'll see "undefined" in place of an author's name (see Figure 2-9). Don't worry; that's

intentional. Remember that an array's index values begin at 0, so the last item is actually the total number of items in the array minus 1. In this case, there are four strings stored in the authors array, so that last item would actually be accessed with *authors[3]*.

Note: If you try to read the value of an item using an index value that doesn't exist, you'll end up with the JavaScript "undefined" value. All that means is that there's no value stored in that index position.

Fortunately, there's an easy technique for retrieving the last item in an array no matter how many items are stored in the array.

Figure 2-9: *If you try to access an array element that doesn't exist, then you'll end up with the value "undefined."*

5. **Return to your text editor and edit the code you just entered. Erase the *4* and add the bolded code in its place:**

```
document.write('<p>The last author is <strong>');
document.write(authors[authors.length-1] + '</strong></p>');
```

As you'll recall from page 60, an array's length property stores the number of items in the array. So the total number of items in the *authors* array can be found with this code *authors.length*. At this point in the script, that turns out to be 4.

Knowing that the index value of the last item in an array is always 1 less than the total number of items in an array, you just subtract one from the total to get the index number of the last item: *authors.length-1*. You can provide that little equation as the index value when accessing the last item in an array: *authors[authors.length-1]*.

You'll finish up by adding one more item to the beginning of the array.

6. **Add another line of code after the ones you added in step 5:**

```
authors.unshift('Stan Lee');
```

As you read on page 62, the *unshift( )* method adds one or more items to the beginning of an array. After this line of code runs the authors array will now be *['Stan Lee', 'Ernest Hemingway',*

Finally, you'll print out the newly added item on the page.

7. **Add the three more lines (bolded below) so that your final code looks like this:**

```
document.write('<p>The first author is <strong>');
document.write(authors[0] + '</strong></p>');
document.write('<p>The last author is <strong>');
document.write(authors[authors.length-1] + '</strong></p>');
authors.unshift('Stan Lee');
document.write('<p>I almost forgot <strong>');
document.write(authors[0]);
document.write('</strong></p>');
```

Save the file and preview it in a Web browser. You should see something like Figure 2-10. If you don't, remember the error console in Firefox can help you locate the error (page 34).

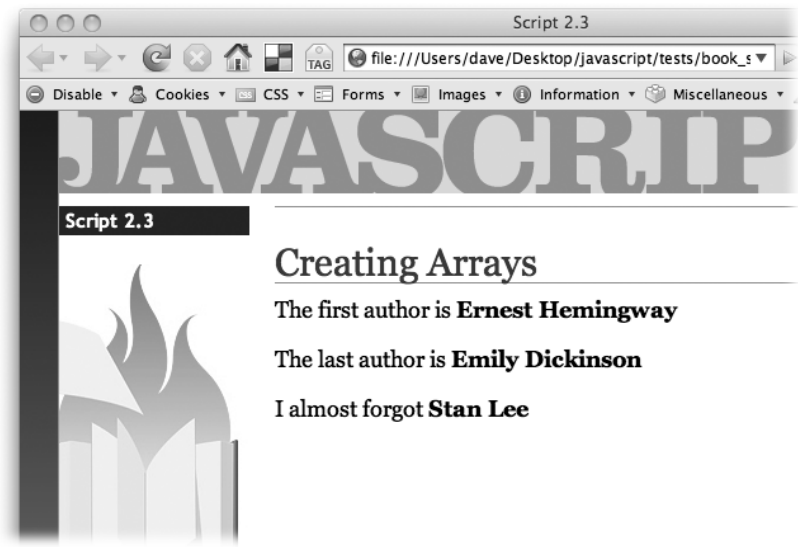


Figure 2-10: *OK, Stan Lee may not be your idea of a literary giant, but at least he's helping you learn how arrays work.*