11.13 Arrays

Overview

Normal objects have different names for each of the object's different data items. For example, a dog object has `name`, `sound`, and `trick` properties for its different data items. Now we'll look at a special type of object that holds multiple data items and uses a common name for all of them. That special type of object is an array.

In the past, we've used the term *collection* to describe a group of items with a common name. An array is not only a special type of object; it's also a special type of collection. Remember in the last chapter how we retrieved a collection of t-shirt color radio buttons using the following code?

```
tshirtRBs = form.elements["color"];
```

And then later, we used the `tshirtRBs` name to access individual elements within the collection. Here's an example:

```
currentRB = tshirtRBs[i];
```

In `tshirtRBs[i]`, the `i` is an *index variable*, and it allows you to access an element within the collection. Arrays also use index variables to access individual elements.

So, what's the difference between a collection and an array? The browser engine builds collections for control groups (like radio buttons and checkboxes) automatically when it creates the DOM node tree. On the other hand, with arrays, it's up to you, the web developer, to instantiate an array before you can use it. To instantiate a class, you first have to define the class and specify the class's properties and methods. Instantiating an array is easier. You use the predefined `Array` object (that's `Array` with a capital A) with the `new` operator to instantiate an array. For example:

```
stuntGroup = new Array("Ellie", "Caiden", "Alexa",
  "Olivia");
```

That code creates an array of girls for a stunt group within my daughter Caiden's cheerleading squad. The instantiation syntax should look familiar because we've used it for instantiating objects in prior examples. But now we're using "Array" instead of a class name.

It's very common to use a `for` loop to loop through the elements of an array. For example, suppose you have a web page with the preceding `stuntGroup` assignment, and within the `body` container, you have `<div id="stunt-group"></div>`. To insert the stunt group girls' names within that `div` container, you could use this code:

```
var names = "";

for (let i=0; i<stuntGroup.length; i++) {
  names += stuntGroup[i] + "<br>";
}
document.getElementById("stunt-group").innerHTML =
  names;
```

In the `for` loop heading, note the `stuntGroup` array's `length` property. It works the same as the `length` property for radio button and checkbox collections—it returns the number of elements in the array. Note that the index variable `i` gets initialized to 0. As with collections, an array's first element is at index position 0.

Continuing with the stunt group example, suppose Caiden suffers a concussion from a fall. She has to sit out, and she gets replaced by Coach Ayers. Here's the code that models that behavior (assuming "Caiden" is stored in the `stuntGroup` array's second element):

```
stuntGroup[1] = "Coach Ayers";
```

After Caiden regains consciousness, she rejoins her teammates as a fifth member of the stunt group, with Coach Ayers continuing to help out when Caiden gets disoriented. Here's the code that models Caiden rejoining the stunt group:

```
stuntGroup[4] = "Caiden";
```

The assignment statement dynamically expands the array's size by adding a fifth element to the array at index position 4.

You've already seen how to instantiate an array with initial values for the array's elements—`new Array(`*comma-separated-list-of-initial-values*`)`. As an alternative, you can instantiate an empty array and then later assign values into the array. For example, we could have started with an empty stuntGroup array and implemented an "Add cheerleader" button with an onclick event handler that adds a user-entered name to the array. Here's a declaration for an empty stuntGroup array, plus a function that could be used for an "Add cheerleader" button's event handler:

```
var stuntGroup = new Array();

function addCheerleader() {
  stuntGroup[stuntGroup.length] =
    document.getElementById("cheerleader").value;
} // end addCheerleader
```

Note how the function uses the array's length property as an index for the new element. Before any cheerleaders have been added, the array is empty with a length of 0. So the first time addCheerleader gets called, the assignment statement's left side refers to an element at index position 0. As a result of the assignment, that element becomes the first element in the array. The assignment statement's right side retrieves the contents of a text control with an id value of "cheerleader".

As you've seen before, JavaScript sometimes provides more than one way to accomplish the same thing. As a more elegant alternative to using stuntGroup.length as shown earlier, we can use the Array object's push method to add an element to the end of the array. Here's the code:

```
stuntGroup.push(document.getElementById("cheerleader").value);
```

Methods

The Array object has quite a few methods, and every array you instantiate inherits those methods. Take a look at **FIGURE 11.16**. It shows a few of the more popular Array methods, starting with the push method, which you've already seen. When you call the push, reverse, sort, or splice method, the primary purpose is to modify the calling object array. That's different from when you call the concat or indexOf methods. With those methods, the purpose is to return a value, and the calling object array remains intact, unmodified.

The reverse method is self-explanatory. It reverses the order of the calling object array's elements.

The sort method can be a little tricky. If you have an array of strings, the sort method uses lexicographical ordering. As you might recall from Chapter 9, lexicographical ordering is pretty much the same as dictionary ordering. Here's an example:

```
stuntGroup = new Array("Ellie", "Caiden", "Alexa", "Olivia");
stuntGroup.sort();
```

array-variable`.concat(`*another-array*`)`
Returns a new array that is formed by concatenating the passed-in array to the end of the calling object array. The calling object array does not change.

array-variable`.indexOf(`*value*`)`
Returns the index of the first element that holds the specified value. Returns -1 if not found.

array-variable`.push(`*value*`)`
Creates an element with the specified value and adds that element to the end of the array. To add multiple elements to the end of an array, use commas to separate the element values. It returns the new length of the array.

array-variable`.reverse()`
Reverses the array's elements, so the first array element becomes the last, and the last array element becomes the first. It returns the reversed array.

array-variable`.sort()`
Sorts the array's elements, so the elements with smaller values move to the start of the array. It returns the sorted array.

array-variable`.splice(`*start, delete-count, value1, value2, ...*`)`
Removes elements from the array and/or inserts elements into the array. The *start* parameter specifies the index position where elements are removed and/or inserted. The *delete-count* parameter specifies the number of elements removed. The value parameters specify the values for inserted elements. It returns an array of the deleted elements or an empty array if no elements are removed.

FIGURE 11.16 Some of the more popular `Array` methods—their headings and descriptions

After this code executes, "Alexa" moves to the first position, followed by "Caiden," "Ellie," and "Olivia." Lexicographical ordering is different from dictionary ordering in that lowercase letters go after uppercase letters. As a new-age hipster, Ellie sometimes prefers lowercase *e* for the first letter in her name ("ellie"). With that spelling, the `sort` method would move "ellie" to the last position in the array.

By default, if you have an array of numbers, the `sort` method first converts the numbers to strings and then sorts using lexicographical ordering. So if you have an array that holds 4, 196, 8, and 23 initially, the array holds 196, 23, 4, and 8 after sorting. That's because the "1" character in "196" comes before the "2" character in "23," and the "2" comes before "4," which comes before "8." In the next section, you'll learn how to sort numbers in standard numerical order.

The `splice` method removes elements from the calling object array and/or inserts elements into the calling object array. When you call the `splice` method, the first argument specifies the index position where the removal and/or insertion takes place. The second argument specifies the number of elements that are to be removed from the array. If there is no second argument, then the `splice` method removes all the elements starting at the first argument's position. If the second argument is 0, that means no elements are removed. Subsequent arguments specify the values for elements that are to be inserted into the array. We'll show an example that uses the `splice` method after first describing the `concat` and `indexOf` methods.

The `concat` method returns a new array that is formed by concatenating the argument array to the end of the calling object array. Here's an example:

```
megaStuntGroup = stuntGroup1.concat(stuntGroup2);
```

After that code executes, the `megaStuntGroup` array holds all of the elements from `stuntGroup1` plus `stuntGroup2`. The `concat` method does not impact the original arrays, so `stuntGroup1` and `stuntGroup2` are unchanged.

The `indexOf` method searches for a specified value within the calling object array's elements. It returns the index of the first element that holds the value. It returns -1 if the value is not found.

To illustrate how the `indexOf` and `splice` methods work, we need another stunt group example. Let's check in on Caiden and see how she's doing with her concussion recovery. After two weeks, she's finally able to remember her name. Way to go, Caiden! With that hurdle cleared, Coach Ayers leaves the stunt group and returns to full coaching duties. The following code models Coach Ayers leaving the stunt group:

```
deletePosition = stuntGroup.indexOf("Coach Ayers");
stuntGroup.splice(deletePosition, 1);
```

Note how we use the `indexOf` method to search for Coach Ayers and then use the `splice` method to remove Coach Ayers's element.