Chapter 17
JavaScript Arrays
Arrays as Objects

earlier, we introduced the concept of software objects: programming structures that encompass both data (properties) and operations (methods)

- e.g., a string object has properties (e.g., length) and methods (e.g., charAt) that enable the programmer to easily store and manipulate text

like strings, JavaScript arrays are objects that encapsulate multiple values and their associated properties and methods

- unlike strings (which store only text characters), the items in an array can be of any data type

- an array consists of a sequence of items, enclosed in square brackets and separated by commas

- when assigning values to arrays:
  - an item can appear in an array more than once
  - array items can be specified as expressions
  - arrays can be empty

```javascript
responses = ['yes', 'no', 'maybe'];
nums = [1, 2, 3, 2+1, 7*7, 2*5-1];
misc = [1.234, 'foo', 7-5, true, 3, 'foo'];
empty = [];
```
since an array stores a series of values, its associated memory cell can be envisioned as divided into components, each containing an individual value

array items are assigned sequential indices, allowing programmers to identify an item by its corresponding index

array access is accomplished by specifying the name of the array object, followed by the index of the desired item in brackets

- the first item is misc[0], the second is misc[1], ..., the last is misc[length-1]
- if an index is specified beyond the scope of an array, the access yields undefined
we now can understand how the RandomOneOf function from random.js works

- the input to the function is a nonempty array of arbitrary items
- the RandomInt function is called to pick a random index from the array (between 0 and the last index, list.length-1)
- the bracket notation is used to access the item at that index, which is then returned by the function

```javascript
function RandomOneOf(list)
// Given : list is a nonempty list (array)
// Returns: a random item from the list
{
    var randomIndex;
    randomIndex = RandomInt(0, list.length-1);
    return list[randomIndex];
}
```
Assigning Items in an Array

array items can be assigned values just like any variable

- suppose the array `misc` has been assigned to store the following

  ```
  misc = [1.234, 'foo', 7-5, true, 3, 'foo'];
  ```

- the assignment `misc[0] = 1000;` would store the value 1000 as the first item in the array, overwriting the value that was previously there
Assigning Items in an Array

Array items can be assigned values just like any variable

- Suppose the array `misc` has been assigned to store the following

```plaintext
misc = [1.234, 'foo', 7-5, true, 3, 'foo'];
```

- The assignment `misc[0] = 1000;` would store the value 1000 as the first item in the array, overwriting the value that was previously there.

- If the index in an assignment statement is beyond the array’s current length, the array will automatically expand to accommodate the new item.
  - The assignment `misc[8] = 'oops';` would store 'oops' at index 8.
From Strings to Arrays

so far, our Web pages have handled each input from the user as a single string

- this approach is limiting since many programming tasks involve separately processing an arbitrary number of words or numbers entered by the user

- recall the Pig Latin page

- we might want to generalize the page so that it translates entire phrases instead of just words

- that is, the user would enter an arbitrary sequence of words and each word would be translated

- this task would be very difficult using our current set of programming tools
JavaScript strings provide a method, `split`, for easily accessing the components of a string

- the only input required by the `split` method is a character (or sequence of characters) that serves as a delimiter for breaking apart the string
- the `split` method separates the string into component substrings at each occurrence of the delimiter, and returns an array consisting of those substrings

```javascript
user = 'Grace Murray Hopper';
arr1 = user.split(' '); // ASSIGNS arr1 to be the array
//   ['Grace', 'Murray', 'Hopper']
```

<table>
<thead>
<tr>
<th><code>phrase.split(' ')</code></th>
<th>breaks the string <code>phrase</code> into an array of items, delimited by a single space</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>phrase.split(': ')</code></td>
<td>breaks the string <code>phrase</code> into an array of items, delimited by a colon followed by a space</td>
</tr>
<tr>
<td><code>phrase.split(/\[ \t\n,\]/)</code></td>
<td>breaks the string <code>phrase</code> into an array of items, delimited by a single space, tab (<code>\t</code>), newline (<code>\n</code>), or comma</td>
</tr>
<tr>
<td><code>phrase.split(/\[ \t\n,\]+)</code></td>
<td>breaks the string <code>phrase</code> into an array of items, delimited by any sequence of spaces, tabs (<code>\t</code>), newlines (<code>\n</code>), and/or commas</td>
</tr>
</tbody>
</table>
JavaScript strings provide a method, `split`, for easily accessing the components of a string:

- The only input required by the `split` method is a character (or sequence of characters) that serves as a delimiter for breaking apart the string.
- The `split` method separates the string into component substrings at each occurrence of the delimiter, and returns an array consisting of those substrings.

```javascript
user = 'Grace Murray Hopper';
arr1 = user.split(' ');  // ASSIGNS arr1 to be the array
                        //   ['Grace', 'Murray', 'Hopper']
```

- As was the case with strings, `[/[ ... ]/]` can be used to specify groups of characters.

```
phrase.split(' ')  // breaks the string phrase into an array of items, delimited by a single space
phrase.split(': ') // breaks the string phrase into an array of items, delimited by a colon followed by a space
phrase.split(/\s/)  // breaks the string phrase into an array of items, delimited by a single space, tab (\t), newline (\n), or comma
phrase.split(/\s+/) // breaks the string phrase into an array of items, delimited by any sequence of spaces, tabs (\t), newlines (\n), and/or commas
```
Example: Generating Acronyms

one interesting application of the split method is in deriving acronyms

- **acronyms** (terms made from the first letters of words in a phrase) are especially popular in the computing field
  - RAM – Random Access Memory
  - GUI – Graphical User Interface
  - WWW – World Wide Web

```javascript
function Acronym(phrase) {
  // Assumes: phrase is a string of words
  // Returns: the acronym made up of first letters from the phrase
  var words, acronym, index, nextWord;
  words = phrase.split(/[^\t,\s]+/); // CONVERT phrase TO AN ARRAY
  acronym = ''; // INITIALIZE THE acronym
  index = 0; // START AT FIRST WORD
  while (index < words.length) {
    nextWord = words[index]; // AS LONG AS WORDS LEFT
    acronym = acronym + nextWord.charAt(0); // ADD FIRST CHAR OF WORD
    index = index + 1; // GO ON TO NEXT WORD
  }
  return acronym.toUpperCase(); // RETURN UPPER CASE acronym
}
```

the Acronym function in arrays.js takes a phrase, such as "What you see is what you get", splits it into an array of individual words, then extracts the first letters to construct the acronym
the Acronym function can be integrated into a Web page
- the user enters a phrase in a text area
- a button is defined to call the Acronym function with the text box contents as input
- the result returned by the function call is displayed in another text box
Arrays of Numbers

some applications involve reading and manipulating sequences of numbers
  ▪ e.g., suppose we wanted to calculate the average of some number of grades
  ▪ we could enter the numbers in one big string, separated by spaces, then
    ▪ use the split function to separate out the individual numbers
    ▪ then, traverse the resulting array and calculate the sum and average of the numbers

note: even if the input string contains numerical digits, split returns each array item as a string (similar to text box/area access)
  ▪ arrays.js contains a function that traverses an array and converts each item to a number

```javascript
function ParseArray(strArray) {
  // Assumes: strArray is an array of strings representing numbers
  // Returns: a copy of array with items converted to numbers
  
  var numArray, index;
  numArray = [ ];   // CREATE EMPTY ARRAY TO STORE NUMS
  index = 0;        // FOR EACH ITEM IN strArray
  while (index < strArray.length) {
    numArray[index] = parseFloat(strArray[index]); // CONVERT TO NUMBER AND STORE
    index = index + 1;
  }
  return numArray;  // FINALLY, RETURN THE NUMS
}
```
this page averages an arbitrary number of grades (entered as a string in a text box)
  • utilizes split to split the string into an array of strings
  • utilizes ParseArray to convert into an array of numbers
Example: Dice Stats

recall our web page for simulating repeated dice rolls and recording the number of times that a specific total was obtained

- statistical analysis predicts that, given a large number of dice rolls, the distribution of totals will closely mirror the percentages listed below

<table>
<thead>
<tr>
<th>Dice total</th>
<th>Likelihood of Obtaining That Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.8%</td>
</tr>
<tr>
<td>3</td>
<td>5.6%</td>
</tr>
<tr>
<td>4</td>
<td>8.3%</td>
</tr>
<tr>
<td>5</td>
<td>11.1%</td>
</tr>
<tr>
<td>6</td>
<td>13.9%</td>
</tr>
<tr>
<td>7</td>
<td>16.7%</td>
</tr>
<tr>
<td>8</td>
<td>13.9%</td>
</tr>
<tr>
<td>9</td>
<td>11.1%</td>
</tr>
<tr>
<td>10</td>
<td>8.3%</td>
</tr>
<tr>
<td>11</td>
<td>5.6%</td>
</tr>
<tr>
<td>12</td>
<td>2.8%</td>
</tr>
</tbody>
</table>
Approach 1: Separate Counters

to obtain a valid distribution of dice totals, we would need to simulate a large number of rolls and simultaneously count the occurrences of every total

- this can be accomplished by defining 11 counters, each corresponding to a particular total
- however, this would be extremely tedious

```java
count2 = 0; // INITIALIZE EACH
count3 = 0;
count4 = 0;

rep = 0; // INITIALIZE rep COUNTER
while (rep < 1000) { // AS LONG AS rolls REMAIN
    roll = RandomInt(1, 6) + RandomInt(1, 6); // GET NEXT ROLL
    if (roll == 2) { // IF ROLLED 2,
        count2 = count2 + 1;
        count3 = count3 + 1;
    } else if (roll == 3) { // ELSE IF ROLLED 3,
        count3 = count3 + 1;
    } else if (roll == 4) { // ELSE IF ROLLED 4,
        count4 = count4 + 1;
        count3 = count3 + 1;
    } else if (roll == 5) { // ELSE IF ROLLED 5,
        count5 = count5 + 1;
        count4 = count4 + 1;
    } else if (roll == 6) { // ELSE IF ROLLED 6,
        count6 = count6 + 1;
        count5 = count5 + 1;
    } else if (roll == 7) { // ELSE IF ROLLED 7,
        count7 = count7 + 1;
        count6 = count6 + 1;
    } else if (roll == 8) { // ELSE IF ROLLED 8,
        count8 = count8 + 1;
        count7 = count7 + 1;
    } else if (roll == 9) { // ELSE IF ROLLED 9,
        count9 = count9 + 1;
        count8 = count8 + 1;
    } else if (roll == 10) { // ELSE IF ROLLED 10,
        count10 = count10 + 1;
        count9 = count9 + 1;
    } else if (roll == 11) { // ELSE IF ROLLED 11,
        count11 = count11 + 1;
        count10 = count10 + 1;
    } else if (roll == 12) { // ELSE IF ROLLED 12,
        count12 = count12 + 1;
        count11 = count11 + 1;
    }
    rep = rep + 1; // GO ON TO NEXT REP
}
```

- requires separate assignment statements for all 11 counters
- requires a cascading if-else statement with 11 cases
- not easily generalized – what if we wanted to use 8-sided dice?
Approach 2: Array of Counters

instead of representing each counter as a separate variable, we can define the counters as items in an array

- all 11 counters can be stored in an array and initialized via a single assignment statement
- any individual counter can be accessed and updated via its corresponding index
  - since the first possible total is 2, its count is stored in index 0
  - the next possible total, 3, has its count stored in index 1
  - ...
  - for an arbitrary roll, its count is stored in index roll-2

- the resulting code is shorter, simpler, and easier to generalize

```c
count = [0,0,0,0,0,0,0,0,0,0];  // INITIALIZE ALL COUNTERS
rep = 0;
while (rep < 1000) {
    roll = RandomInt(1, 6) + RandomInt(1, 6);  // GET RANDOM ROLL OF DICE
    count[roll-2] = count[roll-2] + 1;  // ADD 1 TO COUNTER
    rep=rep+1;  // GO ONTO NEXT REP
}
```

Sunday, December 4, 2011
code for maintaining statistics on repeated dice rolls can be integrated into a Web page

- the number of rolls to be simulated is entered by the user into a text box
- a button is defined to call the code for repeatedly simulating the roll and maintaining stats, with the text box contents as input
- the numbers of rolls for each total are then displayed in a page division
Example: ASCII Animations

ASCII art (pictures made of keyboard characters) was popular in the 70s-80s

- can generalize still pictures to moving animations
- recall: a movie is a sequence of frames (individual images) that are displayed in rapid succession
- can build an animation out of a sequence of ASCII art frames

When played in succession, these ASCII art frames would appear as a stick person doing jumping jacks.
ASCII Animation Page

- the user enters the frames in a text area, separated by '=====
- frames are extracted using split
- setInterval is used to display successive frames every ¼ second

```html
1. <!doctype html>
2. <html>
3. <head>
4. <title>ASCII Animation</title>
5. <script type="text/javascript">
6. function PlayAnimation() {
7.   // Assumes: frameArea contains the text of an ASCII animation
8.   // Results: displays each frame of the animation in succession (0.25 sec apart)
9.   var frameStr;
10.  frameStr = document.getElementById('frameArea').value;
11.  if (frameStr.indexOf('\\n') == -1) {
12.    frameSeq = frameStr.split('====\\n');
13.  } else {
14.    frameSeq = frameStr.split('====\n');
15.  }
16.  currentFrame = 0;
17.  setInterval('ShowNextFrame()', 250);
18. }
19. function ShowNextFrame() {
20.   // Assumes: frameSeq is an array of animation frames, and
21.   // currentFrame is the index of the current frame
22.   // Results: displays the current frame in displayArea & increments the index
23.   document.getElementById('displayArea').value = frameSeq[currentFrame];
24.   currentFrame = (currentFrame + 1) % frameSeq.length;
25. }
26. </script>
27. </head>
28. <body style="text-align:center">
29. <h2>ASCII Animation Editor/Viewer</h2>
30. <table style="margin-left:auto;margin-right:auto">
31.  <tr><td style="text-align:center">
32.   Enter the frames below, separated by "====".
33.  </td></tr>
34.  <tr><td style="text-align:center">
35.   <input type="button" value="Play the Animation" onclick="PlayAnimation();">
36.  </td></tr>
37.  <tr><td id="frameArea" rows=25 cols=55 style="font-size:8pt">
38.     \__
39.     \___
40.     \_____
41.     \_
42.   </td></tr>
43.  <tr><td id="displayArea" rows=25 cols=55 style="font-size:8pt">
44.   \_
45.   \___
46.   \_____
47.   \_
48.   </td></tr>
49. </table>
50. </body>
51. </html>
```