Chapter 15
JavaScript Strings
Strings as Objects

so far, your interactive Web pages have manipulated strings in simple ways
- use text box to input a word or phrase
- store that text in a (string) variable
- incorporate the text in a message, possibly using + to concatenate

strings are different from numbers and Booleans in that they are objects
- a software object is a unit of code that encapsulates both data and operations that can be performed on that data

- a string is a software object that models words and phrases
  - data: a sequence of characters, enclosed in quotes
  - operations include: make upper case, make lower case, determine the number of characters, access a particular character, search for a particular character, …
Object-Oriented Programming

objects are fundamental in the dominant approach to developing software systems: *object-oriented programming (OOP)*

- OOP encourages programmers to design programs around software objects
  - the programmer identifies the real-world objects involved in a system (e.g., for a banking program: bank account, customer, teller, …)
  - then designs and builds software objects to model these real-world objects

- OOP is effective for managing large systems, since individual objects can be assigned to different teams and developed independently
- OOP also supports code reuse, since the same or similar objects can be combined in different ways to solve different kinds of problems
Object-Oriented Programming

objects are fundamental in the dominant approach to developing software systems: *object-oriented programming (OOP)*

- OOP encourages programmers to design programs around software objects
  - the programmer identifies the real-world objects involved in a system (e.g., for a banking program: bank account, customer, teller, …)
  - then designs and builds software objects to model these real-world objects

- OOP is effective for managing large systems, since individual objects can be assigned to different teams and developed independently
- OOP also supports code reuse, since the same or similar objects can be combined in different ways to solve different kinds of problems

*example*: a doorbell button

- has physical components/properties: color, shape, label, …
- has functionality: when you press the button, the bell rings

an HTML button is a software object that models a real-world button

- has physical components/properties: color, shape, label, …
- has functionality: when you click on the button, JavaScript code is executed

Sunday, December 4, 2011
Properties and Methods

using object-oriented terminology,
- the characteristics of an object are called *properties*
  - e.g., a string object has a length property that identifies the number of characters in the string
- the operations that can be performed on the string are called *methods*
  - e.g., the toLowerCase method makes a copy of the string with all upper-case letters converted to lower-case
Properties and Methods

using object-oriented terminology,

- the characteristics of an object are called properties
  - e.g., a string object has a length property that identifies the number of characters in the string
- the operations that can be performed on the string are called methods
  - e.g., the toLowerCase method makes a copy of the string with all upper-case letters converted to lower-case

properties and methods are not new concepts

- a property is a special kind of a variable (it stores a value)
- a method is a special kind of function (it performs some action)
what is special is that they are associated with (or "belong to") an object
- e.g., each string object will have its own variable to stores it length

to access an object property, specify: object name, a period, property name

```javascript
str1 = 'foo'; str2 = 'Hi there';
len1 = str1.length; len2 = str2.length;
```
Properties and Methods

similarly, to call a method: object name, period, method call

- e.g., str.toLowercase() calls the toLowerCase method on str (which *returns* a lowercase copy of the string)
- e.g., str.toUpperCase() calls the toUpperCase method on str (which *returns* an uppercase copy of the string)

```javascript
str = 'Foo 2 You';
len = str.length;
upStr = str.toUpperCase();
downStr = str.toLowerCase();
```

<table>
<thead>
<tr>
<th>str</th>
<th>'Foo 2 You'</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>'Foo 2 You'</td>
<td>9</td>
</tr>
<tr>
<td>upStr</td>
<td>'FOO 2 YOU'</td>
<td>9</td>
</tr>
<tr>
<td>downStr</td>
<td>'foo 2 you'</td>
<td></td>
</tr>
</tbody>
</table>
String Manipulation Page

1. <html>
2. <!-- strdemo.html -->
3. <!-- This page demonstrates several string properties and operations -->
4. <!--================================================================== -->
5. 
6. <head>
7.    <title> String Fun </title>
8.    <script type="text/javascript">
9.        function Process()
10.           // Assumes: strBox contains a string
11.           // Results: displays the outcome of string operations in outputDiv
12.           {
13.               var str;
14.               str = document.getElementById('strBox').value;
15.               
16.               document.getElementById('outputDiv').innerHTML = 'length: ' + str.length + '<br>' + 'uppercase: ' + str.toUpperCase() + '<br>;'
17.           }
18.    </script>
19. </head>
20. 
21. <body>
22.    String Demo</h2>
23.    <p>
24.        Enter a string: <input type="text" id="strBox" size=20 value="">
25.    </p>
26.    <hr>
27.    <input type="button" value="Click to Process" onclick="Process()">;
28.    <hr>
29.    <div id="outputDiv"></div>
30. </body>
31. </html>

String Demo
Enter a string: Foo 2 You

Click to Process

length: 9
uppercase: FOO 2 YOU
Common String Methods

useful methods exist that allow programmers to access and manipulate individual components of a string

- components are identifiable via indices, or numbers that correspond to the order in which individual characters occur in a string
- indices are assigned in ascending order from left to right, so that the first character in the string is at index 0

the charAt method provides access to a single character within the string
- it takes an index as an input and returns the character at that particular index

```java
word = 'foo';
ch = word.charAt(0); // ASSIGNS ch = 'f'
```
Common String Methods

useful methods exist that allow programmers to access and manipulate individual components of a string

- components are identifiable via *indices*, or numbers that correspond to the order in which individual characters occur in a string
- indices are assigned in ascending order from left to right, so that the first character in the string is at index 0

the `charAt` method provides access to a single character within the string

- it takes an index as an input and returns the character at that particular index

```javascript
word = 'foo';
ch = word.charAt(0); // ASSIGNS ch = 'f'
```

the substring method provides access to an entire sequence of characters within the string

- it takes two numbers as inputs, representing the starting (inclusive) and ending (exclusive) indices of the substring, and returns the substring

```javascript
word = 'foo';
sub = word.substring(1, 3); // ASSIGNS sub = 'oo'
```
String Access/Concatenation

recall: the concatenation operator (+) can join strings together

assuming the variable word stores a string value, what affect would the following assignment have?

```
word = word.charAt(0) + word.substring(1, word.length);
```

```javascript
function Capitalize(str) {
  // Assumes: str is a word
  // Returns: str with first letter capitalized, all others lowercase
  var firstLetter, restString, cap;
  firstLetter = str.charAt(0);       // GET FIRST CHAR
  restString = str.substring(1, str.length); // GET REST OF WORD
  cap = firstLetter.toUpperCase() + restString.toLowerCase();   // PUT BACK TOGETHER
  return cap;
}
```
String Access/Concatenation

recall: the concatenation operator (+) can join strings together

assuming the variable word stores a string value, what affect would the following assignment have?

```java
word = word.charAt(0) + word.substring(1, word.length);
```

the following function takes a string as input and uses string method calls to create (and return) a capitalized version of that string

```javascript
function Capitalize(str) {
  // Assumes: str is a word
  // Returns: str with first letter capitalized, all others lowercase
  var firstLetter, restString, cap;
  firstLetter = str.charAt(0);   // GET FIRST CHAR
  restString = str.substring(1, str.length); // GET REST OF WORD
  cap = firstLetter.toUpperCase() + restString.toLowerCase(); // PUT BACK TOGETHER
  return cap;
}
```
Searching Strings

the search method traverses a string in order to locate a given character or substring

- it takes a character or string as input and returns the index at which the character or string first occurs (or -1 if not found)

```
str = 'banana';
num1 = str.search('n'); // ASSIGNS num1 = 2 since the character 'n' first occurs at index 2
num2 = str.search('ana'); // ASSIGNS num2 = 1 since the string 'ana' first occurs at index 1
num3 = str.search('z'); // ASSIGNS num3 = -1 since the character 'z' does not occur anywhere
```
Searching Strings

the search method traverses a string in order to locate a given character or substring

- it takes a character or string as input and returns the index at which the character or string first occurs (or -1 if not found)

```javascript
str = 'banana';
num1 = str.search('n'); // ASSIGN num1 = 2 since the character 'n' first occurs at index 2
num2 = str.search('ana'); // ASSIGN num2 = 1 since the string 'ana' first occurs at index 1
num3 = str.search('z'); // ASSIGN num3 = -1 since the character 'z' does not occur anywhere
```

**simple application:** determine whether a string is a single word or a phrase

- if the string contains no spaces, the call `str.search(' ')` will return -1, indicating that the string value consists of a single word
- if `str.search(' ')` returns a nonnegative value, then the presence of spaces signifies a phrase containing multiple words
there are times when you want to search for a type of character, rather than a specific value

**example:** converting a word into Pig Latin

- if a word contains no vowels or begins with a vowel, the characters 'way' are appended to the end of the word
  
  nth → nthway
  apple → appleway

- if a word begins with a consonant, its initial sequence of consonants is shifted to the end of the word followed by 'ay'
  
  banana → ananabay
  cherry → errychay
there are times when you want to search for a type of character, rather than a specific value

example: converting a word into Pig Latin

- if a word contains no vowels or begins with a vowel, the characters 'way' are appended to the end of the word
  
  nth → nthway  
  apple → appleway

- if a word begins with a consonant, its initial sequence of consonants is shifted to the end of the word followed by 'ay'
  
  banana → ananabay  
  cherry → errychay

in order to distinguish between these two cases, must search for the first vowel
  
  then, use the substring method to break the string into parts and the + operator to put the pieces back together (with 'ay')

  cherry → erry + ch + ay = errychay
rather than having to search for vowels individually, an entire class of characters can be specified using /[^ . . . ]/.

```javascript
phrase.search(/[aeiou]/)  // returns the index of the first occurrence of a lowercase vowel in phrase; returns -1 if not found
phrase.search(/[aeiouAEIOU]/)  // returns the index of the first occurrence of a lowercase or uppercase vowel in phrase; returns -1 if not found
phrase.search(/[a-z]/)  // returns the index of the first occurrence of lowercase letter in phrase; returns -1 if not found
phrase.search(/[a-zA-Z]/)  // returns the index of the first occurrence of lowercase or uppercase letter in phrase; returns -1 if not found
phrase.search(/[0-9]/)  // returns the index of the first occurrence of a digit in phrase; returns -1 if not found
phrase.search(/[ ,;':!'\?]/)  // returns the index of the first occurrence of a space or punctuation mark in phrase; returns -1 if not found
```
Strings and Repetition

some tasks involve repeatedly performing the same operations

- to accomplish such tasks, we can combine while loops with string methods such as `charAt` and `search`

**example:** a while loop used to access and process each character in a string

- the characters that comprise the string are concatenated one-by-one onto another string, resulting in an exact copy

```java
str = 'abcd';
copy = '';
i = 0;
while (i < str.length) {
    copy = copy + str.charAt(i);
    i = i + 1;
}
```

<table>
<thead>
<tr>
<th></th>
<th>copy</th>
<th>i</th>
<th>str.charAt(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>before loop</td>
<td>''</td>
<td>0</td>
<td>'a'</td>
</tr>
<tr>
<td>after 1st loop pass</td>
<td>'a'</td>
<td>1</td>
<td>'b'</td>
</tr>
<tr>
<td>after 2nd loop pass</td>
<td>'ab'</td>
<td>2</td>
<td>'c'</td>
</tr>
<tr>
<td>after 3rd loop pass</td>
<td>'abc'</td>
<td>3</td>
<td>'d'</td>
</tr>
<tr>
<td>after 4th loop pass</td>
<td>'abcd'</td>
<td>4</td>
<td>''</td>
</tr>
</tbody>
</table>
Example: Substitution Ciphers

a substitution cipher is a code for encrypting/decrypting messages
- one letter of the alphabet is substituted for another in the message
- Atbash cipher (500 B.C.) was used by Hebrew scribes
- Caesar cipher (50-60 B.C.) was used by Julius Caesar

Atbash cipher:

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Z | Y | X | W | V | U | T | R | Q | P | O | N | M | L | K | J | I | H | G | F | E | D | C | B | A |

HELLO → SVOOL

Caesar cipher:

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |

HELLO → KHOOR

substitution ciphers are easy to understand and use
- $26! \approx 4 \times 10^{26}$ possible substitution keys
Text Areas

a text area is similar to a text box but it can contain any number of text lines

general form of a text area element:

```
<textarea id='AREA_ID' rows=NUM_ROWS cols=NUM_COLS>
INITIAL_TEXT
</textarea>
```

- the ID attribute gives the element an identifier so that it can be referenced
- the ROWS attribute specifies the height (number of text lines) of the area
- the COLS attribute specifies the width (number of characters) of the area

unlike a text box, opening and closing tags are used to define a text area

- any text appearing between the tags will be the initial text in the text area
- otherwise, the contents of a text area are accessed/assigned in the same way

```javascript
document.getElementById('AREA_ID').value
```
Encoding a pseudocode:

- get the next character in the message
- find its position in the alphabet
- find the corresponding letter in the key
- use that letter to encode the current letter in the message