Objectives

When you complete this chapter, you will be able to:

- Manipulate strings with properties and methods of the `String` object
- Create regular expressions and use them to validate user input
- Manipulate arrays with properties and methods of the `Array` object
- Convert between strings and arrays, and between strings and JSON

Manipulating Strings

- **String**
  - Text contained within double or single quotation marks
  - Literal values or assigned to a variable
  - Begin and end with same type of quotation mark
- **Example:**
  ```javascript
  document.getElementById("mainHeading").innerHTML = "24-Hour Forecast";
  var highSurfAdvisory = "Watch out for high waves and strong rip currents.";
  ```

Formatting Strings

- Using special characters
  - For basic types: use escape sequences
    ```javascript
    var mainHead = "Today's Forecast";
    ```
  - For other special characters: use Unicode
    ```javascript
    ```
- Using special characters (cont’d.)
  ```javascript
  copyrightInfo = String.fromCharCode(169) + " 2017";
  ```

Manipulating Strings (cont’d.)

- Parsing
  - Extracting characters or substrings from a larger string
- Use **String** class to parse text strings in scripts
  - Represents all literal strings and string variables in JavaScript
  - Contains methods for manipulating text strings

Formatting Strings (cont’d.)

- Using special characters (cont’d.)
  - `fromCharCode()` method
    - Constructs a text string from Unicode character codes
  ```javascript
  var copyrightInfo = String.fromCharCode(169, 32, 10, 169, 32, 97, 99, 116, 105, 110, 115, 116);
  ```
- Examples:
  ```javascript
  copyrightInfo = String.fromCharCode(169) + " 2017";
  ```
Formatting Strings (cont’d.)

• Changing case
  – toLowerCase() and toUpperCase() methods
  – Examples:
    ```javascript
    var agency = "noaa";
    agencyName.innerHTML = agency.toUpperCase();
    // browser displays "NOAA" but value of agency is still "noaa"
    ```

Counting Characters in a String

• length property
  – Returns the number of characters in a string
  – Example:
    ```javascript
    var country = "Kingdom of Morocco";
    var stringLength = country.length;
    // value of stringLength is 18
    ```

Finding and Extracting Characters and Substrings

Table 8-1 Search and extraction methods of the String class (continues)

Finding and Extracting Characters and Substrings (cont’d.)

• Two types of string search methods
  – Those that return a numeric position in a text string
    • Character position in text string begins with a value of zero
    • Can pass a second optional argument specifying the position in the string to start searching to the indexOf() method
  – Example: search() method
    ```javascript
    var email = "president@whitehouse.gov";
    var atPosition = email.search("@"); // returns 9
    ```
Finding and Extracting Characters and Substrings (cont’d.)

• Two types of string search methods (cont’d.)
  – Those that return a numeric position in a text string (cont’d.)
    • Example: `indexOf()` method
      ```javascript
      var email = "president@whitehouse.gov";
      var atIndex = email.indexOf("@", 10); // returns -1
      ```

Finding and Extracting Characters and Substrings (cont’d.)

• Two types of string search methods (cont’d.)
  – Those that return a character or substring
    • `substring()` or `slice()` method
      ```javascript
      var email = "president@whitehouse.gov";
      var nameEnd = email.search("@");
      // value of nameEnd is 9
      var nameText = email.substring(0, nameEnd);
      // value of nameText is "president"
      ```

Finding and Extracting Characters and Substrings (cont’d.)

• Extracting characters from the middle or end of a string
  – Use the `search()`, `indexOf()`, `lastIndexOf()` methods
    • `lastIndexOf()` method returns position of the last occurrence of one string in another string
      ```javascript
      var email = "president@whitehouse.gov";
      var startDomainID = email.lastIndexOf(".");
      // startDomainID value is 20
      var domainID = email.substring(startDomainID + 1);
      // domainID value is "gov"
      ```

Finding and Extracting Characters and Substrings (cont’d.)

• `slice()` method allows negative argument values for the index arguments
  – Specifying a negative value for the starting index
    • `slice()` method starts at the end of the text string
  – Specifying a negative value for the ending index
    • Number of characters the `slice()` method extracts also starts at the end of the text string
    • `slice()` method does not return the character represented by the ending index
      – Returns the character immediately before the ending index
      ```javascript
      var email = "president@whitehouse.gov";
      var nameText = email.slice(0, 9);
      // nameText value is "president"
      var domain = email.slice(-14, -4);
      // domain value is "whitehouse"
      ```

Finding and Extracting Characters and Substrings (cont’d.)

• `replace()` method
  – Creates a new string with the first instance of a specified pattern replaced with the value of the text argument
    • Syntax: `string.replace(pattern, text)`
      ```javascript
      var email = "president@whitehouse.gov";
      var newEmail = email.replace("president", "vice.president");
      // value of newEmail is "vice.president@whitehouse.gov"
      ```
Combining Characters and Substrings

• Combining strings
  – Use the concatenation operator (+) and compound assignment operator (+=)
  – Use the `concat()` method
    • Creates a new string by combining strings passed as arguments
    • Syntax: `string.concat(value1, value2, ...)`
  – To combine text strings
    • Easier to use the concatenation operator and the compound assignment operator

Comparing Strings

• Comparison operator (===) can be used with strings
  – Compare individual characters according to their Unicode position
• `localeCompare()` method
  – Compares strings according to the particular sort order of a language or country
  – Performs a case-sensitive comparison of two strings

Working with Regular Expressions

• Regular expressions
  – Patterns used for matching and manipulating strings according to specified rules
  – With scripting languages, most commonly used for validating submitted form data

Defining Regular Expressions in JavaScript

• Regular expressions
  – Must begin and end with forward slashes
  – Example: `var urlProtocol = /https/`
• Approaches to creating regular expressions
  – Use regular expressions with several `String` class methods
  – Pass pattern directly to a method
  – Use the `RegExp()` constructor
    • Contains methods and properties for working with regular expressions in JavaScript

Defining Regular Expressions in JavaScript (cont’d.)

• Approaches to creating regular expressions (cont’d.)
  – Syntax for creating a regular expression with the `RegExp()` constructor
    `var regExpName = new RegExp("pattern", attributes));`
  – Example:
    `var urlProtocol = new RegExp("https");
    var url = "http://www.cengagebrain.com";
    var searchResult = url.search(urlProtocol);
    // returns -1`
Using Regular Expression Methods

• RegExp object
  – Includes two methods
    • test() and exec()
• test() method: returns a value of true
  – If a string contains text that matches a regular expression
  – Otherwise returns false value
  – Syntax: var pattern = test(string);

• Real power of regular expressions
  – Comes from the patterns written

Writing Regular Expression Patterns

• Hardest part of working with regular expressions
  – Writing the patterns and rules
• Example:
  var emailPattern = /^[\_a-zA-Z0-9.\-]+(\[a-zA-Z0-9.\-\]+)*\@[a-zA-Z0-9.\-]+(\[a-zA-Z0-9.\-\]+)*\$/;
  var email = "president@whitehouse.gov";
  var result;
  if (emailPattern.test(email)) {
    result = true;
  } else {
    result = false;
  }
  // value of result is true

Writing Regular Expression Patterns (cont’d.)

• Regular expression patterns consist of literal characters and metacharacters
  – Metacharacters: special characters that define the pattern matching rules in a regular expression

Writing Regular Expression Patterns (cont’d.)

• Matching any character
  – Period (.)
    • Matches any single character in a pattern
    • Specifies that the pattern must contain a value where the period located
• Matching characters at the beginning or end of a string
  – ^ metacharacter
    • Matches characters at the beginning of a string
  – $ metacharacter
    • Matches characters at the end of a string

Writing Regular Expression Patterns (cont’d.)

• Matching characters at the beginning or end of a String (cont’d.)
  – Anchor
    • Pattern that matches the beginning or end of a line
    • Specifying an anchor at the beginning of a line
    • Pattern must begin with the ^ metacharacter
  – Matching special characters
    • Precede the character with a backslash

Writing Regular Expression Patterns (cont’ d.)

<table>
<thead>
<tr>
<th>METACHARACTER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>Identifies the next character as a literal value</td>
</tr>
<tr>
<td>^</td>
<td>Matches characters at the beginning of a string</td>
</tr>
<tr>
<td>$</td>
<td>Matches characters at the end of a string</td>
</tr>
<tr>
<td>()</td>
<td>Specifies required characters to include in a pattern match</td>
</tr>
<tr>
<td>[^]</td>
<td>Specifies alternate characters allowed in a pattern match</td>
</tr>
<tr>
<td>()</td>
<td>Specifies characters to exclude in a pattern match</td>
</tr>
<tr>
<td>+</td>
<td>Identifies a possible range of characters in a match</td>
</tr>
<tr>
<td>\</td>
<td>Specifies alternate set of characters to include in a pattern match</td>
</tr>
</tbody>
</table>

Table 8-2 JavaScript regular expression metacharacters

JavaScript, Sixth Edition
Writing Regular Expression Patterns (cont’d.)

• Specifying quantity
  – Quantifiers
    • Metacharacters that specify the quantity of a match

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Specifies the preceding character is optional</td>
</tr>
<tr>
<td>+</td>
<td>Specifies one or more of the preceding characters must match</td>
</tr>
<tr>
<td>*</td>
<td>Specifies that zero or more of the preceding characters can match</td>
</tr>
<tr>
<td>(n)</td>
<td>Specifies that the preceding character repeat exactly n times</td>
</tr>
<tr>
<td>(n, m)</td>
<td>Specifies that the preceding character repeat at least n times but no more than m times</td>
</tr>
</tbody>
</table>

Table 8-3 JavaScript regular expression quantifiers

Writing Regular Expression Patterns (cont’d.)

• Defining character classes
  – Character classes
    • Used in regular expressions to treat multiple characters as a single item
    • Created by enclosing the characters that make up the class with bracket [ ] metacharacters
    – Use a hyphen metacharacter (-) to specify a range of values in a character class
    – To specify optional characters to exclude in a pattern match
      • Include the ^ metacharacter immediately before the characters in a character class

Writing Regular Expression Patterns (cont’d.)

• Specifying subexpressions
  – Subexpression or subpattern
    • Characters contained in a set of parentheses within a regular expression
    • Allows determination of the format and quantities of the enclosed characters as a group

Writing Regular Expression Patterns (cont’d.)

• Defining character classes (cont’d.)
  – Regular expressions include special escape characters in character classes
    • To represent different types of data

<table>
<thead>
<tr>
<th>EXPLANATION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphanumeric characters</td>
<td>[a-zA-Z0-9]</td>
</tr>
<tr>
<td>any character that is not an alphanumeric character</td>
<td>[^a-zA-Z0-9]</td>
</tr>
<tr>
<td>numeric characters</td>
<td>[0-9]</td>
</tr>
<tr>
<td>nonnumeric characters</td>
<td>[^0-9]</td>
</tr>
<tr>
<td>white space characters</td>
<td>[\s]</td>
</tr>
<tr>
<td>all printable characters</td>
<td>[\p{IsPrint}]</td>
</tr>
<tr>
<td>backspace character</td>
<td>[\x08]</td>
</tr>
</tbody>
</table>

Table 8-4 JavaScript character class expressions

Setting Regular Expression Properties

<table>
<thead>
<tr>
<th>PROPERTY</th>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>global</td>
<td>g</td>
<td>Determines whether to search for all possible matches within a string</td>
</tr>
<tr>
<td>ignoreCase</td>
<td>i</td>
<td>Determines whether to ignore-case when executing a regular expression</td>
</tr>
<tr>
<td>lastIndex</td>
<td></td>
<td>Stores the index of the first character from the last match (no flag)</td>
</tr>
<tr>
<td>multiline</td>
<td>m</td>
<td>Determines whether to search across multiple lines of text</td>
</tr>
<tr>
<td>source</td>
<td></td>
<td>Contains the regular expression pattern (no flag)</td>
</tr>
</tbody>
</table>

Table 8-5 Properties of the RegExp object
Setting Regular Expression Properties (cont’d.)

• Options for setting the values of these properties
  – Assign a value of true or false to the property
  • By creating a regular expression with the RegExp() constructor
  – Use the flags when assigning a regular expression to a variable without using the RegExp() constructor

Manipulating Arrays

• Use the Array class length property and methods
  • Creating an array
    – Instantiates an object from the Array class

Manipulating Arrays (cont’d.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>push()</td>
<td>Adds one or more elements to the end of an array</td>
</tr>
<tr>
<td>pop()</td>
<td>Removes the last element from the end of an array</td>
</tr>
<tr>
<td>slice()</td>
<td>Removes and returns a portion of an array, starting at the index specified by the start argument and ending at the index specified by the end argument</td>
</tr>
<tr>
<td>shift()</td>
<td>Removes and returns the first element from the beginning of an array</td>
</tr>
<tr>
<td>unshift()</td>
<td>Adds one or more elements to the beginning of an array</td>
</tr>
</tbody>
</table>

Table 8-6 Methods of the Array class

Finding and Extracting Elements and Values

• Primary method for finding a value in an array
  – Use a looping statement to iterate through the array until a particular value found
• Extract elements and values from an array
  – Use the slice() method to return (copy) a portion of an array and assign it to another array
• Syntax for the slice() method
  ```javascript
  array_name.slice(start, end);
  ```

Finding and Extracting Elements and Values (cont’ d.)

```javascript
var largestStates = ['Alaska', 'Texas', 'California', 'Montana', 'New Mexico', 'Arizona', 'Nevada', 'Colorado', 'Oregon', 'Wyoming'];
var fiveLargestStates = largestStates.slice(0, 5);
for (var i = 0; i < fiveLargestStates.length; i++) {
  var newli = document.createElement("li");
  newli.innerHTML = fiveLargestStates[i];
  document.body.appendChild(newli);
}
```

Figure 8-11 List of states extracted using the slice() method

Manipulating Elements

• Adding and removing elements to and from the beginning of an array
  – shift() method removes and returns the first element from the beginning of an array
  – unshift() method adds one or more elements to the beginning of an array
Manipulating Elements

• Adding and removing elements to and from the end of an array
  − Use array’s length property to determine the next available index

```javascript
var colors = ['mauve', 'periwinkle', 'silver', 'cherry', 'lemon'];
colors.colors.length = 'lemon';
// color.value now ['mauve', 'periwinkle', 'silver', 'cherry']
```

• Adding and removing elements to and from the end of an array (cont’d.)
  − `pop()` method removes the last element from the end of an array
  − `push()` method adds one or more elements to the end of an array

```javascript
var colors = ['mauve', 'periwinkle', 'silver', 'cherry'];
colors.pop();
// colors value now ['mauve', 'periwinkle', 'silver', 'cherry', 'lemon']
```

• Adding and removing elements within an array
  − Use the `splice()` method
    • Also renames the indexes in the array
    • To add an element, include 0 as second argument

```javascript
var hospitalDepts = ['Anesthesia', 'Molecular Biology',
                     'Neurology', 'Pediatrics'];
hospitalDepts.splice(1, 2, 'Ophthalmology');
// value now ['Anesthesia', 'Molecular Biology',
// 'Neurology', 'Ophthalmology', 'Pediatrics']
```

• Adding and removing elements within an array (cont’d.)
  − Use the `splice()` method (cont’d.)
    • To delete elements, omit third argument
    • Indexes renumbered just like when elements added

```javascript
var hospitalDepts = ['Anesthesia', 'Molecular Biology',
                     'Neurology', 'Pediatrics'];
hospitalDepts.splice(1, 2);
// value now ['Anesthesia', 'Pediatrics']
```

Manipulating Elements (cont’d.)

• Adding and removing elements within an array
  − Use the `splice()` method (cont’d.)

```javascript
var scientificFishNames = ['Quadratus taiwanae',
                           'Macquaria australasica', 'Jordania zonope',
                           'Abudefduf sparoides', 'Dactylopterus volitans',
                           'Wattsia mossambica', 'Bagrus urostigma'];
scientificFishNames.sort();
// scientificFishNames value now
// ['Abudefduf sparoides', 'Bagrus urostigma',
//  'Dactylopterus volitans', 'Jordania zonope',
//  'Macquaria australasica', 'Quadratus taiwanae',
//  'Wattsia mossambica']
```

Sorting and Combining Arrays

• Sorting arrays
  − Sort elements of an array alphabetically
    • Use the `sort()` method

```javascript
var scientificFishNames = ['Quadratus taiwanae',
                           'Macquaria australasica', 'Jordania zonope',
                           'Abudefduf sparoides', 'Dactylopterus volitans',
                           'Wattsia mossambica', 'Bagrus urostigma'];
scientificFishNames.sort();
// scientificFishNames value now
// ['Abudefduf sparoides', 'Bagrus urostigma',
//  'Dactylopterus volitans', 'Jordania zonope',
//  'Macquaria australasica', 'Quadratus taiwanae',
//  'Wattsia mossambica']
```
Sorting and Combining Arrays (cont'd.)

• Sorting arrays (cont'd.)
  – reverse() method
    • Transposes, or reverses, the order of the elements in an array
      
      ```javascript
      scientificFishNames.reverse();
      // scientificFishNames value now 
      // ["Wattsia mossambica", "Quadratus taiwanae", 
      // "Macquaria australasica", "Xandopsis zonoper", 
      // "Dactylopterus volitans", "Bagnus urostigma", 
      // "Abudelfadl sparoides"]
      ```

  

Sorting and Combining Arrays (cont'd.)

• Combining arrays
  – Use the concat() method
    
    ```javascript
    array1.concat(array2, array3, ...);
    ```

Converting Between Data Types

• Common task to convert strings and arrays to different data types
  – strings to arrays
  – arrays to strings
  – objects to strings
  – strings to objects

Converting Between Strings and Arrays

• split() method of the String class
  – Splits a string into an indexed array
    
    ```javascript
    array = string.split(separator, limit);
    ```
  
• To split individual characters in a string into an array
  – Pass an empty string ("\"") as the separator argument
    
    ```javascript
    var OPEC = "Algeria, Angola, Ecuador, Iran, Iraq, Kuwait, 
    Libya, Nigeria, Qatar, Saudi Arabia, " 
    United Arab Emirates, Venezuela";
    // The value of OPEC is a string
    var opecArray = OPEC.split("\", ");
    // The value of opecArray is the following array:
    // ["Algeria", "Angola", "Ecuador", "Iran", "Iraq", 
    "Kuwait", "Libya", "Nigeria", "Qatar", "Saudi Arabia", 
    "United Arab Emirates", "Venezuela"]
    ```
Converting Between Strings and Arrays (cont’d.)

• `join()` method of the Array class
  – Combines array elements into a string, separated by a comma or specified characters
• Syntax
  
```
array.join(“separator”);
```
• To prevent elements from being separated by any characters in the new string
  – Pass an empty string (“”) as the `separator` argument

```javascript

// value of OPEC is an array
var opecString = OPEC.join();

// value of opecString is the following string:
// “Algeria, Angola, Ecuador, Iran, Iraq, Kuwait, Libya, Nigeria, Qatar, Saudi Arabia, United Arab Emirates, Venezuela”
```

Converting Between Strings and Arrays (cont’d.)

• `join()` method does not include a separator argument
  – Previous example OPEC nations automatically separated by commas
    • Can include a `separator` argument of “,”
      ```javascript
      // value of OPEC is an array
      var opecString = OPEC.join(“,”);
      // value of opecString is the following string:
      // “Algeria, Angola, Ecuador, Iran, Iraq, Kuwait, Libya, Nigeria, Qatar, Saudi Arabia, United Arab Emirates, Venezuela”
      ```

Converting Between Strings and JSON (cont’d.)

• JavaScript Object Notation (JSON)
  – Represents a JavaScript object as a string
  – Exchanges data between application and server
• JSON object
  – Supported in modern browsers, including IE8

<table>
<thead>
<tr>
<th>METHOD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parse()</code></td>
<td>Converts a string value to an object</td>
</tr>
<tr>
<td><code>stringify()</code></td>
<td>Converts an object to a string value</td>
</tr>
</tbody>
</table>

Table 8-7 Methods of the `JSON` object

Converting Between Strings and JSON (cont’d.)

• Converting an Object to a String
  ```javascript
  function stringify() {
    var string = JSON.stringify(value [, replacer [, space]]);
    // string is name of variable that will contain string
    // value represents JavaScript object to be converted
  }
  ```
Converting Between Strings and JSON (cont’d.)

• Converting an Object to a String (cont’d.)
  ```javascript
  var newUser = {
    fName: "Tony",
    lName: "Chu"
  },
  newUserString = JSON.stringify(newUser);
  // value of newUserString is
  // {"fName": "Tony", "lName": "Chu"}
  ```

Converting Between Strings and JSON (cont’d.)

• Converting a String to an Object (cont’d.)
  ```javascript
  var newUser = '{
    fName: "Tony",
    lName: "Chu"
  }',
  newUserObject = JSON.parse(newUser);
  // value of newUserObject is
  // { fName: "Tony", lName: "Chu" }
  ```

Summary

• Parsing
  - Act of extracting characters or substrings from a larger string
  - All literal strings and string variables in JavaScript
    - Represented by the String class
      - fromCharCode() method of the String class
        - Constructs a text string from Unicode character codes
        - toLowerCase() and toUpperCase() methods
          - Change the case of letters in a string

Summary (cont’d.)

• String class
  - length property
  - Methods: replace(), concat(), localeCompare()
• Regular expressions
  - Patterns used for matching and manipulating strings according to specified rules
  - RegExp object
    - Contains methods and properties for working with regular expressions in JavaScript

Summary (cont’d.)

• Use the Array class methods and length property to manipulate arrays in scripts
  - Methods: slice(), shift() and unshift(), pop() and push(), splice(), sort(), reverse(), concat(), and join()
• split() method of the String class
  - Splits a string into an indexed array
• join() method of the Array class
  - Combines array elements into a string
Summary (cont’d.)

• Use the JSON class methods to convert between string values and object values
  • stringify() method of the JSON class  
    – Converts JavaScript object to JSON string
  • parse() method of the JSON class  
    – Converts JSON string to JavaScript object