COURSE TECHNOLOGY
CENGAGE Learning

JavaScript, Sixth Edition

Chapter 8
Manipulating Data in Strings and Arrays

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:

```
document.getElementById("mainHeading").innerHTML = "24-Hour↵
  Forecast";
var highSurfAdvisory = "Watch out for high waves and strong↵
  rip currents.";
```

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

- Using special characters
  - For basic types: use escape sequences

```
var mainHead = 'Today\'s Forecast';
```

  - For other special characters: use Unicode
    - Standardized set of characters from many of the world's languages

```
copyrightInfo = "<p>&#169; 2006-2017</p>";
  // numeric character ref.
copyrightInfo = "<p>&copy; 2006-2017</p>";
  // character entity
```

Formatting Strings (cont'd.)

- Using special characters (cont'd.)
  - `fromCharCode()` method
    - Constructs a text string from Unicode character codes
  - Syntax:

```
String.fromCharCode(char1, char2, ...)
```

  - Examples:

```
String.fromCharCode(74,97,118,97,83,99,114,105,112,116)
copyrightInfo = String.fromCharCode(169) + " 2017";
```

Formatting Strings (cont'd.)

- Changing case
  - `toLowerCase()` and `toUpperCase()` methods
  - Examples:

```
var agency = "noaa";
agencyName.innerHTML = agency.toUpperCase();

// browser displays "NOAA" but value of agency is still "noaa"
```

```
var agency = "noaa";
agency = agency.toUpperCase();
// value of agency is "NOAA"
agencyName.innerHTML = agency;

// browser displays "NOAA"
```

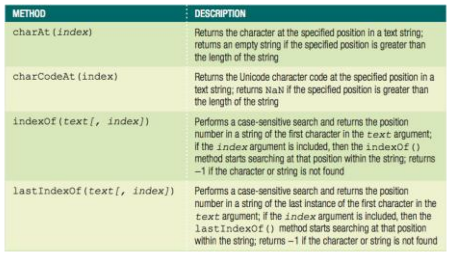
Counting Characters in a String

- `length` property
  - Returns the number of characters in a string
  - Example:

```
var country = "Kingdom of Morocco";
var stringLength = country.length;

// value of stringLength is 18
```

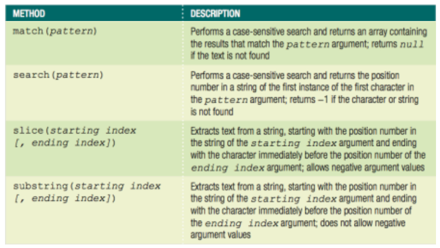
Finding and Extracting Characters and Substrings



METHOD	DESCRIPTION
`charAt(index)`	Returns the character at the specified position in a text string; returns an empty string if the specified position is greater than the length of the string
`charCodeAt(index)`	Returns the Unicode character code at the specified position in a text string; returns `NaN` if the specified position is greater than the length of the string
`indexOf(text[, index])`	Performs a case-sensitive search and returns the position number in a string of the first character in the `text` argument; if the `index` argument is included, then the `indexOf()` method starts searching at that position within the string; returns –1 if the character or string is not found
`lastIndexOf(text[, index])`	Performs a case-sensitive search and returns the position number in a string of the last instance of the first character in the `text` argument; if the `index` argument is included, then the `lastIndexOf()` method starts searching at that position within the string; returns –1 if the character or string is not found

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

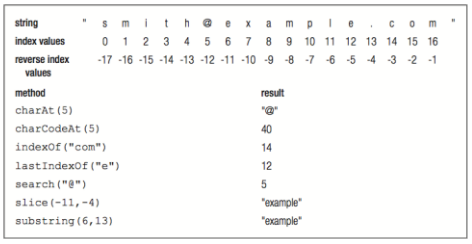
Finding and Extracting Characters and Substrings (cont' d.)



METHOD	DESCRIPTION
`match(pattern)`	Performs a case-sensitive search and returns an array containing the results that match the `pattern` argument; returns `null` if the text is not found
`search(pattern)`	Performs a case-sensitive search and returns the position number in a string of the first instance of the first character in the `pattern` argument; returns –1 if the character or string is not found
`slice(starting index [, ending index])`	Extracts text from a string, starting with the position number in the string of the `starting index` argument and ending with the character immediately before the position number of the `ending index` argument; allows negative argument values
`substring(starting index [, ending index])`	Extracts text from a string, starting with the position number in the string of the `starting index` argument and ending with the character immediately before the position number of the `ending index` argument; does not allow negative argument values

Table 8-1 Search and extraction methods of the `String` class

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



string	"	s	m	i	t	h	@	e	x	a	m	p	l	e	.	c	o	m	"
index values		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
reverse index values		-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

method	result
`charAt(5)`	"@"
`charCodeAt(5)`	40
`indexOf("com")`	14
`lastIndexOf("e")`	12
`search("@")`	5
`slice(-11,-4)`	"example"
`substring(6,13)`	"example"

Figure 8-5 Example uses of `String` class methods

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

```
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

```
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

```
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
  - Example:

```
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

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

```
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"
```

Replacing Characters and Substrings

- `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)`
  - Example:

```
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

Combining Characters and Substrings (cont' d.)

```
var name = "Theodor Seuss Geisel";
var penName = "Dr. Seuss";
var bio = penName.concat(" was the pen name of ", name);
// value of bio is
// "Dr. Seuss was the pen name of Theodor Seuss Geisel"

var name = "Theodor Seuss Geisel";
var penName = "Dr. Seuss";
var bio = penName + " was the pen name of " + name;
// value of bio is
// "Dr. Seuss was the pen name of Theodor Seuss Geisel"
```

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:

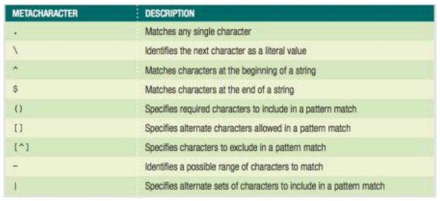
```
var emailPattern = /^[_a-zA-Z0-9\\-]+(\.[_a-zA-Z0-9\\-]+)*↵
    @[a-zA-Z0-9\\-]+(\.[a-zA-Z0-9\\-]+)*(\.[a-z]{2,6})$/;
var email = "president@whitehouse.gov";
var result;
if (emailPattern.test(email)) {
  result = true;
} else {
  result = false;
}

// 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.)



METACHARACTER	DESCRIPTION
.	Matches any single character
\	Identifies the next character as a literal value
^	Matches characters at the beginning of a string
$	Matches characters at the end of a string
()	Specifies required characters to include in a pattern match
[]	Specifies alternate characters allowed in a pattern match
[^]	Specifies characters to exclude in a pattern match
-	Identifies a possible range of characters to match
|	Specifies alternate sets of characters to include in a pattern match

Table 8-2 JavaScript regular expression metacharacters

Writing Regular Expression Patterns (cont'd.)

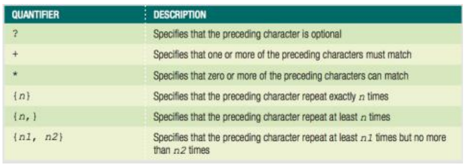
- Matching any character
  - Period (`.`)
    - Matches any single character in a pattern
    - Specifies that the pattern must contain a value where the period located
- Matching characters at the beginning or end of a string
  - `^` metacharacter
    - Matches characters at the beginning of a string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

- Specifying quantity
  - Quantifiers
    - Metacharacters that specify the quantity of a match



QUANTIFIER	DESCRIPTION
?	Specifies that the preceding character is optional
+	Specifies that one or more of the preceding characters must match
*	Specifies that zero or more of the preceding characters can match
{n}	Specifies that the preceding character repeat exactly *n* times
{n,}	Specifies that the preceding character repeat at least *n* times
{n1, n2}	Specifies that the preceding character repeat at least *n1* times but no more than *n2* times

Table 8-3 JavaScript regular expression quantifiers

Writing Regular Expression Patterns (cont'd.)

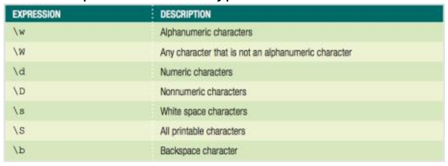
- Specifying subexpressions
  - Subexpression or subpattern
    - Characters contained in a set of parentheses within a regular expression
    - Allows determination of the format and quantities of the enclosed characters as a group

Writing Regular Expression Patterns (cont'd.)

- Defining character classes
  - 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.)

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



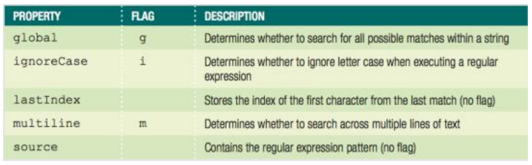
EXPRESSION	DESCRIPTION
\w	Alphanumeric characters
\W	Any character that is not an alphanumeric character
\d	Numeric characters
\D	Nonnumeric characters
\s	White space characters
\S	All printable characters
\b	Backspace character

Table 8-4 JavaScript character class expressions

Writing Regular Expression Patterns (cont'd.)

- Defining character classes (cont'd.)
  - Matching multiple pattern choices
  - Allow a string to contain an alternate set of substrings
    - Separate the strings in a regular expression pattern with the `|` metacharacter

Setting Regular Expression Properties



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

Table 8-5 Properties of the `RegExp` object

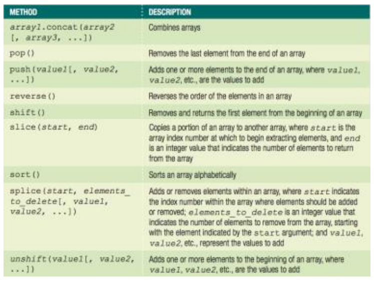
Setting Regular Expression Properties (cont'd.)

- 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.)



METHOD	DESCRIPTION
`array1.concat(array2 [, array3, ...])`	Combines arrays
`pop()`	Removes the last element from the end of an array
`push(value1[, value2, ...])`	Adds one or more elements to the end of an array, where *value1*, *value2*, etc., are the values to add
`reverse()`	Reverses the order of the elements in an array
`shift()`	Removes and returns the first element from the beginning of an array
`slice(start, end)`	Copies a portion of an array to another array, where *start* is the array index number at which to begin extracting elements, and *end* is an integer value that indicates the number of elements to return from the array
`sort()`	Sorts an array alphabetically
`splice(start, elements_to_delete[, value1, value2, ...])`	Adds or removes elements within an array, where *start* indicates the index number within the array where elements should be added or removed; *elements_to_delete* is an integer value that indicates the number of elements to remove from the array, starting with the element indicated by the *start* argument; and *value1*, *value2*, etc., represent the values to add
`unshift(value1[, value2, ...])`	Adds one or more elements to the beginning of an array, where *value1*, *value2*, etc., are the values to add

Table 8-6 Methods of the `Array` class

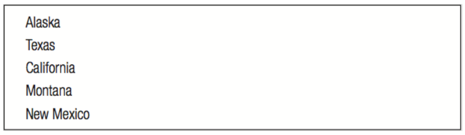
Finding and Extracting Elements and Values

- 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

  `array_name``.slice(`*`start, end`*`);`

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

```
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 newItem = document.createElement("p");
   newItem.innerHTML = fiveLargestStates[i];
   document.body.appendChild(newItem);
}
```



```
Alaska
Texas
California
Montana
New Mexico
```

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 (cont’d.)

```
var colors = ["mauve", "periwinkle", "silver", "cherry",↵
    "lemon"];
colors.shift(); // colors value now
// ["periwinkle", "silver", "cherry", "lemon"]
colors.unshift("yellow-orange", "violet");
// colors value now ["yellow-orange", "violet",
// "mauve", "periwinkle", "silver", "cherry", "lemon"]
```

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

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

Manipulating Elements (cont’d.)

- 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

```
var colors = ["mauve", "periwinkle", "silver", "cherry"];
colors.pop();
// colors value now ["mauve", "periwinkle", "silver"]
colors.push("yellow-orange", "violet");
// colors value now ["mauve", "periwinkle", "silver",
// "yellow-orange", "violet"]
```

Manipulating Elements (cont’d.)

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

```
var hospitalDepts = ["Anesthesia", "Molecular Biology",↵
    "Neurology", "Pediatrics"];
hospitalDepts.splice(3, 0, "Ophthalmology");
// value now ["Anesthesia", "Molecular Biology",
// "Neurology", "Ophthalmology", "Pediatrics"]
```

Manipulating Elements (cont’d.)

- 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

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

Sorting and Combining Arrays

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

```
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

```
scientificFishNames.reverse();
// scientificFishNames value now
// ["Wattsia mossambica", "Quadratus taiwanae",
// "Macquaria australasica", "Jordania zonope",
// "Dactylopterus volitans", "Bagrus urostigma",
// "Abudefduf sparoides"]
```

Sorting and Combining Arrays (cont'd.)

- Combining arrays
  - Use the `concat()` method
  - Syntax

```
array1.contact(array2, array3, ...);
```

Sorting and Combining Arrays (cont'd.)

```
var Provinces = ["Newfoundland and Labrador",↵
    "Prince Edward Island", "Nova Scotia",↵
    "New Brunswick", "Quebec", "Ontario",↵
    "Manitoba", "Saskatchewan", "Alberta",↵
    "British Columbia"];
var Territories = ["Nunavut", "Northwest Territories",↵
    "Yukon"];
var Canada = [];
Canada = Provinces.concat(Territories);
// value of Canada now ["Newfoundland and Labrador",
// "Prince Edward Island", "Nova Scotia",
// "New Brunswick", "Quebec", "Ontario",
// "Manitoba", "Saskatchewan", "Alberta",
// "British Columbia", "Nunavut",
// "Northwest Territories", "Yukon"];
```

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
- Syntax

  array = *string*.split(*separator*[, *limit*]);

- To split individual characters in a string into an array
  - Pass an empty string (`""`) as the separator argument

Converting Between Strings and Arrays (cont'd.)

```
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

Converting Between Strings and Arrays (cont' d.)

```
var OPEC = ["Algeria", "Angola", "Ecuador", "Iran",↵
  "Iraq", "Kuwait", "Libya", "Nigeria", "Qatar",↵
  "Saudi Arabia", "United Arab Emirates", "Venezuela"];
// 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 `";"`

```
var OPEC = ["Algeria", "Angola", "Ecuador", "Iran",↵
  "Iraq", "Kuwait", "Libya", "Nigeria", "Qatar",↵
  "Saudi Arabia", "United Arab Emirates", "Venezuela"];
// 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.)

- Can also use the `toString()` and `toLocaleString()` method
  - Convert an array to a string
  - *array*.toString();
  - *array*.toLocaleString();

Converting Between Strings and JSON

- 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

METHOD	DESCRIPTION
`parse()`	Converts a string value to an object
`stringify()`	Converts an object to a string value

Table 8-7 Methods of the `JSON` object

Converting Between Strings and JSON (cont'd.)

- Converting an Object to a String
  - `stringify()` method
  - *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.)

```
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
  - `parse()` method
  - *object* = JSON.parse(*string* [, *function*]);
  - `object` is namve of variable that will contain object
  - `string` represents JSON string to be converted

Converting Between Strings and JSON (cont'd.)

- Converting a String to an Object (cont'd.)
  - JSON string definition:

    ```
    var newUser = '{"fName":"Tony","lName":"Chu"}';
    ```

    - String because enclosed in quotes
  - To convert string to JavaScript object:

    ```
    var newUserObject = JSON.parse(newUser);
    // value of newUserObject is
    // {
    //   fName: "Tony",
    //   lName: "Chu"
    // };
    ```

Summary

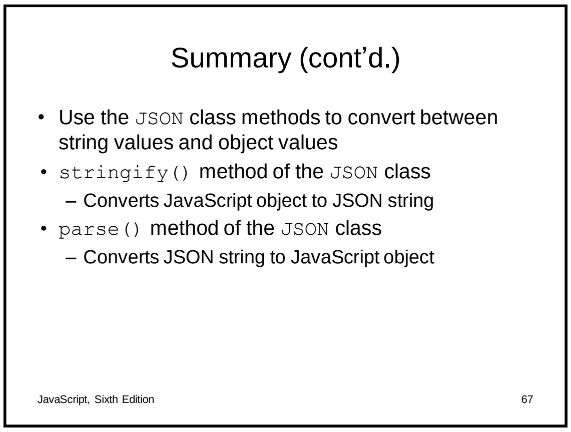
- Parsing
  - Act of extracting characters or substrings from a larger string
- All literal strings and string variables in JavaScript
  - Represented by the `String` class
- `fromCharCode()` method of the `String` class
  - Constructs a text string from Unicode character codes
- `toLowerCase()` and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