# HTML5 and CSS3

## 7th Edition

INTRODUCTORY

# Tutorial 9

# Getting Started with JavaScript

Carey

# Objectives

- Insert a script element
- Write JavaScript comments
- Display an alert dialog box
- Use browser debugging tools
- Reference browser and page objects
- Use JavaScript properties and methods

# Objectives (continued)

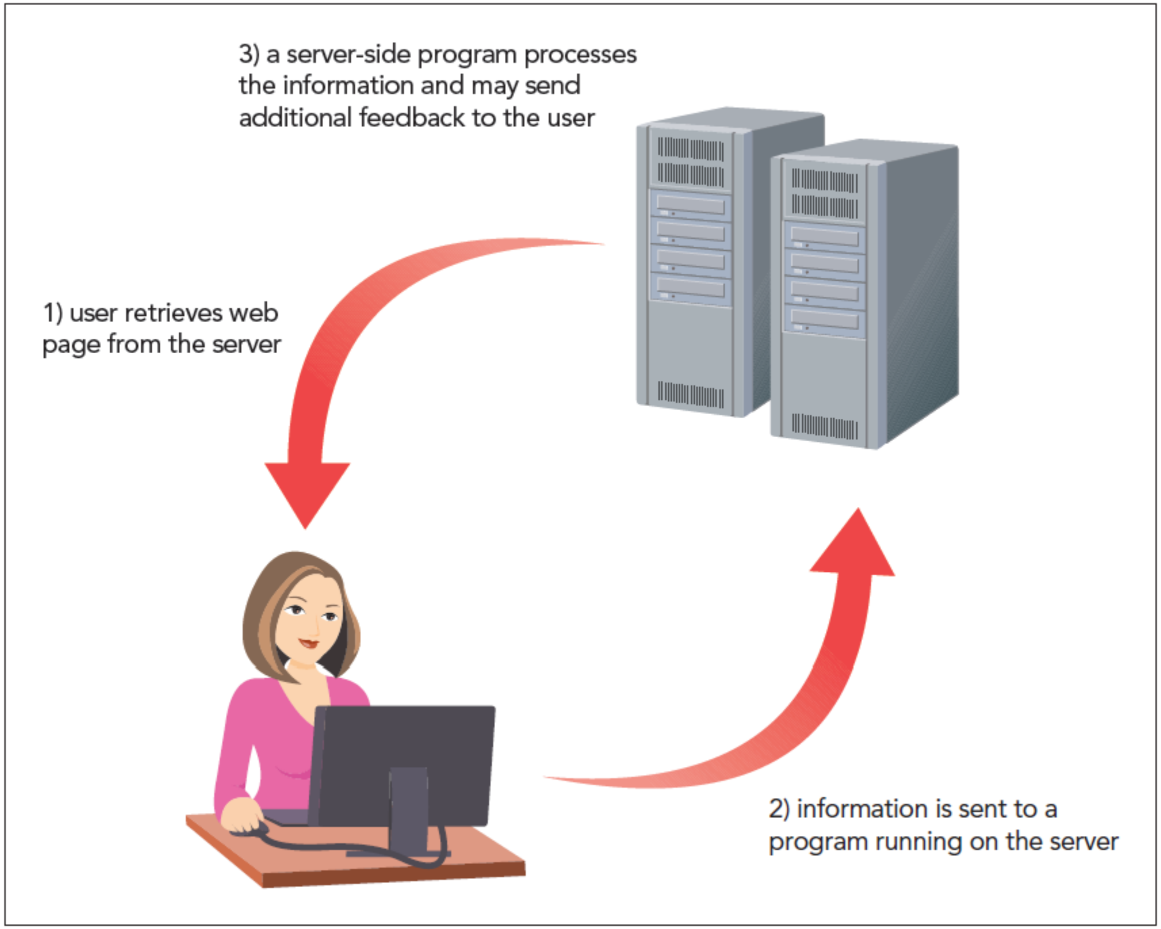
- Write HTML code and text content into a page
- Work with a Date object
- Use JavaScript operators
- Create a JavaScript function
- Create timed commands

# Server-Side and Client-Side Programming

- **Server-side programming:** Program code runs from the server hosting the website
- Advantage
  - Connects a server to an online database containing information not directly accessible to end users
- Disadvantages
  - Use server resources and requires Internet access
  - Long delays in cases of system over-load

# Server-Side and Client-Side Programming (continued 1)

**Figure 9-1** Server-side programming


3) a server-side program processes the information and may send additional feedback to the user
1) user retrieves web page from the server
2) information is sent to a program running on the server

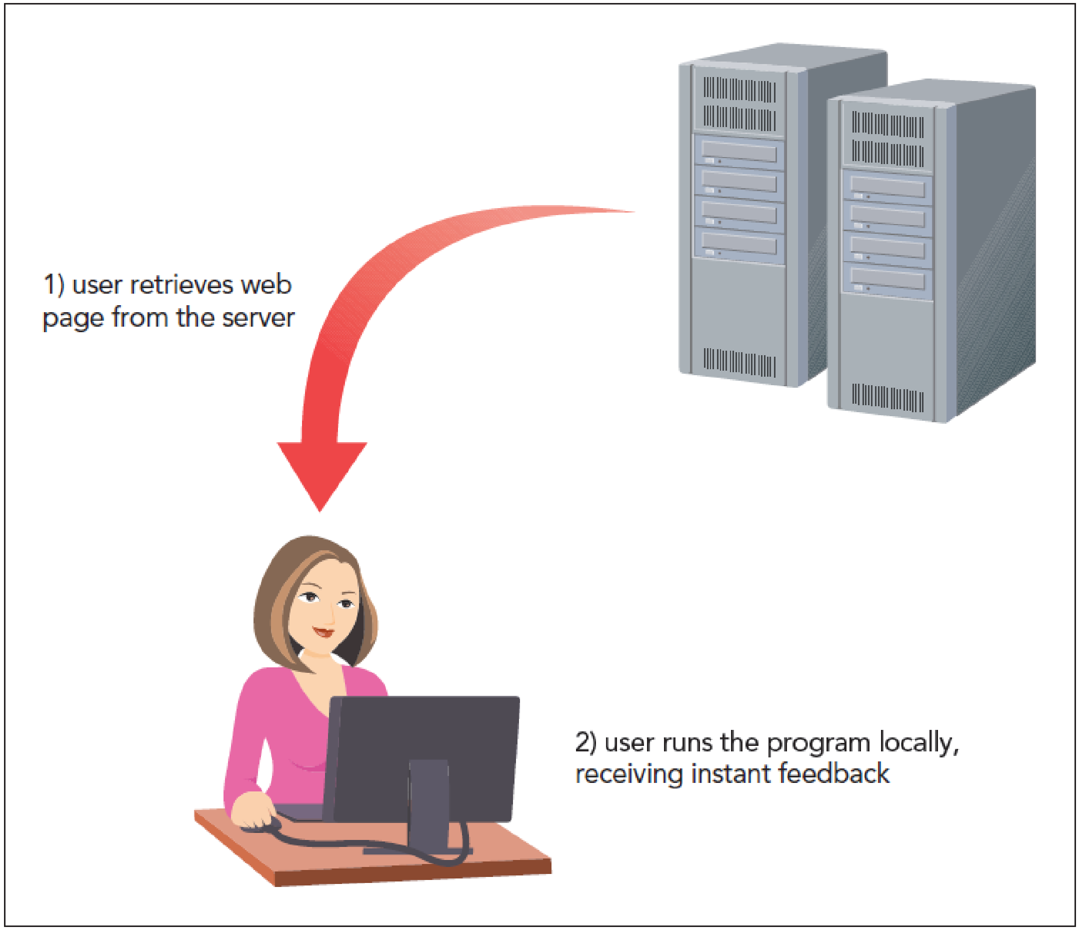
© 2016 Cengage Learning

# Server-Side and Client-Side Programming (continued 2)

- **Client-side programming:** Programs run on the user's computer using downloaded scripts with HTML and CSS files
- Distributes load to avoid overloading of program-related requests
- Client-side programs can never replace server-side programming

# Server-Side and Client-Side Programming (continued 3)

Figure 9-2 Client-side programming


1) user retrieves web page from the server
2) user runs the program locally, receiving instant feedback

© 2016 Cengage Learning

# The Development of JavaScript

- **JavaScript** is a programming language for client-side programs
- It is an **interpreted language** that executes a program code without using an application
- **Compiler** is an application that translates a program code into machine language
- JavaScript code can be directly inserted into or linked to an HTML file

# Working with the `script` Element

- JavaScript code is attached to an HTML file using the `script` element

```
<script src=”url”></script>
```

  where *`url`* is the URL of the external file containing the JavaScript code

- An **embedded script** can be used instead of an external file by omitting the `src` attribute

```
<script>
    code
</script>
```

# Loading the `script` Element

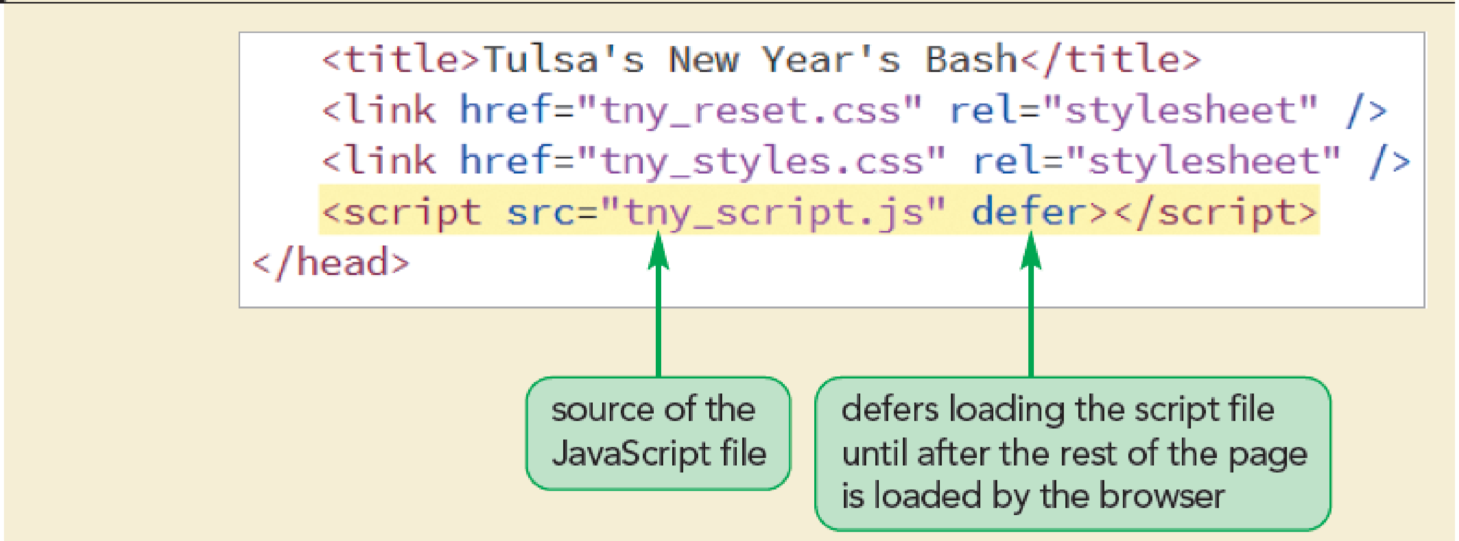
- `script` element can be placed anywhere within an HTML document
- When a browser encounters a script, it immediately stops loading the page and begins loading and then processing the script commands
- `async` and `defer` attributes can be added to `script` element to modify its sequence of processing

# Loading the `script` Element (continued)

- `async` attribute tells a browser to parse the HTML and JavaScript code together
- `defer` attribute defers script processing until after the page has been completely parsed and loaded
- `async` and `defer` attributes are ignored for embedded scripts

# Inserting the `script` Element

Figure 9-4 Inserting the script element



```
   <title>Tulsa's New Year's Bash</title>
   <link href="tny_reset.css" rel="stylesheet" />
   <link href="tny_styles.css" rel="stylesheet" />
   <script src="tny_script.js" defer></script>
</head>
```

source of the JavaScript file

defers loading the script file until after the rest of the page is loaded by the browser

# Creating a JavaScript Program

- JavaScript programs are created using a standard text editor
- Adding Comments to your JavaScript Code
  - Comments help understand the design and purpose of programs
  - JavaScript comments can be entered on single or multiple lines

# Creating a JavaScript Program (continued 1)

- Syntax of a single-line comment is as follows:

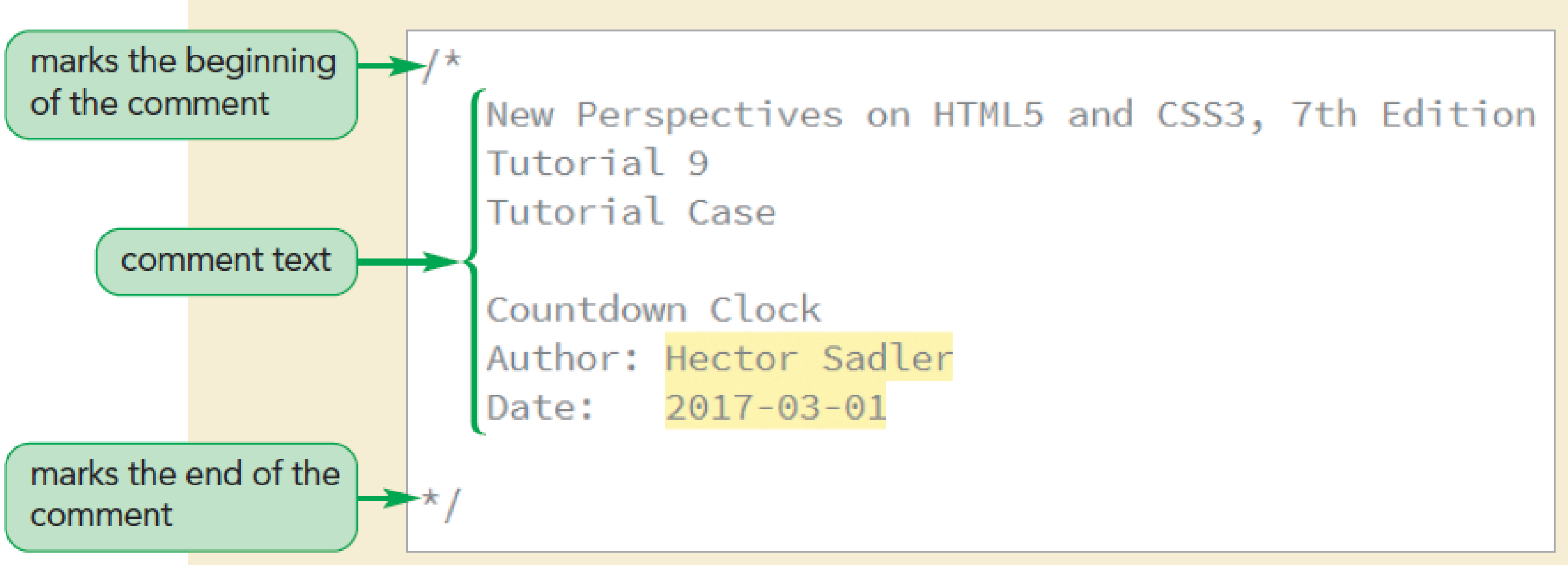
```
// comment text
```

- Syntax of multiple-line comments is as follows:

```
/*
     comment text spanning
     several lines
*/
```

# Creating a JavaScript Program (continued 2)

**Figure 9-6** Adding a JavaScript comment



marks the beginning of the comment
comment text
marks the end of the comment

```
/*
   New Perspectives on HTML5 and CSS3, 7th Edition
   Tutorial 9
   Tutorial Case

   Countdown Clock
   Author: Hector Sadler
   Date:   2017-03-01
*/
```

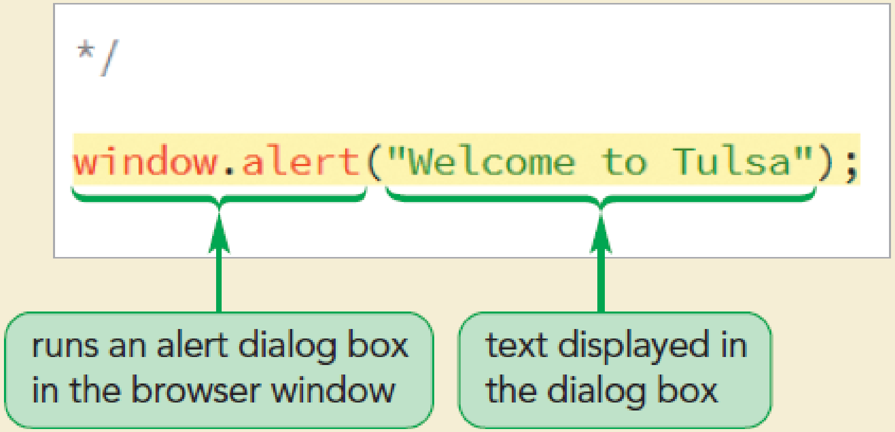
# Creating a JavaScript Program (continued 3)

- Writing a JavaScript Command
  - A command indicates an action for a browser to take
  - A command should end in a semicolon

```
JavaScript command;
```

# Creating a JavaScript Program (continued 4)

**Figure 9-7** Displaying a dialog box



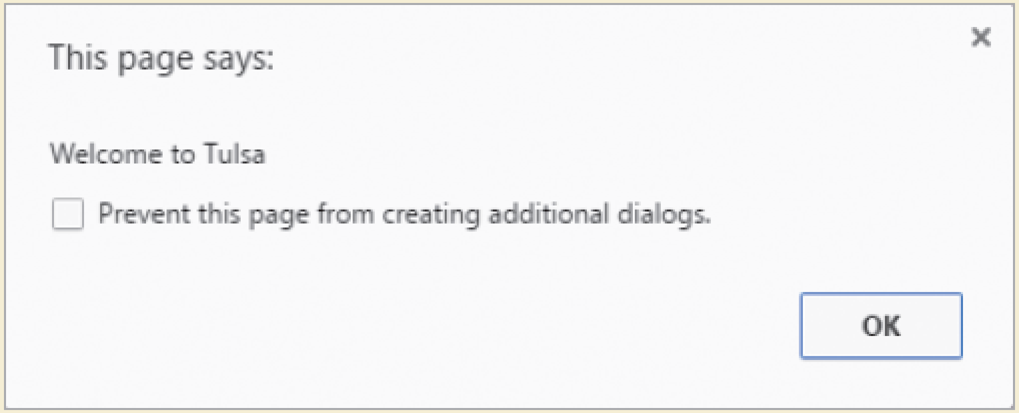
```
*/

window.alert("Welcome to Tulsa");
```

runs an alert dialog box in the browser window

text displayed in the dialog box

**Figure 9-8** Google Chrome dialog box



This page says:

Welcome to Tulsa

□ Prevent this page from creating additional dialogs.

# Creating a JavaScript Program (continued 5)

- Understanding JavaScript Syntax
  - JavaScript is case sensitive
  - Extra white space between commands is ignored
  - Line breaks placed within the name of a JavaScript command or a quoted text string cause an error

# Debugging your Code

- **Debugging:** Process of locating and fixing a programming error
- Types of errors
  - Load-time errors – occur when a script is first loaded by a browser
  - Run-time errors – occur during execution of a script without syntax errors
  - Logical errors – are free from syntax and executable mistakes but result in an incorrect output

# Opening a Debugger

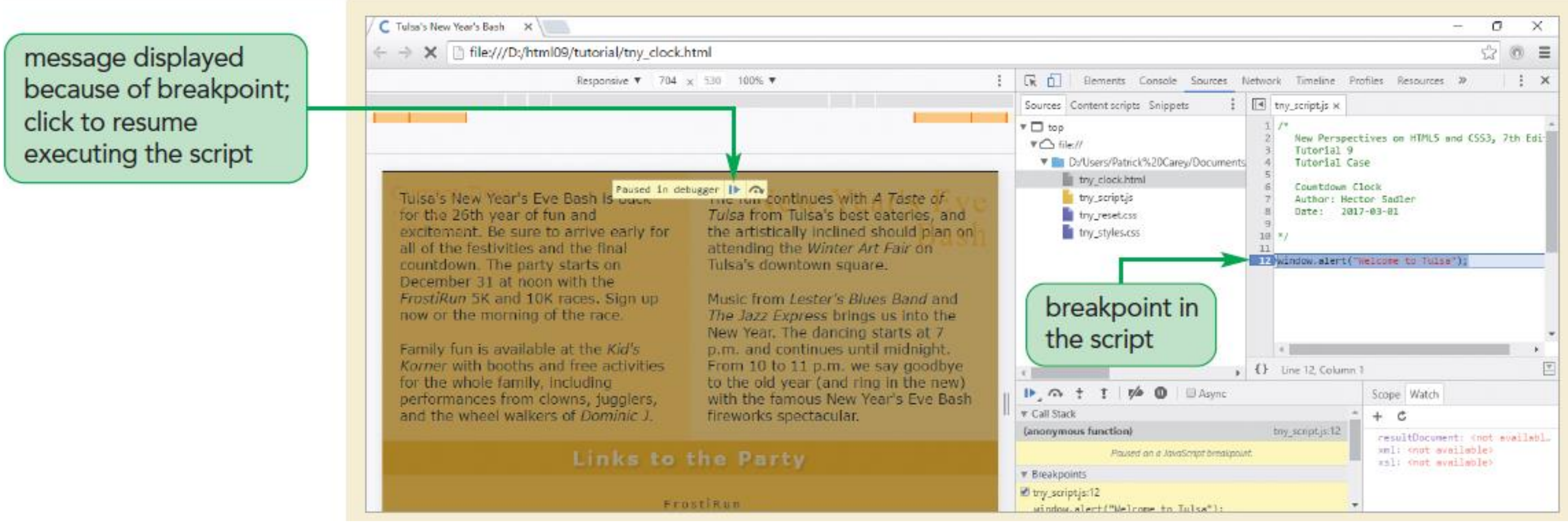
- Debugging tools locate and fix errors in JavaScript codes
- Shortcut to open a debugging tool is F12 key
- The tools can also be opened by selecting Developer Tools from the browser menu

# Inserting a Breakpoint

- A useful technique to locate the source of an error is to set up **breakpoints**
- **Breakpoints** are locations where a browser pauses a program to determine whether an error has occurred at that point during execution

# Inserting a Breakpoint (continued)

**Figure 9-10** Setting a breakpoint in Google Chrome



message displayed because of breakpoint; click to resume executing the script

breakpoint in the script

file:///D:/html09/tutorial/tny_clock.html

Tulsa's New Year's Eve Bash is back for the 26th year of fun and excitement. Be sure to arrive early for all of the festivities and the final countdown. The party starts on December 31 at noon with the *FrostiRun* 5K and 10K races. Sign up now or the morning of the race.

Family fun is available at the *Kid's Korner* with booths and free activities for the whole family, including performances from clowns, jugglers, and the wheel walkers of *Dominic J.*

The fun continues with *A Taste of Tulsa* from Tulsa's best eateries, and the artistically inclined should plan on attending the *Winter Art Fair* on Tulsa's downtown square.

Music from *Lester's Blues Band* and *The Jazz Express* brings us into the New Year. The dancing starts at 7 p.m. and continues until midnight. From 10 to 11 p.m. we say goodbye to the old year (and ring in the new) with the famous New Year's Eve Bash fireworks spectacular.

Links to the Party

FrostiRun

Paused in debugger

```
/*
   New Perspectives on HTML5 and CSS3, 7th Edi
   Tutorial 9
   Tutorial Case

   Countdown Clock
   Author: Hector Sadler
   Date:   2017-03-01

*/

window.alert("Welcome to Tulsa");
```

Line 12, Column 1

Call Stack

(anonymous function) tny_script.js:12

*Paused on a JavaScript breakpoint.*

Breakpoints

☑ tny_script.js:12
window.alert("Welcome to Tulsa");

```
resultDocument: <not availabl…
xml: <not available>
xsl: <not available>
```

# Applying Strict Usage of JavaScript

- **Strict mode** enables all lapses in syntax to result in load-time or run-time errors
- To run a script in strict mode, add the following statement to the first line of the file:

```
"use strict";
```

# Introducing Objects

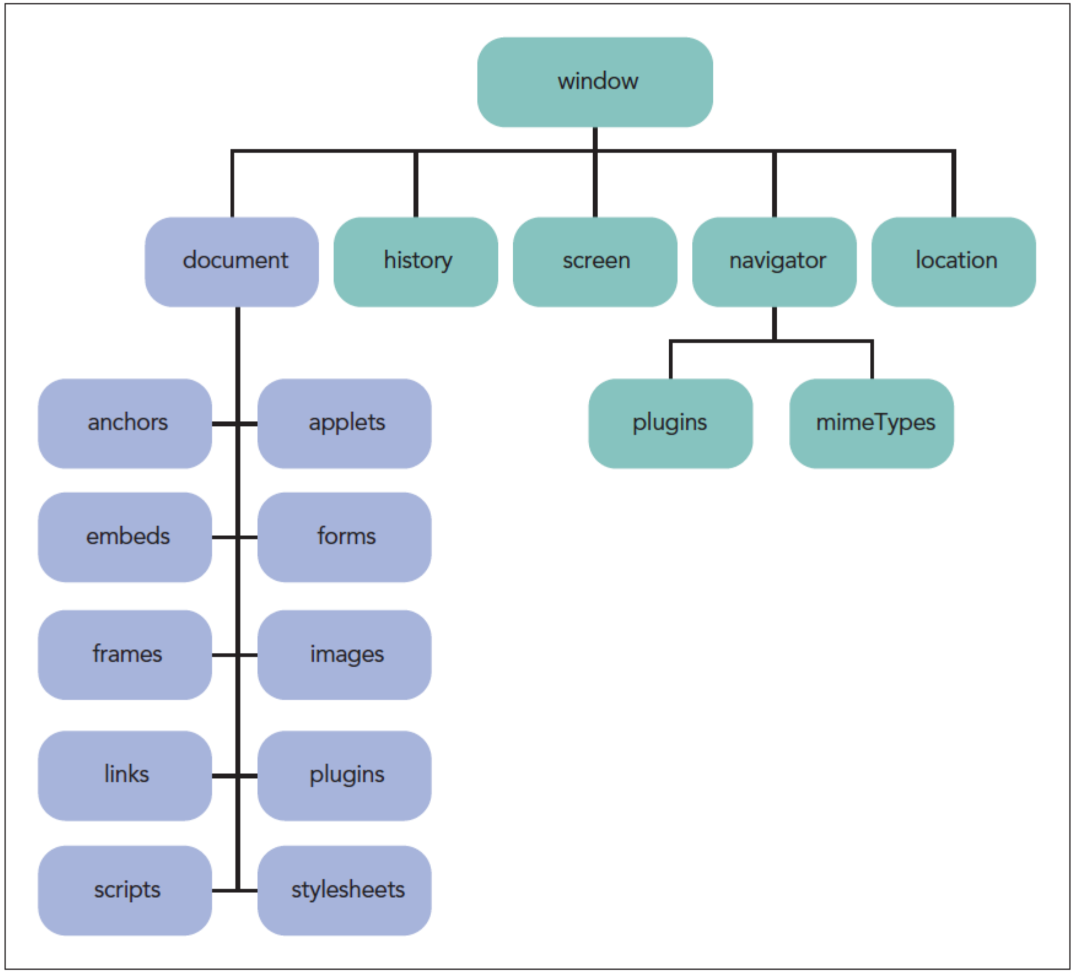
- **Object:** Entity within a browser or web page that has **properties** and **methods**
- **Properties:** Define objects
- **Methods:** Act upon objects
- JavaScript is an **object-based language** that manipulates an object by changing one or more of its properties

# Introducing Objects (continued 1)

- Types of JavaScript objects
  - **Built-in objects** – intrinsic to JavaScript language
  - **Browser objects** – part of browser
  - **Document objects** – part of web document
  - **Customized objects** – created by a programmer to use in an application
- **Browser object model (BOM)** and **document object model (DOM)** organize browser and document objects in hierarchical structures, respectively

# Introducing Objects (continued 2)

**Figure 9-12** Object hierarchy



# Object References

- Objects within the object hierarchy are referenced by their object names such as `window`, `document`, or `navigator`
- Objects can be referenced using the notation

  *`object1.object2.object3`*…

  where *`object1`* is at the top of the hierarchy, *`object2`* is a child of *`object1`*, and so on

# Referencing Object Collections

- **Object collections:** Objects organized into groups
- To reference a specific member of an object collection, use

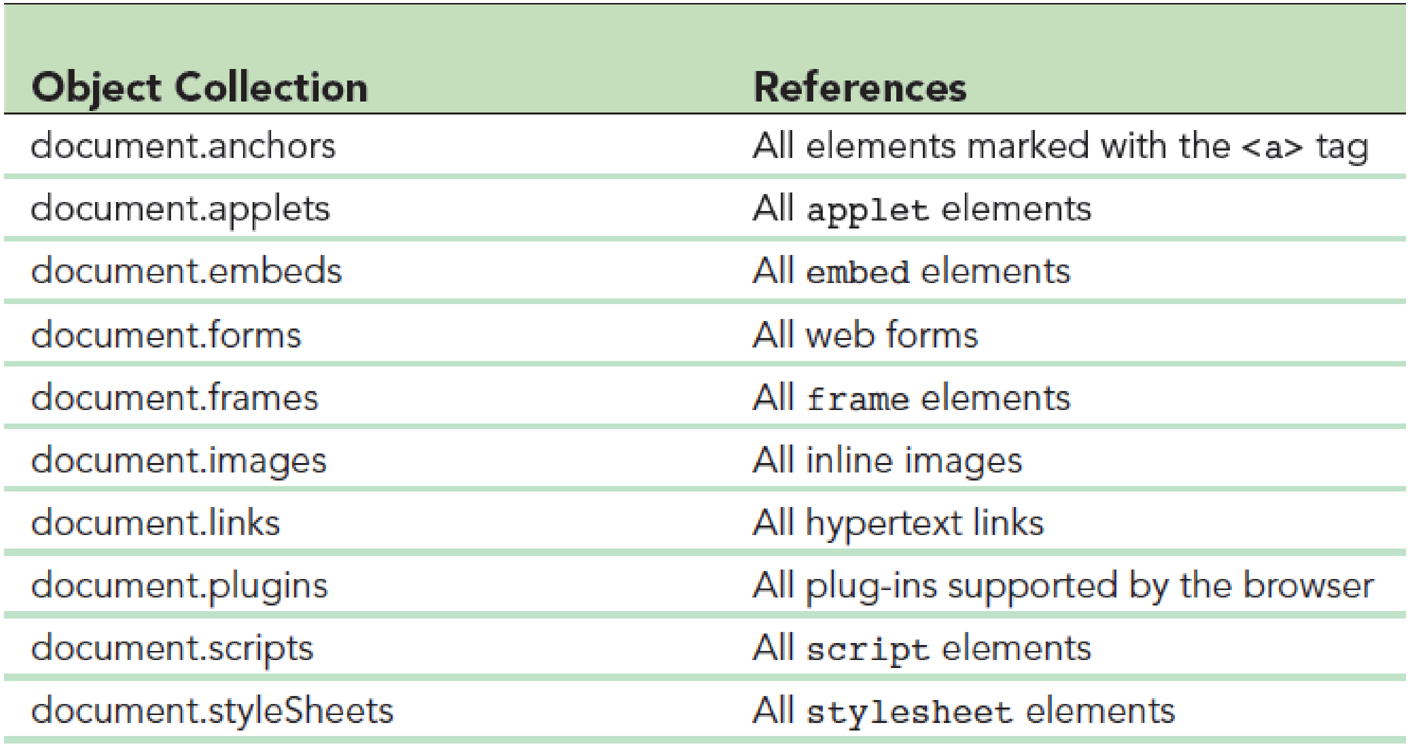
  `collection[idref]`

  or `collection.idref`

  where `collection` is a reference to the object collection and `idref` is either an index number or the value of `id` attribute

# Referencing Object Collections (continued)

Figure 9-13 Document object collections



| Object Collection | References |
|---|---|
| document.anchors | All elements marked with the `<a>` tag |
| document.applets | All `applet` elements |
| document.embeds | All `embed` elements |
| document.forms | All web forms |
| document.frames | All `frame` elements |
| document.images | All inline images |
| document.links | All hypertext links |
| document.plugins | All plug-ins supported by the browser |
| document.scripts | All `script` elements |
| document.styleSheets | All `stylesheet` elements |

# Referencing an Object by ID and Name

- An efficient approach to reference an element is to use its `id` attribute using the expression

  `document.getElementById(`*`id`*`)`

  where *`id`* is the value of `id` attribute

# Changing Properties and Applying Methods

- Object Properties
  - Object property is accessed using

    *`object.property`*

    where *`object`* is a reference to an object and *`property`* is a property associated with that object
  - **Read-only properties** cannot be modified

# Changing Properties and Applying Methods (continued)

- Applying a Method
  - Objects can be modified using methods
  - Methods are applied using the expression

    *`object`*`.`*`method`*`(`*`values`*`)`

    where *`object`* is a reference to an object, *`method`* is the name of the method applied to the object, and *`values`* is a comma-separated list of values associated with the method

# Writing HTML Code

- HTML code stored within a page element is referenced using

  ```
      element.innerHTML
  ```

  where *`element`* is an object reference to an element within a web document

# Writing HTML Code (continued 1)

- HTML code stored within a page element is referenced using

  ```
  element.innerHTML
  ```

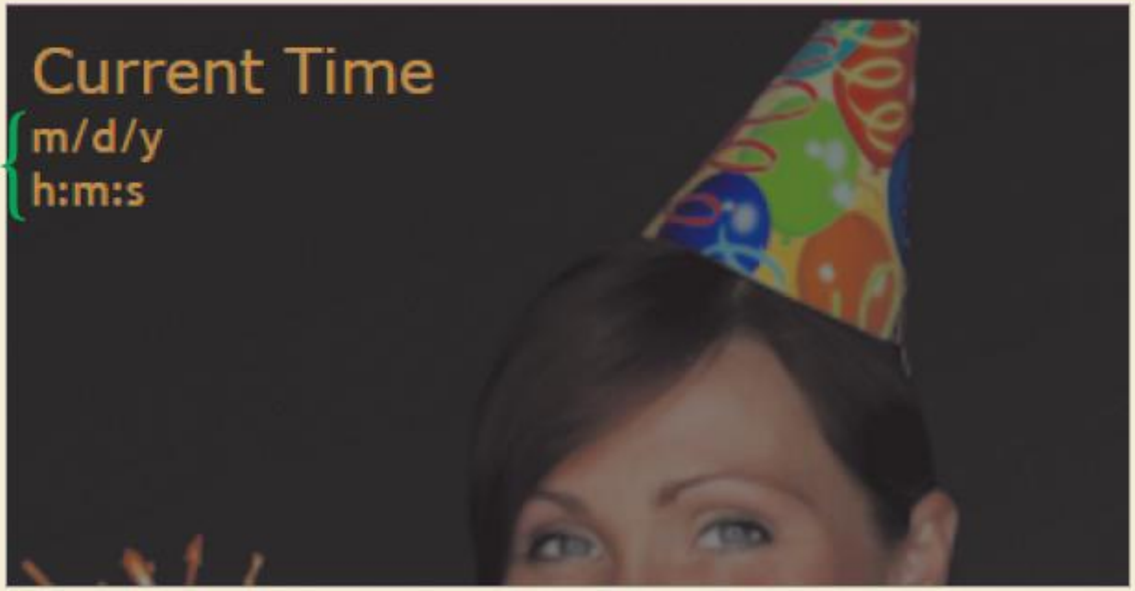
  where *element* is an object reference to an element within a web document

- For example,

  ```
  /* Display the current date and time */
  document.getElementById(“dateNow”).inne
  rHTML = “m/d/y<br />h:m:s”;
  ```

# Writing HTML Code (continued 2)

**Figure 9-15** Revised date and time content



Current Time

m/d/y

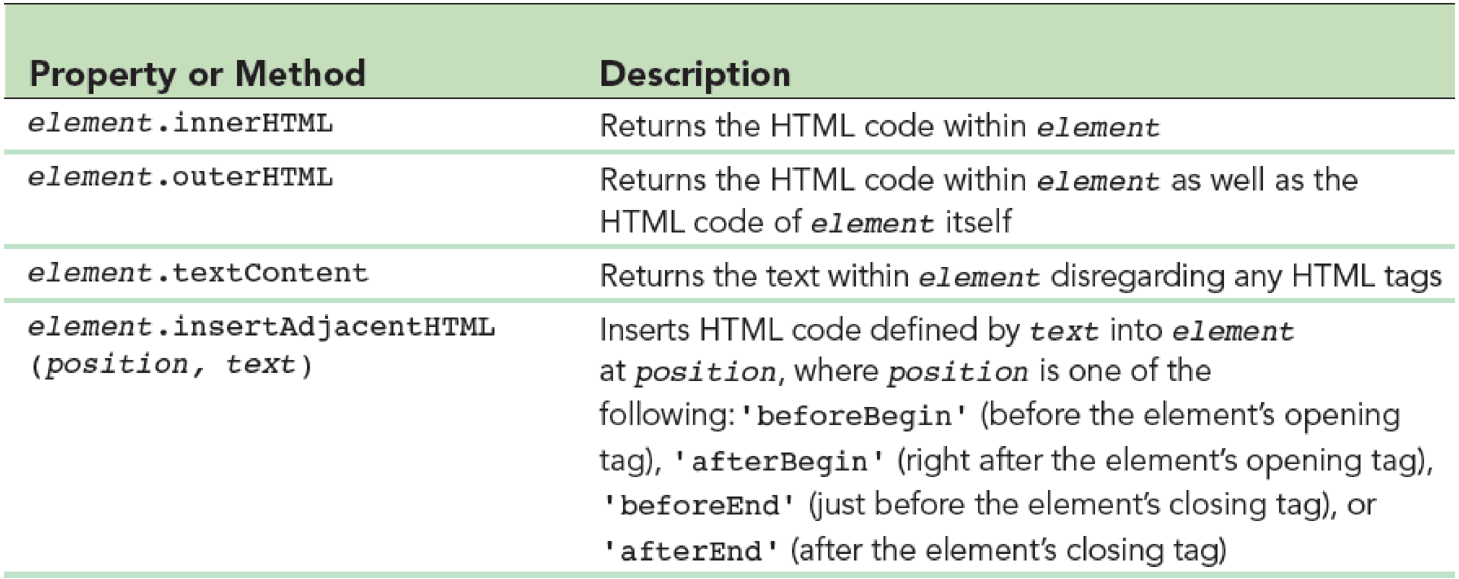
h:m:s

content written with JavaScript

© altafulla/Shutterstock.com

# Writing HTML Code (continued 3)

**Figure 9-16** Properties and methods to insert content



| Property or Method | Description |
|---|---|
| *element*.innerHTML | Returns the HTML code within *element* |
| *element*.outerHTML | Returns the HTML code within *element* as well as the HTML code of *element* itself |
| *element*.textContent | Returns the text within *element* disregarding any HTML tags |
| *element*.insertAdjacentHTML (*position*, *text*) | Inserts HTML code defined by *text* into *element* at *position*, where *position* is one of the following: 'beforeBegin' (before the element's opening tag), 'afterBegin' (right after the element's opening tag), 'beforeEnd' (just before the element's closing tag), or 'afterEnd' (after the element's closing tag) |

# Working with Variables

- **Variable:** Named item in a program that stores a data value
- Declaring a Variable
  - Introduced into a script by **declaring** the variable using the `var` keyword

    ```
    var variable = value;
    ```

    where *`variable`* is the name assigned to the variable and *`value`* is the variable’s initial value

# Working with Variables (continued)

- Conditions to assign variable names in JavaScript
  - First character must be either a letter or an underscore character ( _ )
  - The characters after the first character can be letters, numbers, or underscore characters
  - No spaces
  - No using names that are part of JavaScript language

# Variables and Data Types

- **Data type:** Type of information stored in a variable
- Supported data types
  - Numeric value
  - Text string
  - Boolean value
  - Object
  - null value

# Variables and Data Types (continued)

- **Numeric value:** Any number
- **Text string:** Group of characters enclosed within either double or single quotation marks
- **Boolean value:** Indicates the truth or falsity of a statement

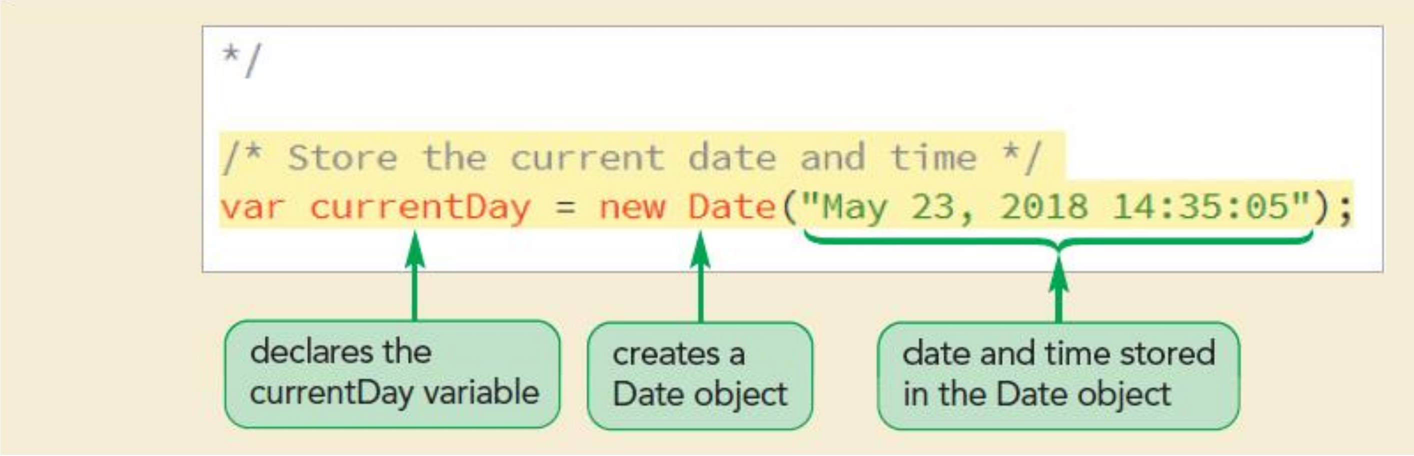
# Variables and Data Types (continued 1)

- Object – Simplifies code by removing the need to rewrite complicated object references
- `null` value – Indicates that no value has yet been assigned to a variable

# Working with Date Objects

- **Date object:** Built-in JavaScript object used to store information about dates and times

Figure 9-19 Creating a Date object



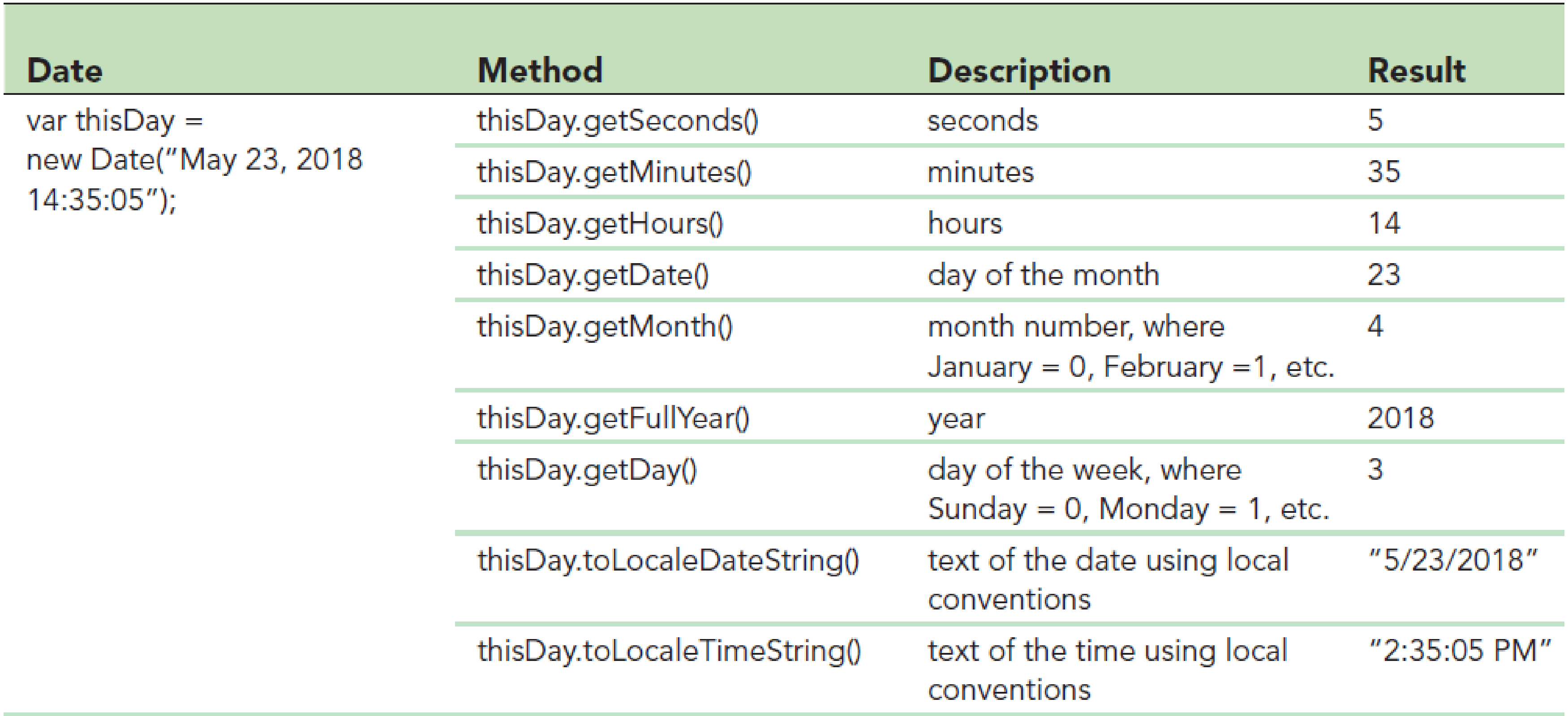
```
*/

/* Store the current date and time */
var currentDay = new Date("May 23, 2018 14:35:05");
```

declares the currentDay variable
creates a Date object
date and time stored in the Date object

# Working with Date Objects (continued 1)

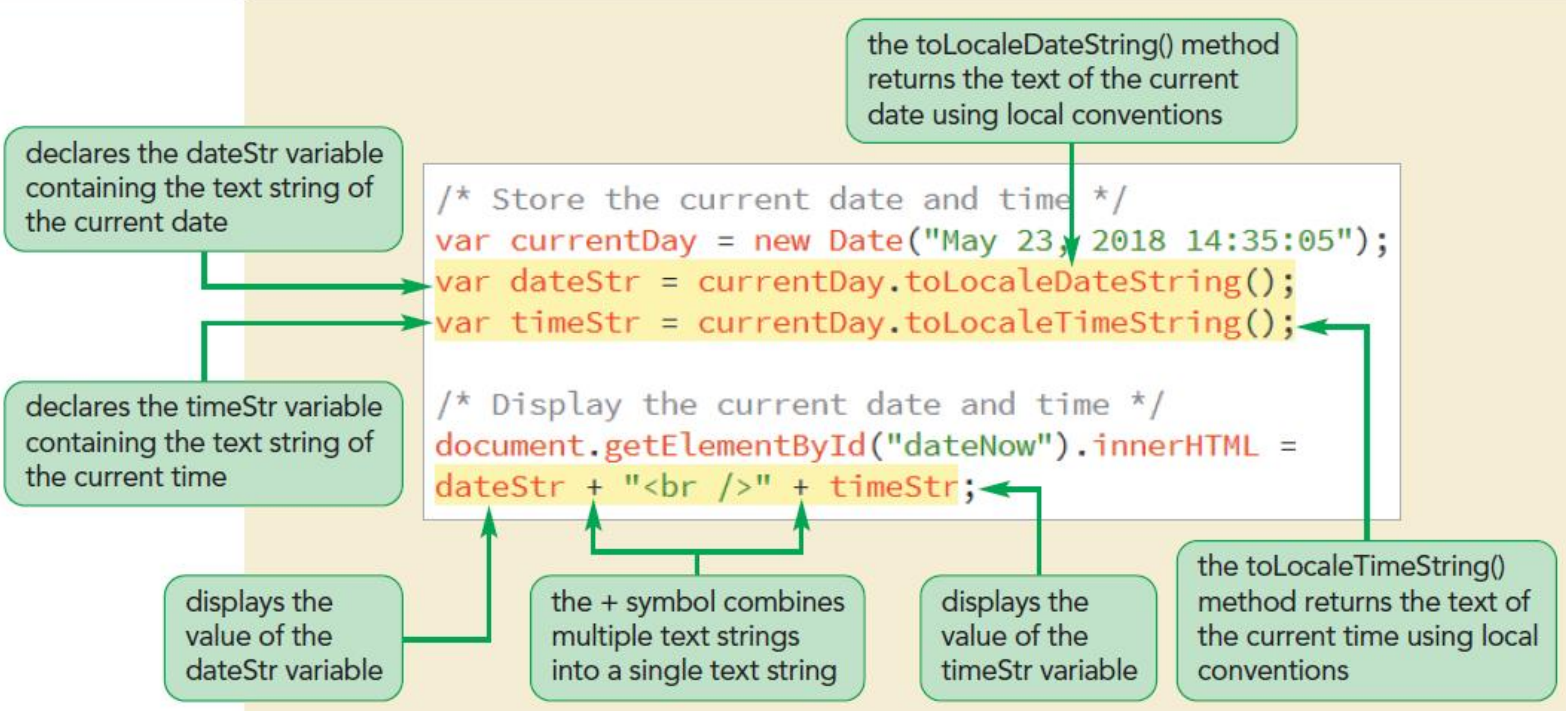
**Figure 9-20** Methods of the Date object



| Date | Method | Description | Result |
|---|---|---|---|
| var thisDay =<br>new Date("May 23, 2018<br>14:35:05"); | thisDay.getSeconds() | seconds | 5 |
| | thisDay.getMinutes() | minutes | 35 |
| | thisDay.getHours() | hours | 14 |
| | thisDay.getDate() | day of the month | 23 |
| | thisDay.getMonth() | month number, where January = 0, February =1, etc. | 4 |
| | thisDay.getFullYear() | year | 2018 |
| | thisDay.getDay() | day of the week, where Sunday = 0, Monday = 1, etc. | 3 |
| | thisDay.toLocaleDateString() | text of the date using local conventions | "5/23/2018" |
| | thisDay.toLocaleTimeString() | text of the time using local conventions | "2:35:05 PM" |

# Working with Date Objects (continued 2)

**Figure 9-21** Displaying dates and times



```
/* Store the current date and time */
var currentDay = new Date("May 23, 2018 14:35:05");
var dateStr = currentDay.toLocaleDateString();
var timeStr = currentDay.toLocaleTimeString();

/* Display the current date and time */
document.getElementById("dateNow").innerHTML =
dateStr + "<br />" + timeStr;
```

the toLocaleDateString() method returns the text of the current date using local conventions

declares the dateStr variable containing the text string of the current date

declares the timeStr variable containing the text string of the current time

the toLocaleTimeString() method returns the text of the current time using local conventions

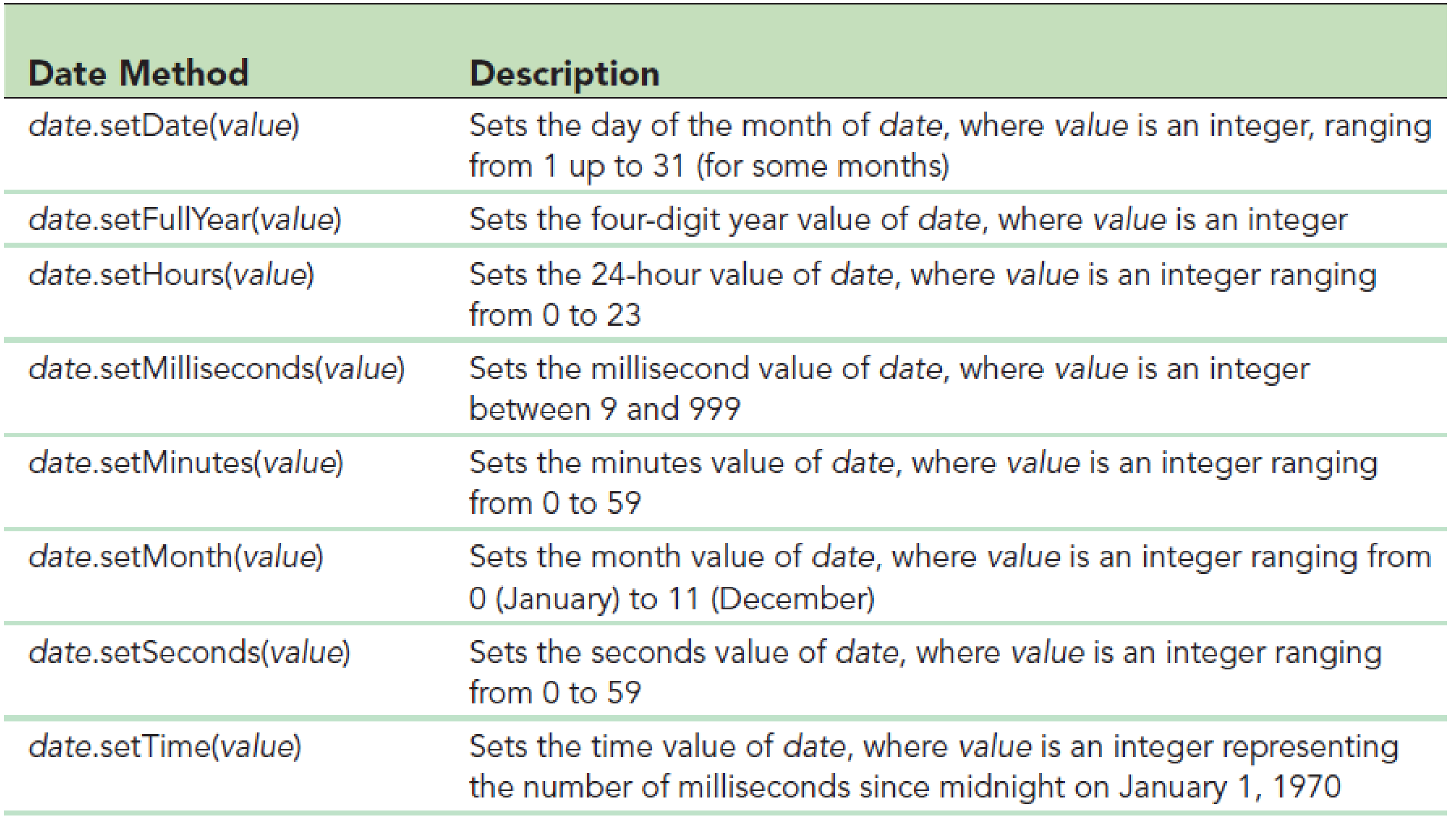
displays the value of the dateStr variable

the + symbol combines multiple text strings into a single text string

displays the value of the timeStr variable

# Setting Date and Time Values

**Figure 9-23** JavaScript methods to set values of the Date object



| Date Method | Description |
|---|---|
| *date*.setDate(*value*) | Sets the day of the month of *date*, where *value* is an integer, ranging from 1 up to 31 (for some months) |
| *date*.setFullYear(*value*) | Sets the four-digit year value of *date*, where *value* is an integer |
| *date*.setHours(*value*) | Sets the 24-hour value of *date*, where *value* is an integer ranging from 0 to 23 |
| *date*.setMilliseconds(*value*) | Sets the millisecond value of *date*, where *value* is an integer between 9 and 999 |
| *date*.setMinutes(*value*) | Sets the minutes value of *date*, where *value* is an integer ranging from 0 to 59 |
| *date*.setMonth(*value*) | Sets the month value of *date*, where *value* is an integer ranging from 0 (January) to 11 (December) |
| *date*.setSeconds(*value*) | Sets the seconds value of *date*, where *value* is an integer ranging from 0 to 59 |
| *date*.setTime(*value*) | Sets the time value of *date*, where *value* is an integer representing the number of milliseconds since midnight on January 1, 1970 |

# Working with Operators and Operands

- **Operator:** Symbol used to act upon an item or a variable within an expression
- **Operands**: Variables or expressions that operators act upon
- Types of operators
  - **Binary operators** – require two operands in an expression

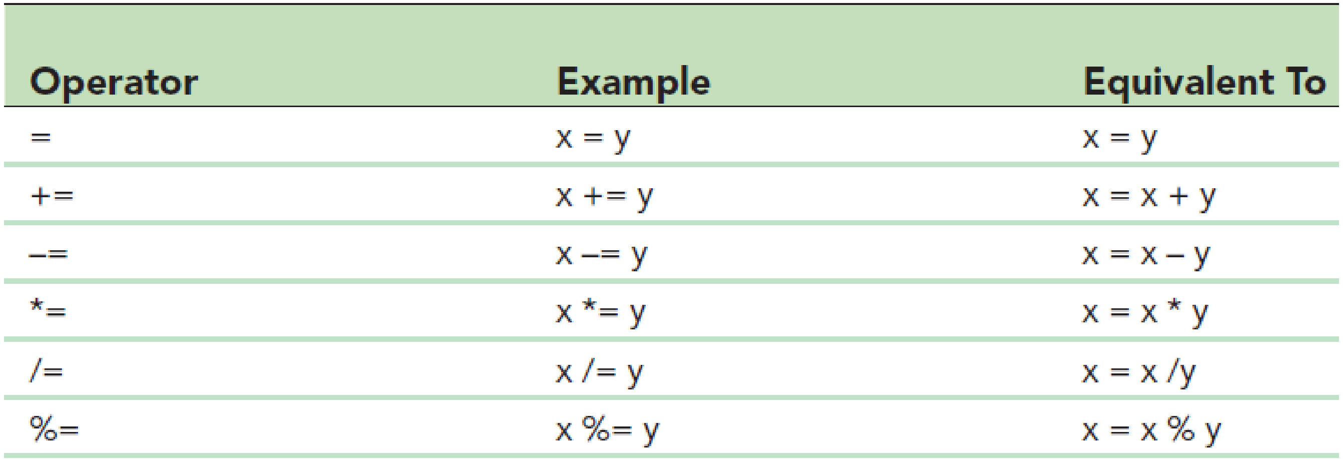
# Working with Operators and Operands (continued)

- **Unary operators** – require only one operand
  - **Increment operator (++) –** increases the value of an operand by 1
  - **Decrement operator (--) –** decreases the value of an operand by 1

# Using Assignment Operators

- **Assignment operator:** Assigns a value to an item

**Figure 9-25** JavaScript assignment operators



| Operator | Example | Equivalent To |
|---|---|---|
| = | x = y | x = y |
| += | x += y | x = x + y |
| –= | x –= y | x = x – y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x /y |
| %= | x %= y | x = x % y |

# Working with the Math Object

- **Math object:** Built-in object used to perform mathematical tasks and store mathematical values
- Syntax to apply a Math method is

  ```
  Math.method(expression)
  ```

  where *`method`* is the method applied to a mathematical expression

# Working with the Math Object (continued 1)

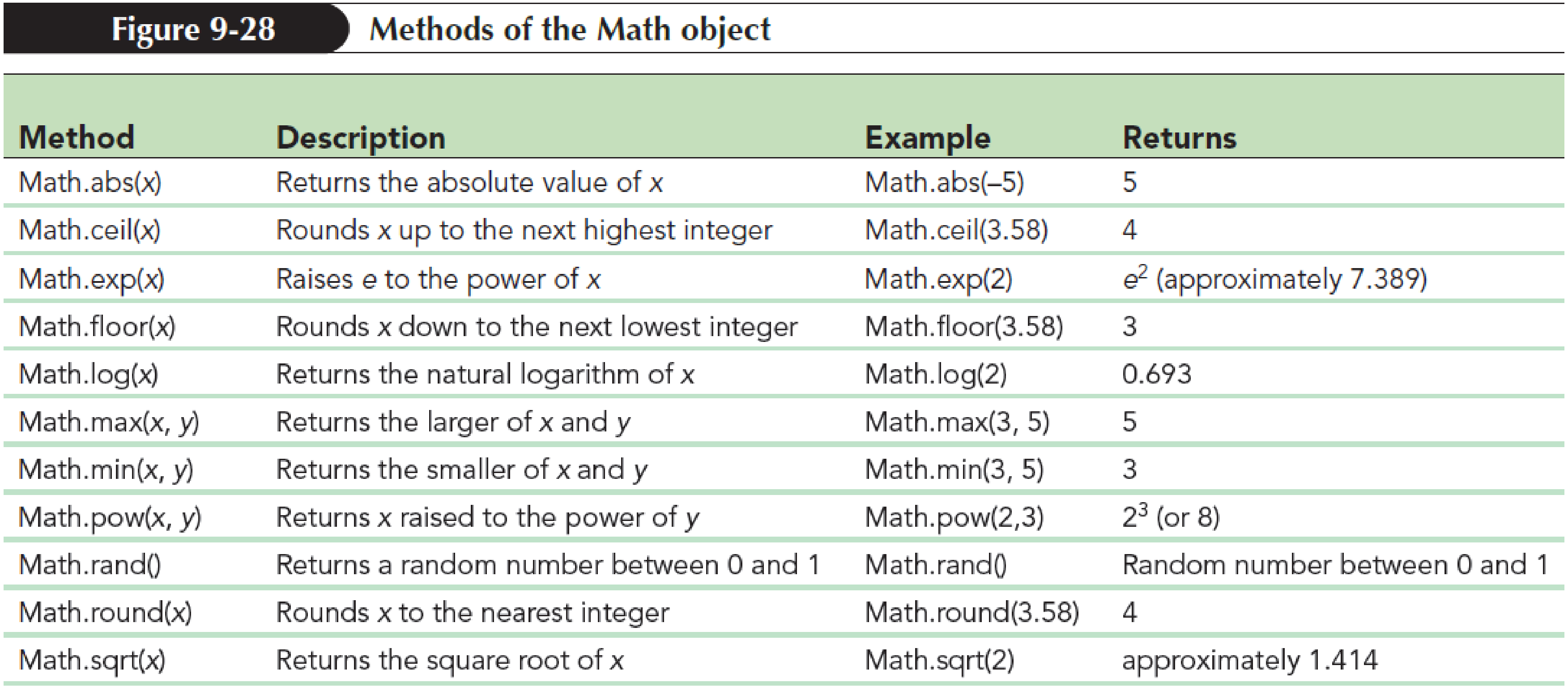
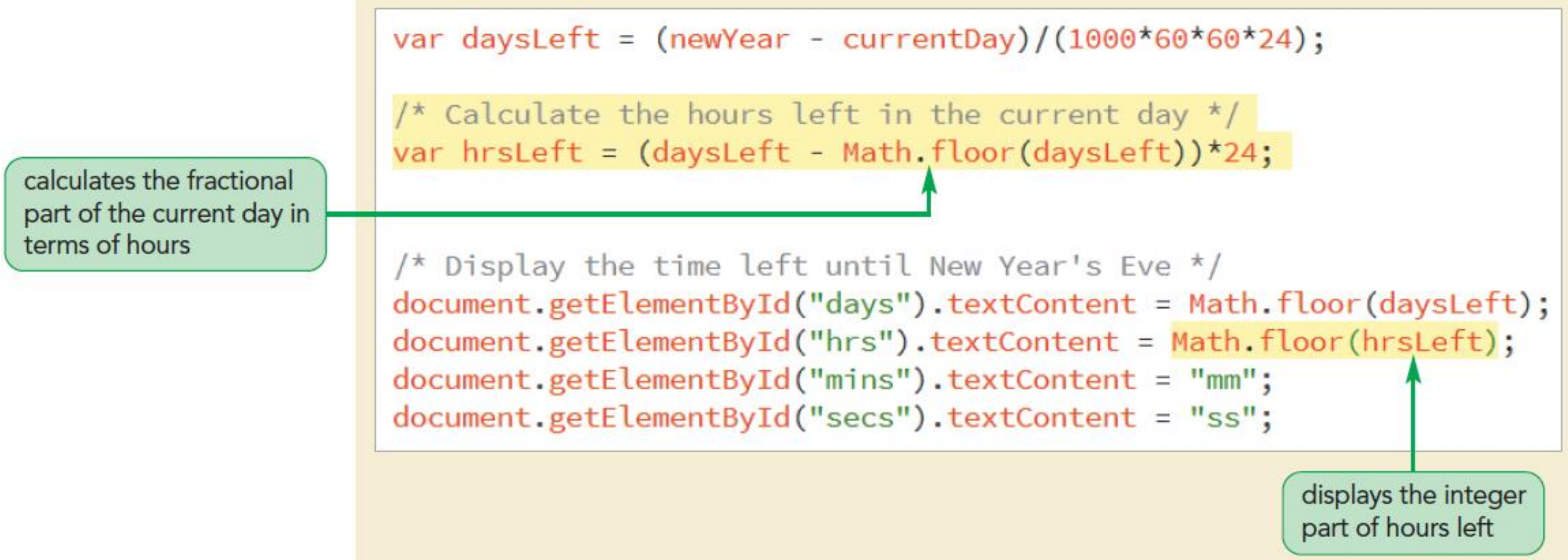


**Figure 9-28** Methods of the Math object

| Method | Description | Example | Returns |
|---|---|---|---|
| Math.abs(*x*) | Returns the absolute value of *x* | Math.abs(–5) | 5 |
| Math.ceil(*x*) | Rounds *x* up to the next highest integer | Math.ceil(3.58) | 4 |
| Math.exp(*x*) | Raises *e* to the power of *x* | Math.exp(2) | $e^2$ (approximately 7.389) |
| Math.floor(*x*) | Rounds *x* down to the next lowest integer | Math.floor(3.58) | 3 |
| Math.log(*x*) | Returns the natural logarithm of *x* | Math.log(2) | 0.693 |
| Math.max(*x*, *y*) | Returns the larger of *x* and *y* | Math.max(3, 5) | 5 |
| Math.min(*x*, *y*) | Returns the smaller of *x* and *y* | Math.min(3, 5) | 3 |
| Math.pow(*x*, *y*) | Returns *x* raised to the power of *y* | Math.pow(2,3) | $2^3$ (or 8) |
| Math.rand() | Returns a random number between 0 and 1 | Math.rand() | Random number between 0 and 1 |
| Math.round(*x*) | Rounds *x* to the nearest integer | Math.round(3.58) | 4 |
| Math.sqrt(*x*) | Returns the square root of *x* | Math.sqrt(2) | approximately 1.414 |

# Working with the Math Object (continued 2)

**Figure 9-30** Calculating the hours left in the current day



```
var daysLeft = (newYear - currentDay)/(1000*60*60*24);

/* Calculate the hours left in the current day */
var hrsLeft = (daysLeft - Math.floor(daysLeft))*24;


/* Display the time left until New Year's Eve */
document.getElementById("days").textContent = Math.floor(daysLeft);
document.getElementById("hrs").textContent = Math.floor(hrsLeft);
document.getElementById("mins").textContent = "mm";
document.getElementById("secs").textContent = "ss";
```

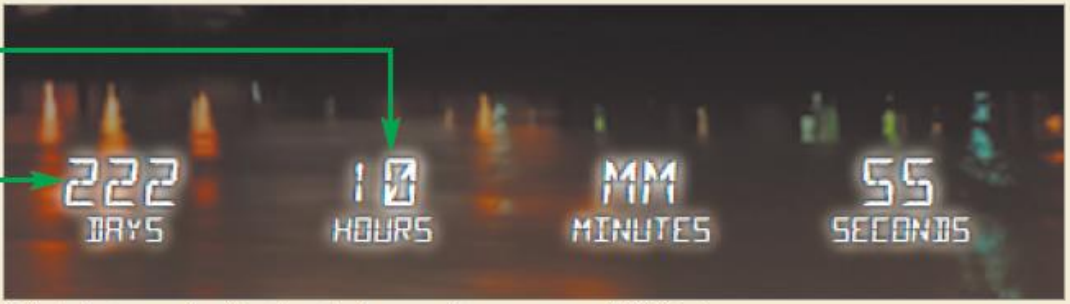
calculates the fractional part of the current day in terms of hours

displays the integer part of hours left

**Figure 9-31** Days and hours left until January 1st

number of hours left in the current day

number of days left in the current year



© jbdphotography/Shutterstock.com; Source: www.1001fonts.com

# Using Math Constants

- Math functions refer to built-in constants stored in JavaScript Math object
- Syntax to access mathematical constants is

```
Math.CONSTANT
```

where `CONSTANT` is the name of one of the mathematical constants supported by `Math` object

# Using Math Constants (continued)

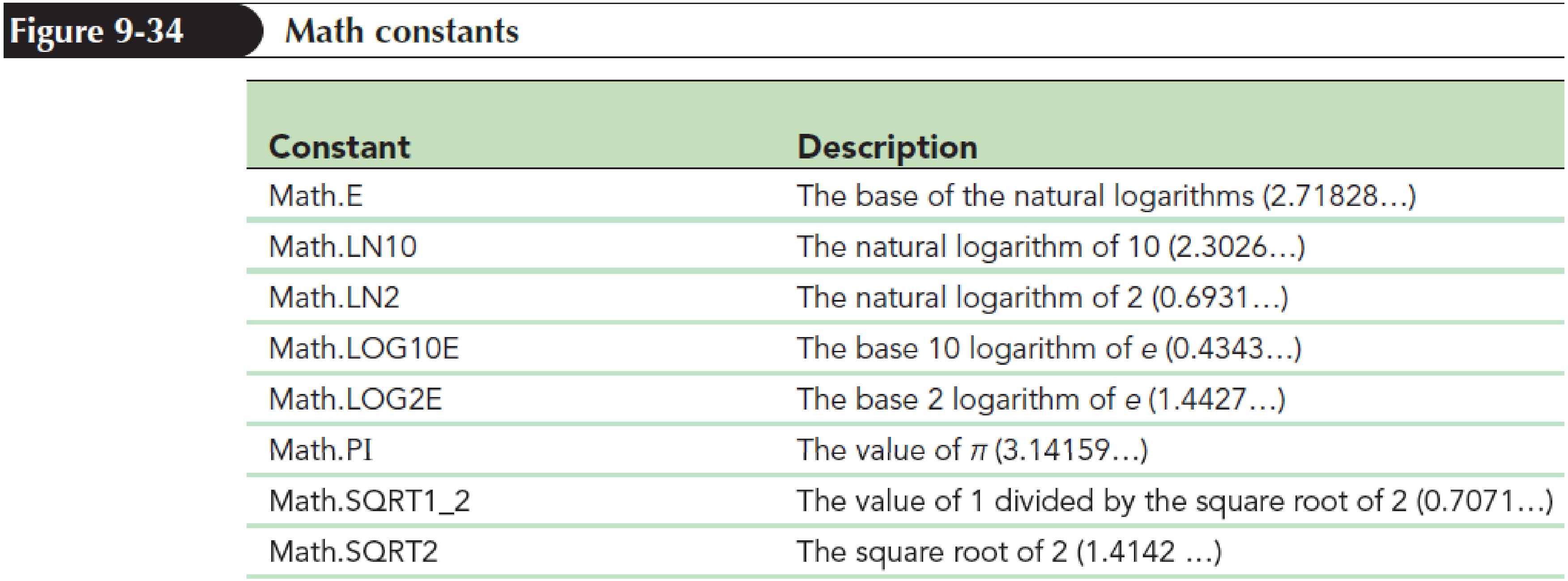


**Figure 9-34** Math constants

| Constant | Description |
|---|---|
| Math.E | The base of the natural logarithms (2.71828…) |
| Math.LN10 | The natural logarithm of 10 (2.3026…) |
| Math.LN2 | The natural logarithm of 2 (0.6931…) |
| Math.LOG10E | The base 10 logarithm of *e* (0.4343…) |
| Math.LOG2E | The base 2 logarithm of *e* (1.4427…) |
| Math.PI | The value of $\pi$ (3.14159…) |
| Math.SQRT1_2 | The value of 1 divided by the square root of 2 (0.7071…) |
| Math.SQRT2 | The square root of 2 (1.4142 …) |

# Working with JavaScript Functions

- **Function:** Collection of commands that performs an action or returns a value
- A function name identifies a function and a set of commands that are run when the function is called
- **Parameters:** Variables associated with the function

# Working with JavaScript Functions (continued)

- General syntax of a JavaScript function is

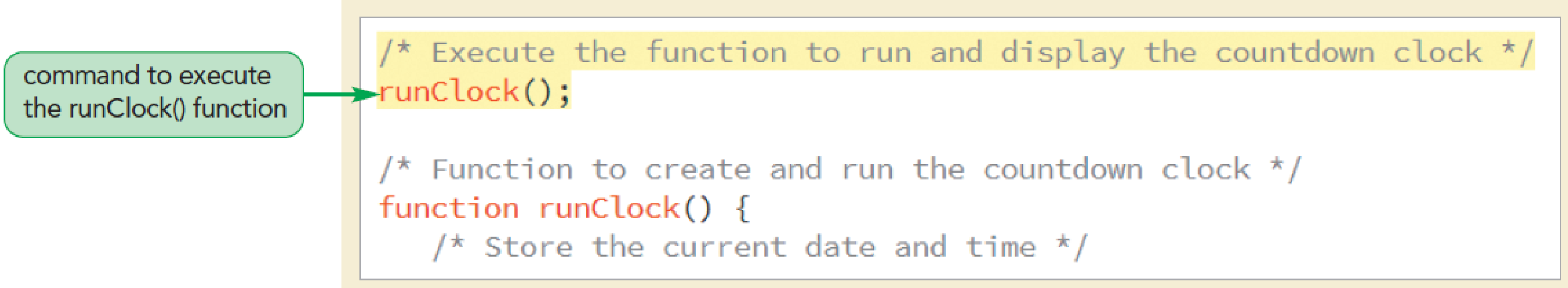
```
function function_name(parameters){
         commands
}
```

  where,

  - *function_name* is the name of the function
  - *parameters* is a comma-separated list of variables used in the function
  - *commands* is the set of statements run by the function

# Calling a Function

**Figure 9-37** Calling the runClock() function



command to execute the runClock() function

```
/* Execute the function to run and display the countdown clock */
runClock();

/* Function to create and run the countdown clock */
function runClock() {
   /* Store the current date and time */
```

# Creating a Function to Return a Value

- Functions return values using `return` statement

```
function function_name(parameters){
   commands
   return value;
}
```

where *value* is the calculated value that is returned by the function

# Running Timed Commands

- Methods to update the current and the remaining time constantly
  - Time-delayed commands
  - Timed-interval commands
- Working with Time-Delayed Commands
  - **Time-delayed commands:** JavaScript commands run after a specified amount of time has passed

# Running Timed Commands (continued 1)

- Time delay is defined using

  ```
  setTimeout("command", delay);
  ```

  where *command* is a JavaScript command and *delay* is the delay time in milliseconds before a browser runs the command

- Running Commands at Specified Intervals
  - The timed-interval command instructs browsers to run a command repeatedly at a specified interval

# Running Timed Commands (continued 2)

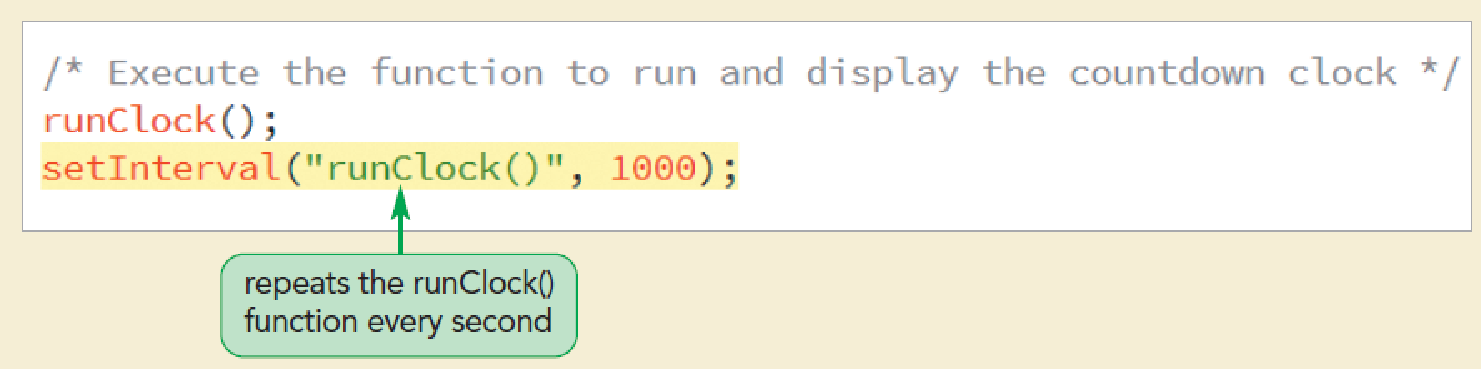
- Timed-interval commands are applied using `setInterval()` method

```
setInterval(“command”, interval);
```

  where *`interval`* is the interval in milliseconds before the command is run again

# Running Timed Commands (continued 3)

Figure 9-38 Repeating the runClock() function



```
/* Execute the function to run and display the countdown clock */
runClock();
setInterval("runClock()", 1000);
```

repeats the runClock() function every second

# Controlling How JavaScript Works with Numeric Values

- Handling Illegal Operations
  - Mathematical operations can return results that are not numeric values
  - JavaScript returns `NaN` if an operation does not involve only numeric values

# Controlling How JavaScript Works with Numeric Values (continued)

- `isNaN()` function returns a Boolean value of `true` if the value is not numeric and `false` if otherwise
- `Infinity` value is generated for an operation whose result is less than the smallest numeric value and greater than the largest numeric value supported by JavaScript

# Defining a Number Format

- JavaScript stores a numeric value to 16 decimal places of accuracy
- The number of digits displayed by browsers is controlled using `toFixed()` method

```
    value.toFixed(n)
```

where *`value`* is the value or variable and *`n`* is the number of decimal places displayed in the output

# Defining a Number Format (continued)

- `toFixed()` limits the number of decimals displayed by a value and converts the value into a text string
- `toFixed()` rounds the last digit in an expression rather than truncating it

# Converting Between Numbers and Text

- + operator adds a text string to a number
- For example,

```
testNumber = 123; // numeric value
testString = testNumber + “”; // text
string
```

where + operator concatenates a numeric value with an empty text string resulting in a text string

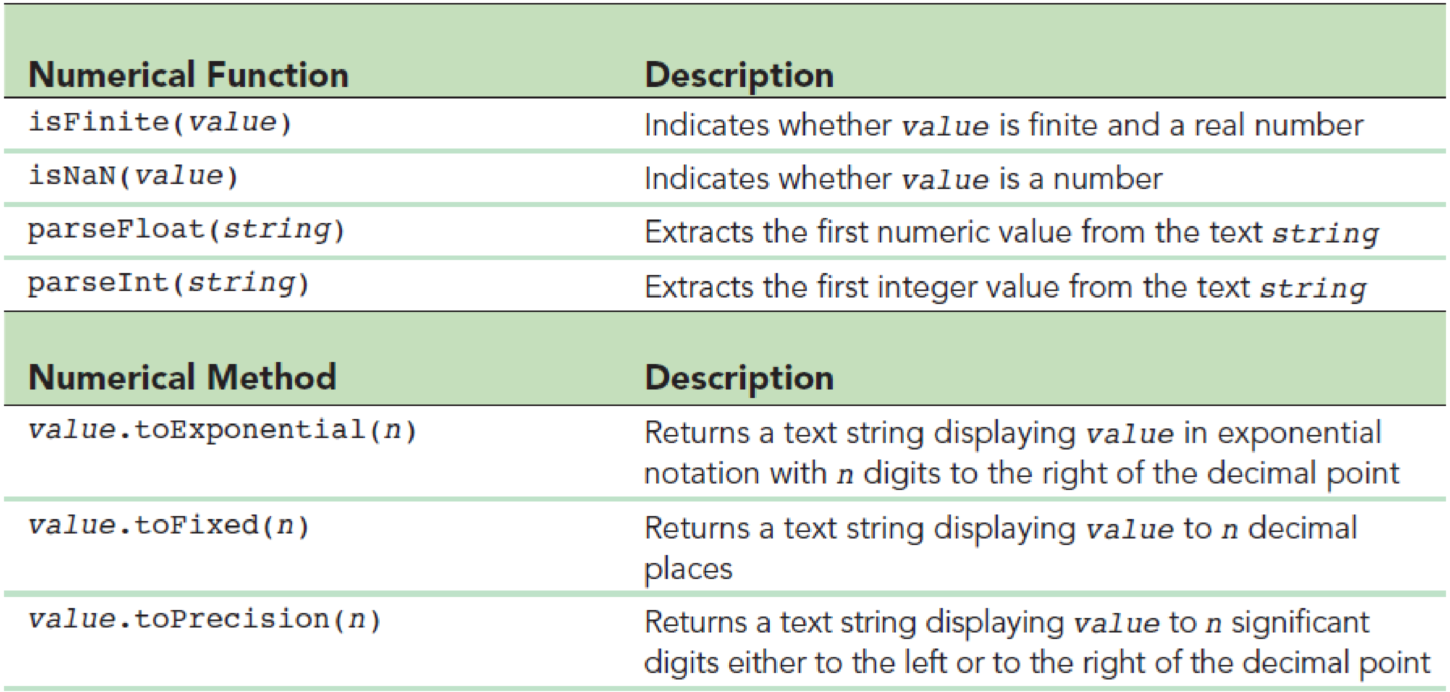
# Converting Between Numbers and Text (continued 1)

- `parseInt()` function extracts the leading integer value from a text string
- It returns the integer value from the text string by discarding any non-integer characters
- Example,

```
parseInt(“120.88 lbs”); // returns 120
parseInt(“weight equals 120 lbs”); //
returns NaN
```

# Converting Between Numbers and Text (continued 2)

**Figure 9-39** Numerical functions and methods



| Numerical Function | Description |
|---|---|
| `isFinite(`*`value`*`)` | Indicates whether *`value`* is finite and a real number |
| `isNaN(`*`value`*`)` | Indicates whether *`value`* is a number |
| `parseFloat(`*`string`*`)` | Extracts the first numeric value from the text *`string`* |
| `parseInt(`*`string`*`)` | Extracts the first integer value from the text *`string`* |

| Numerical Method | Description |
|---|---|
| *`value`*`.toExponential(`*`n`*`)` | Returns a text string displaying *`value`* in exponential notation with *`n`* digits to the right of the decimal point |
| *`value`*`.toFixed(`*`n`*`)` | Returns a text string displaying *`value`* to *`n`* decimal places |
| *`value`*`.toPrecision(`*`n`*`)` | Returns a text string displaying *`value`* to *`n`* significant digits either to the left or to the right of the decimal point |