COURSE TECHNOLOGY
CENGAGE Learning

JavaScript, Sixth Edition

Chapter 2
Working with Functions, Data Types, and Operators

Objectives

When you complete this chapter, you will be able to:
- Use functions to organize your JavaScript code
- Use expressions and operators
- Identify the order of operator precedence in an expression

Working with Functions

- Methods
  - Procedures associated with an object
- Functions
  - Related group of JavaScript statements
  - Executed as a single unit
  - Virtually identical to methods
    - Not associated with an object
  - Must be contained within a `script` element

Defining Functions

- Named function
  - Related statements assigned a name
  - Call, or reference, named function to execute it
- Anonymous function
  - Related statements with no name assigned
  - Work only where they are located in code
- Use named function when you want to reuse code
- Use anonymous function for code that runs only once

Defining Functions (cont'd.)

- Function definition
  - Lines making up a function
- Named function syntax

```
function name_of_function(parameters) {
    statements;
}
```

- Anonymous function syntax

```
function (parameters) {
    statements;
}
```

Defining Functions (cont'd.)

- Parameter
  - Variable used within a function
  - Placed within parentheses following a function name
  - Multiple parameters allowed

```
calculateVolume(length, width, height)
```

Defining Functions (cont'd.)

- Function statements
  - Do the actual work
  - Contained within function braces
- Put functions in an external .js file
  - Reference at bottom of body section

```
function calculateVolume(length, width, height) {
    var volume = length * width * height;
    document.write(volume);
}
```

Calling Functions

- To execute a named function:
  - Must invoke, or call, it
- Function call
  - Code calling a function
  - Consists of function name followed by parentheses
    - Contains any variables or values assigned to the function parameters
- Arguments (actual parameters)
  - Variables (values) placed in the function call statement parentheses

Calling Functions (cont'd.)

- Passing arguments
  - Sending arguments to parameters of a called function
    - Argument value assigned to the corresponding parameter value in the function definition

Calling Functions (cont'd.)

- Handling events
  - Three options
    - Specify function as value for HTML attribute

```
<input type="submit" onclick="showMessage()" />
```

    - Specify function as property value for object

```
document.getElementById("submitButton").onclick =↵    showMessage;
```

    - Use `addEventListener()` method

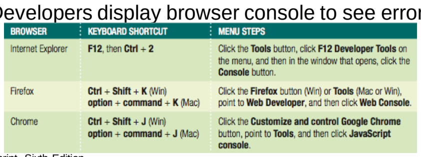
```
var submit = document.getElementById("submitButton");
submit.addEventListener("click", showMessage, false);
```

Calling Functions (cont'd.)

- Adding an event listener is most flexible
  - Separates HTML and JavaScript code
  - Can specify several event handlers for a single event
- IE8 requires use of the `attachEvent()` method instead of `addEventListener()` (see Chapter 3)

Locating Errors with the Browser Console

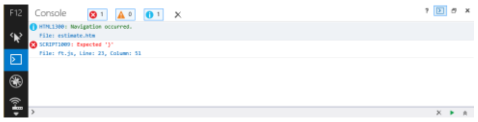
- Unintentional coding mistakes keep code from working
  - Browsers generate error messages in response
  - Messages displayed in browser console pane
  - Hidden by default to avoid alarming users
- Developers display browser console to see errors



BROWSER	KEYBOARD SHORTCUT	MENU STEPS
Internet Explorer	**F12**, then **Ctrl** + **2**	Click the **Tools** button, click **F12 Developer Tools** on the menu, and then in the window that opens, click the **Console** button.
Firefox	**Ctrl** + **Shift** + **K** (Win) **option** + **command** + **K** (Mac)	Click the **Firefox** button (Win) or **Tools** (Mac or Win), point to **Web Developer**, and then click **Web Console**.
Chrome	**Ctrl** + **Shift** + **J** (Win) **option** + **command** + **J** (Mac)	Click the **Customize and control Google Chrome** button, point to **Tools**, and then click **JavaScript console**.

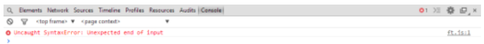
Locating Errors with the Browser Console (cont'd.)

- Consoles specify a line number with each error



```
HTML1300: Navigation occurred.
File: estimate.htm
SCRIPT1009: Expected '}'
File: ft.js, Line: 23, Column: 51
```

Figure 2-3: Internet Explorer browser console



```
Uncaught SyntaxError: Unexpected end of input                ft.js:1
```

Figure 2-4: Chrome browser console

Using Return Statements

- Can return function value to a calling statement
- Return statement
  - Returns a value to the statement calling the function
  - Use the `return` keyword with the variable or value to send to the calling statement
- Example:

```
function averageNumbers(a, b, c) {
   var sum_of_numbers = a + b + c;
   var result = sum_of_numbers / 3;
   return result;
}
```

Understanding Variable Scope

- Variable scope
  - Where in code a declared variable can be used
- Global variable
  - Declared outside a function
    - Available to all parts of code
- Local variable
  - Declared inside a function
    - Only available within the function in which it is declared
  - Cease to exist when the function ends
  - Keyword `var` required

Understanding Variable Scope (cont' d.)

- Good programming technique
  - Always use the `var` keyword when declaring variables
    - Clarifies where and when variable used
- Poor programming technique
  - Declaring a global variable inside of a function by not using the `var` keyword
    - Harder to identify global variables in your scripts

Understanding Variable Scope (cont' d.)

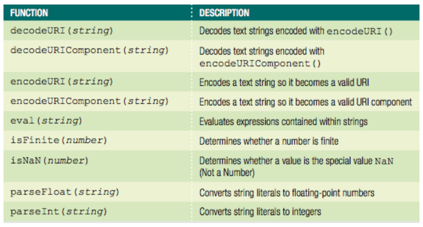
- If variable declared within a function and does not include the `var` keyword
  - Variable automatically becomes a global variable
- Program may contain global and local variables with the same name
  - Local variable takes precedence
  - Value assigned to local variable of the same name
    - Not assigned to global variable of the same name

Understanding Variable Scope (cont' d.)

```
var color = "green";
function duplicateVariableNames() {
   var color = "purple";
   document.write(color);
   // value printed is purple
}
duplicateVariableNames();
document.write(color);
// value printed is green
```

Using Built-in JavaScript Functions

- Called the same way a custom function is called

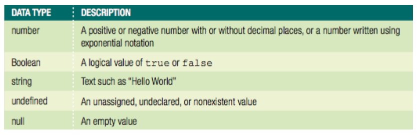


FUNCTION	DESCRIPTION
`decodeURI(string)`	Decodes text strings encoded with `encodeURI()`
`decodeURIComponent(string)`	Decodes text strings encoded with `encodeURIComponent()`
`encodeURI(string)`	Encodes a text string so it becomes a valid URI
`encodeURIComponent(string)`	Encodes a text string so it becomes a valid URI component
`eval(string)`	Evaluates expressions contained within strings
`isFinite(number)`	Determines whether a number is finite
`isNaN(number)`	Determines whether a value is the special value `NaN` (Not a Number)
`parseFloat(string)`	Converts string literals to floating-point numbers
`parseInt(string)`	Converts string literals to integers

Table 2-2 Built-in JavaScript functions

Working with Data Types

- Data type
  - Specific information category a variable contains
- Primitive types
  - Data types assigned a single value



DATA TYPE	DESCRIPTION
number	A positive or negative number with or without decimal places, or a number written using exponential notation
Boolean	A logical value of `true` or `false`
string	Text such as "Hello World"
undefined	An unassigned, undeclared, or nonexistent value
null	An empty value

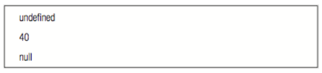
Table 2-3 Primitive JavaScript data types

Working with Data Types (cont'd.)

- The `null` value: data type and a value
  - Can be assigned to a variable
  - Indicates no usable value
  - Use: ensure a variable does not contain any data
- Undefined variable
  - Never had a value assigned to it, has not been declared, or does not exist
  - Indicates variable never assigned a value: not even `null`
  - Use: determine if a value being used by another part of a script

Working with Data Types (cont'd.)

```
var stateTax;
document.write(stateTax);
stateTax = 40;
document.write(stateTax);
stateTax = null;
document.write(stateTax);
```



```
undefined
40
null
```

Figure 2-7 Variable assigned values of `undefined` and `null`

Working with Data Types (cont'd.)

- Strongly typed programming languages
  - Require declaration of the data types of variables
  - Strong typing also known as static typing
    - Data types do not change after declared
- Loosely typed programming languages
  - Do not require declaration of the data types of variables
  - Loose typing also known as dynamic typing
    - Data types can change after declared

Working with Data Types (cont'd.)

- JavaScript interpreter automatically determines data type stored in a variable
- Examples:

```
diffTypes = "Hello World"; // String
diffTypes = 8;            // Integer number
diffTypes = 5.367;        // Floating-point number
diffTypes = true;         // Boolean
diffTypes = null;         // Null
```

Understanding Numeric Data Types

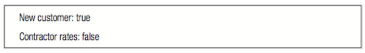
- JavaScript supports two numeric data types
  - Integers and floating-point numbers
- Integer
  - Positive or negative number with no decimal places
- Floating-point number
  - Number containing decimal places or written in exponential notation
  - Exponential notation (scientific notation)
    - Shortened format for writing very large numbers or numbers with many decimal places

Using Boolean Values

- Logical value of true or false
  - Used for decision making
    - Which parts of a program should execute
  - Used for comparing data
- JavaScript programming Boolean values
  - The words `true` and `false`
    - JavaScript converts `true` and `false` values to the integers 1 and 0 when necessary

Using Boolean Values (cont'd.)

```
var newCustomer = true;
var contractorRates = false;
document.write("<p>New customer: " + newCustomer + "</p>");
document.write("<p>Contractor rates: " + contractorRates +↵
"</p>");
```



```
New customer: true
Contractor rates: false
```

Figure 2-9 Boolean values

Working with Strings

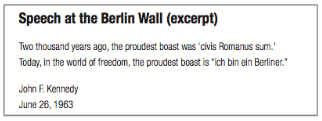
- Text string
  - Contains zero or more characters
    - Surrounded by double or single quotation marks
  - Can be used as literal values or assigned to a variable
- Empty string
  - Zero-length string value
  - Valid for literal strings
    - Not considered to be `null` or `undefined`

Working with Strings (cont'd.)

- To include a quoted string within a literal string surrounded by double quotation marks
  - Surround the quoted string with single quotation marks
- To include a quoted string within a literal string surrounded by single quotation marks
  - Surround the quoted string with double quotation marks
- String must begin and end with the same type of quotation marks

Working with Strings (cont'd.)

```
document.write("<h1>Speech at the Berlin Wall↵
  (excerpt)</h1>");
document.write("<p>Two thousand years ago, the proudest boast↵
  was 'civis Romanus sum.'<br />");
document.write('Today, in the world of freedom, the proudest↵
  boast is "Ich bin ein Berliner."</p>');
var speaker = "<p>John F. Kennedy</br>";
var date = 'June 26, 1963</p>';
document.write(speaker);
document.write(date);
```



Speech at the Berlin Wall (excerpt)

Two thousand years ago, the proudest boast was 'civis Romanus sum.'
Today, in the world of freedom, the proudest boast is "Ich bin ein Berliner."

John F. Kennedy
June 26, 1963

Figure 2-10 String examples in a browser

Working with Strings (cont’d.)

- String operators
  - Concatenation operator (+): combines two strings

```
var destination = "Honolulu";
var location = "Hawaii";
destination = destination + " is in " + location;
```

- Compound assignment operator (+=): combines two strings

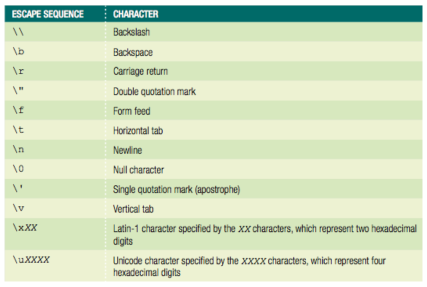
```
var destination = "Honolulu";
destination += " is in Hawaii";
```

- Plus sign
  - Concatenation operator and addition operator

Working with Strings (cont’d.)

- Escape characters and sequences
  - Escape character
    - Tells the compiler or interpreter that the character that follows has a special purpose
    - In JavaScript, escape character is backslash (\)
  - Escape sequence
    - Escape character combined with other characters
    - Most escape sequences carry out special functions

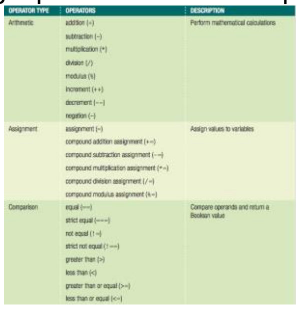
Working with Strings (cont’d.)



ESCAPE SEQUENCE	CHARACTER
`\\`	Backslash
`\b`	Backspace
`\r`	Carriage return
`\"`	Double quotation mark
`\f`	Form feed
`\t`	Horizontal tab
`\n`	Newline
`\0`	Null character
`\'`	Single quotation mark (apostrophe)
`\v`	Vertical tab
`\xXX`	Latin-1 character specified by the *XX* characters, which represent two hexadecimal digits
`\uXXXX`	Unicode character specified by the *XXXX* characters, which represent four hexadecimal digits

Table 2-4 JavaScript escape sequences

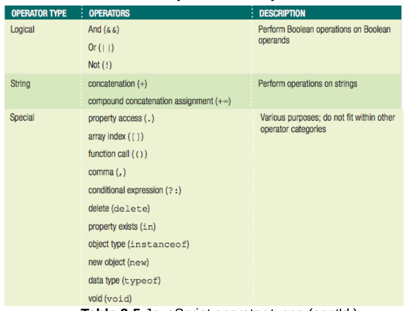
Using Operators to Build Expressions



OPERATOR TYPE	OPERATORS	DESCRIPTION
Arithmetic	addition (+)	Perform mathematical calculations
	subtraction (-)	
	multiplication (*)	
	division (/)	
	modulus (%)	
	increment (++)	
	decrement (--)	
	negation (-)	
Assignment	assignment (=)	Assign values to variables
	compound addition assignment (+=)	
	compound subtraction assignment (-=)	
	compound multiplication assignment (*=)	
	compound division assignment (/=)	
	compound modulus assignment (%=)	
Comparison	equal (==)	Compare operands and return a Boolean value
	strict equal (===)	
	not equal (!=)	
	strict not equal (!==)	
	greater than (>)	
	less than (<)	
	greater than or equal (>=)	
	less than or equal (<=)	

Table 2-5 JavaScript operator types *(continues)*

Using Operators to Build Expressions (cont’d.)



OPERATOR TYPE	OPERATORS	DESCRIPTION
Logical	And (&&)	Perform Boolean operations on Boolean operands
	Or (||)	
	Not (!)	
String	concatenation (+)	Perform operations on strings
	compound concatenation assignment (+=)	
Special	property access (.)	Various purposes; do not fit within other operator categories
	array index ([])	
	function call (())	
	comma (,)	
	conditional expression (?:)	
	delete (`delete`)	
	property exists (`in`)	
	object type (`instanceof`)	
	new object (`new`)	
	data type (`typeof`)	
	void (`void`)	

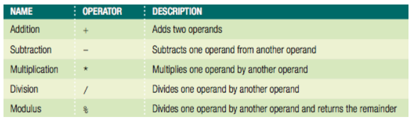
Table 2-5 JavaScript operator types (cont'd.)

Using Operators to Build Expressions (cont’d.)

- Binary operator
  - Requires an operand before and after the operator
- Unary operator
  - Requires a single operand either before or after the operator

Arithmetic Operators

- Perform mathematical calculations
  - Addition, subtraction, multiplication, division
  - Returns the modulus of a calculation
- Arithmetic binary operators



NAME	OPERATOR	DESCRIPTION
Addition	+	Adds two operands
Subtraction	-	Subtracts one operand from another operand
Multiplication	*	Multiplies one operand by another operand
Division	/	Divides one operand by another operand
Modulus	%	Divides one operand by another operand and returns the remainder

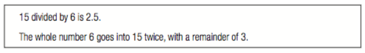
Table 2-6 Arithmetic binary operators

Arithmetic Operators (cont'd.)

- Arithmetic binary operators (cont'd.)
  - Value of operation on right side of the assignment operator assigned to variable on the left side
  - Example: `arithmeticValue = x + y;`
    - Result assigned to the `arithmeticValue` variable
  - Division operator (/)
    - Standard mathematical division operation
  - Modulus operator (%)
    - Returns the remainder resulting from the division of two integers

Arithmetic Operators (cont'd.)

```
var divisionResult = 15 / 6;
var modulusResult = 15 % 6;
document.write("<p>15 divided by 6 is "↵
  + divisionResult + ".</p>"); // prints '2.5'
document.write("<p>The whole number 6 goes into 15 twice,↵
  with a remainder of "+ modulusResult + ".</p>");↵ // prints '3'
```



```
15 divided by 6 is 2.5.
The whole number 6 goes into 15 twice, with a remainder of 3.
```

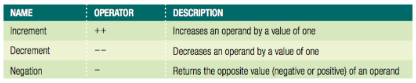
Figure 2-13 Division and modulus expressions

Arithmetic Operators (cont'd.)

- Arithmetic binary operators (cont'd.)
  - Assignment statement
    - Can include combination of variables and literal values on the right side
    - Cannot include a literal value as the left operand
  - JavaScript interpreter
    - Attempts to convert the string values to numbers
    - Does not convert strings to numbers when using the addition operator

Arithmetic Operators (cont'd.)

- Prefix operator
  - Placed before a variable
- Postfix operator
  - Placed after a variable



NAME	OPERATOR	DESCRIPTION
Increment	++	Increases an operand by a value of one
Decrement	--	Decreases an operand by a value of one
Negation	-	Returns the opposite value (negative or positive) of an operand

Table 2-7 Arithmetic unary operators

Arithmetic Operators (cont'd.)

- Arithmetic unary operators
  - Performed on a single variable using unary operators
  - Increment (++) unary operator: used as prefix operators
    - Prefix operator placed before a variable
  - Decrement (--) unary operator: used as postfix operator
    - Postfix operator placed after a variable
  - Example: `++count;` and `count++;`
    - Both increase the count variable by one, but return different values

Arithmetic Operators (cont’d.)

```
var studentID = 100;
var curStudentID;
curStudentID = ++studentID; // assigns '101'
document.write("<p>The first student ID is " +↵
   curStudentID + "</p>");
curStudentID = ++studentID; // assigns '102'
document.write("<p>The second student ID is " +↵
   curStudentID + "</p>");
curStudentID = ++studentID; // assigns '103'
document.write("<p>The third student ID is " +↵
   curStudentID + "</p>");
```

```
The first student ID is 101
The second student ID is 102
The third student ID is 103
```

Figure 2-14 Output of the prefix version of the student ID script

Arithmetic Operators (cont’d.)

```
var studentID = 100;
var curStudentID;
curStudentID = studentID++; // assigns '100'
document.write("<p>The first student ID is " +↵
   curStudentID + "</p>");
curStudentID = studentID++; // assigns '101'
document.write("<p>The second student ID is " +↵
   curStudentID + "</p>");
curStudentID = studentID++; // assigns '102'
document.write("<p>The third student ID is " +↵
   curStudentID + "</p>");
```

```
The first student ID is 100
The second student ID is 101
The third student ID is 102
```

Figure 2-15 Output of the postfix version of the student ID script

Assignment Operators

- Used for assigning a value to a variable
- Equal sign (=)
  - Assigns initial value to a new variable
  - Assigns new value to an existing variable
- Compound assignment operators
  - Perform mathematical calculations on variables and literal values in an expression
    - Then assign a new value to the left operand

Assignment Operators (cont’d.)

NAME	OPERATOR	DESCRIPTION
Assignment	=	Assigns the value of the right operand to the left operand
Compound addition assignment	+=	Combines the value of the right operand with the value of the left operand (if the operands are strings), or adds the value of the right operand to the value of the left operand (if the operands are numbers), and assigns the new value to the left operand
Compound subtraction assignment	−=	Subtracts the value of the right operand from the value of the left operand, and assigns the new value to the left operand
Compound multiplication assignment	*=	Multiplies the value of the right operand by the value of the left operand, and assigns the new value to the left operand
Compound division assignment	/=	Divides the value of the left operand by the value of the right operand, and assigns the new value to the left operand
Compound modulus assignment	%=	Divides the value of the left operand by the value of the right operand, and assigns the remainder (the modulus) to the left operand

Table 2-8 Assignment operators

Assignment Operators (cont’d.)

- += compound addition assignment operator
  - Used to combine two strings and to add numbers
- Examples:

```
var x, y;
// += operator with string values
x = "Hello ";
x += "World"; // x changes to "Hello World"
// += operator with numeric values
x = 100;
y = 200;
x += y; // x changes to 300
// -= operator
x = 10;
y = 7;
x -= y; // x changes to 3
// *= operator
x = 2;
y = 6;
x *= y; // x changes to 12
```

Assignment Operators (cont’d.)

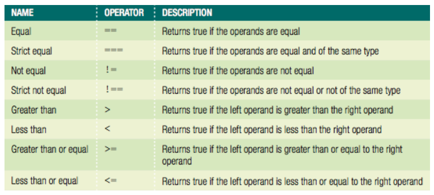
- Examples: (cont’d.)

```
// /= operator
x = 24;
y = 3;
x /= y; // x changes to 8
// %= operator
x = 3;
y = 2;
x %= y; // x changes to 1
// *= operator with a number and a convertible string
x = "100";
y = 5;
x *= y; // x changes to 500
// *= operator with a number and a nonconvertible string
x = "one hundred";
y = 5;
x *= y; // x changes to NaN
```

Comparison and Conditional Operators

- Comparison operators
  - Compare two operands
    - Determine if one numeric value is greater than another
  - Boolean value of true or false returned after compare
- Operands of comparison operators
  - Two numeric values: compared numerically
  - Two nonnumeric values: compared in alphabetical order
  - Number and a string: convert string value to a number
    - If conversion fails: value of false returned

Comparison and Conditional Operators (cont'd.)



NAME	OPERATOR	DESCRIPTION
Equal	`==`	Returns true if the operands are equal
Strict equal	`===`	Returns true if the operands are equal and of the same type
Not equal	`!=`	Returns true if the operands are not equal
Strict not equal	`!==`	Returns true if the operands are not equal or not of the same type
Greater than	`>`	Returns true if the left operand is greater than the right operand
Less than	`<`	Returns true if the left operand is less than the right operand
Greater than or equal	`>=`	Returns true if the left operand is greater than or equal to the right operand
Less than or equal	`<=`	Returns true if the left operand is less than or equal to the right operand

Table 2-9 Comparison operators

Comparison and Conditional Operators (cont'd.)

- Conditional operator
  - Executes one of two expressions based on conditional expression results
  - Syntax
    conditional expression ? *expression1* : *expression2*;
  - If conditional expression evaluates to true:
    - Then `expression1` executes
  - If the conditional expression evaluates to false:
    - Then `expression2` executes

Comparison and Conditional Operators (cont'd.)

- Example of conditional operator:

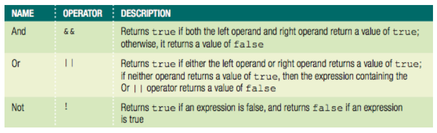
```
var intVariable = 150;
var result;
intVariable > 100 ?↵
   result = "intVariable is greater than 100" :↵
   result = "intVariable is less than or equal to 100";
document.write(result);
```

Falsy and Truthy Values

- Six falsy values treated like Boolean `false`:
  - ""
  - -0
  - 0
  - `NaN`
  - `null`
  - `undefined`
- All other values are truthy, treated like Boolean `true`

Logical Operators

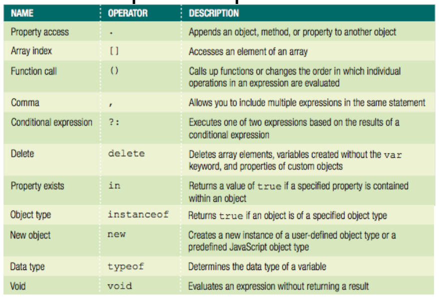
- Compare two Boolean operands for equality



NAME	OPERATOR	DESCRIPTION
And	`&&`	Returns `true` if both the left operand and right operand return a value of `true`; otherwise, it returns a value of `false`
Or	`||`	Returns `true` if either the left operand or right operand returns a value of `true`; if neither operand returns a value of `true`, then the expression containing the Or `||` operator returns a value of `false`
Not	`!`	Returns `true` if an expression is false, and returns `false` if an expression is true

Table 2-10 Logical operators

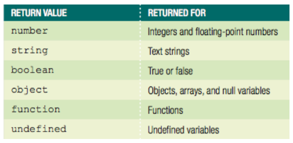
Special Operators



NAME	OPERATOR	DESCRIPTION
Property access	`.`	Appends an object, method, or property to another object
Array index	`[]`	Accesses an element of an array
Function call	`()`	Calls up functions or changes the order in which individual operations in an expression are evaluated
Comma	`,`	Allows you to include multiple expressions in the same statement
Conditional expression	`?:`	Executes one of two expressions based on the results of a conditional expression
Delete	`delete`	Deletes array elements, variables created without the `var` keyword, and properties of custom objects
Property exists	`in`	Returns a value of `true` if a specified property is contained within an object
Object type	`instanceof`	Returns `true` if an object is of a specified object type
New object	`new`	Creates a new instance of a user-defined object type or a predefined JavaScript object type
Data type	`typeof`	Determines the data type of a variable
Void	`void`	Evaluates an expression without returning a result

Table 2-11 Special operators

Special Operators (cont’d.)



RETURN VALUE	RETURNED FOR
`number`	Integers and floating-point numbers
`string`	Text strings
`boolean`	True or false
`object`	Objects, arrays, and null variables
`function`	Functions
`undefined`	Undefined variables

Table 2-12 Values returned by `typeof` operator

Understanding Operator Precedence

- Operator precedence
  - Order in which operations in an expression evaluate
- Associativity
  - Order in which operators of equal precedence execute
  - Left to right associativity
  - Right to left associativity

Understanding Operator Precedence (cont’d.)

- Evaluating associativity
  - Example: multiplication and division operators
    - Associativity of left to right

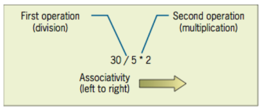

First operation (division)
Second operation (multiplication)
30 / 5 * 2
Associativity (left to right)

Figure 2-16 Conceptual illustration of left to right associativity

Understanding Operator Precedence (cont’d.)

- Evaluating associativity (cont’d.)
  - Example: Assignment operator and compound assignment operators
    - Associativity of right to left
    - `x = y *= ++x`

```
var x = 3;
var y = 2;
x = y *= ++x;
```

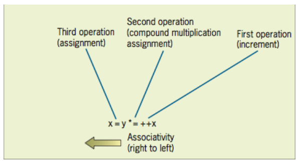
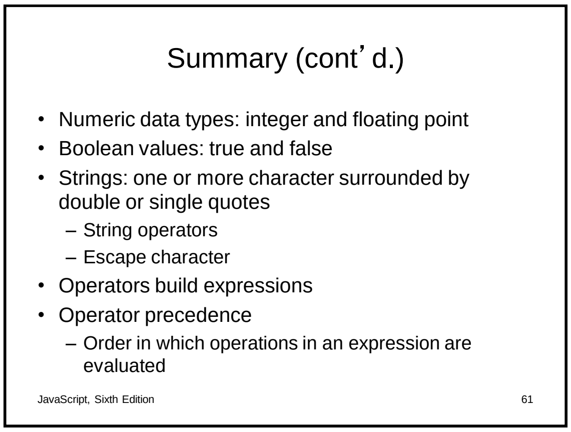

Third operation (assignment)
Second operation (compound multiplication assignment)
First operation (increment)
x = y *= ++x
Associativity (right to left)

Figure 2-17 Conceptual illustration of right-to-left associativity

Summary

- Functions
  - Similar to methods associated with an object
  - Pass parameters
  - To execute, must be called
- Variable scope
  - Where a declared variable can be used
  - Global and local variables
- Data type
  - Specific category of information a variable contains
  - Static typing and dynamic typing



Summary (cont’d.)

- Numeric data types: integer and floating point
- Boolean values: true and false
- Strings: one or more character surrounded by double or single quotes
  - String operators
  - Escape character
- Operators build expressions
- Operator precedence
  - Order in which operations in an expression are evaluated