JavaScript

Chapter 2
Working with Data Types and Operators

Lecturer: Thiều Quang Trung

Objectives

- Work with variables
- Study data types
- Use expressions and operators
- Work with strings
- Study operator precedence

Using Variables

- **Variable**
  - Specific location in computer’s memory
- Before using a variable:
  - Write a statement that creates the variable and assigns it a name

Naming Variable Names

- **Identifier**
  - Name you assign to a variable
- Rules and conventions when naming a variable
  - Identifiers must begin with an uppercase or lowercase ASCII letter, dollar sign ($), or underscore (_)
  - Can use numbers in an identifier, but not as first character
  - Cannot include spaces in an identifier
  - Cannot use reserved words for identifiers

Naming Variable Names (continued)

- **Reserved words** (or **keywords**)
  - Special words that are part of JavaScript syntax
- Variable names are case sensitive
  - `myVariable`, `myvariable`, `MyVariable`, and `MYVARIABLE` are all different variables

Declaring and Initializing Variables

- Use the reserved keyword `var` to create variables
  - To create a variable named `myVariable`:

```
var myVariable;
```

- **Declaring** a variable
  - Using a statement to create a variable
- **Initializing** a variable
  - Assigning a specific value to it
  - Can be done when you declare the variable

```
var variable_name = value;
```

Declaring and Initializing Variables (continued)

- **Assignment operator**
  - Equal sign (=)
  - Assigns the value on the right side of expression to the variable on the left side of expression
- Value assigned to a variable can be a literal string or a numeric value
  - Literal string must be enclosed in quotation marks
    - `var myName = "Don“;`
  - Numeric value is not enclosed in quotation marks
    - `var retirementAge = 59;`

Declaring and Initializing Variables (continued)

- Can declare multiple variables using a single `var` keyword

```
var customerName = "Don Gosselin",
     orderQuantity = 100, salesTax
= .05;
```

- Can assign value of one variable to another

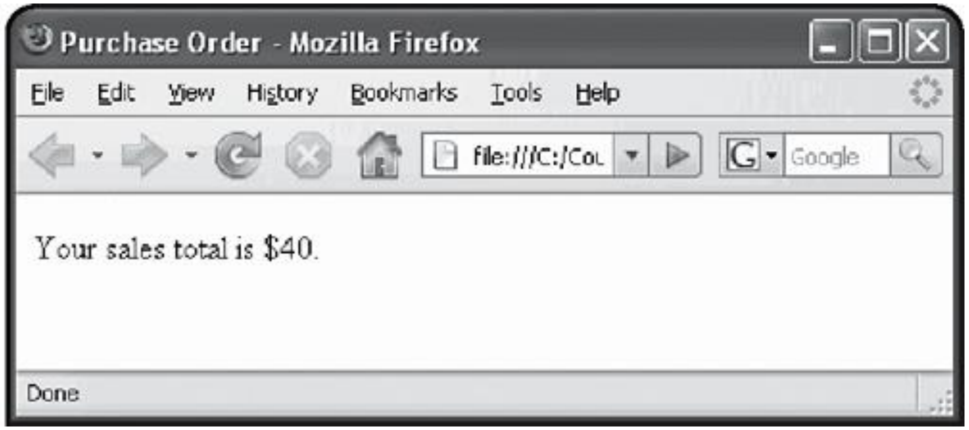
```
var salesTotal;
var curOrder = 40;
salesTotal = curOrder;
```

Displaying Variables

- To print a variable, pass variable name to `document.write()` or `document.writeln()` method
- Example

```
document.write("<p>Your sales total is $“ +
salesTotal + ".</p>");
```

Displaying Variables (continued)


Purchase Order - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Cou
Google
Done

```
Your sales total is $40.
```

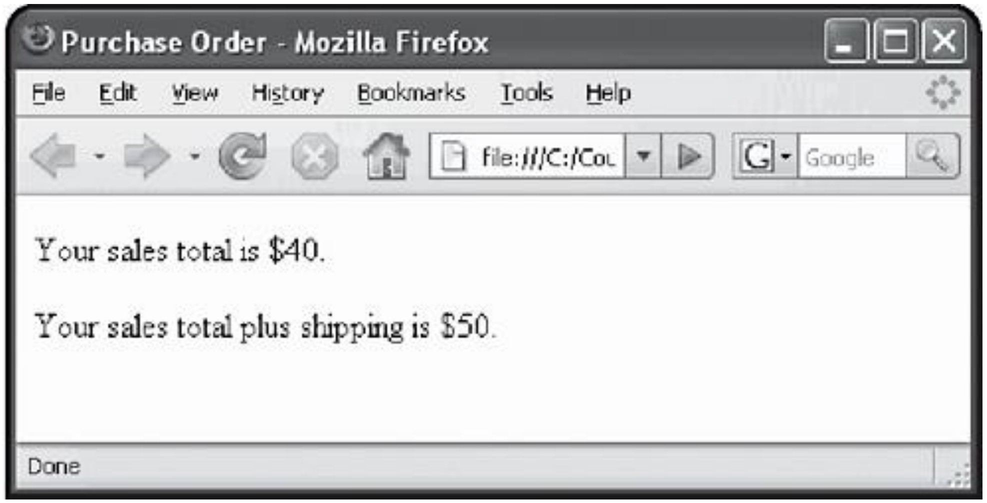
Figure 2-1: Results of script that assigns the value of one variable to another

Modifying Variables

- To change a variable’s value, use a statement with variable’s name, equal sign, and new value

```
var salesTotal = 40;
document.write("<p>Your sales total is $" +
  salesTotal + ".</p>");
var shipping = 10;
salesTotal = salesTotal + shipping;
document.write("<p>Your sales total plus
  shipping is $" + salesTotal + ".</p>");
```

Modifying Variables (continued)


Purchase Order - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Cou
Google
Done

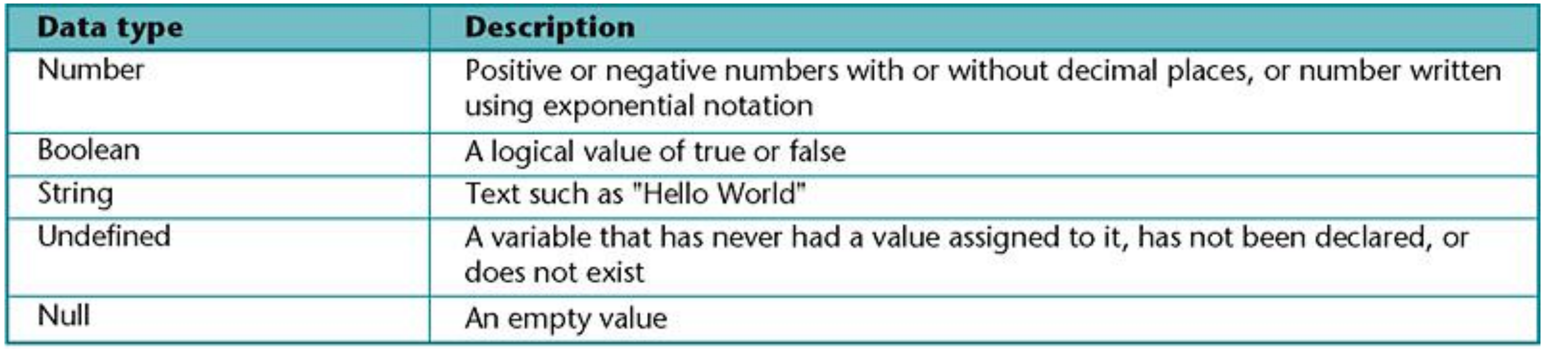
```
Your sales total is $40.

Your sales total plus shipping is $50.
```

Figure 2-3: Results of script that includes a changing variable

Working with Data Types

- **Data type**
  - Category of information that a variable contains
- **Primitive types**
  - Data types that can be assigned only a single value



Data type	Description
Number	Positive or negative numbers with or without decimal places, or number written using exponential notation
Boolean	A logical value of true or false
String	Text such as "Hello World"
Undefined	A variable that has never had a value assigned to it, has not been declared, or does not exist
Null	An empty value

Table 2-2: Primitive JavaScript data types

Working with Data Types (continued)

- **Reference**, or **composite**, data types
  - Can contain multiple values or complex types of information
  - Functions, objects, arrays
- **Strongly typed programming languages**
  - Must declare data types of variables
- **Loosely typed programming languages**
  - Not required to declare data types of variables
- JavaScript is a loosely typed language

Numeric Data Types

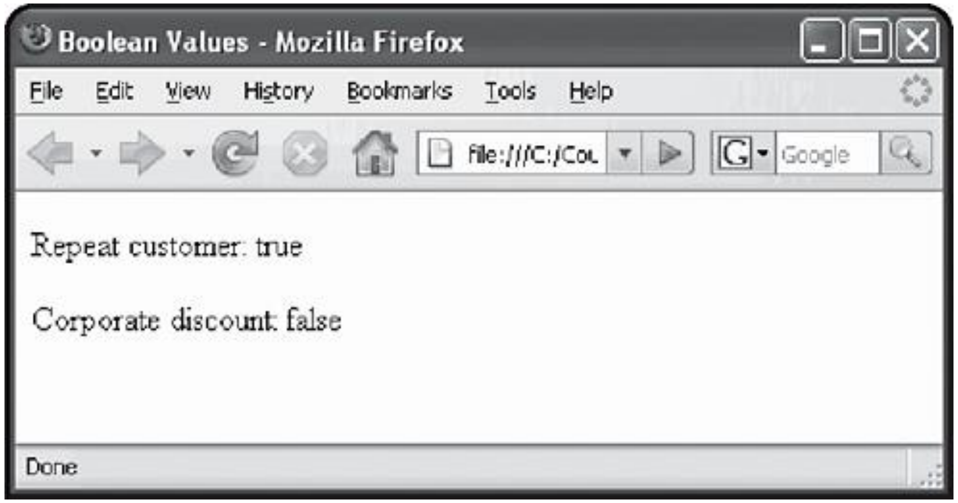
- JavaScript supports two numeric data types
  - Integers and floating-point numbers
- **Integer**
  - Positive or negative number with no decimal places
- **Floating-point number**
  - Decimal places (or written in exponential notation)
  - **Exponential notation**, or **scientific notation**
    -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

Boolean Values

- **Boolean value**
  - Logical value of true or false
  - In JavaScript, words true and false indicate Boolean values
- Example

```
var repeatCustomer = true;
var corporateDiscount = false;
document.write("<p>Repeat customer: " +
  repeatCustomer + "</p>");
document.write("<p>Corporate discount: " +
  corporateDiscount + "</p>");
```

Boolean Values (continued)


Boolean Values - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Cou
Google
Done

```
Repeat customer: true

Corporate discount: false
```

Figure 2-7: Boolean values

Arrays

- Array
  - Set of data represented by a single variable name

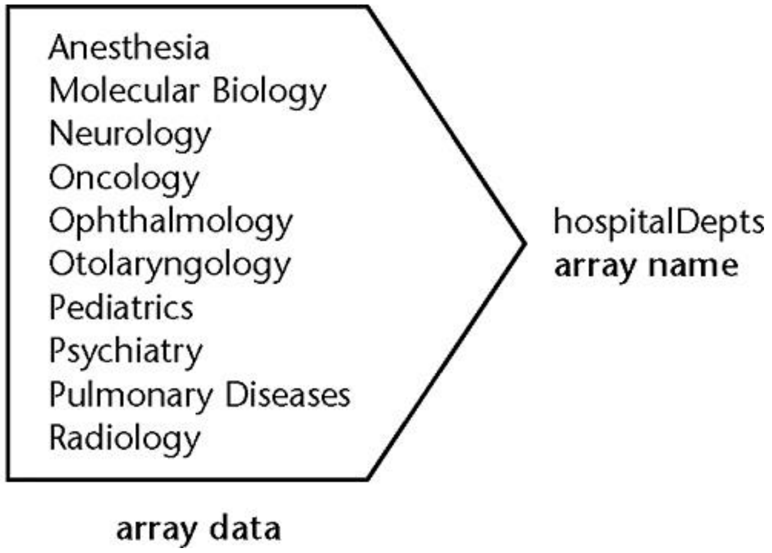


Figure 2-8: Conceptual example of an array

Declaring and Initializing Arrays

- **Element:** each piece of data in an array
- Create an array named `hospitalDepts[]` that has 10 elements

```
var hospitalDepts = new Array(10);
```

- Assign value to first element in: `hospitalDepts[]`

```
hospitalDepts[0] = "Anesthesia";
```

- Can assign value to elements when array is created

```
hospitalDepts = new Array("Anesthesia",
  "Molecular Biology", "Neurology");
```

Accessing Element Information

- To access an element’s value, include brackets and element index
- Examples

```
document.writeln(hospitalDepts[0]);
  // prints "Anesthesia"
document.writeln(hospitalDepts[1]);
  // prints "Molecular Biology"
document.writeln(hospitalDepts[2]);
  // prints "Neurology"
```

Modifying Elements

- To modify values in existing array elements, include brackets and element index
- Examples

```
hospitalDepts[0] = "Anesthesia";
  // first element
hospitalDepts[1] = "Molecular Biology";
  // second element
hospitalDepts[2] = "Neurology";
  // third element
```

Determining the Number of Elements in an Array

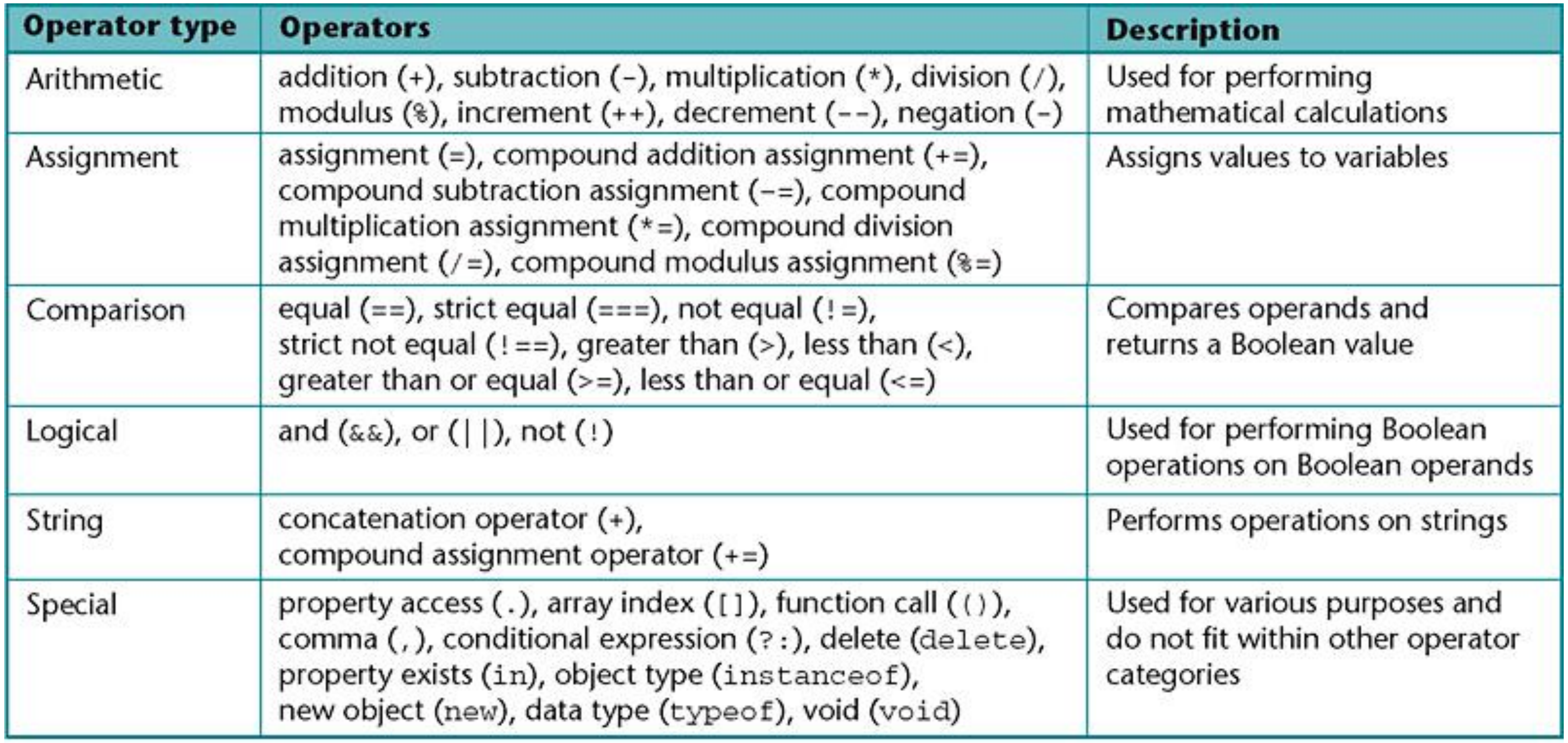
- **`length` property** of Array class returns the number of elements in an array
- Syntax

```
    array_name.length;
```

Building Expressions

- **Expression**
  - Literal value or variable or a combination of literal values, variables, operators, and other expressions
  - Can be evaluated by the JavaScript interpreter to produce a result
- **Operands**
  - Variables and literals contained in an expression
  - **Literal:** value such as a literal string or a number
- **Operators**
  - Symbols used in expressions to manipulate operands

Building Expressions (continued)



Operator type	Operators	Description
Arithmetic	addition (+), subtraction (-), multiplication (*), division (/), modulus (%), increment (++), decrement (--), negation (-)	Used for performing mathematical calculations
Assignment	assignment (=), compound addition assignment (+=), compound subtraction assignment (-=), compound multiplication assignment (*=), compound division assignment (/=), compound modulus assignment (%=)	Assigns values to variables
Comparison	equal (==), strict equal (===), not equal (!=), strict not equal (!==), greater than (>), less than (<), greater than or equal (>=), less than or equal (<=)	Compares operands and returns a Boolean value
Logical	and (&&), or (||), not (!)	Used for performing Boolean operations on Boolean operands
String	concatenation operator (+), compound assignment operator (+=)	Performs operations on strings
Special	property access (.), array index ([]), function call (()), comma (,), conditional expression (?:), delete (`delete`), property exists (`in`), object type (`instanceof`), new object (`new`), data type (`typeof`), void (`void`)	Used for various purposes and do not fit within other operator categories

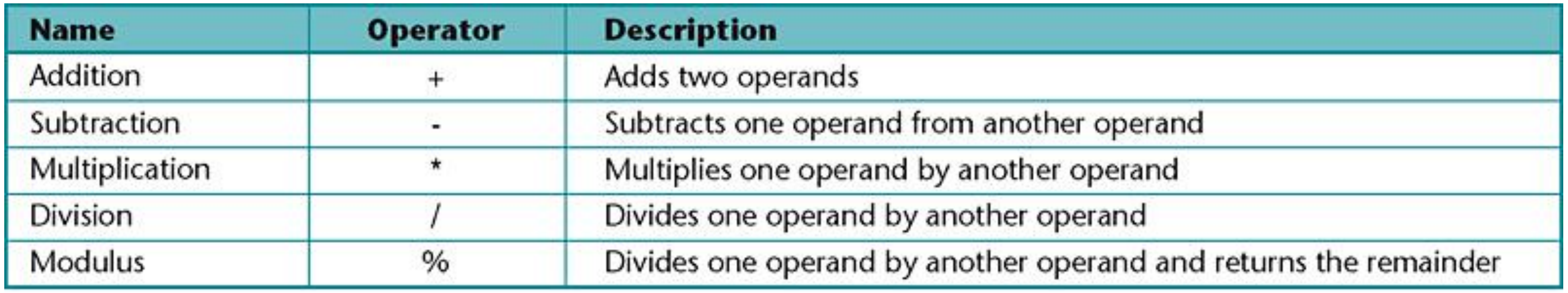
Table 2-3: JavaScript operator types

Building Expressions (continued)

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

Arithmetic Operators

- Used to perform mathematical calculations
  - Addition, subtraction, multiplication, division, etc.

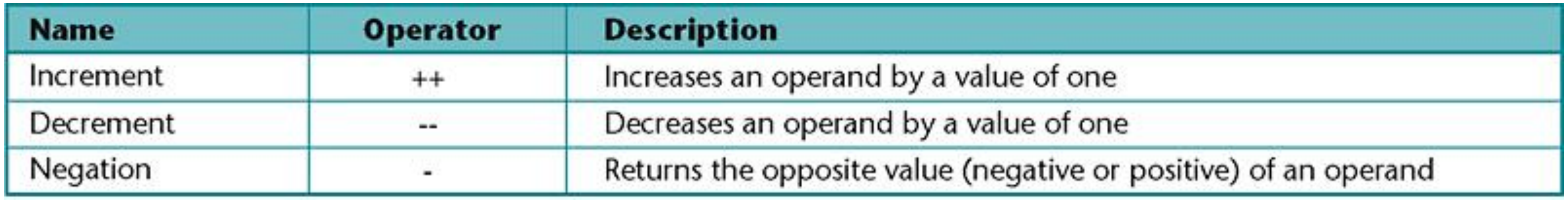


Name	Operator	Description
Addition	+	Adds two operands
Subtraction	-	Subtracts one operand from another operand
Multiplication	*	Multiplies one operand by another operand
Division	/	Divides one operand by another operand
Modulus	%	Divides one operand by another operand and returns the remainder

Table 2-4: Arithmetic binary operators

Arithmetic Unary Operators

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



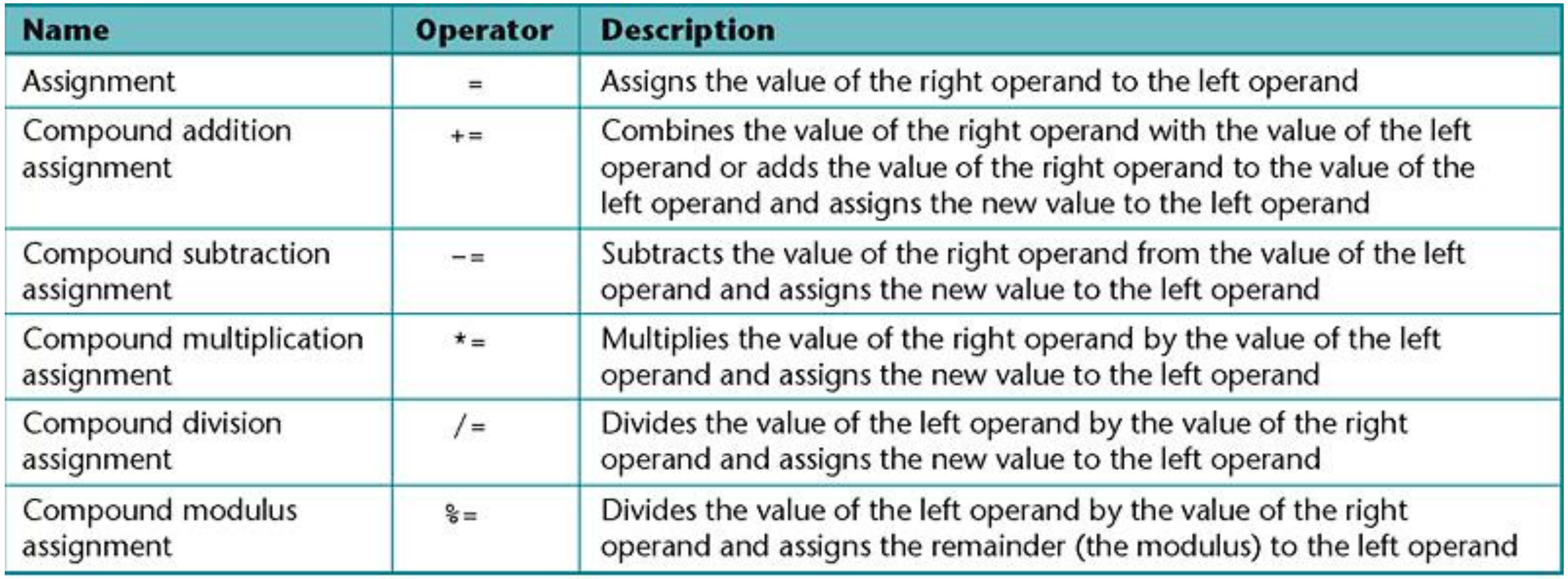
Name	Operator	Description
Increment	++	Increases an operand by a value of one
Decrement	--	Decreases an operand by a value of one
Negation	-	Returns the opposite value (negative or positive) of an operand

Table 2-5: Arithmetic unary operators

Assignment Operators

- Used for assigning a value to a variable
- Equal sign (=)
- **Compound assignment operators**
  - Perform mathematical calculations on variables and literal values, and then assign a new value to the left operand

Assignment Operators (continued)



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 or adds the value of the right operand to the value of the left operand 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

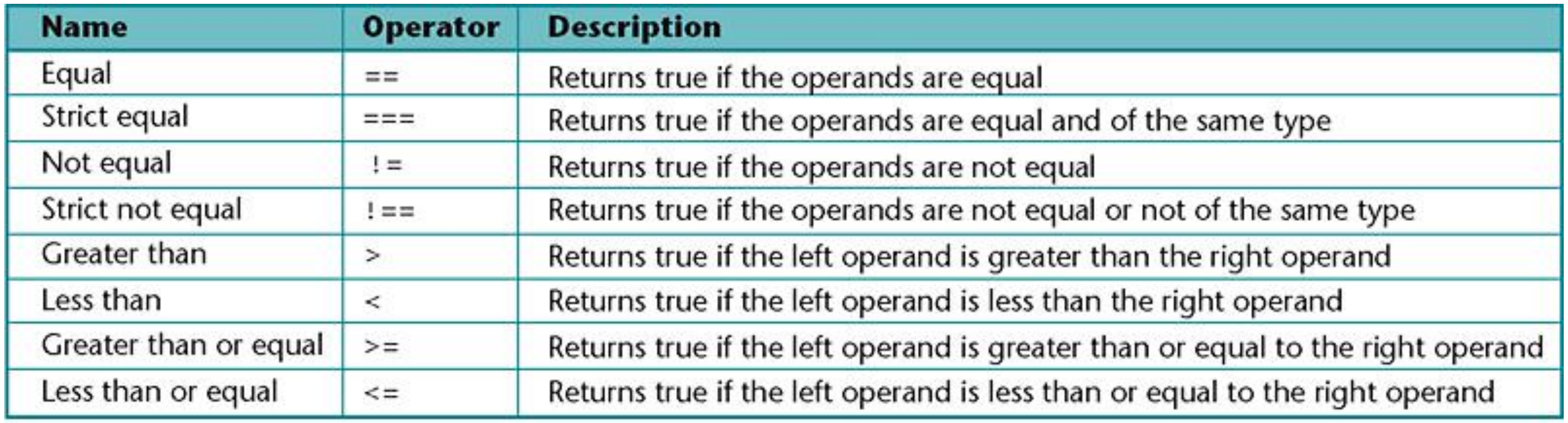
Table 2-6: Assignment operators

Comparison and Conditional Operators

- **Comparison operators**
  - Compare two operands and determine if one numeric value is greater than another
  - Boolean value of true or false is returned
- **Conditional operator**
  - Executes one of two expressions, based on the results of a conditional expression
  - Syntax

```
conditional expression ? expression1:
expression2;
```

Comparison and Conditional Operators (continued)

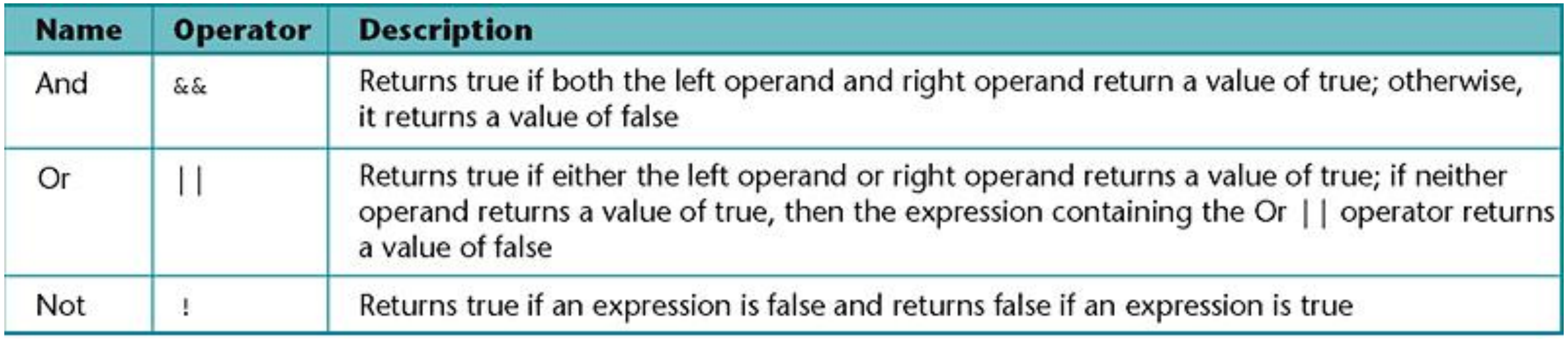


Name	Operator	Description
Equal	==	Returns true if the operands are equal
Strict equal	===	Returns true if the operands are equal and of the same type
Not equal	!=	Returns true if the operands are not equal
Strict not equal	!==	Returns true if the operands are not equal or not of the same type
Greater 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 2-7: Comparison operators

Logical Operators

- **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 2-8: Logical operators

Working with Strings

- Text string is text contained within double or single quotation marks
- Can use text strings as literal values or assign them to a variable
- **Empty string**
  - Zero-length string value
  - Valid value for literal strings

String Operators

- Operators used to combine two strings
  - Concatenation operator (+)

```
var destination = "Jakarta";
var location = "Indonesia";
destination = destination + " is in "
 + location;
```

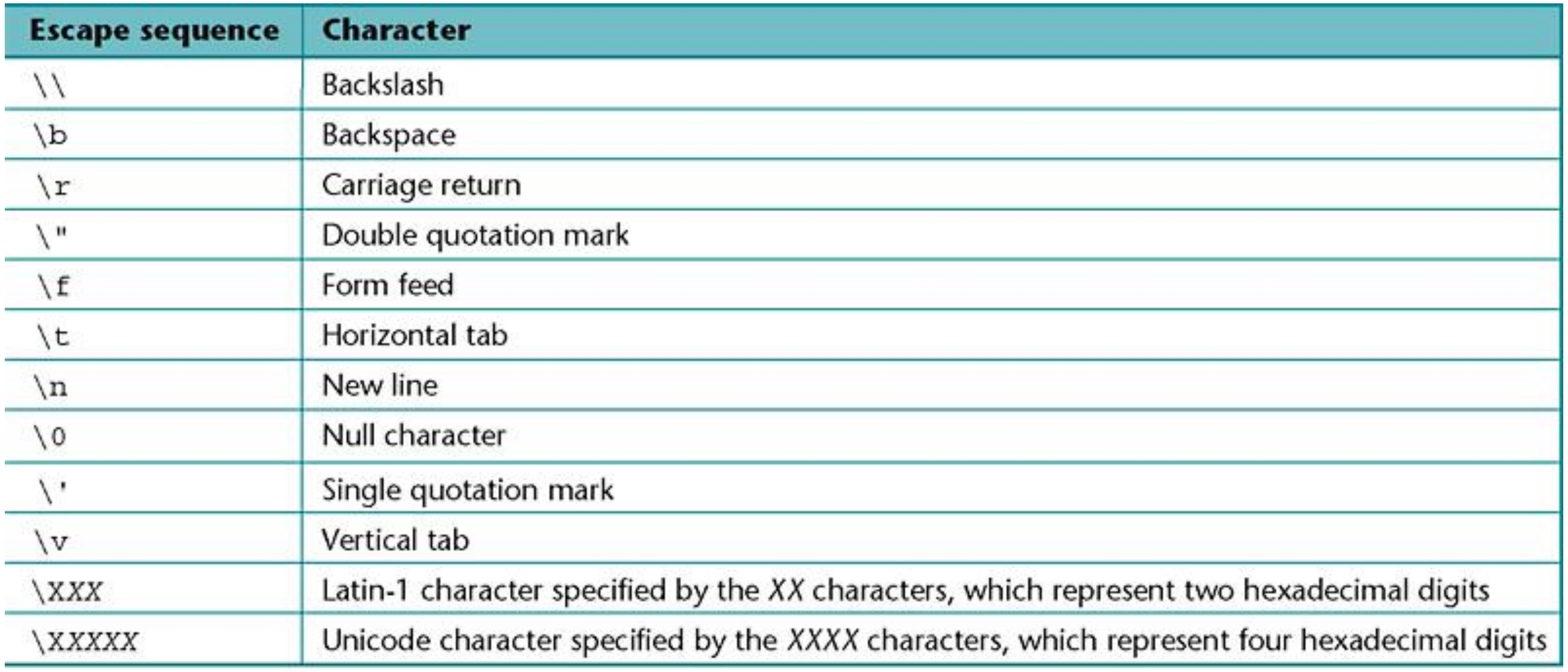
  - Compound assignment operator (+=)

```
var destination = "Jakarta";
destination += " is in Indonesia";
```

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

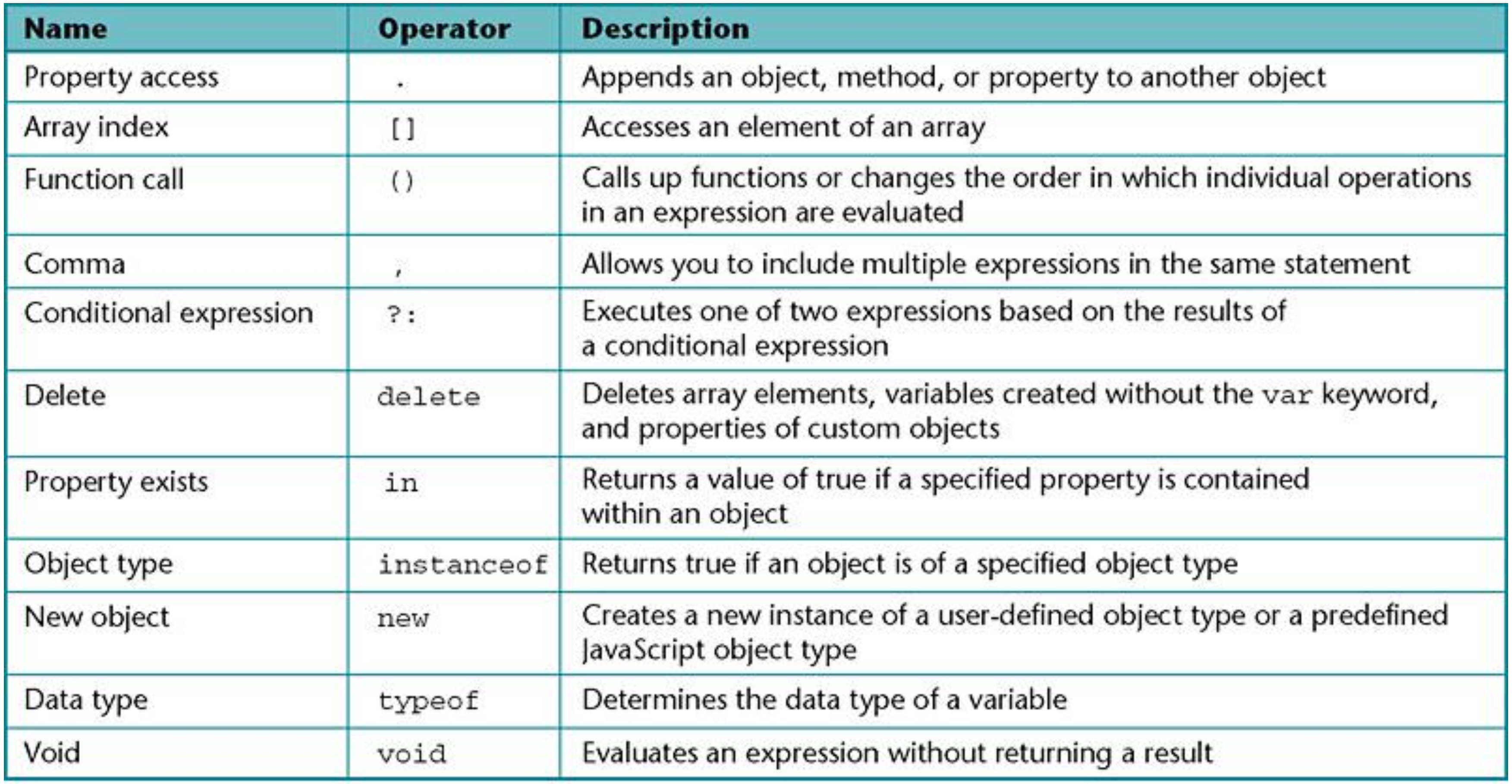
Escape Characters and Sequences (continued)



Escape sequence	Character
`\\`	Backslash
`\b`	Backspace
`\r`	Carriage return
`\"`	Double quotation mark
`\f`	Form feed
`\t`	Horizontal tab
`\n`	New line
`\0`	Null character
`\'`	Single quotation mark
`\v`	Vertical tab
`\XXX`	Latin-1 character specified by the *XX* characters, which represent two hexadecimal digits
`\XXXXX`	Unicode character specified by the *XXXX* characters, which represent four hexadecimal digits

Table 2-9: JavaScript escape sequences

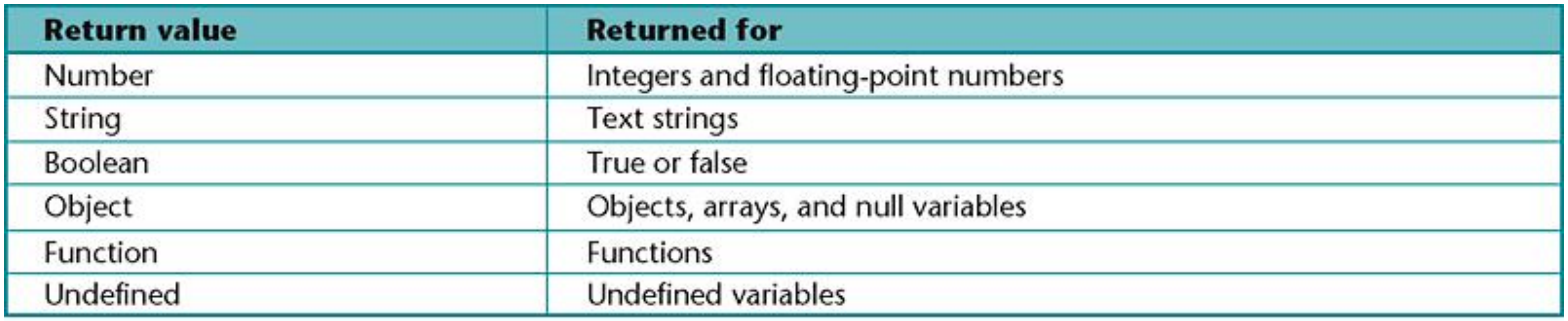
Special Operators



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 2-10: Special operators

Special Operators (continued)



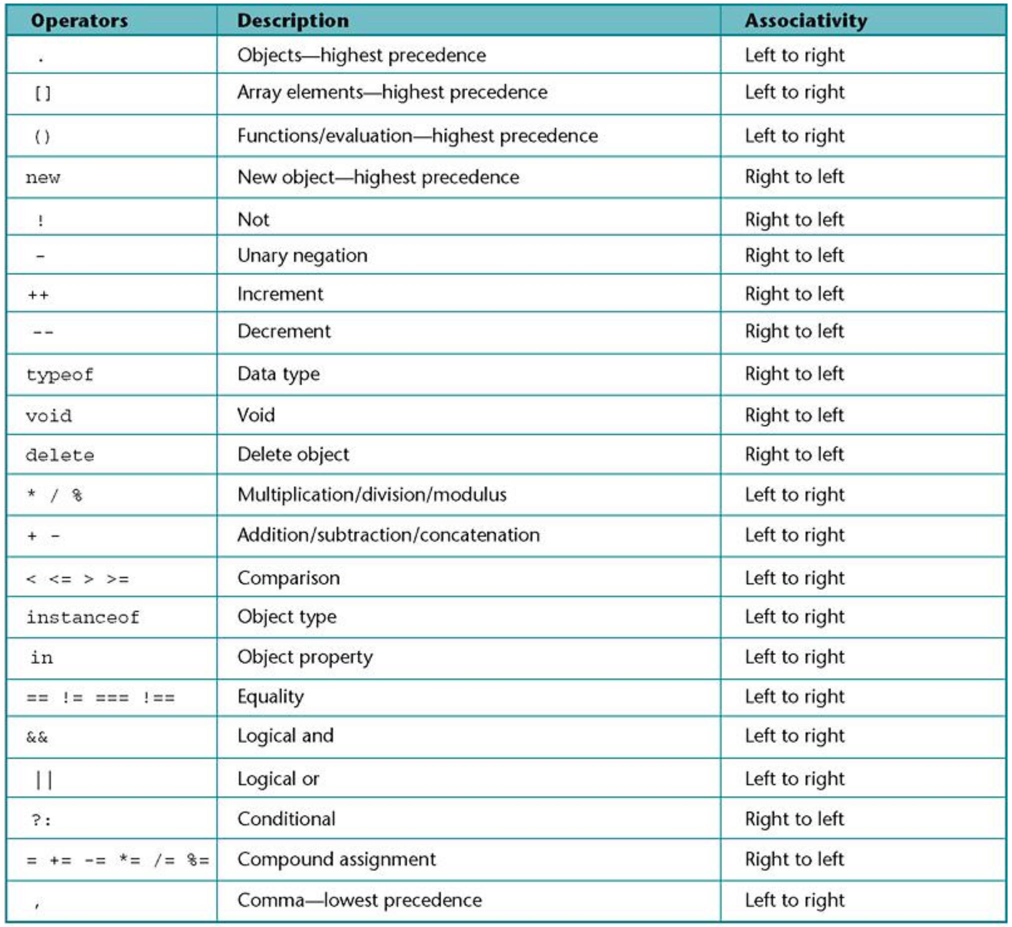
Return value	Returned for
Number	Integers and floating-point numbers
String	Text strings
Boolean	True or false
Object	Objects, arrays, and null variables
Function	Functions
Undefined	Undefined variables

Table 2-11: Values returned by `typeof` operator

Understanding Operator Precedence

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

Understanding Operator Precedence (continued)



Operators	Description	Associativity
`.`	Objects—highest precedence	Left to right
`[]`	Array elements—highest precedence	Left to right
`()`	Functions/evaluation—highest precedence	Left to right
`new`	New object—highest precedence	Right to left
`!`	Not	Right to left
`-`	Unary negation	Right to left
`++`	Increment	Right to left
`--`	Decrement	Right to left
`typeof`	Data type	Right to left
`void`	Void	Right to left
`delete`	Delete object	Right to left
`* / %`	Multiplication/division/modulus	Left to right
`+ -`	Addition/subtraction/concatenation	Left to right
`< <= > >=`	Comparison	Left to right
`instanceof`	Object type	Left to right
`in`	Object property	Left to right
`== != === !==`	Equality	Left to right
`&&`	Logical and	Left to right
`||`	Logical or	Left to right
`?:`	Conditional	Right to left
`= += -= *= /= %=`	Compound assignment	Right to left
`,`	Comma—lowest precedence	Left to right

Table 2-12: Operator precedence

Understanding Operator Precedence (continued)

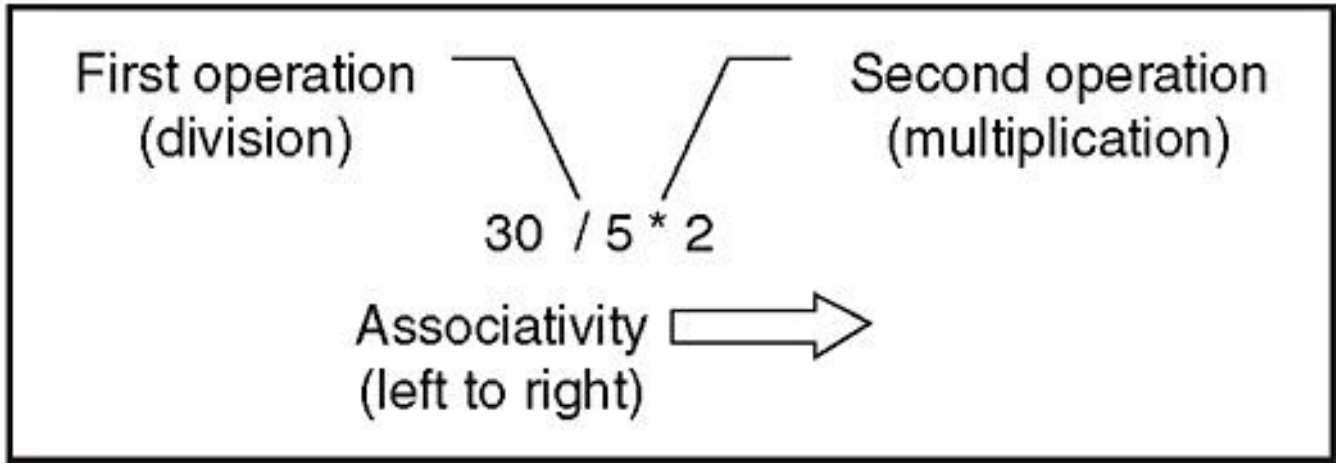


Figure 2-20: Conceptual illustration of left to right associativity

Understanding Operator Precedence (continued)

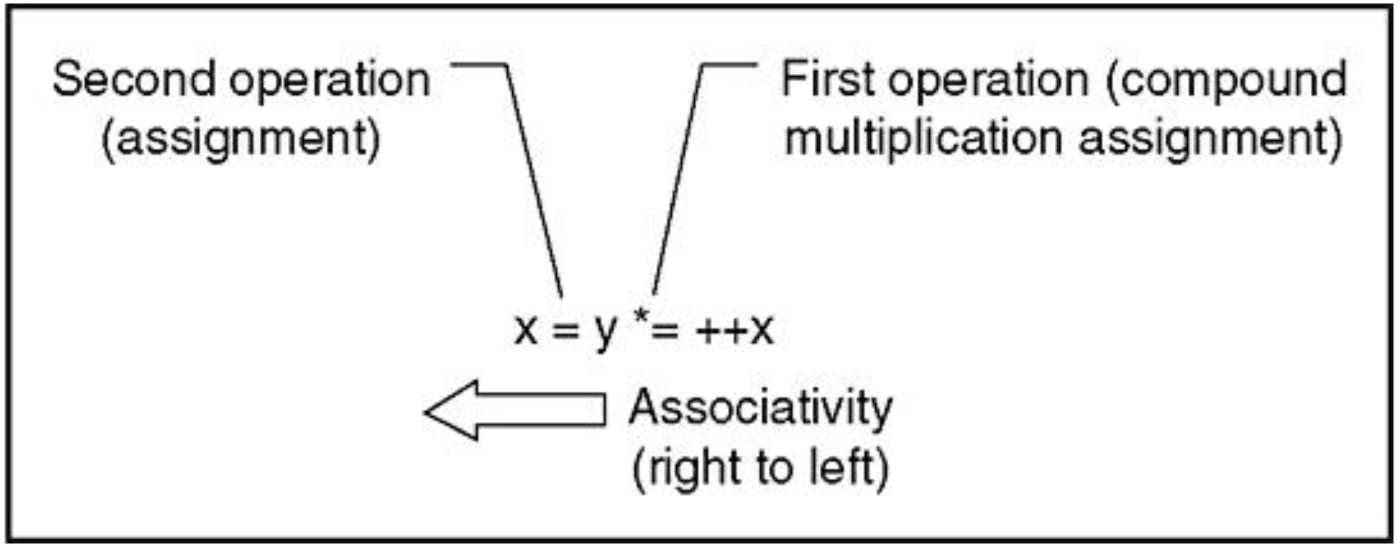

Second operation (assignment)
First operation (compound multiplication assignment)
x = y *= ++x
Associativity (right to left)

Figure 2-21: Conceptual illustration of right to left associativity

Summary

- Values a program stores in computer memory are called variables
- Name assigned to a variable is called an identifier
- Reserved words (or keywords) are special words that are part of the JavaScript language syntax
- Data type is the specific category of information that a variable contains
- Array contains a set of data represented by a single variable name

Summary (continued)

- Expression is a single literal value or variable or a combination of literal values, variables, operators, and other expressions that can be evaluated by JavaScript interpreter to produce a result
- Operands are variables and literals contained in an expression
- Operators are symbols used in expressions to manipulate operands
- Operator precedence is order in which operations in an expression are evaluated