CHAPTER 7

JavaScript Operators Reference

In JavaScript, *operators* perform operations on expressions. The expressions that the operators operate on are called *operands*. JavaScript supports unary operators (operators that work on one expression, like the increment operator), binary operators (operators that require two expressions, like most mathematical operators), and one ternary operator (which requires three expressions).

Operators in JavaScript can be grouped into seven broad categories:

- **Assignment**: Assign values to variables
- **Comparison**: Compare values
- **Arithmetic**: Perform basic arithmetic—addition, subtraction, modulus, etc.
- **Bitwise**: Modify operands based on their binary representations
- **Logical**: Logical constructions like AND and OR
- **String**: Modifies strings (this category includes only one operator)
- **Miscellaneous**: Catch-all group for remaining operators that don’t fit in the other categories

We’ll cover each of these categories in order in this reference.

Assignment Operators

JavaScript assignment operators are used to assign values to their left operand based on the value of their right operand, the simplest example of which is the basic assignment operator `=`:

```javascript
x = 1;
strLocation = "California";
boolSuccess = false;
```

JavaScript has several other assignment operators that are shorthand for other operations. These shorthand operators, listed in Table 7-1, provide a way to write more concise code.
### Examples

```javascript
var num1 = 5,
    num2 = 9;
num1 += num2; // num1 is now 13
num1 -= num2; // num1 (which was 13) is now 5 again
num1 *= num2; // num1 is now 45
num1 /= num2; // num1 is now 5 again
```

Note that if either of the operands is a string, the shorthand operator `+=` will perform a string concatenation rather than an arithmetical addition. If the other operand is not a string, it will be cast as one first. See “String Operator,” later in the chapter, for more information.

JavaScript also supports several shorthand binary operators, listed in Table 7-2. Binary operators operate on their operands by modifying the bits that compose them. For a full discussion of binary operators, see the “Bitwise Operators” section later in the chapter.

### Table 7-2. Shorthand Bitwise Operators

<table>
<thead>
<tr>
<th>Shorthand Operator</th>
<th>Equivalent Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>operand1 %= operand2</td>
<td>operand1 = operand1 % operand2</td>
</tr>
<tr>
<td>operand1 ^= operand2</td>
<td>operand1 = operand1 ^ operand2</td>
</tr>
<tr>
<td>operand1 &lt;&lt;= operand2</td>
<td>operand1 = operand1 &lt;&lt; operand2</td>
</tr>
<tr>
<td>operand1 &gt;&gt;= operand2</td>
<td>operand1 = operand1 &gt;&gt; operand2</td>
</tr>
<tr>
<td>operand1 &gt;&gt;&gt; operand2</td>
<td>operand1 = operand1 &gt;&gt;&gt; operand2</td>
</tr>
<tr>
<td>operand1</td>
<td>= operand2</td>
</tr>
</tbody>
</table>

**Example**

```javascript
var num1 = 100,
    num2 = 50;
num1 %= num2; // num1 is now 0
```

### Comparison Operators

Comparison operators are used to compare two operands. Since JavaScript is weakly typed, it has two different kinds of comparisons: strict and coerced.

In strict comparisons, the operator compares both the value of the operands and their type. If either doesn’t match, the comparison returns `false`. For example, the strict comparison `true === "true"` evaluates to `false` because `true` is a boolean, which is a different type than "true", which is a string.