Conditional Logic and Control Flow

David Tucker

CTO Consultant

@_davidtucker_ | davidtucker.net

Comparison Operators

index.js

```
// Example 1
let example1 = (10 > 4);
console.log(example1); // Output: true

// Example 2
let example2 = (5 >= 5);
console.log(example2); // Output: true
```

Equality Operators

index.js

```
// Example 3
let example3 = ("Hello" === "Hi");
console.log(example3); // Output: false

// Example 4
let example4 = (4 !== 7);
console.log(example4); // Output: true
```

Using Conditionals

index.js

```
// Using Conditionals
let badgeColor;

if (numYearsService < 5) {
   badgeColor = "blue";
}
   badgeColor = "yellow";
}
   badgeColor = "red";
}
   badgeColor = "purple";
}
   badgeColor = "silver";
}
```

Mathematical Operators

Assignment Operators

Increment and Decrement Operators

If used prefix, with operator before operand (for example, ++x), the increment operator increments and returns the value after incrementing.

If used postfix, with operator after operand (for example, x++), the increment operator increments and returns the value before incrementing.

Comparison Operators

Standard and Strict Equality

In JavaScript, there are two different sets of equality operators. The == and != operators are the standard equality operators. The === and !== are the strict equality operators. The latter requires that both the value and the type match.

Always use strict equality operators unless you fully understand the different behavior of the standard equality operators.

Strict equality (===)

The **strict equality** (`===`) operator checks whether its two operands are equal, returning a Boolean result. Unlike the equality operator, the strict equality operator always considers operands of different types to be different.

Try it

JavaScript Demo: Expressions - Strict equality operator

```
1  console.log(1 === 1);
2  // Expected output: true
3
4  console.log('hello' === 'hello');
5  // Expected output: true
6
7  console.log('1' ===  1);
8  // Expected output: false
9
10 console.log(0 === false);
```

Falsy Values

Falsy Values

A falsy (sometimes written falsey) value is a value that is considered false when encountered in a Boolean context. JavaScript uses type conversion to coerce any value to a Boolean in contexts that require it, such as conditionals and loops.

Falsy

A **falsy** (sometimes written **falsey**) value is a value that is considered false when encountered in a Boolean context.

JavaScript uses type conversion to coerce any value to a Boolean in contexts that require it, such as conditionals and loops.

The following table provides a complete list of JavaScript falsy values:

Value	Description
`false`	The keyword `false`.
`0`	The `Number` zero (so, also `0.0`, etc., and `0x0`).
`-0`	The `Number` negative zero (so, also `-0.0`, etc., and `-0x0`).

Conditional Logic

Switch Statement

Switch Statement

The switch statement evaluates an expression, matching the expression's value against a series of case clauses, and executes statements after the first case clause with a matching value, until a break statement is encountered. The default clause of a switch statement will be jumped to if no case matches the expression's value.

Working with the Exercise Files

Demo

Reviewing the organization of the exercise files

Installing a third-party module with npm

Configuring VS Code to debug in the integrated terminal

Implementing User Input Validation

Demo

Creating a command line application to add an employee to the directory

Utilizing conditional logic to validate user input against specific rules