Day 2: Conditional Statements: If-Else

by AvmnuSng

If-Else Conditional Statements

JavaScript Comparison Operators

Equality Operators

Equality (==)

The equality operator is a binary operator that compares two operands, returning *true* if they are deemed to be equal. It works by converting the operands if they are not of the same type, then applying strict comparison. If both operands are primitive types, it will compare their values (i.e., `1 == 1` evaluates to *true*). If both operands are objects, then JavaScript compares their internal references; this means it checks to see if both operands point to the same object (i.e., location) in memory. For example:

- EXAMPLE

```
1 console.log(1 == 1);
2 console.log(1 == "1");
3 console.log('1' == 1);
4 console.log(0 == false);
5 console.log(0 == null);
6 console.log(0 == undefined);
7 console.log(null == undefined);
```

Output

Run

The code above produces the following output:

```
true
true
true
true
false
false
true
```

Inequality (!=)

The inequality operator is a binary operator that returns *true* if the operands are *not equal*. If the two operands are of different types, JavaScript attempts to convert the operands to an appropriate type to compare them. If both operands are objects, then JavaScript compares the internal references to see if they are not equal (i.e., refer to different objects in memory).

Table Of Contents

- EXAMPLE

```
1 console.log(1 != 1);
2 console.log(1 != "1");
3 console.log('1' != 1);
4 console.log(0 != false);
5 console.log(0 != null);
6 console.log(0 != undefined);
7 console.log(null != undefined);
```

Output

Run

The code above produces the following output:

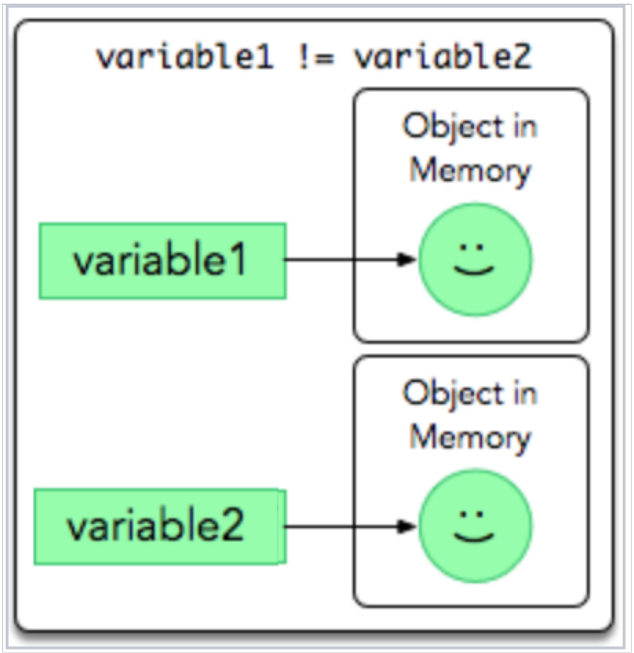
```
false
false
false
false
true
true
false
```

Example: Comparing Objects

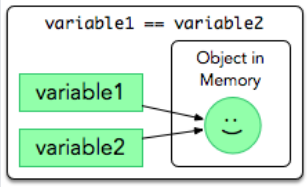
- EXAMPLE

Consider the following diagrams:

1. In this diagram, we have two references that point to different objects that *look* the same:



2. In this diagram, we have two references that point to the same object:



Click *Run* below to see this in code.

```
// This creates a custom object named MyObject
class MyObject {
    // Each object of this type has an attribute named 'magic'
    constructor(magic) {
        this.magic = magic;
    }
    // We'll discuss this syntax in more detail later
}

// Create two objects
var variable1 = new MyObject(":)");
var variable2 = new MyObject(":)");

// Print the result of an equality comparison
console.log( variable1.magic + " == " + variable2.magic
            + " evaluates to "
            + (variable1 == variable2)
            );

// Set variable1 to reference the same object as variable2
variable1 = variable2;
// Print the result of an equality comparison
console.log( variable1.magic + " == " + variable2.magic
            + " evaluates to "
            + (variable1 == variable2)
            );
```

Output

Run

Identity or Strict Equality (===)

The identity operator returns *true* if both of the following conditions are satisfied:

- The operands are strictly equal.
- The operands are of the same type.

\- EXAMPLE

```
console.log(1 === 1);
console.log(1 === "1");
console.log('1' === 1);
console.log(0 === false);
console.log(0 === null);
console.log(0 === undefined);
console.log(null === undefined);
```

Output

Run

The code above produces the following output:

```
true
false
false
false
false
false
false
```

Non-Identity or Strict Inequality (!==)

The non-identity operator returns *true* if the operands satisfy any of the following conditions:

- The operands are not equal.
- The operands are not of the same type.

EXAMPLE

```
console.log(1 !== 1);
console.log(1 !== "1");
console.log('1' !== 1);
console.log(0 !== false);
console.log(0 !== null);
console.log(0 !== undefined);
console.log(null !== undefined);
```

Output

Run

The code above produces the following output:

```
false
true
true
true
true
true
true
```

Relational Operators

Greater Than Operator (`>`)

This binary operator returns *true* if the left operand is greater than the right operand; otherwise, it returns *false*.

Greater Than or Equal Operator (`>=`)

This binary operator returns *true* if the left operand is greater than or equal to the right operand; otherwise, it returns *false*.

Less Than Operator (`<`)

This binary operator returns *true* if the left operand is less than the right operand; otherwise, it returns *false*.

Less Than or Equal Operator (`<=`)

This binary operator returns *true* if the left operand is less than or equal to the right operand; otherwise, it returns *false*.

Example: Relational Operators

EXAMPLE

```
console.log(5 > 5);
console.log(5 >= 5);
console.log(7 < 6);
console.log(4 <= 6);
```

Output

Run

The code above produces the following output:

```
false
true
false
true
```

Logical Operators

Logical AND (`&&`)

Usage: `expression1 && expression2`

If both expressions evaluate to *true*, then it returns *true*; otherwise, it returns *false*.

Logical OR (`||`)

Usage: `expression1 || expression2`

If both expressions evaluate to *false*, then it returns *false*; otherwise, it returns *true*.

Logical NOT (`!`)

Usage: `!expression` If the expression (by itself) evaluates to *false*, it returns *true* (i.e., the logical negation of *false*); otherwise, it returns *false*.

\- EXAMPLE

```
(5 < 7) && (4 < 4)
(5 < 7) && (4 >= 4)
(5 < 7) || (4 < 4)
(5 >= 7) || (4 > 4)
!(2**3)
```

Output

Run

The code above produces the following output:

```
false
true
true
false
false
```

Short-Circuit Evaluation

As logical expressions are evaluated from left to right, they are tested for possible short-circuit evaluation using the following rules:

- `false && expression` is short-circuit evaluated to *false*.
- `true || expression` is short-circuit evaluated to *true*.

If-Else Statements

Use the *if* statement to execute a statement if a logical condition (i.e., some statement that evaluates to *true* or *false*) is *true*. Use the optional *else* clause to execute a statement *only in the event that the if condition evaluates to false*. The code below demonstrates the basic syntax for this:

```
if (condition) {
    statement1;
}
else {
    statement2;
}
```

In the code above:

- ***condition*** can be any expression that evaluates to *true* or *false*.
- If ***condition*** evaluates to *true*, then ***statement1*** is executed; otherwise, ***statement2*** is executed.
- ***statement1*** and ***statement2*** represent any statement (or sequence of statements), including additional nested *if* statements.

The code below demonstrates multiple statements inside an *if-else* block:

```
if (condition1) {
    statement1;
    statement4;
    statement5;
}
else {
    statement2;
    statement3;
    if (condition2) {
        statement6;
    }
}
```

Additionally, you can compound the statements using the *else-if* clause to test multiple conditions in sequence:

```
if (condition1) {
    statement1;
}
else if (condition2) {
    statement2;
```

```
}
else if (conditionN) {
    statementN;
}
else {
    statementLast;
}
```

Chaining related logic conditions using *else-if* in this way has a few benefits:

- When there are multiple conditions being checked within a chained sequence of statements, only the first logical condition to evaluate to true will be executed. This also means that after one of the logical conditions evaluates to true, any subsequent logical statements in the block will be skipped over. For example, let's say $condition1$ in the code above evaluates to *false*, but $condition2$ evaluates to *true*. If this happens, the program will execute $statement2$ and then jump to the end of the chain of statements and continue executing (meaning it skips over $conditionN$ and the last *else*).
- If a later condition check is reached, you know that all the preceding condition checks within that chain all evaluated to false. This means you don't have to re-check certain conditions. Try changing the integer in the *Input* box below and clicking *Run* to see this in code:

```
process.stdin.on('data', function (data) {
    main(+(data));
});
/**** Ignore above this line. ****/

function classifyAge(age) {
    /* First, let's check the lower bound on our age range: */
    if (age < 13) {
        return age + " is a child.";
    }
    /* If this condition is checked, we know that age < 13 is false:
    else if (age < 20) {
        return age + " is a teenager.";
    }
    /* If this condition is checked, we know both of these are false
    *   age < 13 is false
    *   age < 20 is false
    *  This tells us that either age >= 20 is true, or age is not a
    */
    else if (age >= 20){
        return age + " is an adult.";
    }
    /* The input wasn't a number. */
    else {
        return "Your input must be an integer.";
    }
}

function main(input) {
    console.log(classifyAge(input));
}
```

Input

```
12
```

Run

Output

Falsy Values

The following six values are known as *Falsy* values, meaning they evaluate to *false*:

- `false`

- `undefined`
- `null`
- `0`
- `NaN`
- `""` (i.e., the empty string)

All other values, including all objects, evaluate to true when used as the *condition* in a conditional statement. Click *Run* below to see this in code.

```
var a = true;
var b = new Boolean(false);
var c = "";

console.log(a);
console.log(b);
console.log("\"" + c + "\"");

if (a) {
    console.log("Hello from a");
}

if (b) {
    console.log("Hello from b");
}

if (c) {
    console.log("Hello from c");
}
else {
    console.log("c is false");
}
```

Output

Run

Conditional (Ternary) Operator

The conditional (ternary) operator is the only JavaScript operator that takes three operands, and it's used as a shortcut for the *if* statement. The basic syntax is:

```
condition ? trueStatement : falseStatement
```

You can essentially read the `?` as the word "then" and the `:` as the word "else". If ***condition*** evaluates to true, then `trueStatement` is executed; else, `falseStatement` is executed. For example, try changing the integer in the *Input* box below and clicking *Run* to see this in code:

```
process.stdin.on('data', function (data) {
    main(+(data));
});
/**** Ignore above this line. ****/

function main(input) {
    // The examples below all accomplish the same thing.

    // Example 1:
    input % 2 == 0 ? console.log(input + " is EVEN") : console.log(i

    // Example 2:
    console.log( input + " is " + ((input % 2 == 0) ? "EVEN" : "ODD"

    // Example 3:
    var parity = input % 2 == 0 ? "EVEN" : "ODD";
    console.log(input + " is " + parity);
```

```
18 }
```

Input

4

Run

Output

- SWITCH CONDITIONAL STATEMENTS Recommended Article

Switch Conditional Statements View