JavaScript Comparison Operators

Summary: in this tutorial, you will learn how to use JavaScript comparison operators to compare two values.

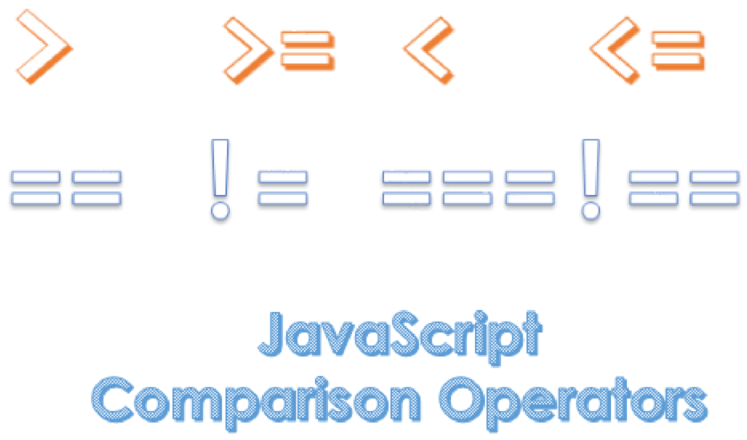
Introduction to JavaScript comparison operators

To compare two values, you use a comparison operator. The following table shows the comparison operators in JavaScript:

Operator	Meaning
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

A comparison operator returns a Boolean value indicating whether the comparison is true or not. See the following example:

```
let r1 = 20 > 10; // true
let r2 = 20 < 10; // false
let r3 = 10 == 10; // true
```



A comparison operator takes two values. If the types of values are not comparable, the comparison operator converts them into values of comparable types according to specific rules.

Compare numbers

If values are numbers, the comparison operators perform a numerical comparison. For example:

```
let a = 10,
    b = 20;

console.log(a >= b);  // false
console.log(a == 10); // true
```

This example is straightforward. The variable `a` is `10`, `b` is `20`. The expression `a >= b` expression returns `false` and the expression `a == 10` expression returns `true`.

Compare strings

If the operands are strings, JavaScript compares the character codes numerically one by one in the string.

```
let name1 = 'alice',
    name2 = 'bob';

let result = name1 < name2;
console.log(result); // true
console.log(name1 == 'alice'); // true
```

Since JavaScript compares the character codes in the strings numerically, you may receive an unexpected result, for example:

```
let f1 = 'apple',
    f2 = 'Banana';
let result = f2 < f1;
console.log(result); // true
```

In this example, `f2` is less than `f1` because the letter `B` has the character code `66` while the letter `a` has the character code `97`.

To fix this, you need to:

- First, convert the strings into a common format, either lowercase or uppercase
- Second, compare the converted values

For example:

```
let f1 = 'apple',
    f2 = 'Banana';

let result = f2.toLowerCase() < f1.toLowerCase();
console.log(result); // false
```

Note that the `toLowerCase()` is a method of the String object that converts the string to lowercase.

Compare a number with a value of another type

If one value is a number and the other is not, the comparison operator will convert the non-numeric value into a number and compare them numerically. For example:

```
console.log(10 < '20'); // true
```

In this example, the comparison operator converts the string `'20'` into the number `20` and compares with the number 10. Here is an example:

```
console.log(10 == '10'); // true
```

In this example, the comparison operator converts the string `'10'` into the number `10` and compares them numerically.

Compare an object with a non-object

If a value is an object, the `valueOf()` method of that object is called to return the value for comparison. If the object doesn't have the `valueOf()` method, the `toString()` method is called instead. For example:

```
let apple = {
  valueOf: function () {
    return 10;
  },
};

let orange = {
  toString: function () {
    return '20';
  },
};
console.log(apple > 10); // false
console.log(orange == 20); // true
```

In this first comparison, the `apple` object has the `valueOf()` method that returns `10`. Therefore, the comparison operator uses the number 10 for comparison.

In the second comparison, JavaScript first calls the `valueOf()` method. However, the `orange` object doesn't have the `valueOf()` method. So JavaScript calls the `toString()` method to get the returned value of `20` for comparison.

Compare a Boolean with another value

If a value is a Boolean value, JavaScript converts it to a number and compares the converted value with the other value; `true` is converted to `1` and `false` is converted to `0`. For example:

```
console.log(true > 0); // true
console.log(false < 1); // true
console.log(true > false); // true
console.log(false > true); // false
console.log(true >= true); // true
console.log(true <= true); // true
console.log(false <= false); // true
console.log(false >= false); // true
```

In addition to the above rules, the equal (`==`) and not equal (`!=`) operators also have the following rules.

Compare null and undefined

In JavaScript, `null` equals `undefined` . It means that the following expression returns `true` .

```
console.log(null == undefined); // true
```

Compare NaN with other values

If either value is `NaN` , then the equal operator(`==`) returns `false` .

```
console.log(NaN == 1); // false
```

Even

```
console.log(NaN == NaN); // false
```

The not-equal (`!=`) operator returns `true` when comparing the `NaN` with another value:

```
console.log(NaN != 1); // true
```

And also

```
console.log(NaN != NaN); // true
```

Strict equal (===) and not strict equal (!==)

Besides the comparison operators above, JavaScript provides strict equal (`===`) and not strict equal (`!==`) operators.

Operator	Meaning
===	strict equal
!==	not strict equal

The strict equal and not strict equal operators behave like the equal and not equal operators except that they don't convert the operand before comparison. See the following example:

```
console.log("10" == 10); // true
console.log("10" === 10); // false
```

In the first comparison, since we use the equality operator, JavaScript converts the string into a number and performs the comparison.

However, in the second comparison, we use the strict equal operator (`===`), JavaScript doesn't convert the string before comparison, therefore the result is `false` .

In this tutorial, you have learned how to use the JavaScript comparison operators to compare values.

Quiz