Comparisons in JavaScript

[(https://github.com/learn-co-curriculum/phase-0-pac-1-comparisons-in-javascript)](https://github.com/learn-co-curriculum/phase-0-pac-1-comparisons-in-javascript) [(https://github.com/learn-co-curriculum/phase-0-pac-1-comparisons-in-javascript/issues/new)](https://github.com/learn-co-curriculum/phase-0-pac-1-comparisons-in-javascript/issues/new)

Learning Goals

- Identify equality operators
- Compare numbers using relational operators

Introduction

Now that we know what *Boolean expressions* are we'll start learning how to create them. In this lesson we'll learn about comparison operators, which enable us to check to see if a value is what we're expecting. Follow along with the examples below in **replit** (https://replit.com/languages/javascript).

Note: JavaScript includes both *strict* and *loose* comparison operators. When writing JavaScript, you should strongly prefer the **strict** operators.

The reason for this is the loose operators will return true even if the data types aren't the same, which can result in unexpected behavior and bugs that can be difficult to track down. Even if you find you need to compare two values of different data types, you should avoid using loose operators. You will be better off converting the data type of one of the variables so they have the same type, then comparing them using a strict operator.

Identify equality operators

JavaScript includes four equality operators:

- **strict equality operator** (`===`)
- **strict inequality operator** (`!==`)
- **loose equality operator** (`==`)
- **loose inequality operator** (`!=`)

These operators allow us to compare values and determine whether they are the same.

Strict Equality Operator `===` and Strict Inequality Operator `!==`

The **strict equality operator** returns `true` if two values are equal *without performing type conversions*. Even if the values on both sides of the operator look similar (e.g., `'42' === 42`), the `===` operator will only return `true` if the data types also match:

```
42 === 42;
// => true

42 === "42";
// => false

true === 1;
// => false

"0" === false;
// => false

null === undefined;
// => false

" " === 0;
// => false
```

This is logical and accurate!

The **strict inequality operator** returns `true` if two values are *not* equal *without* performing type conversions:

```
9000 !== 9001
// => true

9001 !== '9001'
```

```
// => true

[] !== ''
// => true
```

You should prefer `===` and `!==` for comparisons.

Loose Equality Operator `==` and Loose Inequality Operator `!=`

The **loose equality operator** returns `true` if two values are equal:

```
42 == 42;
// => true
```

However, it will *also* return `true` if it can perform a type conversion (e.g., changing the string `'42'` into the number `42`) that makes the two values equal:

```
42 == "42";
// => true

true == 1;
// => true

"0" == false;
// => true

null == undefined;
// => true

" " == 0;
// => true
```

The **loose inequality operator** is the opposite of `==`. It returns `true` if two values are *not* equal, performing type conversions as necessary:

```
9000 != 9001
// => true

9001 != '9001'
// => false

[] != ''
// => false
```

This is confusing and inaccurate! It makes no sense that the string `'0'` is equal to the boolean `false` or that `null` and `undefined` — two **completely different** data types — are equivalent.

You should prefer `===` and `!==` for comparisons. There are **a lot of rules (https://dorey.github.io/JavaScript-Equality-Table/)** that JavaScript follows when performing type coercion using the `==` operator, and it's not worth keeping track of them all.

Compare Numbers with Relational Operators

JavaScript includes four relational operators:

- **greater than** (`>`)
- **greater than or equals** (`>=`)
- **less than** (`<`)
- **less than or equals** (`<=`)

The behavior of these operators is consistent with the meaning of the corresponding symbols in mathematics:

```
88 > 9;
// => true

88 >= 88;
// => true
```

```
88 < 9;
// => false
```

However, beware of type conversion when comparing non-numbers against numbers. For instance, when a string is compared with a number, the JavaScript engine tries to convert the string to a number:

```
88 > "9";
// => true
```

If the engine can't convert the string into a number, the comparison will always return `false` :

```
88 >= "hello";
// => false

88 <= "hello";
// => false
```

Strings are compared with other strings lexicographically, meaning character-by-character from left-to-right. The following returns `false` because the Unicode value of `8` , the first character in `88` , is less than the Unicode value of `9` .

```
"88" > "9";
// => false
```

If you aren't sure what data type you are going to be receiving, but you still need to compare them, make sure that you tell JavaScript to **convert the string to a number first** (https://gomakethings.com/converting-strings-to-numbers-with-vanilla-javascript/) , and then compare.

> **Top Tip**: Stick to comparing *numerical* values with the relational operators and you'll be golden.

Conclusion

JavaScript contains both equality and relational operators that we can use in writing code to compare two values. The resulting statements are *Boolean expressions* — they always return `true` or `false`.

Make sure you're using the strict equality operators whenever possible, and only comparing numerical values using the relational operators, and you'll avoid annoying errors that can be time consuming to troubleshoot!

Resources

- MDN
  - **Comparison operators (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators)**
  - **Equality comparisons and sameness (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness)**
- **JavaScript Equality Table (https://dorey.github.io/JavaScript-Equality-Table/)**
- **freeCodeCamp Forum — JavaScript Comparison Operators (https://forum.freecodecamp.org/t/javascript-comparison-operators/14660)**