**Why Use the Triple-Equals Operator in JavaScript?**

By Louis Lazaris on March 1st, 2012 | [31 Comments](https://www.impressivewebs.com/why-use-triple-equals-javascipt/#comments)

“Determining whether two variables are equivalent is one of the most important operations in programming.” That’s according to [Nicholas Zakas](http://www.nczonline.net/) in his book [JavaScript for Web Developers](http://www.amazon.com/Professional-JavaScript-Developers-Wrox-Programmer/dp/047022780X/).

In other words, throughout your scripts you’ll probably have lines resembling this:

if (x == y) {

// do something here

}

Or, if you’re conforming to best practices, this:

if (x === y) {

// do something here

}

The difference between those two examples is that the second example uses the triple-equals operator, also called “strict equals” or “identically equal”.

JavaScript beginners who try to adhere to best practices may be using triple-equals and not double-equals, but might not fully understand what the difference is or why it’s important to stick to triple-equals.

**What’s the Difference?**

In a comparison using the double-equals operator, the result will return true if the two things being compared are equal. But there’s one important catch: If the comparison being made is between two different “types” of values, type coercion will occur.

Each JavaScript value belongs to a specific “type”. These types are: Numbers, strings, Booleans, functions, and objects. So if you try comparing (for example) a string with a number, the browser will try to convert the string into a number before doing the comparison. Similarly, if you compare true or false with a number, the true or false value will be converted to 1 or 0, respectively.

This can bring unpredictable results. Here are a few examples:

console.log(99 == "99"); // true

console.log(0 == false); // true

Although this can initially feel like a good thing (because the browser seems to be doing you a favour), it can cause problems. For example:

console.log(' \n\n\n' == 0); // true

console.log(' ' == 0); // true

In light of this, most JavaScript experts recommend always using the triple-equals operator, and never using double-equals.

The triple-equals operator, as you’ve probably figured out by now, never does type coercion. So whenever you use triple-equals, you’re doing an exact comparison of the actual values. You’re ensuring the values are ‘strictly equal’ or ‘identically equal’.

This means that, using triple-equals, all the examples from above will produce the correct results:

console.log(99 === "99"); // false

console.log(0 === false); // false

console.log(' \n\n\n' === 0); // false

console.log(' ' === 0); // false

**What About Inequality?**

When doing a not-equals-to expression, the same rules apply. Except this time, instead of triple-equals vs. double-equals, you’re using double-equals vs. single.

Here are the same examples from above, this time expressed with the != operator:

console.log(99 != "99"); // false

console.log(0 != false); // false

console.log(' \n\n\n' != 0); // false

console.log(' ' != 0); // false

Notice now that the desired result in each case should be “true”. Instead, they’re false — because of type coercion.

If we change to double-equals, we get the correct results:

console.log(99 !== "99"); // true

console.log(0 !== false); // true

console.log(' \n\n\n' !== 0); // true

console.log(' ' !== 0); // true

**Conclusion**

As mentioned, you’ve probably already used triple-equals pretty exclusively. While researching this article, I learned a few things about this concept myself.

I think the best summary comes from Zakas again, where, after recommending always using strict equals, he says: “This helps to maintain data type integrity throughout your code.”