Equality

So far, you've seen how you can use `==` and `!=` to compare numbers and strings for equality. However, if you use `==` and `!=` in situations where the data you're comparing is mixed, it can lead to some interesting results. For example,

```
"1" == 1
```

> **Returns:** true

and

```
0 == false
```

> **Returns:** true

both evaluate to true. Why is that?

Implicit type coercion

JavaScript is known as a *loosely typed language*.

Basically, this means that when you're writing JavaScript code, you do not need to specify data types. Instead, when your code is interpreted by the JavaScript engine it will automatically be converted into the "appropriate" data type. This is called *implicit type coercion* and you've already seen examples like this before when you tried to concatenate strings with numbers.

```
"julia" + 1
```

> **Returns:** "julia1"

In this example, JavaScript takes the string `"julia"` and adds the number `1` to it resulting in the string `"julia1"`. In other programming languages, this code probably would have returned an error, but in JavaScript the number `1` is converted into the string `"1"` and then is concatenated to the string `"julia"`.

It's behavior like this which makes JavaScript unique from other programming languages, but it can lead to some quirky behavior when doing operations and comparisons on mixed data types.

QUESTION 1 OF 2

What value do you think the result of `"Hello" % 10` will be?

- ○ 0
- ○ "Hello10"
- ○ 10
- ○ SyntaxError
- ○ NaN

SUBMIT

Mentorship
Get support and stay on track

Mentorship
Get support and stay on track

need to specify data types; however, this can lead to errors that are hard to diagnose due to implicit type coercion.

Example of strongly typed programming language code

```
int count = 1;
string name = "Julia";
double num = 1.2932;
float price = 2.99;
```

Equivalent code in JavaScript

```
// equivalent code in JavaScript
var count = 1;
var name = "Julia";
var num = 1.2932;
var price = 2.99;
```

In the example below, JavaScript takes the string `"1"`, converts it to `true`, and compares it to the boolean `true`.

```
"1" == true
```

Returns: true

When you use the `==` or `!=` operators, JavaScript first converts each value to the same type (if they're not already the same type); this is why it's called "type coercion"! This is often not the behavior you want, and **it's actually considered bad practice to use the `==` and `!=` operators when comparing values for equality**.

Strict equality

Instead, in JavaScript it's better to use **strict equality** to see if numbers, strings, or booleans, etc. are identical in *type* and *value* without doing the type conversion first. To perform a strict comparison, simply add an additional equals sign `=` to the end of the `==` and `!=` operators.

```
"1" === 1
```

Returns: false

This returns false because the string `"1"` is not the same type *and* value as the number `1`.

```
0 === false
```

Returns: false

This returns false because the number `0` is not the same type *and* value as the boolean `false`.

QUESTION 2 OF 2

Check the expressions that evaluate to `true`.

- [ ] "3" > 1
- [ ] 3 != "3"
- [ ] true >= 0

Lesson 2:
Data Types & Variables

Equality

- [ ] "false" === 0
- [ ] 3 === 3

SUBMIT

NEXT

Mentorship
Get support and stay on track