Lesson Monday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / A Common Data Type Error and Data Type Coercion

Text Cheat sheet

The most common error beginning JavaScript students experience is attempting to call methods or functions meant for numbers on strings. This happens especially frequently because **all values returned from `window.prompt()` are saved as strings**. Even if the user enters a number into a prompt without quotation marks! This may be confusing, but this is simply how JavaScript works.

If we want to collect a value from a user with `window.prompt()`, and perform arithmetic with this value, we will need to convert it into a number with JavaScript's `parseInt()` function (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt). If not, we will receive some very unexpected output.

In this lesson, we'll review this common data type error, and then we'll learn about the **data type coercion** that JavaScript automatically performs in certain situations.

A Common Data Type Error

Here's what can happen if you *don't* parse integers (whole numbers) correctly. In the example below, we use `window.prompt()` to ask the user how old they are. The user enters 35. We then attempt to perform some arithmetic on `age`:

```
> const age = window.prompt("Enter your age: ");
> age + age;
"3535"
```

Remember, **all values returned from `window.prompt()` are strings!** So, even though the user entered a *number*, `age` is actually equal to the *string* `"35"`. Since the `+` operator can be used to concatenate strings `age + age` actually returns `"3535"`, instead of `70`.

Using parseInt() to Avoid Data Type Errors

To convert a string into a number, we parse it using the `parseInt()` function like this:

```
> const number = parseInt("5");
> number;
5
```

So, to change our `age` from the example above into an integer, we do the following:

```
> const inputtedAge = window.prompt("Enter your age: ");
> const age = parseInt(inputtedAge);
> age + age;
70
```

Alternatively, we can write our code on on a single line:

```
> const age = parseInt(window.prompt("Enter your age: "));
> age + age;
70
```

Here we're immediately passing the string returned by the `window.prompt()` method as the argument to the `parseInt()` function.

Use `parseFloat()` for Decimals

The *Int* part of `parseInt()` is short for integer, which means "whole number". If we wanted to convert a string into a number with a decimal, we'd use JavaScript's `parseFloat()`:

```
> const pi = "3.14";
> parseFloat(pi);
3.14
```

Floating point numbers are simply numbers with decimals. If you're parsing a whole number, use `parseInt()`, if you're parsing a number containing a decimal point, use `parseFloat()`.

Visit the MDN documentation to learn more about `parseFloat()`:

- **`parseFloat` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseFloat)**

Data Type Coercion

So far, we've discussed data type conversion with handy JS functions and methods:

- `parseInt()`
- `Number.prototype.toString()`
- `Boolean.prototype.toString()`
- `parseFloat()`

When we use these methods we are **explicitly** converting the data type we have. This process is also called **typecasting**. At this point, we should be familiar with when and why we convert from one data type to another.

However, JavaScript also performs **implicit** data type conversion. This is usually called **coercion** (https://developer.mozilla.org/en-US/docs/Glossary/Type_coercion), which is when JavaScript automatically decides on its own to force a piece of data to change into another type. Let's see what this looks like (try this in the DevTools console if you like):

```
> 54 + 3 + "45";
"5745"
```

When JavaScript reads this code from left to right, it first performs the addition of `54 + 3`, and then it turns the result `57` into a string that it then concatenates with `"45"`, returning the string `"5745"`. Yikes! That's unexpected!

So why doesn't JavaScript return an error instead? Well, JavaScript is just not set up that way. Indeed, implicit data type coercion can lead to weird bugs and frustration. This is why some developers prefer "strongly typed" languages (like Typescript, a strongly typed version of JavaScript), where all variables need to be labeled with their data type, and any data type conversion only happens explicitly by the choice of the developer writing the code.

However, JavaScript's data type coercion can also be harnessed to write less code that's (arguably) more readable. We'll see more cases of this as we move through this section's material and get to branching. As a developer, your job is simply to be aware that JavaScript performs implicit data type coercion and of some of the cases in which it can pop up and cause issues.

Examples of Data Type Coercion

We see data type coercion come up when we use JavaScript arithmetic, comparison, and equality operators on incompatible data types:

- `+`, `-`, `*`, `/`
- `>`, `<`, `==` (the loose equality operator, which we don't use)

Let's see how this works with booleans:

```
> true + true;
2;
> true + false;
1;
> false - false;
0;
> true * false;
0
> true * 6;
6
> true + " or false: do you like data type coercion?";
"true or false: do you like data type coercion?"
> true * "false";
NaN
```

With arithmetic operators applied to booleans, `true` is converted to the number `1` and `false` is converted to the number `0`. In that way, we can use any arithmetic operator between a boolean and a number, and JavaScript will implicitly convert the boolean into the number.

When we use the addition `+` operator between a boolean and a string, JavaScript implicitly converts the boolean into a string. If we use any other arithmetic operator between a boolean and a string, we'll get `NaN` returned to us, which is one of the 5 different types of operations that return `NaN` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN#description).

Let's see how numbers and strings interact:

```
> "I am " + 54 + " years old.";
"I am 54 years old."
> 54 + " years old am I.";
"54 years old am I."
> 6 + 2 + 1 + 1 + "45";
"1045"
> "45" + 54 + 3;
"45543"
> "45" - 54 + 3;
-6
> "55" * 2;
110
> 10 / "2";
5
```

Because the addition `+` operator is used for concatenation of strings, no matter how many numbers are added together, once JavaScript hits a string, it implicitly converts the resulting value or subsequent operations into a string. However, the operators `-`, `/`, and `*` are never used with strings, so any string with a number in it is implicitly converted to a number.

We also see data type coercion come up when we mix up data types when working with (some) methods and properties. Let's see some examples:

```
> window.alert(window.innerHeight);
```

The `window.alert()` method takes a string as an argument, but we're passing in `window.innerHeight`, which is a number! What do you think happens when we input the above code? Well, it works! This is a case where we can harness JavaScript's data type coercion to our benefit, because it allows us to write less code. It's debatable whether this code is easier to read than the longer form `window.alert(window.innerHeight.toString());`.

We'll see more examples of when data type coercion can be helpful throughout this section and beyond. Remember, as a developer, your job is simply to be aware that JavaScript performs implicit data type coercion and of some of the cases in which it can pop up and cause issues. Don't feel like you need to remember all of the examples that we've covered in this lesson. If you are ever unsure about what data type you are working with, use the `typeof` operator to check.

Previous (/introduction-to-programming/javascript-and-web-browsers/interactivity-with-window-methods)
Next (/introduction-to-programming/javascript-and-web-browsers/practice-interactivity-with-window-methods)

Lesson 34 of 75
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)