Lesson Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Static versus Instance with Built-In JS Objects

Text

You may have noticed references to "instance" or "static" when perusing JavaScript string or number methods on MDN documentation. Or maybe you decided to explore JavaScript's Math library and used a method that looks like this:

```
// This method finds the absolute value of a number.
> Math.abs(-3);
3
```

If you haven't yet noticed this distinction, this lesson will explain it to you now. We also want to introduce a few additional JS objects that provide a lot of helpful functionality. The goal of covering this information is to expose you to new JavaScript tools that you can explore on your own time. It's totally optional to use any of the concepts covered in your projects — you just might not need them!

Standard Built-in JS Objects

All standard, built-in JS objects can be found listed at this resource:

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

As the name suggests these "Global Objects" can be called on from anywhere in our JS or DevTools console. We want to highlight two of these objects:

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

The `Math` object has a bunch of useful functionality related to Math:

- a method that randomizes numbers
- a method that rounds a number to the nearest integer
- a property that contains pi

Okay, well maybe these aren't so useful if you are not doing mathematical calculations — that's okay. It's still good to know these exist.

The `Date` object also has useful functionality. Things like:

- calculating the day of week of a given date
- calculating the month of the year for a given date
- calculating time zone differences based on universal time

The `Date` object also may not be helpful if you are not creating functionality around dates, but it is good to be aware of its existence. There's a small learning curve to working with the `Date` object, since it uses numbers to represent each date, and these numbers are the milliseconds elapsed since 1 January 1970 UTC. There's a reason for this system, and you'll get to explore it when the time comes!

What's important now is to explain the difference between **static** and **instance** methods so that you are more prepared to explore both of these objects.

A Distinction Among Properties and Methods: Instance and Static

Before we continue, take note that identifying methods and properties as "instance" or "static" is used in describing JavaScript, and not any of the Web APIs we've discussed thus far. This doesn't mean that this concept doesn't exist for Web APIs, it just means that it's not an important distinction to make. Generally speaking, you won't see the terms "instance" and "static" on Web API reference pages.

So far when we've talked about string or number methods, we've technically been talking about instance methods. An **instance method** is a method that is called on an instance of an object type. The same is true for **instance properties**. For example, we can create an instance of the `String` object like so:

```
> const wifiPassCode = "iLovePizza2000";
```

The `wifiPassCode` is actually an instance of the `String` object type. We can call instance methods on it like:

```
> wifiPassCode.toUpperCase();
"ILOVEPIZZA2000"
```

Strings also have one instance property that we can access on every single instance of a string:

```
> wifiPassCode.length;
14
```

The `String` instance property `length` returns the number of characters in the string.

So what is a static property or method? A **static property** or method is one that is called on the object type itself. For example, an instance of a `Number` and an instance method looks like this:

```
> let howManyPaintingsIveMade = 32;
> howManyPaintingsIveMade.toString();
"32"
```

A `Number` static method and property looks like this:

```
> Number.MAX_SAFE_INTEGER;
9007199254740991
> Number.isNaN(0/0);
true
```

As we can see, static properties and methods are called on the object type itself. These object types are always capitalized. If we don't capitalize them, we'll get `undefined` or an error returned:

```
> number.MAX_SAFE_INTEGER;
undefined
> number.isNaN(0/0);
Uncaught TypeError: number.isNan is not a function at <anon
ymous>:1:8
```

The easiest way to distinguish between an instance method and a static one is to look at the method definition on MDN. If a method is listed with `prototype`, it is an instance method. For example,

```
String.prototype.toUpperCase()
Number.prototype.toString()
Array.prototype.concat()
```

However, there's no easy way to distinguish between instance and static properties. Sometimes static properties are all uppercased, but that's not consistent across all JS objects. The best thing to do is just reference the MDN page for an object you are working with (or want to work with). Each reference page distinguishes between static and instance when it comes to listing properties and methods. Also note that JS object types can have any number of instance or static properties and methods. For example, the `Math` object consists solely of static properties and methods.

Previous (/introduction-to-programming/arrays-and-looping/the-basics-of-prototypes)
Next (/introduction-to-programming/arrays-and-looping/introduction-to-arrays)

Lesson 4 of 50
Last updated February 28, 2023

disable dark mode

Epicodus (http://www.epicodus.com)