Lesson Weekend

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Methods

Text Cheat sheet

In this lesson, we'll learn about key functionality for JavaScript: methods and functions. We'll start with briefly defining functions, because methods are a type of function.

A **function** is a bundle of code that performs a set of procedures. It describes actions that our code can perform. For instance, we could write a custom function to take a string, like "hello", and uppercase it so the original string "hello" is turned into "HELLO". When we **call a function**, we are asking it to perform its actions.

A **method** is a type of function that belongs to a specific type of data, like strings or numbers. We **call** a method **on** something, asking the method to perform its actions on the data type it belongs to. When a method belongs to a data type, that means we can only call it on that data type.

In JavaScript, there are methods and functions that are built-in to the language, and we can also write our own custom functions and methods. When functionality is **built-in** to JavaScript, it means that

it is a part of the language and all we need to do is learn the correct syntax to incorporate that functionality into our code. On the other hand, **custom** functions and methods are ones that we design using the tools that JavaScript provides us.

In this lesson, we'll start using built-in JavaScript methods to do things for us. We will practice a lot more in future lessons, too, so if you do not understand something in this lesson, trust that your understanding will develop soon enough. The goal is to introduce you to these concepts and the terminology associated with them. Then, in subsequent lessons we'll review and practice.

Take notes on anything you don't understand so that you can discuss it with your dev team, pair, or in Scrum. The time it takes to understand methods and functions will be different for everyone in your cohort, and discussing points of confusion is instrumental in getting to a solid understanding of these JavaScript concepts. Again, **if you are confused about any of these concepts, just remember that we will be covering them in greater depth later and that you don't need to know everything about how JavaScript works in order to use basic methods and functions.**

Using Built-In JavaScript Methods

First, let's take a look at an example of a JavaScript method to uppercase a string.

```
> "This is a string.".toUpperCase();
```

Try this out in the console. If you're working alone or in person with a pair, open Chrome's DevTools console. A quick refresher on how to access it:

- Click on the three dots at the upper right corner of the browser screen.

- Go to the *More Tools* menu.
- Click *Developer Tools*.
- Go to the *Console* tab within the window that pops up.

You can also access the DevTools console with these shortcuts:

- Windows:
  - `Ctrl + Shift + j`
  - `Ctrl + Shift + i`
- Mac:
  - `Cmd + Option + i`
  - `F12`

If you're working remotely with others, remember to take turns screen sharing the DevTools console on Discord.

Here's what happens in the console when we call this method:

```
> "This is a string.".toUpperCase();
"THIS IS A STRING."
```

`toUpperCase` is a method that can be called on *any* JavaScript string.

`"THIS IS A STRING."` is the **return value** from calling the `toUpperMethod` on the string `"This is a string." A return value is anything that's returned from calling a method or function.

Methods Are Called *On* Something

So here's the important distinction about methods. As we mentioned at the beginning of this lesson, methods are always called *on* something. The thing they are called on is known as the **receiver**. In contrast, functions don't necessarily have to be called on something. That's why a method is always a function, but a function isn't necessarily a method. We will continue to discuss this distinction in future lessons.

So what do we mean by receiver? The receiver is the thing to the *left* of the method. In the example above, the receiver is `"This is a string."`. The `toUpperCase` method is called *on* `"This is a string."`

Here's the syntax of a method:

```
// This isn't real code!
// This is **pseudocode** that describes the syntax of actu
al JS code.
// In this case, we're showing where the receiver is.

receiver.method();
```

As we can see, the receiver is on the left. The method that is called on the receiver is on the right and always includes parentheses.

JavaScript Comments

Also, in JavaScript, everything after the `//` is a **comment**. We can also write inline comments with this syntax: `/* */`. Here's an example:

```
receiver.method(/* any arguments are passed in here */);
```

When JavaScript interprets our code, it will ignore these comments. Comments are a convenient way to leave notes in your code for yourself or other programmers. Note — we'll learn about arguments in just a few paragraphs!

How To Call a Method

To call a method means that we're telling it to perform its functionality. To call a method, we need to include parentheses after the name of the method. By the way, **parens** is a shorthand

word for "parentheses".

```
> "hello".toUpperCase() // calling the toUpperCase method o
n the string "hello"
'HELLO'
```

The above example is familiar — we're calling the `toUpperCase` method on the string `"hello"`. Now, let's see what happens when we don't include parens.

```
> "hello".toUpperCase // there are no parens, so we're NOT
calling our method
ƒ toUpperCase() { [native code] }
```

This is new! What's going on here? In this case, when we input `toUpperCase` without the parens, the definition of the `toUpperCase` method is returned: `ƒ toUpperCase() { [native code] }`. A function or method definition tells us what the code does. Here's a breakdown:

- `ƒ` stands for "function", which includes methods (as we've learned, methods are a type of function).
- `toUpperCase` is the name of the function.
- `() { [native code] }` is the syntax for the function definition. In this case, `[native code]` is a placeholder for the JavaScript source code for this built-in method.

If you don't completely understand the above, that is okay — we'll have more chance to practice this in the coming section and weeks ahead. Just remember to include parens when you are calling a method. When we don't, we're effectively asking "what does this method/function do?".

Arguments

Both functions and methods can have **arguments**. Arguments are passed into the parentheses `()` when we call the function/method. It's not required for a method or function to have arguments. Some take no arguments, some take optional arguments, and some take one or many arguments.

Remember the following code snippet? It's the explanation of method syntax in pseudocode. The comment `/* any arguments are passed in here */` is indicating that arguments are passed into the parentheses `( )` of a method or function call.

```
// This is pseudocode to show the syntax of calling a metho
d on a receiver, and where arguments go.
receiver.method(/* any arguments are passed in here */);
```

The `toUpperCase` method doesn't take any arguments. It just uppercases a string. So let's try out a built-in JavaScript method that does take arguments.

We can use the `concat` string method to **concatenate** strings. That's a fancy way of saying "combine separate strings into one string". Let's look at an example in the console:

```
> "This is a string".concat("!!!");
"This is a string!!!"
```

In the example above, `"This is a string"` is the receiver, `concat` is the method, and `"!!!"` is the argument we pass into the method. As we can see, the `concat` method takes the argument (in this case `"!!!"`) and attaches it to the receiver (`"This is a string!!!"`)

We can pass in as many additional arguments to this method as we want. For example, we can do the following:

```
> "This is a string".concat("!!!", "I like strings!");
"This is a string!!!I like strings!"
```

As this example shows, the `concat` method can take multiple arguments. Be careful, though. If you want spaces, you'll need to account for that. The method won't add them automatically. To add spaces you could do any of these:

```
> "This is a string".concat("!!!", " ", "I like strings!");
// add a new argument
"This is a string!!! I like strings!"
> "This is a string".concat("!!!", " I like strings!"); //
add a space before the 2nd argument
"This is a string!!! I like strings!"
> "This is a string".concat("!!! ", "I like strings!"); //
add a space after the 1st argument
"This is a string!!! I like strings!"
> "This is a string".concat("!!! I like strings!"); // only
use 1 argument
"This is a string!!! I like strings!"
```

The above should be a healthy reminder that there are many ways to code a solution to a problem.

Calling Methods on Variables

We can call methods *on* variables. For instance, try this in the console:

```
> let stringVariable = "hi!";
> stringVariable.toUpperCase();
"HI!"
```

As we can see, we can call a method on a receiver that contains a variable.

Chaining Methods

It is possible in JavaScript to **chain methods** together, in order. Let's look at an example: if we want to upper case *and* concat a string, we could do it like this:

```
> const stringCombo = "This is a string".concat("!!!", " ",
"I like strings!");
> stringCombo;
"This is a string!!! I like strings!"
> stringCombo.toUpperCase();
"THIS IS A STRING!!! I LIKE STRINGS!"
```

Or we could chain the methods together and do it like this:

```
> "This is a string".concat("!!!", " ", "I like strings!").
toUpperCase();
"THIS IS A STRING!!! I LIKE STRINGS!"
```

Note that the `concat` method will be called before the `toUpperCase` method, so the order in which you chain methods matters! Try out the above code in your browser DevTools console.

Chaining methods is common and pretty cool — however, it's not necessary. If you don't feel comfortable chaining methods now, that is totally okay and you don't have to do it. You will get more practice

in this section and the coming weeks.

Summary

In this lesson, we've covered what methods are along with key concepts. Here's a summary:

- Methods are a type of function, which perform a set of procedures.
- Methods are **called *on*** something and always belong to a specific data type.
- When we **call a method** we need to include **parens**.
- JavaScript has a bunch of **built-in methods**. We can also write custom methods, but we won't learn about that until a later section.
- Some methods take one or more **arguments** that provide the method with additional information to help it perform its action.
- We can call methods on variables and optionally **chain** methods.

In the next lesson, we'll introduce functions. This will help us better understand what methods are, and the difference between methods and functions.

Previous (/introduction-to-programming/javascript-and-web-browsers/practice-variables-and-strings)
Next (/introduction-to-programming/javascript-and-web-browsers/functions)

Lesson 12 of 75
Last updated March 24, 2023

(http://www.epicodus.com)