Lesson Monday

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

Text Cheat sheet

Previously we learn that information about our web browser window is accessed through the `window` object. Let's review:

- The `window` object represents a browser window or tab and every browser automatically creates a new `window` object every time we open a new window or tab.
- The `window` object represents our current browsing context and lets us access it with JavaScript. So, if we navigate to a new tab, there will be a separate `window` object for that tab.
- Just like with JavaScript objects, the `window` object has properties that describe what a browser `window` is and does.

In this lesson, we'll learn how to access `window` properties and put them to use in our code! As you read through this lesson, try out the example code in the DevTools console.

Properties of the `window` Object

We can find the height and width of our browser window in pixels by accessing `window` properties. Try this in your DevTools console:

```
> window.innerHeight;
655
> window.innerWidth;
1123
```

`innerHeight` and `innerWidth` are both properties of the `window` object and they return numbers. The numbers that are returned will be different for you depending on what size your browser window current is. Notice how property names just like variables names are written in lower camelCase.

We are accessing these properties by using **dot notation**, which follows this syntax:

```
// This is not real code!
// This is pseudocode to explain the syntax of dot notatio
n.
object.property
```

Where `object` is the name of the object and `property` is the name of the property.

Note that `innerHeight` and `innerWidth` don't describe the height and width of the entire web browser, just the height and width of the webpage we are on. If we want to include the entire browser application with the browser toolbars and scroll bars, then we'd need to use the `outerHeight` and `outerWidth` properties. Try it!

We Can Use JavaScript Methods on `window` Properties

Even though `window` is not a part of the JavaScript programming language, it is still completely accessible to any JavaScript code. For example, `window` property values are all data types that JavaScript can recognize and interact with. Let's say that we wanted to retrieve our browser window's inner height and use it in a string. We can do that like this:

```
> const height = window.innerHeight;
> height;
655
> const stringHeight = height.toString();
> stringHeight;
"655"
> "The inner height of your browser window is " + stringHei
ght + ".";
"The inner height of your browser window is 655."
```

Or we could simplify the above code by accessing the property and chaining the method call on the end:

```
> "The inner height of your browser window is " + window.in
nerHeight.toString() + ".";
"The inner height of your browser window is 655."
```

In this case, because `window.innerHeight` returns a string, we can call JavaScript string methods on it.

Nested Objects in `window`

We can also learn about the website we are currently on by accessing the `window.location` property.

```
> window.location;
Location {ancestorOrigins: DOMStringList, href: 'https://ww
w.learnhowtoprogram.com/introduction-to-…/getting-started-a
t-epicodus/learn-how-to-program', origin: 'https://www.lear
nhowtoprogram.com', protocol: 'https:', host: 'www.learnhow
toprogram.com', …}
```

Woah! That's a lot of information. What's happening here is that the `location` property of the `window` object is set to the value of another object, itself with lots of properties. This is called a **nested object**: `window` is an object and `location` is also an object that is nested inside of the `window` object.

We can expand this object in the DevTools console to look at all of the information by clicking the triangle symbol to the left of the object, like in the image below.

```
Elements  Console  Sources  »  3
top ▼  Filter  Default levels ▼
3 Issues: 3
> window.location
<· Location {ancestorOrigins: DOMStringList, href: 'https://www.l
earnhowtoprogram.com/introduction-to-…/getting-started-at-epic
▼odus/learn-how-to-program', origin: 'https://www.learnhowtopro
gram.com', protocol: 'https:', host: 'www.learnhowtoprogram.co
m', …}
  ▶ancestorOrigins: DOMStringList {length: 0}
  ▶assign: ƒ assign()
   hash: ""
   host: "www.learnhowtoprogram.com"
   hostname: "www.learnhowtoprogram.com"
   href: "https://www.learnhowtoprogram.com/introduction-to-prog
   origin: "https://www.learnhowtoprogram.com"
   pathname: "/introduction-to-programming/getting-started-at-ep
   port: ""
   protocol: "https:"
  ▶reload: ƒ reload()
  ▶replace: ƒ replace()
   search: ""
  ▶toString: ƒ toString()
  ▶valueOf: ƒ valueOf()
   Symbol(Symbol.toPrimitive): undefined
  ▶[[Prototype]]: Location
```

As we can see, `window.location` holds information about the webpage we are on, and some properties like `host` or `href` have string values, and some properties have methods as values like `reload: ƒ reload()`.

Let's try accessing the `window.location` properties we just listed: `host`, `href`, and `reload`. Since we're accessing a property that belongs to an object which itself is a property that belongs to an object, we need to combine a series of dot notation to move through the objects to get to the property we are looking for. When we do this, we always start with the parent object, in this case `window`.

```
> window.location.host      // the host corresponds to the
domain name of the site
'www.learnhowtoprogram.com'
> window.location.href      // the href is the full url of
the webpage you are on
'https://www.learnhowtoprogram.com/tracks'
```

Note that in the example above, the responses I get are based off of the current webpage I was on, which was `learnhowtoprogram.com/tracks`, so your response for `window.location.href` will be different.

To summarize, the syntax to access a property in a nested object looks like this:

```
// This is not real code!
// This is pseudocode to demonstrate dot notation.
parentObject.childObject.targetProperty
```

Where the `childObject` is a property of the `parentObject`, and `targetProperty` is the data that we're seeking.

Now let's try out this `window.location` method:

```
> window.location.reload()
```

This command will reload the page! It does the exact same thing as hitting the your button's refresh/reload button.

Let's try another `window` method called `open()`. Can you guess what this does? Try this out in your console:

```
> window.open();
```

You should see a new tab open in your browser. If you navigate back to your DevTools console, you'll also see a return value, which is the `window` object of the new window that you opened. As we can see in the code snippet below, when an object is not expanded in the console, the console lists its first few properties on 1 – 2 lines, followed by an ellipses `...` :

```
> window.open()
Window {window: Window, self: Window, document: document, n
ame: '', location: Location, …}
```

Now try entering this into your DevTools console:

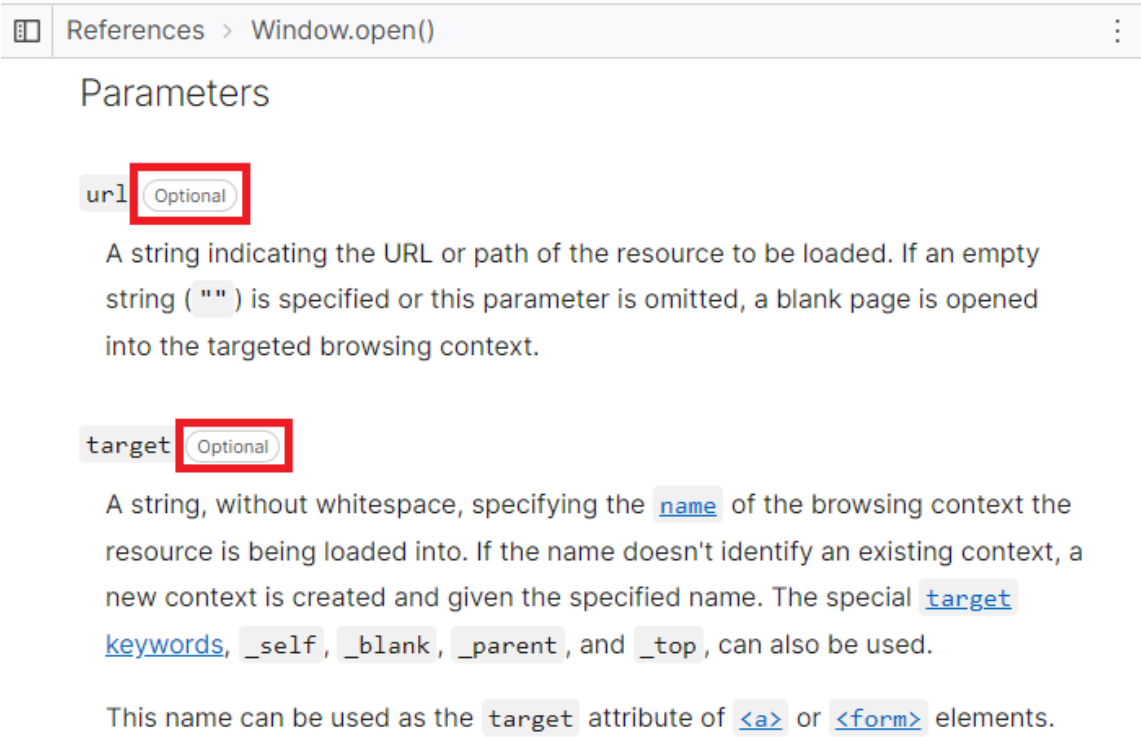
```
> window.open("https://www.learnhowtoprogram.com/tracks")
```

Now with the argument `"https://www.learnhowtoprogram.com/tracks"` included in the method call, a new tab is opened to learnhowtoprogram.com/tracks.

Working with Optional Parameters

The two ways to call `window.open()` is showing us how methods (both in JavaScript and in web browser tools) can have optional parameters. When we are able to optionally include a different number of arguments when we call a method, this means that the method is defined as having optional parameters. MDN documentation does a great job at tracking optional parameters, so if you are ever unsure if a parameter is optional, start by reviewing the docs.

Using `window.open()` as an example, see the section called "Parameters" on the MDN reference page for `window.open()` (https://developer.mozilla.org/en-US/docs/Web/API/Window/open#parameters). This is also pictured in the image below. Here, we can see two parameters, `url` and `target`, are listed as "Optional". (There are also other optional parameters that are not pictured.)



We won't worry about learning about how to write custom functions with optional parameters, but it is helpful to know that they exist and how to spot them. In the next lesson, we'll take time to review more aspects about MDN documentation for `window`, `document`, and events.

Summary

As we can see the `window` object lets us access information about the browsing session and interact with it, like reloading the webpage.

The `window` object has properties with different values: primitives, objects, and methods! We can explore these in the DevTools console, and we can use JavaScript to manipulate that data, just like we did with `window.innerHeight.toString()`.

Object properties, like variable and function names, are written in lower camelCase. We use dot notation to access `window` properties, like `window.innerHeight` or `window.open()`.

When a `window` property itself is an object with properties, we can chain dot notation access those properties. A good example of this is `window.location.href`, where we're accessing the `window` object's property called `location`, and then we're accessing the `location` object's property called `href`.

Previous (/introduction-to-programming/javascript-and-web-browsers/how-web-browsers-work)
Next (/introduction-to-programming/javascript-and-web-browsers/web-apis)

Lesson 29 of 75
Last updated more than 3 months ago.

disable dark mode

(http://www.epicodus.com)