CONTENTS

RELATED

CodeIgniter: Getting Started With a Simple Example

View

How To Install Express, a Node.js Framework, and Set Up Socket.io on a VPS

View

Tutorial Series: Understanding the DOM — Document Object Model

6/8 How To Modify Attributes, C...

7/8 Understanding Events in Jav...

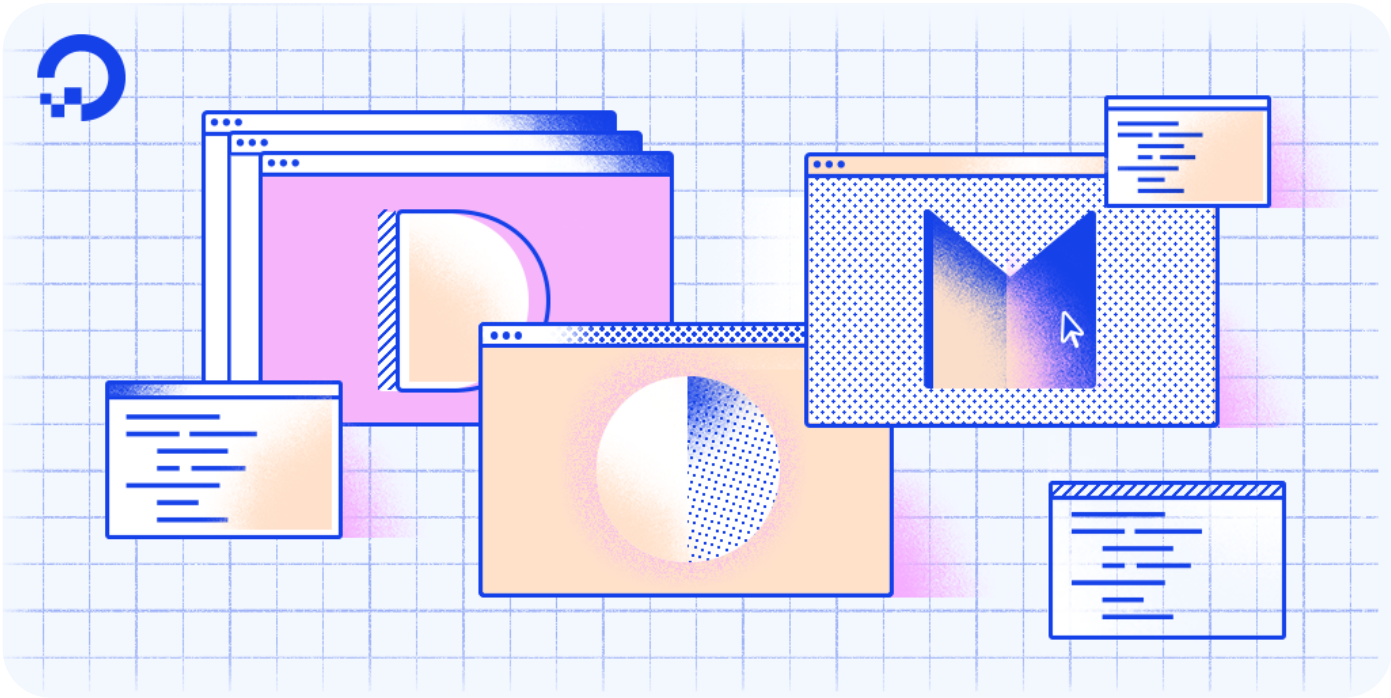
How To Modify Attributes, Classes, and Styles in the DOM

Published on May 17, 2018 · Updated on July 5, 2022

JavaScript Development

By Tania Rascia

English



Introduction

In the previous tutorial in this series, "How To Make Changes to the DOM," we covered how to create, insert, replace, and remove elements from the Document Object Model (DOM) with built-in methods. By increasing your proficiency in manipulating the DOM, you are better able to utilize JavaScript's interactive capabilities and modify web elements.

In this tutorial, you will learn how to further alter the DOM by modifying styles, classes, and other attributes of HTML element nodes. This will give you a greater understanding of how to manipulate essential elements within the DOM.

Until recently, a popular JavaScript library called jQuery was most often used to select and modify elements in the DOM. jQuery simplified the process of selecting one or more elements and applying changes to all of them at the same time. In "How To Access Elements in the DOM," we reviewed the DOM methods for grabbing and working with nodes in vanilla JavaScript.

To review, `document.querySelector()` and `document.getElementById()` are the methods that are used to access a single element. Using a `div` with an `id` attribute in the example below, we can access that element either way. The `querySelector()` method is more robust in that it can select an element on the page by any type of selector.

Given the HTML:

```
<div id="demo-id">Demo ID</div>
```

Copy

We can access the element using the `querySelector()` method as follows:

```
// Both methods will return a single element
const demoId = document.querySelector('#demo-id');
```

Copy

Accessing a single element, we can easily update a part of the element such as the text inside.

```
// Change the text of one element
demoId.textContent = 'Demo ID text updated.';
```

Copy

However, when accessing multiple elements by a common selector, such as a specific class, you have to loop through all the elements in the list. In the code below, there are two `div` elements with a common class value.

```
<div class="demo-class">Demo Class 1</div>
<div class="demo-class">Demo Class 2</div>
```

Copy

You can use `querySelectorAll()` to grab all elements with `demo-class` applied to them, and `forEach()` to loop through them and apply a change. It is also possible to access a specific element with `querySelectorAll()` the same way you would with an array, by

```
// Change the text of multiple elements with a loop
demoClasses.forEach(element => {
  element.textContent = 'All demo classes updated.';
});

// Access the first element in the NodeList
demoClasses[0];
```

Note: The methods `getElementsByClassName()` and `getElementsByTagName()` will return HTML collections which do not have access to the `forEach()` method that `querySelectorAll()` has. In these cases, you will need to use a standard `for` loop to iterate through the collection.

This is one of the most important differences to be aware of when progressing from jQuery to vanilla JavaScript. It is important to note process of applying those methods and properties to multiple elements.

This tutorial covers the properties and methods used to modify the **attributes** of the elements. These properties and methods will often be attached to event listeners in order to respond to clicks, hovers, or other triggers.

Modifying Attributes

Attributes are values that contain additional information about HTML elements. They usually come in **name/value** pairs, and may be essential depending on the element.

Some of the most common HTML attributes are the `src` attribute of an `img` tag, the `href` of an `a` tag, and the `class`, `id`, and `style` attributes. For a full list of HTML attributes, view the attribute list on the Mozilla Developer Network. Custom elements that are not part of the HTML standard will be prepended with `data-` or `aria-`.

In JavaScript, we have four methods for modifying element attributes:

Method	Description	Example

Method	Description	Example
`getAttribute()`	Returns the value of a specified attribute or `null`	`element.getAttribute('href');`
`setAttribute()`	Adds or updates value of a specified attribute	`element.setAttribute('href', 'index.html');`
`removeAttribute()`	Removes an attribute from an element	`element.removeAttribute('href');`

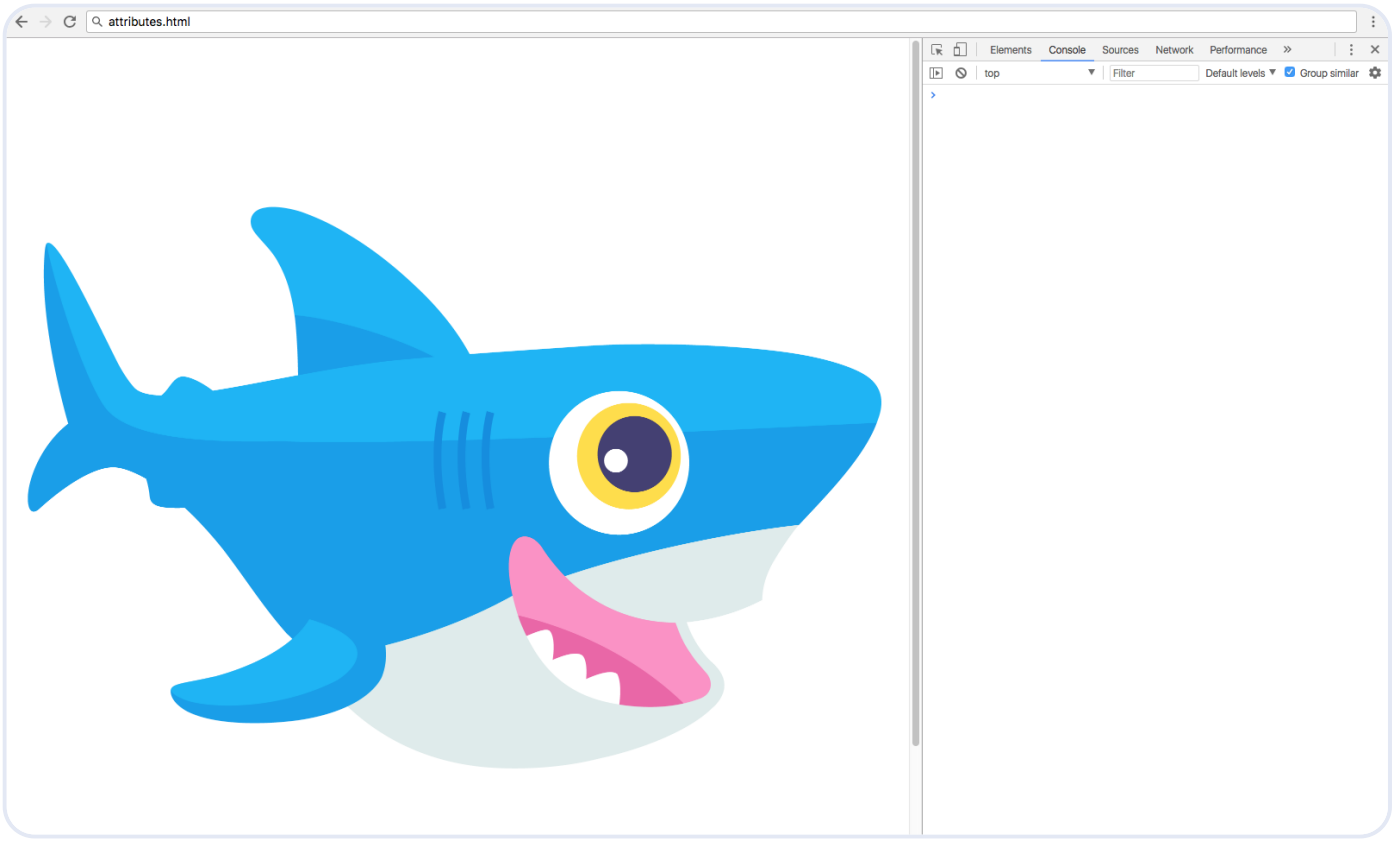
Let's create a new HTML file with an `img` tag with one attribute. We'll link to a public image available via a URL, but you can swap it out for an alternate local image if you're working offline.

attributes.html

Copy

```
<!DOCTYPE html>
<html lang="en">
<body>

        <img src="https://js-tutorials.nyc3.digitaloceanspaces.com/shark.png">

</body>

</html>
```

When you load the above HTML file into a modern web browser and open the built-in Developer Console, you should see something like this:



Now, you can test all the attribute methods on the fly.

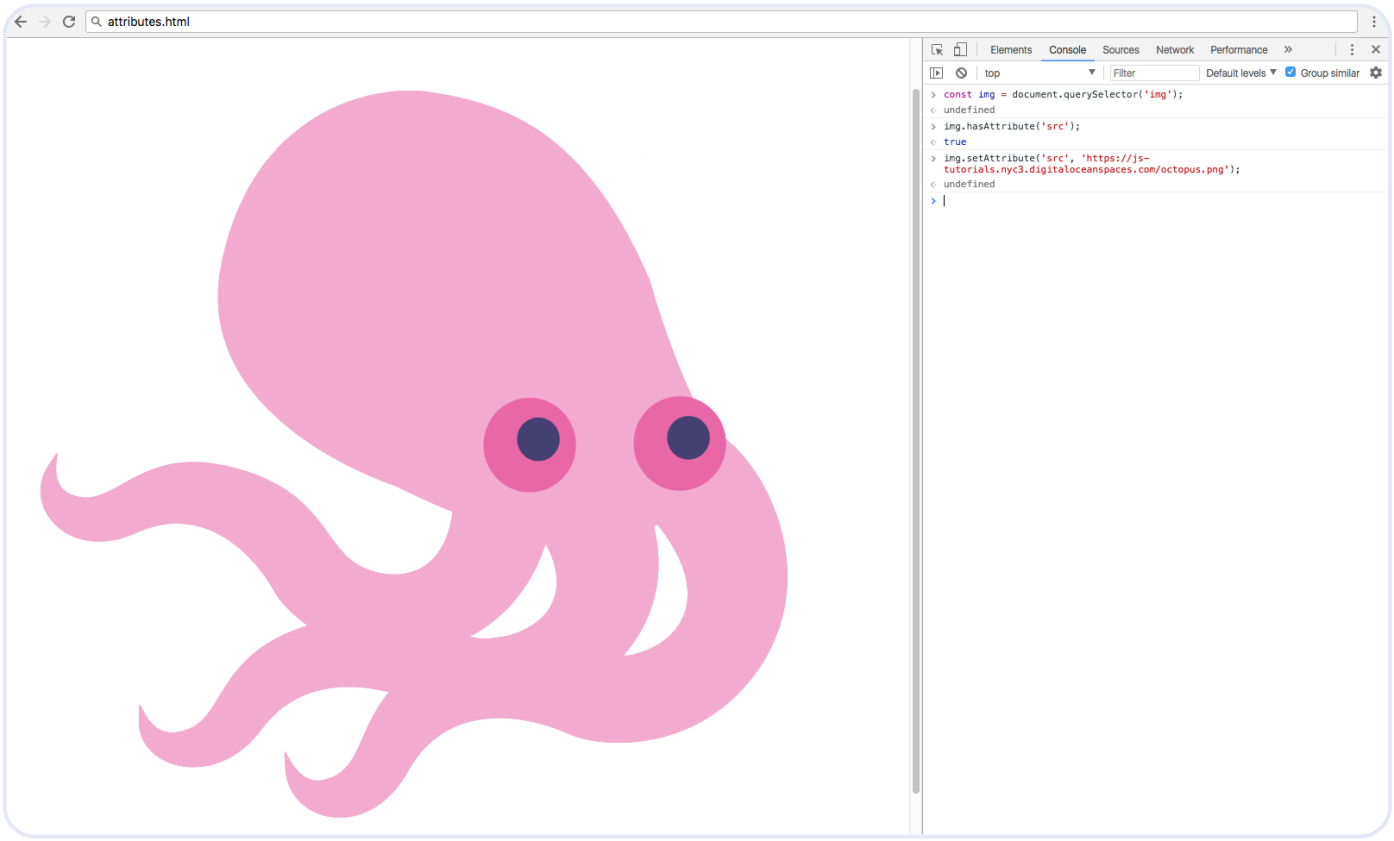
```
// Assign image element
const img = document.querySelector('img');

img.hasAttribute('src');              // returns true
img.getAttribute('src');              // returns "...shark.png"
img.removeAttribute('src');           // remove the src attribute and value
```

Copy

At this point, you will have removed the `src` attribute and value associated with `img`, but you can reset that attribute and assign the value to an alternate image with `img.setAttribute()`:

```
img.setAttribute('src', 'https://js-tutorials.nyc3.digitaloceanspaces.com/
```

Copy



Finally, we can modify the attribute directly by assigning a new value to the attribute as a property of the element, setting the `src` back to the `shark.png` file

```
img.src = 'https://js-tutorials.nyc3.digitaloceanspaces.com/shark.png';
```
Copy

Any attribute can be edited this way as well as with the above methods.

The `hasAttribute()` and `getAttribute()` methods are usually used with conditional statements, and the `setAttribute()` and `removeAttribute()` methods are used to directly modify the DOM.

Modifying Classes

The class attribute corresponds to CSS class selectors. This is not to be confused with ES6 classes, a special type of JavaScript function.

CSS classes are used to apply styles to multiple elements, unlike IDs which can only exist

Method/Property	Description	Example
`className`	Gets or sets class value	`element.className;`
`classList.add()`	Adds one or more class values	`element.classList.add('active');`
`classList.toggle()`	Toggles a class on or off	`element.classList.toggle('active');`
`classList.contains()`	Checks if class value exists	`element.classList.contains('active');`
`classList.replace()`	Replace an existing class value with a new class value	`element.classList.replace('old', 'new');`
`classList.remove()`	Remove a class value	`element.classList.remove('active');`

Make another HTML file to work with the class methods containing two elements and a few classes. Additionally, include an inline CSS stylesheet to provide some styles to help us see the results of our work.

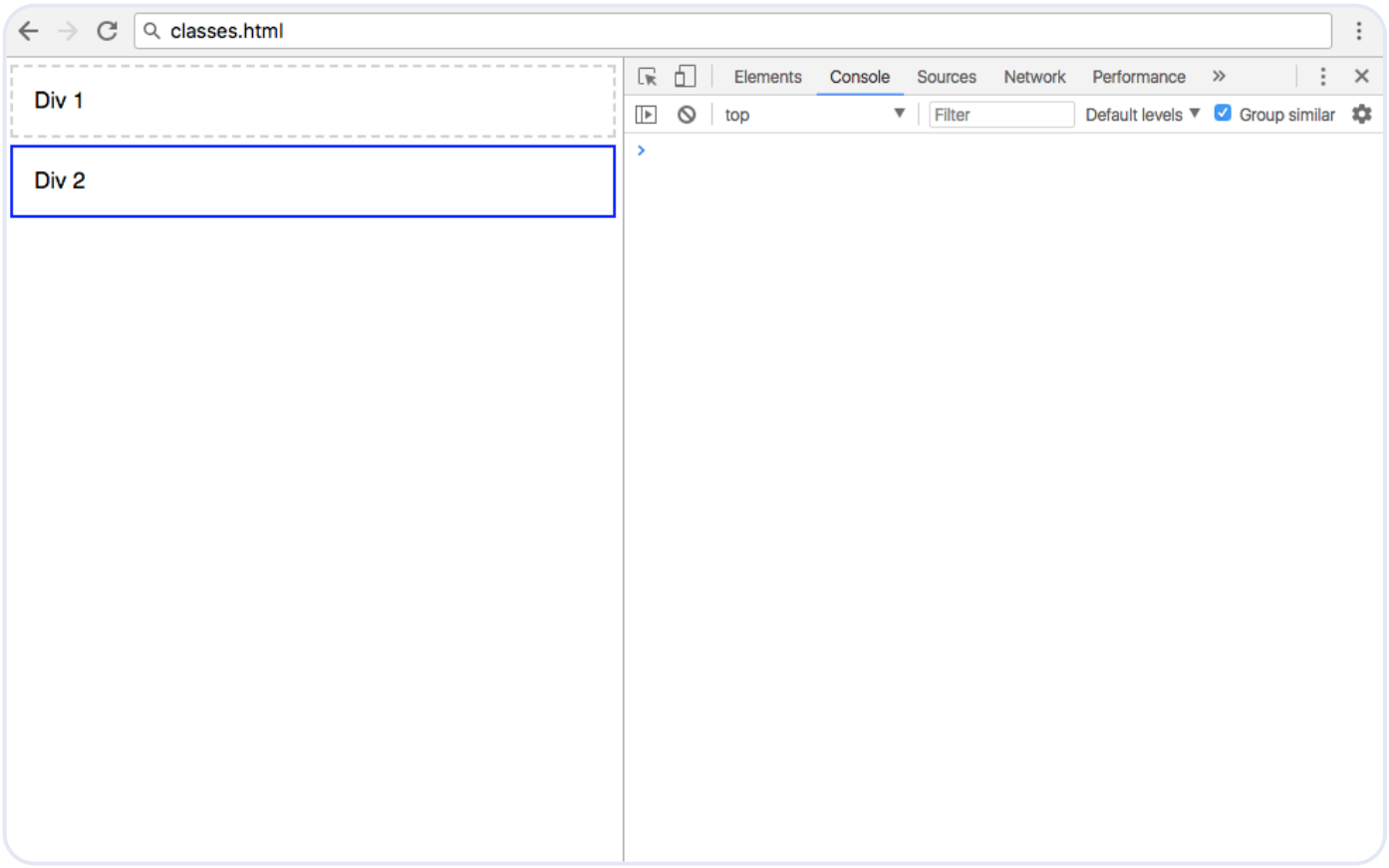
classes.html

Copy

```
<!DOCTYPE html>
<html lang="en">

<style>
```

```
                border: 2px solid blue;
        }

        .warning {
                border: 2px solid red;
        }

        .hidden {
                display: none;
        }

        div {
                border: 2px dashed lightgray;
                padding: 15px;
                margin: 5px;
        }
</style>

<body>

        <div>Div 1</div>
        <div class="active">Div 2</div>

</body>

</html>
```

When you open the `classes.html` file into a web browser, you should receive a rendering that looks similar to the following:

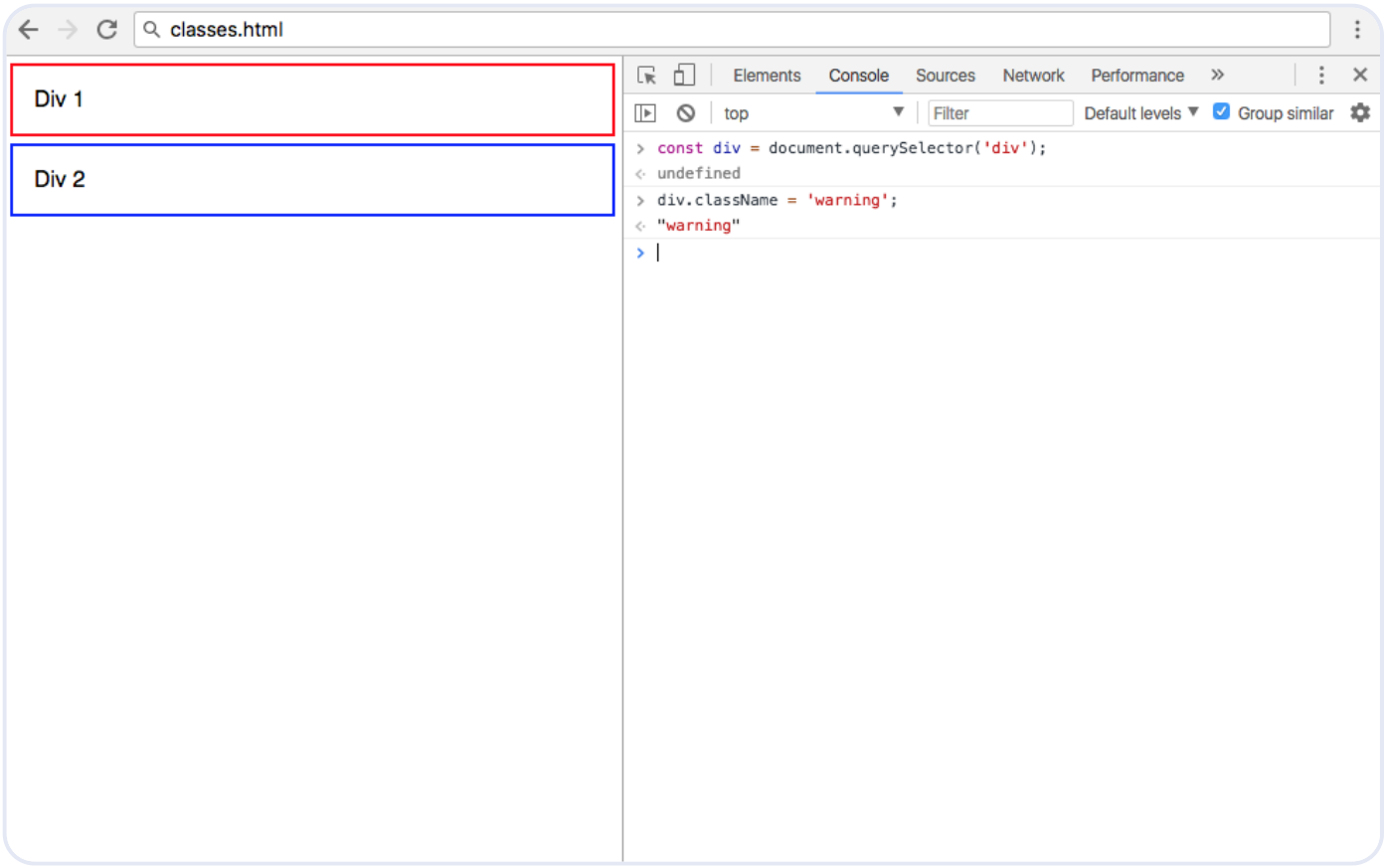


The `className` property was introduced to prevent conflicts with the `class` keyword found in JavaScript and other languages that have access to the DOM. You can use `className` to assign a value directly to the class.

```
// Select the first div
const div = document.querySelector('div');

// Assign the warning class to the first div
div.className = 'warning';
```

Copy

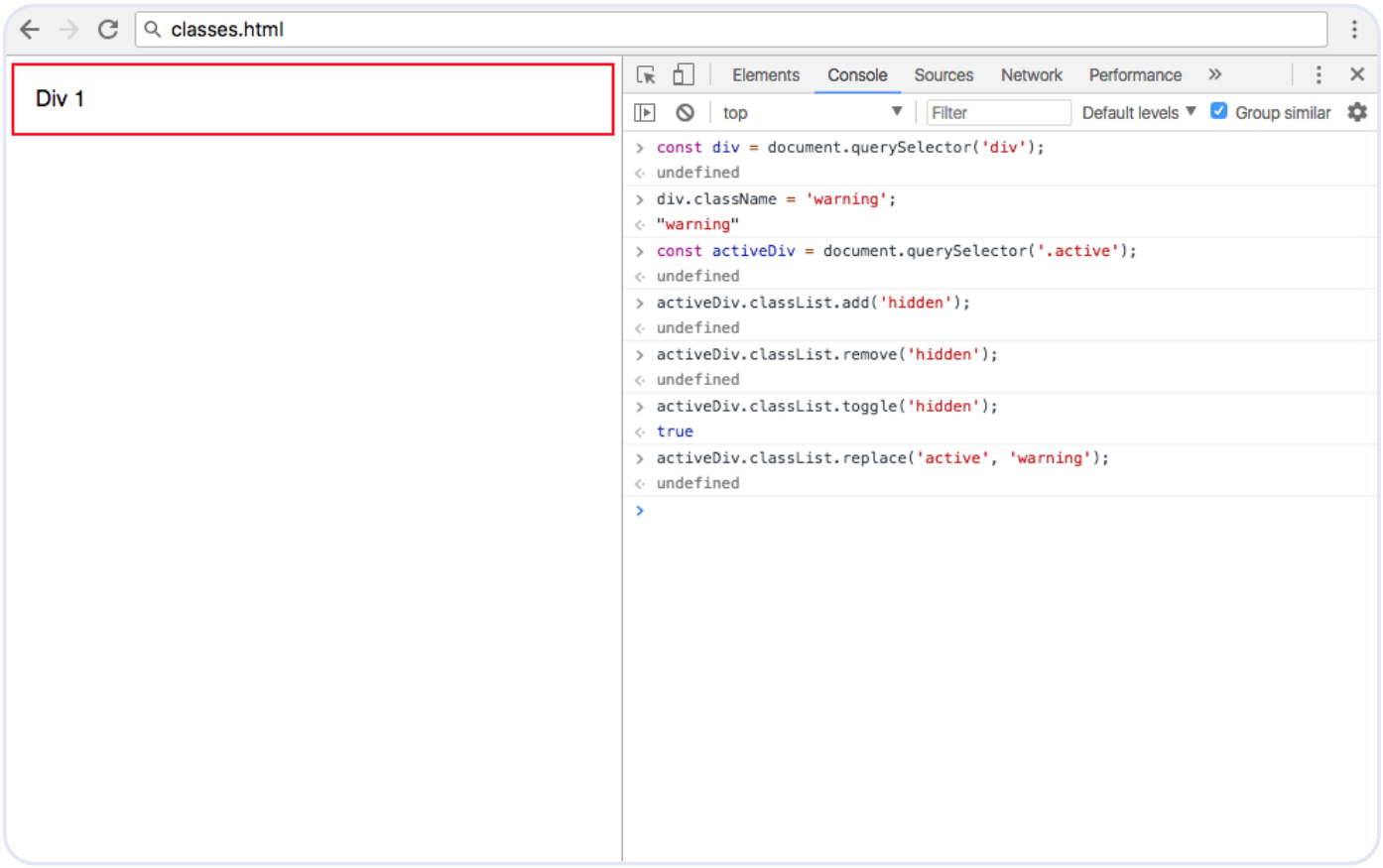
By assigning the `warning` class defined in the CSS values of `classes.html` to the first `div`, you'll receive the following output:



Note: If any classes already exist on the element, this will override them. You can add multiple space delimited classes using the `className` property, or use it without assignment operators to get the current value of the class on the element.

The other way to modify classes is via the `classList` property, which comes with a few helpful methods. These methods are similar to the jQuery `addClass`, `removeClass`, and `toggleClass` methods.

```
// Select the second div by class name                                    Copy
const activeDiv = document.querySelector('.active');

activeDiv.classList.add('hidden');                  // Add the hidden class
activeDiv.classList.remove('hidden');               // Remove the hidden class
activeDiv.classList.toggle('hidden');               // Switch between hidden true a
activeDiv.classList.replace('active', 'warning'); // Replace active class with wa
```



Because the `activeDiv` element still has the class `hidden` applied to it, it is not displayed on the page.

Unlike in the `className` example, using `classList.add()` will add a new class to the list of existing classes. You can also add multiple classes as comma-separated strings. It is also possible to use `setAttribute` to modify the class of an element.

Modifying Styles

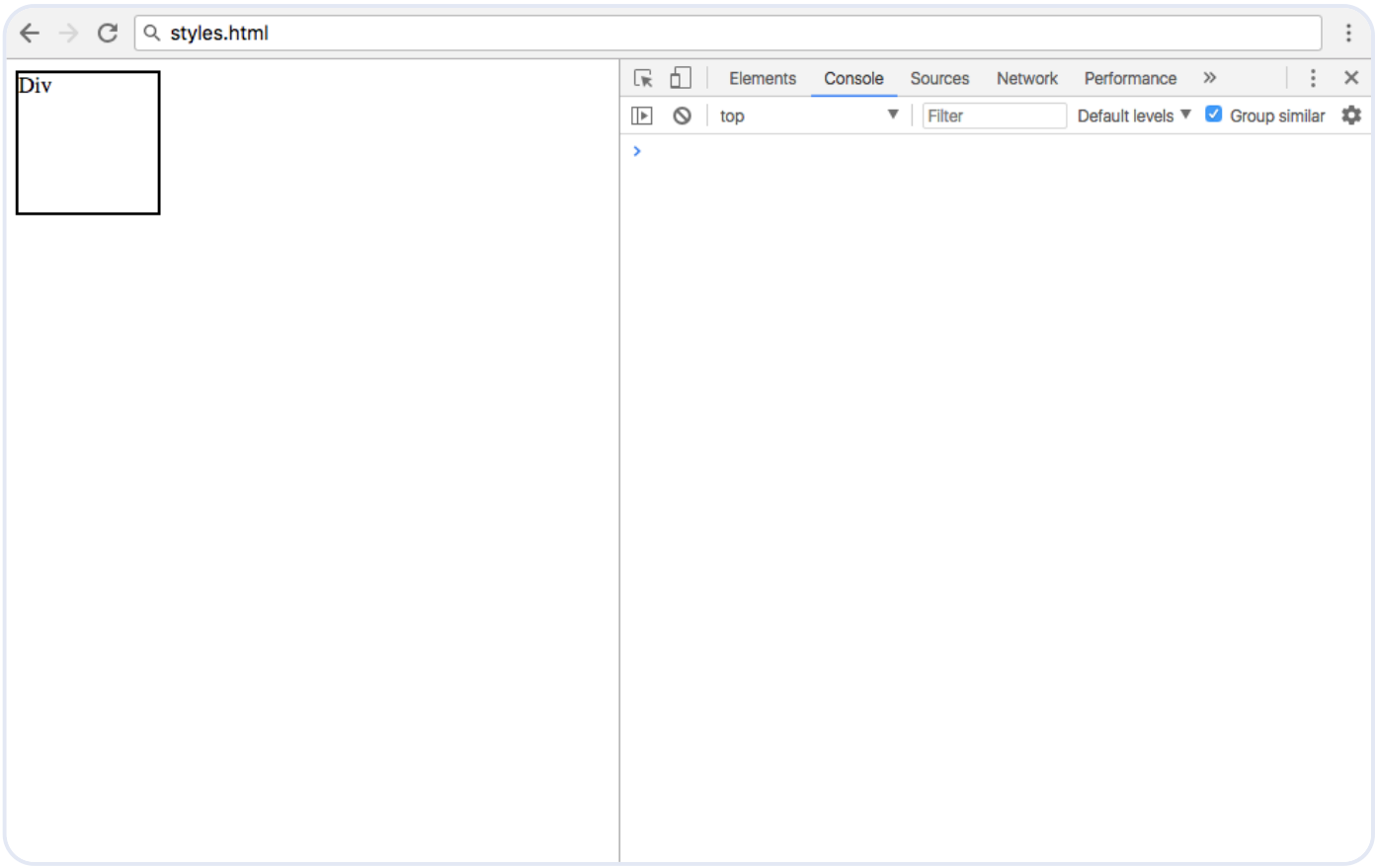
The style property represents the inline styles on an HTML element. Often, styles will be applied to elements via a stylesheet as was done previously in this article, but sometimes you will have to add or edit an inline style directly.

Make a new file to demonstrate editing styles with JavaScript. Use the contents below containing a `div` that has some inline styles applied to display a square.

styles.html

```
        <div style="height: 100px;
                    width: 100px;
                    border: 2px solid black;">Div</div>

</body>

</html>
```

When opened in a web browser, the `styles.html` will look something like this:



One option to edit the styles is with `setAttribute()`.

```
// Select div
const div = document.querySelector('div');

// Apply style to div
div.setAttribute('style', 'text-align: center');
```

Copy

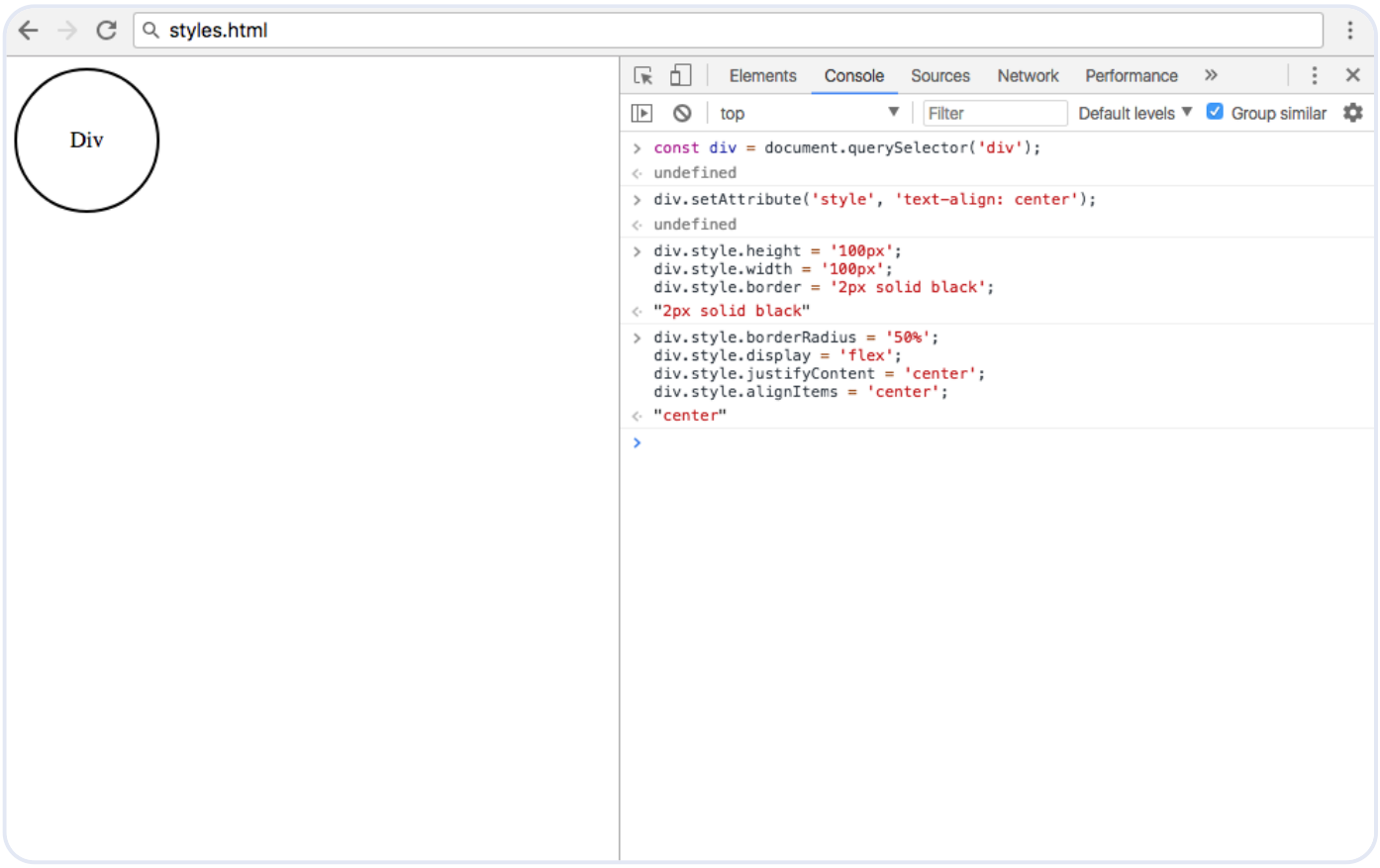
```
div.style.border = '2px solid black';
```

CSS properties are written in kebab-case, which is lowercase words separated by dashes. However, kebab-case CSS properties cannot be used on the JavaScript style property, as the dash `-` is used for subtraction. Instead, they will be replaced with their camelCase equivalent, which is when the first word is lowercase, and all subsequent words are capitalized. In other words, instead of `text-align` use `textAlign` for the JavaScript style property.

```
// Make div into a circle and vertically center the text
div.style.borderRadius = '50%';
div.style.display = 'flex';
div.style.justifyContent = 'center';
div.style.alignItems = 'center';
```

Copy

After completing the above style modifications, your final rendering of `styles.html` will show a circle:



Conclusion

HTML elements often have additional information assigned to them in the form of attributes. Attributes may consist of name/value pairs, and a few of the most common attributes are `class` and `style`.

In this tutorial, you learned how to access, modify, and remove attributes on an HTML element in the DOM using plain JavaScript. You also learned how to add, remove, toggle, and replace CSS classes on an element, and how to edit inline CSS styles. For additional reading, check out the documentation on attributes on the Mozilla Developer Network.

Next in series: Understanding Events in JavaScript →

the content, structure, and style of a website. JavaScript is the client-side scripting language that connects to the DOM in an internet browser.

Subscribe

JavaScript Development

Browse Series: 8 articles

1/8 Introduction to the DOM

2/8 Understanding the DOM Tree and Nodes

3/8 How To Access Elements in the DOM

Expand to view all

About the authors

Tania Rascia Author

Lisa Tagliaferri Editor

Madison Scott-Clary Editor

Tech Writer at DigitalOcean

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

4 Comments

Leave a comment...

This textbox defaults to using **Markdown** to format your answer.

You can type **!ref** in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

Victor • August 22, 2019

Thanks for this Tutorial: I am hoping you can tell me why this statement doesn't work ? intoCount[0].innerHtml = copyFromVar; [code] <!DOCTYPE html> <html>

<button onclick=“myFunction()”>Copy text From field1 to field2</button> <button onclick=“saveItAll()”>Copy text From textarea to contentEditable-div</button>

 <textarea class=“copyFrom” rows=“4” cols=“20”>TextArea1 </textarea>
 <textarea class=“copyFrom” rows=“4” cols=“20” >TextArea2 </textarea>
 pasteInto 1 pasteInto 2

A function is triggered when the button is clicked. The function copies the text from Field1 into Field2. <script> var copyFromVar = “”; var fromCount; // = document.getElementsByClassName(“copyFrom”); var intoCount; // = document.getElementsByClassName(“pasteInto”); function saveItAll() { fromCount = document.getElementsByClassName(“copyFrom”); intoCount = document.getElementsByClassName(“pasteInto”);

copyFromVar = fromCount[0].value ; intoCount[0].innerHtml = copyFromVar; alert("0 copyFromVar = " + copyFromVar);

copyFromVar = fromCount[1].value ; intoCount[1].innerHtml = copyFromVar; alert("1 copyFromVar = " + copyFromVar);

} </script> <script> function myFunction() { document.getElementById(“field2”).value = document.getElementById(“field1”).value; } </script>

</body> </html>

[/code] Thanks

Reply

richardbushell • March 29, 2019

Great Article, very useful...

Ideally, you should also include:- removeProperty

orangetreasure • December 1, 2021

This comment has been deleted

Reply

leoJ • January 2, 2021

The shark and octopus image links appear to be broken.

Reply

Popular Topics

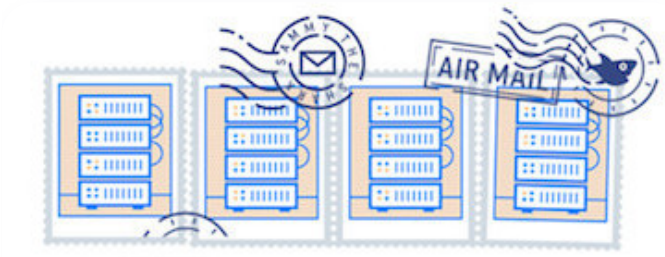
MySQL

Docker

Kubernetes

All tutorials →

Free Managed Hosting →



Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

Sign up →

Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →