## **Dom Manipulation**

### **Introduction**

One of the most unique and useful abilities of JavaScript is its ability to manipulate the DOM. But what *is* the DOM, and how do we go about changing it? Let’s jump right in…

### **Learning Outcomes**

By the end of this lesson, you should be able to do the following:

* Explain what the DOM is in relation to a webpage.
* Explain the difference between a “node” and an “element”.
* Explain how to target nodes with “selectors”.
* Explain the basic methods for finding/adding/removing and altering DOM nodes.
* Explain the difference between a “nodelist” and an “array of nodes”.
* Explain what “bubbling” is and how it works.

### **DOM - Document Object Model**

The DOM (or Document Object Model) is a tree-like representation of the contents of a webpage - a tree of “nodes” with different relationships depending on how they’re arranged in the HTML document.



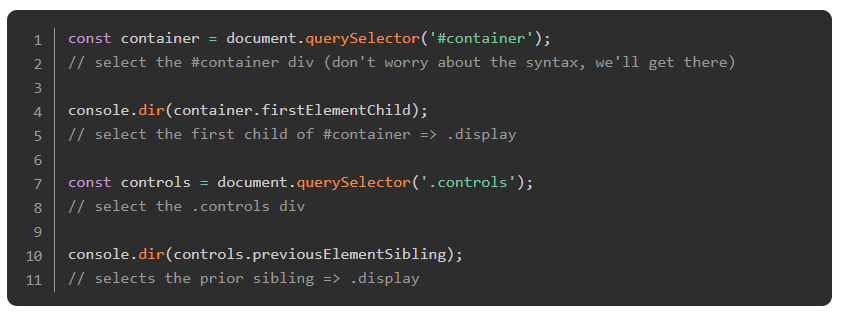
In the above example, the <div class="display"></div> is a “child” of <div id="container"></div> and a sibling to <div class="controls"></div>. Think of it like a family tree. <div id="container"></div> is a parent, with its children on the next level, each on their own “branch”.

### **Targeting Nodes with Selectors**

When working with the DOM, you use “selectors” to target the nodes you want to work with. You can use a combination of CSS-style selectors and relationship properties to target the nodes you want. Let’s start with CSS-style selectors. In the above example, you could use the following selectors to refer to <div class="display"></div>:

* div.display
* .display
* #container > .display
* div#container > div.display

You can also use relational selectors (i.e. firstElementChild or lastElementChild etc.) with special properties owned by the nodes.



So you’re identifying a certain node based on its relationships to the nodes around it.

### **DOM methods**

When your HTML code is parsed by a web browser, it is converted to the DOM as was mentioned above. One of the primary differences is that these nodes are objects that have many properties and methods attached to them. These properties and methods are the primary tools we are going to use to manipulate our webpage with JavaScript. We’ll start with the query selectors - those that help you target nodes.

#### **Query Selectors**

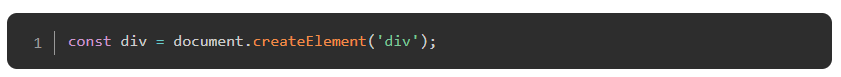
* *element*.querySelector(*selector*) returns reference to the first match of *selector*
* *element*.querySelectorAll(*selectors*) returns a “nodelist” containing references to all of the matches of the *selectors*

\**There are several other, more specific queries, that offer potential (marginal) performance benefits, but we won’t be going over them now.*

It’s important to note that when using querySelectorAll, the return value is not an array. It looks like an array, and it somewhat acts like an array, but it’s really a “nodelist”. The big distinction is that several array methods are missing from nodelists. One solution, if problems arise, is to convert the nodelist into an array. You can do this with Array.from() or the [spread operator.](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator)

#### **Element Creation**

* document.createElement(tagName, [options]) creates a new element of tag type tagName. [options] in this case means you can add some optional parameters to the function. Don’t worry about these at this point.



This function does NOT put your new element into the DOM - it simply creates it in memory. This is so that you can manipulate the element (by adding styles, classes, ids, text etc.) before placing it on the page. You can place the element into the DOM with one of the following methods.

#### **Append Elements**

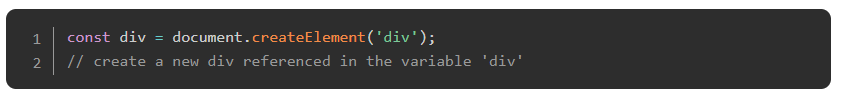
* *parentNode*.appendChild(*childNode*) appends *childNode* as the last child of *parentNode*
* *parentNode*.insertBefore(*newNode*, *referenceNode*) inserts *newNode* into *parentNode* before *referenceNode*

#### **Remove Elements**

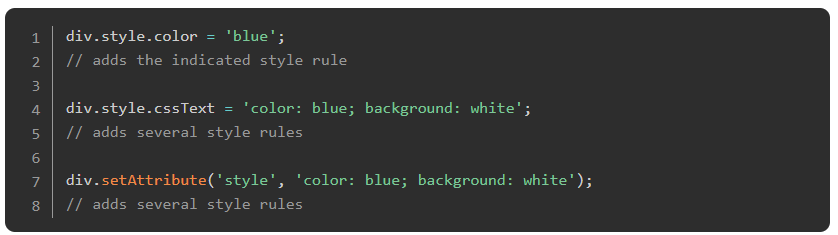
* *parentNode*.removeChild(*child*) removes *child* from *parentNode* on the DOM and returns reference to *child*

#### **Altering Elements**

When you have a reference to an element, you can use that reference to alter the element’s own properties. This allows you to do many useful alterations, like adding/removing and altering attributes, changing classes, adding inline style information and more.

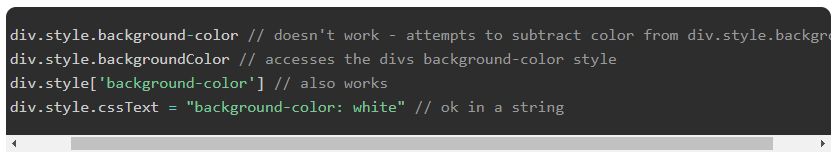


#### **Adding inline style**

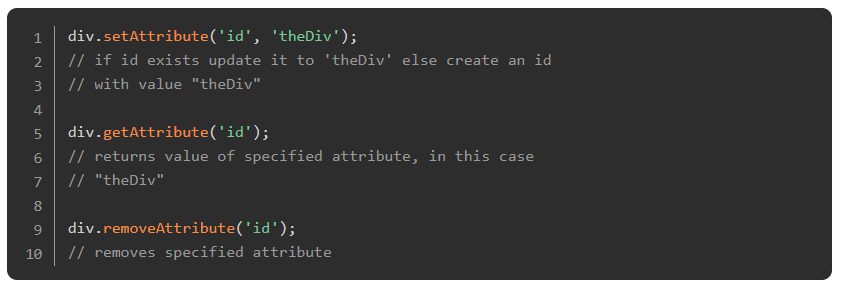


See DOM Enlightenment’s [section on CSS Style rules](http://domenlightenment.com/#6.2) for more info on inline styles.

Note that if you’re accessing a kebab-cased css rule from JS, you’ll either need to use camelcase or you’ll need to use bracket notation instead of dot notation.

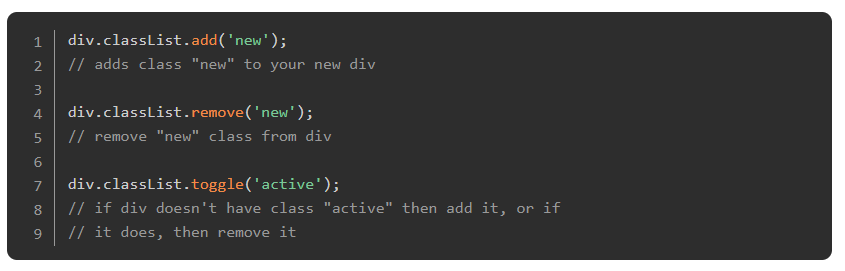


#### **Editing Attributes**



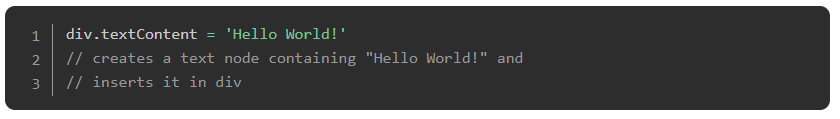
See MDNs section on [HTML Attributes](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes) for more info on available attributes.

#### **Working with classes**



It is often standard (and more clean) to toggle a CSS style rather than adding and removing inline CSS.

#### **Adding text content**

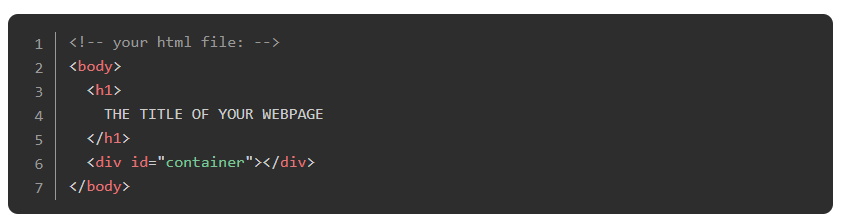


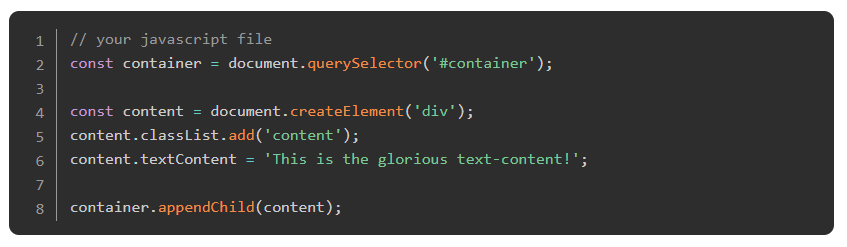
#### **Adding HTML content**



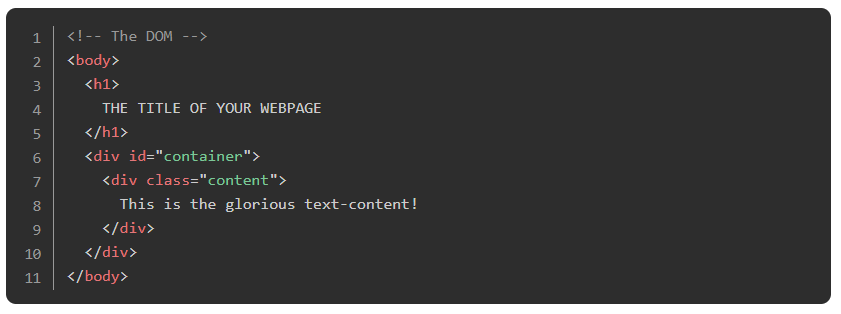
\**Note that textContent is preferable for adding text, and innerHTML should be used sparingly as it can create security risks if misused. Check out* [*this video*](https://www.youtube.com/watch?v=ns1LX6mEvyM&ab_channel=WebDevSimplified) *if you want to see an example of how.*

Let’s take a minute to review what we’ve covered and give you a chance to practice this stuff before moving on. Check out this example of creating and appending a DOM element to a webpage.





In the JavaScript file, first we get a reference to the container div that already exists in our HTML. Then we create a new div and store it in the variable content. We add a class and some text to the content div and finally append that div to container. All in all it’s a simple process. After the JavaScript code is run, our DOM tree will look like this:



Keep in mind that the JavaScript does *not* alter your HTML, but the DOM - your HTML file will look the same, but the JavaScript changes what the browser renders.

### **Important note:**

Your JavaScript, for the most part, is run whenever the JS file is run, or when the script tag is encountered in the HTML. If you are including your JavaScript at the top of your file, many of these DOM manipulation methods will not work because the JS code is being run *before* the nodes are created in the DOM. The simplest way to fix this is to include your JavaScript at the bottom of your HTML file so that it gets run after the DOM nodes are parsed and created.

Alternatively, you can link the JavaScript file in the <head> of your HTML document. Use the <script> tag with the src attribute containing the path to the JS file, and include the defer keyword to load the file *after* the HTML is parsed, as such:



Read the second bullet point in [this MDN article](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/The_head_metadata_in_HTML#applying_css_and_javascript_to_html) for more information, which also includes a link to additional script loading strategies.

### **Exercise**

Copy the example above into files on your own computer. To make it work you’ll need to supply the rest of the HTML skeleton and either link your javascript file, or put the javascript into a script tag on the page. Make sure everything is working before moving on!

Add the following elements to the container using ONLY javascript and the DOM methods shown above.

1. a <p> with red text that says “Hey I’m red!”
2. an <h3> with blue text that says “I’m a blue h3!”
3. a <div> with a black border and pink background color with the following elements inside of it:
   1. another <h1> that says “I’m in a div”
   2. a <p> that says “ME TOO!”
   3. Hint for this one: after creating the div with createElement, append the <h1> and <p> to it before adding it to the container.

### **Events**

Now that we have a handle on manipulating the DOM with JavaScript, the next step is learning how to make that happen dynamically, or on demand! Events are how you make that magic happen on your pages. Events are actions that occur on your webpage such as mouse-clicks or keypresses, and using JavaScript we can make our webpage listen and react to these events.

There are three primary ways to go about this: you can attach functions attributes directly on your HTML elements, you can set the “on\_event\_” property on the DOM object in your JavaScript, or you can attach event listeners to the nodes in your JavaScript. Event listeners are definitely the preferred method, but you will regularly see the others in use, so we’re going to cover all three.

We’re going to create 3 buttons that all alert “Hello World” when clicked. Try them all out using your own HTML file, or using something like [CodePen](https://codepen.io/).

#### **method 1**

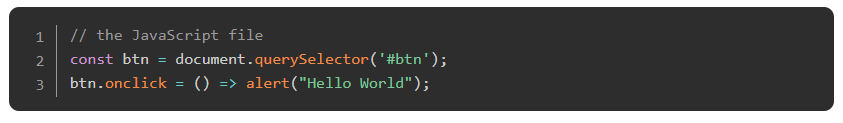


This solution is less than ideal because we’re cluttering our HTML with JavaScript. Also, we can only have 1 “onclick” event per element.

#### 

#### **method 2**



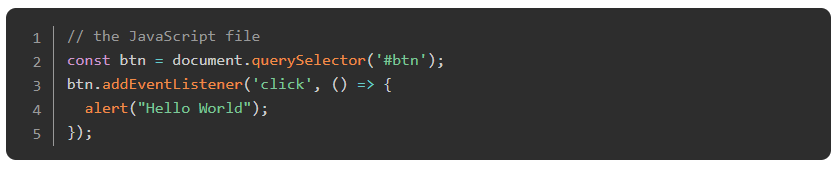


#### **(need to review arrow functions?** [**LINK**](http://javascript.info/arrow-functions-basics)**)**

This is a little better. We’ve moved the JS out of the HTML and into a JS file, but we still have the problem that a DOM element can only have 1 “onclick” property.

#### **method 3**

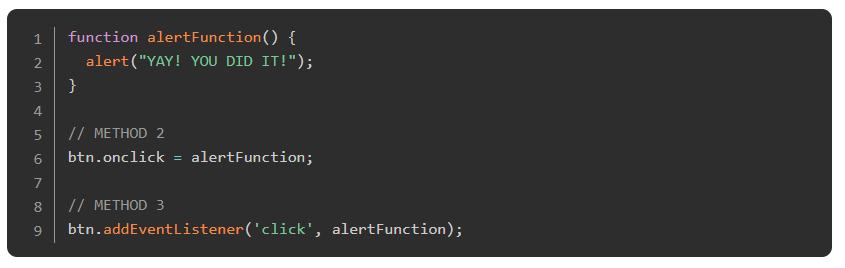




Now, we maintain separation of concerns, and we also allow multiple event listeners if the need arises. Method 3 is much more flexible and powerful, though it is a bit more complex to set up.

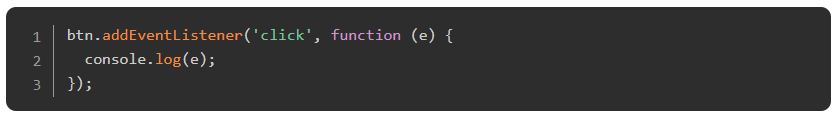
Note that all 3 of these methods can be used with named functions like so:





Using named functions can clean up your code considerably, and is a *really* good idea if the function is something that you are going to want to do in multiple places.

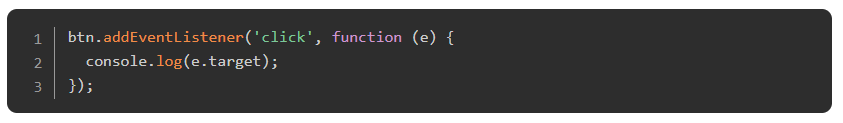
With all three methods we can access more information about the event by passing a parameter to the function that we are calling. Try this out on your own machine:



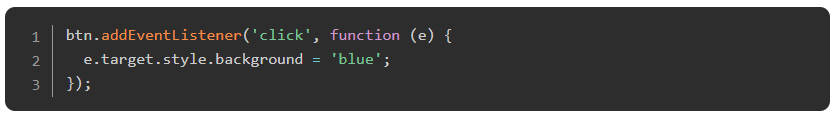
\**Note that function (e) is a callback from addEventListener. Further explanation of callbacks can be found* [*HERE*](https://briggs.dev/blog/understanding-callbacks)*.*

The e in that function is an object that references the event itself. Within that object you have access to many useful properties and functions such as which mouse button or key was pressed, or information about the event’s target - the DOM node that was clicked.

Try this:



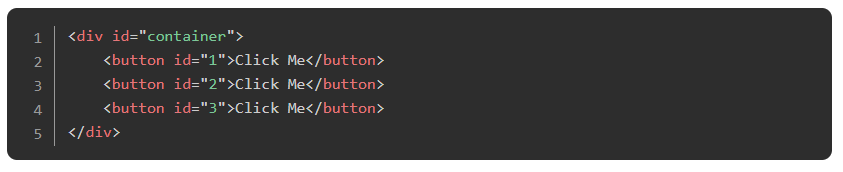
and now this:

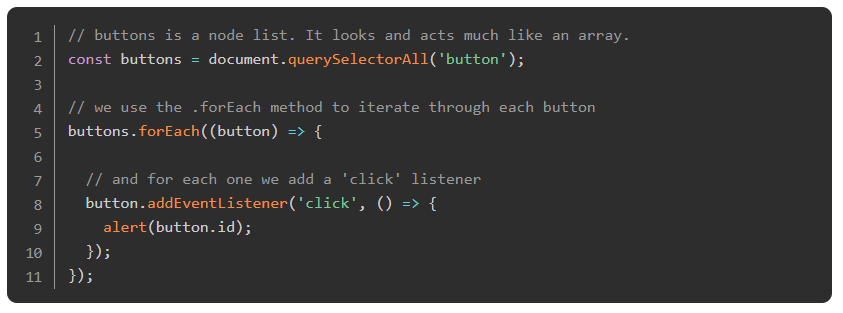


pretty cool eh?

#### **Attaching listeners to groups of nodes**

This might seem like a lot of code if you’re attaching lots of similar event listeners to many elements. There’s a few ways to go about doing that more efficiently. We learned above that we can get a nodelist of all of the items matching a specific selector with querySelectorAll('selector'). In order to add a listener to each of them we simply need to iterate through the whole list like so:





This is just the tip of the iceberg when it comes to DOM manipulation and event handling, but it’s enough to get you started with some exercises. In our examples so far we have been using the ‘click’ event exclusively, but there are *many* more available to you.

Some useful events include:

* click
* dblclick
* keypress
* keydown
* keyup

You can find a more complete list with explanations of each event on [this page](https://www.w3schools.com/jsref/dom_obj_event.asp).

### **Practice**

Manipulating web pages is the primary benefit of the JavaScript language! These techniques are things that you are likely to be messing with *every day* as a front-end developer, so let’s practice!

1. Do the first exercise in Wes Bos’s JavaScript30 program by cloning the repo at <https://github.com/wesbos/JavaScript30>. Check out the [Video Tutorial](https://www.youtube.com/watch?v=VuN8qwZoego) for instructions on the project.
2. Go back to your “Rock Paper Scissors” game from a previous lesson and give it a simple UI. Before we proceed, we need to get familiar with a very popular Git workflow. [Read this article](https://blog.scottlowe.org/2015/01/27/using-fork-branch-git-workflow/) for everything that you need to know for now.  
   a. The above article shows us how to use a “fork and branch” workflow to contribute to another user’s repo. But since we’ll be adding a new feature to our own repo, we don’t need to make a fork or add a remote since we already have it set up. We can start with the ‘Making a branch’ step.  
   b. Since we’ll be making a UI for our Rock Paper Scissors game, make a new branch and switch to it with the command git checkout -b rps-ui  
   c. You are now working in the rps-ui branch, locally. However, this branch does not exist in your remote repo yet. If you go to your github repo page, you’ll see that you only have 1 branch, which would be main. Let’s push this new branch to your remote repo with the command git push origin rps-ui. Now, you’ll see two branches in your Github repository!  
   d. You can now make changes in this branch without worrying about your old code. It will be safe and untouched in the main branch. You can think of the main branch as the ‘production’ branch, which will be the most tested, stable version of our code, and our rps-ui branch as a ‘development’ branch where you’re experimenting or trying to implement a new feature without messing up our stable production code. You can see how this process is very convenient for working on a new feature without fear.  
   e. Make sure you are on the rps-ui branch. You can check this, with the git branch command. The branch you are currently on will have an (\*)asterisk next to it. If you’re in another branch for some reason, switch to rps-ui with the command git checkout rps-ui. Now you’re all set to work on your new feature! Note: You can add files, commit to this branch, and push changes to your repo, just like you would with the main branch. Everything is the same except when you push the changes, you’d use git push origin rps-ui instead of git push origin main, since we’re pushing to our new branch.
3. In our UI, the player should be able to play the game by clicking on buttons rather than typing their answer in a prompt.  
   a. For now, remove the logic that plays exactly five rounds.  
   b. Create three buttons, one for each selection. Add an event listener to the buttons that calls your playRound function with the correct playerSelection every time a button is clicked. (you can keep the console.logs for this step)  
   c. Add a div for displaying results and change all of your console.logs into DOM methods.  
   d. Display the running score, and announce a winner of the game once one player reaches 5 points.  
   e. You will likely have to refactor (rework/rewrite) your original code to make it work for this. That’s OK! Reworking old code is an important part of the programmer’s life.  
   f. Once you’re all done with your UI and make sure everything’s satisfactory, it’s time to merge our current branch rps-ui to our main branch.
4. Now let’s take a look at how we can merge these changes back to our main branch.  
   a. [Watch this short video](https://www.youtube.com/watch?v=S2TUommS3O0) to get an idea of what we’ll be doing. You don’t need to worry about merge conflicts for now, as this will be very unlikely when you’re working on your own and not collaborating with a team of developers.  
   b. As mentioned in the video, let switch to the branch we want to merge INTO i.e. main with the command git checkout main  
   c. Now let’s merge our rps-ui branch into main, our current branch with git merge rps-ui.  
   d. If everything goes fine, our rps-ui branch is now successfully merged with main! Use git log and you’ll see all the commits you’ve made to your feature branch on top of the commits you made to the main branch. Now for our final step!  
   e. Let’s push our main branch into our remote repo by running git push origin main . Go to your Github repo and you’ll see that our main branch will have all the changes and commits you made to the rps-ui branch. Congratulations! You’ve successfully pushed your first feature into your production branch!  
   f. Now that we have all our code in the main branch, we don’t really need our rps-ui branch anymore. Let’s do some cleanup, both locally and in the remote repo. Delete the branch from our local repo with git branch -d rps-ui and also delete it from the remote repo on Github with git push --delete origin rps-ui. Congrats, we’re all done with our cleanup!
5. Watch the [Event Capture, Propagation and Bubbling video](https://www.youtube.com/watch?v=F1anRyL37lE) from Wes Bos’s JavaScript30 program. If you want to code along with the video, you can use the contents of folder #25 from the repo you cloned above.

### **Additional Resources**

This section contains helpful links to other content. It isn’t required, so consider it supplemental.

* [Eloquent JS - DOM](http://eloquentjavascript.net/13_dom.html)
* [Eloquent JS - Handling Events](http://eloquentjavascript.net/14_event.html)
* [DOM Enlightenment](http://domenlightenment.com/)
* [JavaScript30](https://javascript30.com/)
* [Plain JavaScript](https://plainjs.com/javascript/) is a reference of JavaScript code snippets and explanations involving the DOM, as well as other aspects of JS. If you’ve already learned jQuery, it will help you figure out how to do things without it.
* This [W3Schools](https://www.w3schools.com/js/js_htmldom.asp) article offers simple and easy-to-understand lessons on DOM.
* [JS DOM Crash Course](https://www.youtube.com/watch?v=0ik6X4DJKCc&list=PLillGF-RfqbYE6Ik_EuXA2iZFcE082B3s) is an extensive and well explained 4 part video series on the DOM by Traversy Media.
* [Understanding The Dom](https://www.digitalocean.com/community/tutorial_series/understanding-the-dom-document-object-model) is an aptly named article-based tutorial series by Digital Ocean.

### **Knowledge Check**

This section contains questions for you to check your understanding of this lesson. If you’re having trouble answering the questions below on your own, review the material above to find the answer.

* [What is the DOM?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#dom---document-object-model)
* [How do you target the nodes you want to work with?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#targeting-nodes-with-selectors)
* [How do you create an element in the DOM?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#element-creation)
* [How do you add an element to the DOM?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#append-elements)
* [How do you remove an element from the DOM?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#remove-elements)
* [How can you alter an element in the DOM?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#altering-elements)
* [When adding text to a DOM element, should you use textContent or innerHTML? Why?](https://www.youtube.com/watch?v=ns1LX6mEvyM)
* [Where should you include your Javascript tag in your HTML file when working with DOM nodes?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#important-note)
* [How do “events” and “listeners” work?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#events)
* [What are three ways to use events in your code?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#events)
* [Why are event listeners the preferred way to handle events?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#attaching-listeners-to-groups-of-nodes)
* [What are the benefits of using named functions in your listeners?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#method-3)
* [How do you attach listeners to groups of nodes?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#attaching-listeners-to-groups-of-nodes)
* [What is the difference between the return values of querySelector and querySelectorAll?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#query-selectors)
* [What does a “nodelist” contain?](https://www.theodinproject.com/paths/foundations/courses/foundations/lessons/dom-manipulation#query-selectors)
* [Explain the difference between “capture” and “bubbling”.](https://www.youtube.com/watch?v=F1anRyL37lE)