

## 3.14: jQuery

jQuery is a freely available library add-on to JavaScript. It reduces the amount of code and complexity required to produce features and effects commonly used in creating today's sites. This library rapidly extends how much, and how fast, you can improve upon your site. In addition to jQuery, additional library extensions have been created that extend the jQuery library to automate even more tasks.

Before we begin to look at jQuery, we should consider implementation. The jQuery library is hosted for free on Google's servers at `ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.js` (you may need to adjust the version number for future releases). While you are free to save any number of versions to your own site, there are benefits to using Google's copy. Among them are decreased latency (Google's copies are distributed on servers around the world), decreased communication load on your server, and a caching benefit—the more sites using Google as their source increases the chance your user will already have a recent copy on their device.

Once you have connected to Google, you may want a fall back—Google or not, things can happen. To do this, we need a mechanism to detect whether or not our Google hosted copy loaded correctly. To achieve this, we can test for a feature that is only present in jQuery after we have attempted to load it. If this feature fails, we can assume something went wrong, and then load our fallback copy. All of this can be done by adding two lines in our code:

1. `<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>`
2. `<script>window.jQuery || document.write('<script src="link/to/our/jquery-1.10.2.js"></script>')</script>`

Once we have connected to the jQuery library, we can begin to use it in our JavaScript code. jQuery statements begin with a `$` and then expect information on the selector we wish to use or the method we wish to access in order to take action. Using the document object method (DOM) to select a page element in JavaScript bears resemblance to the chain selector approach of Java:

1. `document.getElementById('ourAttributeName');`

Meanwhile, jQuery allows us to identify the same element simply by referencing the same attribute:

1. `$('#ourAttributeName');`

While neither of these examples actually interact with the attribute, both identify the same unique place in our document. With jQuery, the use of the pound sign (`#`) in our selector specifies that we are looking for an ID. The `#`, in this case, takes the place of typing out `document.getElementById`.

Alternatively, if we wanted to select all elements on our page with a particular class, we would exchange our pound sign for a period. This specifies that we want to select all items of the identified class:

1. `$('.ourClassName');`

Once we have declared our selector, it takes on aspects of an object, something that we can interact with whether it represent one or many elements on our page. For example, we can hide all paragraphs on a page by setting a button to do so:

1. `<script>`
2. `$(document).ready(function(){`
3. `$("#button").click(function){`
4. `$("#p").hide();`
5. `});`
6. `});`
7. `</script>`
8. `<h2>Hide those paragraphs!</h2>`
9. `<p>This is a paragraph.</p>`
10. `<p>This is also a paragraph.</p>`
11. `<button>Hide them!</button>`

In this example you will see we have three statements, all nested together. Moving from the inner-most statement out, we first have the action of actually hiding our paragraphs. This is executed as the callback of the statement it sits inside, meaning it is executed when the button element receives a click. Finally, we want to ensure that none of this occurs until the entire page is ready, as it is a response to a user interaction and not part of creating our page. To ensure this happens, all of this is nested inside a ready function

attached to the entire page, which is referred to as document. This means nothing inside the `$(document).ready...` line will be available until after the page is done loading. Let us look at another example, where we change existing content instead of hiding it:

```
1. <script>
2. $(document).ready(function(){
3. $("#btn").click(function(){
4. $("#test").html("<b>We changed it!</b>");
5. });
6. });
7. </script>
8. <p id="test">This is text we will change.</p>
9. <button id="btn">Set new HTML</button>
```

Like newer versions of CSS, we can traverse elements in our page by utilizing concepts like next and closest to move around from our starting point, without having to know exactly where our destination lies in the DOM. For example, if we were to call `closest()` on our `$('#link')` selector, it would traverse up through our page to find the preceding link. In our working example, we do not have one. In this case, the selector would return false, specifying that another link was not found. Otherwise, our selector would now represent that link, and any actions we took would apply to the new, preceding link that we had selected.

Using classes and IDs as our selectors is another best practice approach to using jQuery. While we could specify that we are looking for images that are in paragraphs that are in forms inside of a div tag, the resulting selector ( `$(“div form p img”)`; is actually read in reverse. jQuery will process this by finding all images, eliminating those not immediately wrapped in a paragraph, eliminating from that list items which are not in a form, and then eliminating from what remains anything that is not immediately within a div.

Reading out the explanation is exhausting enough, let alone processing it. Although we could use the example above effectively, if we know the use case we want our selector to impact, we should simply add or implement an ID or class to those element(s) we wish to interact with. By sticking with a class or an ID, the selector can simply traverse the DOM looking for those identifiers. If you still need to use the combined selector approach, set it to a variable so you can refer to that variable in other places. This will save you the effort of finding all of those elements again.

The examples here are only a glimpse of the full power of jQuery. We are keeping it brief for a reason; until you are more comfortable with both JavaScript and CSS, immediately relying on a library can muddle the learning process. That being said, it is a powerful tool that you should embrace when ready to add more complex enhancements to your site.

## Learn more

Keywords, search terms: jQuery, jQuery libraries

50 jQuery Add-ons: <http://tutorialzine.com/2013/04/50-amazing-jquery-plugins/>

Full Documentation: <http://api.jquery.com/>

1 Jacot, de Boinod, Adam. Global Wording. *Smithsonian Magazine*. March 2006. Web. 15 Dec. 2012

---

3.14: jQuery is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- 3.14: jQuery by Michael Mendez has no license indicated.