

jQuery Fundamentals Training

jQuery Basic Concepts

Lesson 1, Activity 2: Including jQuery

To use jQuery on a web page you must, of course, include the jQuery library - a Javascript file. You can do so in one of two ways:

- Download the code from <http://jquery.com/>. Be sure to get the minified and gzipped production version to reduce download time for users; or
- Hotlink from your webpages to a CDN-hosted copy of jQuery, maintained by Google and some other large enterprises; see, for instance, <http://code.google.com/apis/libraries/devguide.html#jquery>

Be sure to include jQuery, either from a downloaded copy or via a CDN-hosted copy, in the head block of your page.

Lesson 1, Activity 3: Using `$(document).ready()`

You cannot safely manipulate a page until the document is "ready." jQuery detects this state of readiness for you; code included inside `$(document).ready()` will only run once the page is ready for JavaScript code to execute.

A `$(document).ready()` Block

Code Sample:

<jqy-concepts/Demos/document-ready.html>

```
<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script src="../../jqy-lib/fix-console.js"></script>
<script>
$(document).ready(function() {
    console.log('ready!');
});
</script>
</head>
<body>
<p>We'll let you know when the document is ready ...</p>
</body>
</html>
```

There is a shorthand for `$(document).ready()` that you will sometimes see; however, I recommend against using it if you are writing code that people who aren't experienced with jQuery may see.

Shorthand for `$(document).ready()`

Code Sample:

<jqy-concepts/Demos/document-ready-short.html>

```
<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script src="../../jqy-lib/fix-console.js"></script>
<script>
$(function() {
    console.log('ready!');
});
</script>
</head>
<body>
<p>We'll let you know when the document is ready ...</p>
</body>
</html>
```

You can also pass a named function to `$(document).ready()` instead of passing an anonymous function.

Passing a Named Function Instead of an Anonymous Function

Code Sample:

<jqy-concepts/Demos/passing-named-function.html>

```
<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script>
function readyFn() {
    alert("Document is ready");
}

$(document).ready(readyFn);
```

```
</script>
</head>
<body>
<p>We'll let you know when the document is ready ...</p>
</body>
</html>
```

Lesson 1, Activity 5: Selecting Elements

The most basic concept of jQuery is to "select some elements and do something with them". jQuery supports most CSS3 selectors, as well as some non-standard selectors. For a complete selector reference, visit <http://api.jquery.com/category/selectors/>.

```
$('#myId'); // note IDs must be unique per page
```

```
$('.myClass'); // all tags with the class myClass
$('div.myClass'); // performance improves if you specify element type
```

```
$('#input[name=first_name]'); // beware, this can be very slow
```

```
$('#contents ul.people li');
```

Pseudo-Selectors

Pseudo-selectors are special terms prefixed with a colon (:) symbol, which select based on some aspect of an element that jQuery can determine. The available selectors are listed below. For any indexed selectors, the indexing is zero-based. Note that the selectors operate on what was found by the selector up to that point -- if the pseudo-selector is in the middle of the selector string, then it will be applied to all the elements found by the part of the selector to the left of that point.

Some of these are defined by CSS, others are specific to jQuery.

Note: many of these operations are also available as jQuery methods that operate on a selection, e.g., `$('li').not('.selected')` as opposed to `$('li:not(.selected)')`

State-Related

Selector	Description
:animated	Is currently animated by jQuery
:visible	Is visible (more than just the display or visibility settings, like has a size greater than 0, and is attached to the DOM, and isn't an element outside the <body> tag
:hidden	Is hidden -- the opposite of the above
:focus	Currently has focus

Form Elements and States

Selector	Description
:button	<input type="button"> or <button>
:checkbox	<input type="checkbox">
:file	<input type="file">
:image	<input type="image">
:input	Any input element (<input>, <select>, <textarea> tags)
:password	<input type="password">
:radio	<input type="radio">
:reset	<input type="reset">
:submit	<input type="submit">
:text	<input type="text"> or <textarea>

:disabled	Any disabled form element
:enabled	Any enabled form element
:checked	Any checked element (radio or checkbox)
:selected	Any <option> elements that are selected

Content-Related

Selector	Description
:empty	Has no children
:contains(text)	Contains the specified text
:not(subselector)	Does not match the subselector
:has(selector)	Contains one or more elements found by the subselector
:first-child	Is the first child of its parent
:last-child	Is the last child of its parent
:only-child	Is the only child of its parent
:nth-child(index)	Is the child at the nth position within its parent
:header	Is one of the header types (<h1>, <h2>, etc.)

Position of Element in Collection

Selector	Description
:first	The first element found by the selector
:last	The last element found by the selector
:even	Even-index elements withing the collection -- the first element is even, since its index is 0
:odd	Odd-index elements withing the collection
:eq(index)	At the index position within the collection
:gt(index)	Position within the collection is greater than index
:lt(index)	Position within the collection is less than index

Code Sample:

jqy-concepts/Demos/pseudo-selectors.html

```

---- C O D E   O M I T T E D ----
<script>
$(document).ready(function() {

    $('tr:odd').addClass('odd');

    // select all input-like elements in a form
    $('#myForm :input').addClass('yellowBg');

    // select all the visible elements
    $('div:visible').addClass('italic');

    // all except the first three divs
    $('div:gt(3)').addClass('redText');

    // all currently animated divs (not implemented here)
    $('div:animated');
});
</script>
<style>
.odd { background-color: #eeeeff; }

```

```

.yellowBg { background-color: yellow; }
.italic { font-style:italic; }
.redText { color: red; }
</style>
</head>
<body>
<div>
<table border="1" cellspacing="0" cellpadding="8">
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
    </tr>
  </thead>

  ---- C O D E   O M I T T E D ----
</table>
</div>
<div>
<form id="myForm">
<input type="text" />
<textarea>Hello</textarea>
<select>
  <option>A</option>
  <option>B</option>
</select>
</form>
</div>
<div style="display:none">Not visible</div>
<div>The fourth div</div>
<div>The fifth div</div>
<hr />
</body>
</html>

```

Choosing Selectors

Choosing good selectors will improve the performance of your JavaScript. You should always consider the browser's JavaScript tools available to jQuery to process selectors:

`document.getElementById (id)` - very highly optimized, so a query starting with an id is always best. A tag-qualified id (`div.#myId`) is actually less efficient, and should be avoided unless you want to ensure that the element is indeed the specified tag type.

`element.getElementsByTagName (tagName)` - optimized by browsers, even for descendants that are not direct children. Use this to reduce the set of tags to be considered for more time-consuming operations, like those based on attributes or content.

`element.querySelectorAll (selector)` - optimized by browsers, and used by jQuery when available and applicable. It appears that the current implementation of jQuery will only use this for an entire selector, as opposed to using it for part of a selector string, and then applying additional logic, so it might be more efficient when using a jQuery-only pseudo-selector to apply it as a separate operation (i.e., invoke the corresponding method on a collection obtained by a selector that can use `querySelectorAll`)

Inspecting attributes and content - generally time-consuming, and not necessarily applied in an efficient fashion, so it is best to try to reduce the number of tags considered by this part of the selection process. For example, `$('input[name=q]')` is more efficient than `$('[name=q]')`, since only input tags need to be inspected for their name.

A little specificity, for example, including an element type such as `div` when selecting elements by class name, can go a long way. If you know that the CSS class `myClass` was only applied to `div` tags, then `$('div.myClass')` is more efficient than `$('.myClass')`, since jQuery can use `getElementsByTagName('div')` before inspecting the `class` attribute, reducing the number of tags to consider. Generally, any time you can give jQuery a hint about where it might expect to find what you're looking for, you should.

On the other hand, too much specificity can be a bad thing. A selector such as `#myTable thead tr th.special` is overkill if a selector such as `#myTable th.special` will get you what you want. Part of efficient jQuery is designing the HTML in a way that makes optimized selection possible.

jQuery offers many attribute-based selectors, including the ability to make selections based on the content of attributes using simplified regular expressions.

Code Sample:

jqy-concepts/Demos/choose-selector.html

```

<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script>
$(document).ready(function() {
  // find all <p>s whose class attribute
  // ends with "text"
  $("p[class$='Text']").addClass("italic");
});
</script>
<style>
.italic { font-style:italic; }
.redText { color: red; }
.blueText { color: blue; }
.greenText { color: green; }
.yellowBg { background-color: yellow; }
</style>
</head>
<body>
<p class="redText">I have red text.</p>
<p class="blueText">I have blue text.</p>
<p class="greenText">I have green text.</p>
<p class="yellowBg">I'm a p tag with some other class.</p>
<p>I'm an ordinary p tag.</p>
</body>
</html>

```

While these can be useful in a pinch, they can also be extremely slow. Wherever possible, make your selections using IDs, class names, and tag names.

Also, possibly the biggest drawback, at least as far as using this approach with classes, is that it tests the ending of the entire string, not the end of each individual class name. So, for example, `<p class="redText special">` would not be found by `[class$=Text]`.

Avoiding Unwanted Elements

You should be careful when choosing a selector to avoid selecting more elements than you want. In particular, with elements that can contain like elements as descendants (like lists within lists, or tables within tables), this can be a problem. The best approach is to use an id selector first, if possible, to select the container for the elements you want to select, and then the CSS *child selector* (`>`) between the container and the children you want to select.

```

// Bad - will select great-grandchildren if there are inner lists
$('#myList li').remove();

// Good - will only select li children
$('#myList>li').remove();

```

Also, you should take precautions to prevent future changes to the HTML from breaking your code, or causing unwanted effects. For example, if a form currently has only one `select` element, you might be tempted to use a selector like `#myForm select`. But, if another `select` is added later, it would be affected as well. It would be better to use the name attribute to ensure that only this `select` is chosen (as in `#myForm select[name=day]`). Although attribute-based selectors can be inefficient when used indiscriminantly, in this case there will only be a limited number of elements (hopefully just one) actually inspected for their name.

Does My Selection Contain Any Elements?

Once you've made a selection, you'll often want to know whether you have anything to work with. You may be inclined to try something like:

```

if ($('#div.foo')) { ... }

```


This won't work. When you make a selection using `$()`, an object is always returned, and objects always evaluate to `true`. Even if your selection doesn't contain any elements, the code inside the `if` statement will still run.

Instead, you should test the selection's `length` property, which tells you how many elements were selected. If the result is 0, the `length` property will be treated as `false` when used as a boolean value.

```
if ($('#div.foo').length) { ... }
```

Saving Selections

Every time you make a selection, a lot of code runs, and jQuery doesn't do caching of selections for you. If you've made a selection that you might need to make again, you should save the selection in a variable rather than making the selection repeatedly.

```
var $divs = $('#div');
```

Once you've stored your selection, you can call jQuery methods on the variable you stored it in just like you would have called them on the original selection.

In the above code, the variable name begins with a dollar sign. Unlike in other languages, there's nothing special about the dollar sign in JavaScript -- it's just another character. We use it here to indicate that the variable contains a jQuery object. This practice -- a sort of *Hungarian notation* -- is merely convention, and is not mandatory.

A selection only fetches the elements that are on the page when you make the selection. If you add elements to the page later, you'll have to repeat the selection or otherwise add them to the selection stored in the variable. Stored selections don't magically update when the DOM changes.

Refining and Filtering Selections

Sometimes you have a selection that contains more than what you're after; in this case, you may want to refine your selection. jQuery offers several methods for zeroing in on exactly what you're looking for.

Selector Contexts

The `$` function can accept an optional second parameter, *context*, that is a jQuery object to use as a starting point for the selection process. This context provides a means to limit the search within a specific node or a jQuery object containing a set of nodes. This is great when you have a very large DOM tree and need to find, for example, all the `<a>` tags within a specific part of the DOM tree.

Let's say we wanted to remove all the nested tables in the DOM, here's what we could do:

```
//get all 'table' tags and put them in a variable called $tables.
// $tables now becomes a "jQuery object".
var $tables = $('table');

//outputs the # of tables in the DOM
console.log($tables.length);

//get the inner tables
//in $('table', $tables), notice the 2nd argument, $tables, which is the "context" in which jQuery is making the selection.
var $innerTables = $('table', $tables);

//filter out, using the "not" operation, all the inner tables from $tables
$tables = $tables.not($innerTables);

//now prints the # of tables without the inner tables
console.log($tables.length);
```

Now, in finding the nested tables to remove, the parameter to the "not" operation is the selection of all 'table' tags **within** the elements in the

`$tables` object itself (in other words, within all the `'table'` elements already selected). Therefore, `$tables` is the context in the case.

In pure JavaScript terms, the engine is going to perform `getElementsByTagName('table')` on each element in `$tables`, and return those elements so that they can be used in the `"not"` method. This approach becomes useful when you have a collection that you have refined to a point where it would no longer be easy or efficient to specify it with a single selector string, and you then want to use it as a context for further selection.

Refining and Filtering Methods

Most of the refining methods accept a selector as a parameter:

```
method( selector )
```

The *selector* is a string containing a selector expression to match the current set of elements against. By default, for most of the refining operations, the selector string operates in the *context* of the document, that is, the selector is evaluated from the root level, not the level reached by the selection. So, for example, using the `not` function to exclude a subset of elements, `$('table').not('table')` will *not* remove nested tables. It will result in an empty set, since the `'table'` selector for the `not` operation will be applied from the root of the document, not from the level reached by the original query, and we will remove exactly the same elements we originally found.

The `find` method, for example, uses the current collection as the context, so that the selector string is evaluated using the current collection as the starting point. So, `$('ul').find('ul')` will find only inner lists.

Additional Method Overloads for Refining and Filtering Methods

Some of the refining/filtering functions accept additional forms of the parameter list. The full set of overload possibilities is below.

```
method( function(index) )
```

A function used as a test for each element in the set. *this* is the current DOM element. If the function returns `true`, the element is included in the result collection.

```
method( element(s) )
```

An element or array of elements to match the current collection against. The matching is done using an equality test. The result will only include items that are in the set of elements, in effect producing the intersection of the filtered current collection with the set of elements.

```
method( jQuery object )
```

An existing jQuery object to match the current set of elements against. Similar to using elements, but using jQuery collection instead, again producing the intersection of the filtered current collection with passed-in collection.

Method	Description	Parameters
<code>filter</code>	filters the collection to only include elements that match the passed-in selector	selector, function, elements, jQuery
<code>find</code>	finds descendant elements that match the parameter; uses the current collection as the context for a selector	selector, elements, jQuery
<code>children</code>	finds all children, optionally filtered by a selector	none, selector
<code>first</code>	finds the first element in the collection	none
<code>last</code>	finds the last element in the collection	none
<code>eq(n)</code>	finds the <i>n</i> th element in the collection	index
<code>is</code>	tests the current collection and returns true if any element in it matches the selector	selector, function, elements, jQuery

has	finds all elements in the current collection that have at least one descendant that matches the selector	selector, elements
not	returns a new collection containing all the original elements except elements that match the selector	selector, function, elements, jQuery

You should note that the refining methods are *not* mutators -- the original selection is not modified by the operation, but the modified set is returned. For example, if we wanted to find `li` elements and eliminate those with the `special` class, we could do:

```
var $items = $('li');
var $notSpecialItems = $items.not('.special');
```

The `$items` collection is unchanged by the `not` operation. If we wanted to use just one variable, we would need to do the following (recognizing that in this case it would be simpler to just use a single chained statement instead of two separate statements, but that would not make this concept clear):

```
var $items = $('li');
$items = $items.not('.special');
```

Code Sample:

jqy-concepts/Demos/refining-selections.html

```
---- C O D E   O M I T T E D ----
<script>
$(document).ready(function() {
  // div.foo elements that contain <p>'s
  $('div.foo').has('p').addClass('big');

  // h1 elements that don't have a class of bar
  $('h1').not('.bar').addClass('italic');

  // unordered list items with class of current
  $('ul li').filter('.current').addClass('redText');

  // just the first unordered list item
  $('ul li').first().addClass('yellowBg');

  // the sixth
  $('ul li').eq(5).addClass('italic');
});
</script>

---- C O D E   O M I T T E D ----
</head>
<body>
<div class="foo">foo div w/o p</div>
<div class="foo"><p>foo div w/ p</p></div>
<h1 class="bar">h1 bar</h1>
<h1>h1</h1>
<ul>
<li class="current">current and the first</li>
<li>not current</li>
<li class="current">current</li>
<li>not current</li>
<li class="current">current</li>
<li>not current, but the sixth</li>
<li class="current">current</li>
<li>not current</li>
</ul>
</body>
</html>
```

Solution:

jqy-concepts/Solutions/js/sandbox-selecting.js

```
$(document).ready(
function() {

    // Select all of the div elements that have a class of 'module'.
    // Add the 'showMe' class to them.
    $('div.module').addClass('showMe');

    // Come up with three selectors that you could use to get the
    // third item in the #myList unordered list. Try each of them
    // using the 'showMe2' class. Which is the best to use? Why?
    $('#myListItem').addClass('showMe2');
    // above is best -- IDs are *always* the fastest selector

    $('#myList li:eq(2)').addClass('showMe2');
    $('#myList li').eq(2).addClass('showMe2');
    // either would be best if the list item didn't have an ID

    // Select the label for the search input using an attribute
    // selector. Add the 'showMe2' class to it.
    $('label[for=q]').addClass('showMe2');

    // Figure out how many elements on the page are hidden
    // (hint: .length). Display in the console.
    console.log($(':hidden').length + " hidden");
    // But that includes head elements-below shows only elements on page
    console.log($('body :hidden').length + " hidden");

    // Figure out how many image elements on the page have an
    // alt attribute. Display in the console.
    console.log($('img[alt]').length + " images with alt");

    // Select all of the odd table rows in the table body.
    // Add the 'oddRow' class to them.
    $('#fruits tbody tr:odd').addClass('oddRow');
    // be sure to specify tbody, otherwise you'll get the tr in the
    // thead too.
}
);
```

An id selector is always the most efficient, since the browser can handle that natively.

The label is tied to the input by the `for` attribute, so we can use that to find the label.

`$('#img[alt]')` will find any `img` tag where the `alt` attribute is present, even though we didn't ask to find any specific value.

Starting the final query with an `id` enables the jQuery engine to quickly reduce the total number of tags it needs to examine.

Chaining

If you call a method on a selection and that method returns a jQuery object, you can continue to call jQuery methods on the object.

Chaining

```
$('#content').find('h3').eq(2).html('new text for the third h3!');
```

If you are writing a chain that includes several steps, you (and the person who comes after you) may find your code more readable if you break the chain over several lines.

Formatting Chained Code

```
$('#content')
    .find('h3')
    .eq(2)
```

```
.html('new text for the third h3!');
```

If you change your selection in the midst of a chain, jQuery provides the `$.fn.end` method to get you back to your original selection.

Restoring Your Original Selection Using `$.fn.end`

Code Sample:

jqy-concepts/Demos/restoring-selection.html

```
<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script>
$(document).ready(function() {
  $('#content')
    .find('h3')
    .eq(2)
    .html('new text for the third h3!')
    .end() // restores the selection to all h3's in #content
    .eq(0)
    .html('new text for the first h3!');
});
</script>
</head>
<body id="content">
<h3>First</h3>
<h3>Second</h3>
<h3>Third</h3>
<h3>Fourth</h3>
</body>
</html>
```

Chaining is extraordinarily powerful, and it's a feature that many libraries have adapted since it was made popular by jQuery. However, it must be used with care. Extensive chaining can make code extremely difficult to modify or debug. There is no hard-and-fast rule to how long a chain should be -- just know that it is easy to get carried away.

Lesson 1, Activity 6: **Selecting**

Duration: 10 to 20 minutes.

Open the file jqy-concepts/Exercises/index.html in your browser. Use jqy-concepts/Exercises/sandbox.js to accomplish the following:

1. Select all of the `div` elements that have a class of `'module'`. Add the `'showMe'` class to them.
2. Come up with three selectors that you could use to get the third item in the `#myList` unordered list. Try each of them using the `'showMe2'` class. Which is the best to use? Why?
3. Select the label for the search input using an attribute selector. Add the `'showMe2'` class to it.
4. Figure out how many elements on the page are hidden (hint: `.length`). Display in the console.
5. Figure out how many image elements on the page have an `alt` attribute. Display in the console.
6. Select all of the odd table rows in the table body. Add the `'oddRow'` class to them.

Lesson 1, Activity 8: Working with Selections

Once you have a selection, you can call methods on the selection. Methods generally come in two different flavors: getters and setters. Getters return a property of the first selected element; setters set a property on all selected elements.

Getting and Setting Information About Elements

There are any number of ways you can change an existing element. Among the most common tasks you'll perform is changing the inner HTML or attribute of an element. jQuery offers simple, cross-browser methods for these sorts of manipulations. You can also get information about elements using many of the same methods in their getter incarnations.

jQuery "overloads" its methods, so the method used to set a value generally has the same name as the method used to get a value. When a method is used to set a value, it is called a *setter*. When a method is used to get (or read) a value, it is called a *getter*.

The getter methods return a single value from the *first* element in the collection, while the setters affect each element in the collection. (The lone exception is the `text()` method; as a getter, it returns the concatenation of the text content of all the elements in the collection.)

The `$.fn.html` Method Used as a Setter

```
$('#h1').html('hello world');
```

The `$.fn.html` Method Used as a Getter

```
var content = $('#h1').html();
```

Setters return a jQuery object, allowing you to continue to call jQuery methods on your selection; getters return whatever they were asked to get, meaning you cannot continue to call jQuery methods on the value returned by the getter.

Note: Changing things about elements is trivial, but remember that the change will affect all elements in the selection, so if you just want to change one element, be sure to specify that in your selection before calling a setter method.

- `$.fn.html` - Get or set the html contents.
- `$.fn.text` - Get or set the text contents; HTML will be stripped when getting, and escaped when setting (e.g., `<` will get turned into `<`).
- `$.fn.attr` - Get or set the value of the provided attribute.
- `$.fn.width` - Get or set the width in pixels of the first element in the selection as an integer.
- `$.fn.height` - Get or set the height in pixels of the first element in the selection as an integer.
- `$.fn.position` - Get an object with position information for the first element in the selection, relative to its first positioned ancestor. This is a getter only.
- `$.fn.val` - Get or set the value of form elements.

Changing the HTML of an Element

Code Sample:

jqy-concepts/Demos/changing-html.html

```
<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script>
$(document).ready(function() {
  $('#myDiv p:first')
    .html('New <strong>first</strong> paragraph!');
});
</script>
</head>
<body>
<div id="myDiv">
```

```
<h1>Changing HTML</h1>  
<p>Original first paragraph.</p>  
<p>Second paragraph.</p>  
</div>  
</body>  
</html>
```

Try changing the `html` method call to `text`.

Lesson 1, Activity 9: CSS, Styling, and Dimensions

jQuery includes a handy way to get and set CSS properties of elements.

Note

CSS properties that normally include a hyphen need to be *camel cased* in JavaScript. For example, the CSS property `font-size` is expressed as `fontSize` in JavaScript.

Getting CSS Properties

```
$('#h1').css('fontSize'); // returns a string such as "19px"
```

Setting CSS Properties

```
// setting an individual property
$('#h1').css('fontSize', '100px');

// setting multiple properties
$('#h1').css({ 'fontSize' : '100px', 'color' : 'red' });
```

Note the style of the argument we use on the second line -- it is an object that contains multiple properties. This is a common way to pass multiple arguments to a function, and many jQuery setter methods accept objects to set multiple values at once.

Using CSS Classes for Styling

As a getter, the `$.fn.css` method is valuable; however, it should generally be avoided as a setter in production-ready code, because you don't want presentational information in your JavaScript. Instead, write CSS rules for classes that describe the various visual states, and then simply change the class on the element you want to affect.

Working with Classes

```
var $h1 = $('#h1');

$h1.addClass('big');
$h1.removeClass('big');
$h1.toggleClass('big');

if ($h1.hasClass('big')) { ... }
```

Classes can also be useful for storing state information about an element, such as indicating that an element is selected.

Dimensions

jQuery offers a variety of methods for obtaining and modifying dimension and position information about an element.

The code in the example below is just a very brief overview of the dimensions functionality in jQuery; for complete details about jQuery dimension methods, visit <http://api.jquery.com/category/dimensions/>.

Basic Dimensions Methods

Code Sample:

jqy-concepts/Demos/basic-dimensions.html

```
---- C O D E   O M I T T E D ----
$(document).ready(function() {
  var msg = "";
  //
  $('#h1').width('300px');

  // gets the width of the first H1
  msg += "Width: " + $('#h1').width() + ", ";
```

```
// sets the height of all H1 elements
$('h1').height('150px');

// gets the height of the first H1
msg += "Height: " + $('h1').height() + "\n";

// returns an object containing position
// information for the first H1 relative to
// its "offset (positioned) parent"
msg +=
"Top: " + $('h1').position().top + ", " +
"Left: " + $('h1').position().left;
alert(msg);
});

---- C O D E   O M I T T E D ----
```

Attributes

An element's attributes can contain useful information for your application, so it's important to be able to get and set them.

The `$.fn.attr` method acts as both a getter and a setter. As with the `$.fn.css` method, `$.fn.attr` as a setter can accept either a key and a value, or an object containing one or more key/value pairs.

Getting Attributes

```
// returns the href for the first a element in the document
$('a').attr('href');
```

Setting Attributes

Manipulating a Single Attribute

The `attr` method will change an attribute or property of a selection.

```
$('#myDiv a:first').attr('href', 'newDestination.html');
```

Manipulating Multiple Attributes

The `attr` method can accept an object instead of two separate parameters -- all of the object's properties will be used to set corresponding properties in the selection.

```
$('#myDiv a:first').attr({
  href : 'newDestination.html',
  rel : 'super-special'
}).html('New Destination');
```

```
$('#myDiv a:last').attr({
  rel : 'super-special',
  href : function() {
    return 'new/' + $(this).attr('href');
  }
}).html('new/Destination');
```

Code Sample:

jqv-concepts/Demos/setting-attributes.html

[Home](#)
[Top](#)
[Up](#)

This time, we broke the object up into multiple lines. Remember, whitespace doesn't matter in JavaScript, so you should feel free to use it liberally to make your code more legible! You can use a minification tool later to strip out unnecessary whitespace for production.

Using a Function to Determine an Attribute's New Value

Code Sample:

[jqy-concepts/Demos/change-attributes.html](#)

```
---- C O D E   O M I T T E D ----
```

```
Old destination (here).  

Old destination (here).  

Old destination (here).
```

Removing Attributes

jQuery provides `$.fn.removeAttr(attributeName)` for this purpose.

Showing and Hiding Elements

While these will be covered in more detail later, the `show()` and `hide()` methods do what you might expect -- show or hide the selected elements.

Iterating Over a Selection

If you have a complex task to perform on your selection, you can iterate over its elements using `$.fn.each()`. The function that you pass to this method is run individually for all of the elements in a selection. The function receives the index of the current element and the DOM element itself as arguments. Inside the function, the DOM element is also available as `this`. (The DOM element is not a jQuery object, though -- it is a plain JavaScript element.)

Code Sample:

[jqy-concepts/Demos/selection-iterating.html](#)

- A
- B
- C
- D

- E

Lesson 1, Activity 10: Traversing

Once you have a jQuery selection, you can find other elements using your selection as a starting point.

For complete documentation of jQuery traversal methods, visit <http://api.jquery.com/category/traversing/>.

Be cautious with traversing long distances in your documents -- complex traversal makes it imperative that your document's structure remain the same, something that's difficult to guarantee even if you're the one creating the whole application from server to client. One or two-step traversal is fine, but you generally want to avoid traversals that take you from one container to another.

Traversal Methods

A useful subset of jQuery's traversal methods is listed below. For most of them, the only parameter choices are none or an optional selector that is used to filter the set of elements returned. For example, the `next()` method returns the element after the current element. `next(selector)` returns the element that follows the current element only if it matches the selector. In other words, `next('p')` doesn't return the next `p` tag -- it returns the next element if and only if it is a `p` tag.

Method	Description	Parameters
<code>parent</code>	finds the parent element of each element in the current set	none, selector
<code>offsetParent</code>	finds first positioned ancestor of each element in the current set	none
<code>closest</code>	finds the closest ancestor element of each element in the current set	none, selector
<code>next</code>	finds the next sibling of each element in the current set	none, selector
<code>prev</code>	finds the previous sibling of each element in the current set	none, selector
<code>nextAll</code>	finds all following siblings of each element in the current set	none, selector
<code>prevAll</code>	finds all preceding sibling of each element in the current set	none, selector
<code>siblings</code>	finds all the siblings of each element in the current set	none, selector

Moving Around the DOM Using Traversal Methods

Code Sample:

jqy-concepts/Demos/traversal-methods.html

```

---- C O D E   O M I T T E D ----
$(document).ready(function() {
  $('h1').next('p').addClass('redText');
  $('input[name=first_name]').closest('form').addClass('yellowBg');
  $('input[name=first_name]').parent().addClass('redText');
  $('#myList').children().addClass('redText');

  // below doesn't work as expected if more than
  // one element is selected
  $('li.selected').siblings().addClass('italic');

  // but this does work as expected
  $('li:not(.selected)').addClass('blueText');
});
</script>

---- C O D E   O M I T T E D ----
</head>
<body>
<h1>Heading</h1>
<p>A paragraph (will be red)</p>
<p>Another paragraph (won't be red)</p>
<h1>Heading</h1>
<h3>A heading 3</h3>
<p>Another paragraph (won't be red)</p>
<form>

```

```

<div>First name: <input type="text" name="first_name" /></div>
<div>Last name: <input type="text" name="last_name" /></div>
</form>
<ul id="myList">
  <li>A</li>
  <li class="selected">B</li>
  <li>C</li>
  <li>D</li>
  <li>E</li>
</ul>
</body>
</html>

```

Solution:

[jqy-concepts/Solutions/js/sandbox-traversing.js](#)

```

$(document).ready(
  function() {

    // Get all the image elements on the page; log each image's
    // alt attribute.
    $('img').each(function(i) {
      console.log($(this).attr('alt'));
      //console.log(this.alt);
    });

    // Get the search input text box, then traverse up to the form
    // and add a class to the form that contains it.
    $('input[name="q"]').closest('form').addClass('showMe');

    // Get the list item inside #myList that has a class of 'current'
    // and remove that class from it; add a class of 'current'
    // to the next list item.
    $('#myList li.current')
      .removeClass('current')
      .next()
      .addClass('current');

    // Get the select element inside #specials; traverse your way to
    // the submit button. Set its disabled attribute to true.
    $('#specials select')
      .parent() // Could also use closest('form')
      .next()   // instead of these two lines
      .find('input.input_submit')
      .attr('disabled', true);

    // Get the first list item in the #slideshow element; add the class
    // 'current'? to it, and then add a class of 'disabled' to its
    // sibling elements.
    $('#slideshow li:first')
      .addClass('current')
      .siblings()
      .addClass('disabled');
  }
);

```

Lesson 1, Activity 11: Traversing

Duration: 10 to 20 minutes.

Open [index.html](#) again in your browser. Use [sandbox.js](#) to accomplish the following:

1. Select all of the image elements on the page; log each image's `alt` attribute.
2. Select the search input text box, then traverse up to the form and add the `'showMe'` class to the form.
3. Select the list item inside `#myList` that has a class of `'current'` and remove that class from it; add a class of `'current'` to the next list item.
4. Select the `select` element inside `#specials`; traverse your way to the submit button. Set its `disabled` attribute to `true`.
5. Select the first list item in the `#slideshow` element; add the class `'current'` to it, and then add a class of `'disabled'` to its sibling elements.

Lesson 1, Activity 13: Manipulating the DOM

Once you've made a selection, the fun begins. You can change, move, remove, and clone elements. You can also create new elements via a simple syntax.

For complete documentation of jQuery manipulation methods, visit <http://api.jquery.com/category/manipulation/>.

Moving, Copying, and Removing Elements

There are a variety of ways to move elements around the DOM; generally, there are two approaches:

- Place the selected element(s) relative to another element
- Place an element relative to the selected element(s)

For example, jQuery provides `$.fn.insertAfter` and `$.fn.after`. The `$.fn.insertAfter` method places the selected element(s) after the element that you provide as an argument; the `$.fn.after` method places the element provided as an argument after the selected element. Several other methods follow this pattern: `$.fn.insertBefore` and `$.fn.before`; `$.fn.appendTo` and `$.fn.append`; and `$.fn.prependTo` and `$.fn.prepend`.

The method that makes the most sense for you will depend on what elements you already have selected, and whether you will need to store a reference to the elements you're adding to the page. If you need to store a reference, you will generally want to take the first approach -- placing the selected elements relative to another element -- as it returns the element(s) you're placing. In this case, `$.fn.insertAfter`, `$.fn.insertBefore`, `$.fn.appendTo`, and `$.fn.prependTo` will be your tools of choice.

Moving Elements Using Different Approaches

Code Sample:

jqy-concepts/Demos/moving-elements.html

```

---- C O D E   O M I T T E D ----
<script>
$(document).ready(function() {
// make the first list item the last list item
$('#myList li:first').appendTo('#myList').addClass('current');

// another approach to the same problem
$('#myList').append($('#myList li:first')).addClass('italic');

// note that there's no way to access the
// list item that we moved, as this returns
// the list itself
});
</script>

---- C O D E   O M I T T E D ----
</html>

```

Cloning Elements

When you use methods such as `$.fn.appendTo`, you are moving the element; sometimes you want to make a copy of the element instead. In this case, you'll need to use `$.fn.clone` first. The clone method is overloaded -- with no parameters, it makes a shallow clone, which does not clone event handlers and data (jQuery has a facility for associating data with an element, which we will cover later). If a `true` value is passed as a parameter, event handlers and data are cloned as well.

Note: ids will get cloned as well as other attributes, but you shouldn't have two page elements with the same id. So, after cloning an element with an id, you should either remove the id or set it to some unique value, before inserting it into the document.

```

$('#myDiv')
.clone().removeAttr('id')
.appendTo('body');

// or...

$('#myDiv')

```



```
.clone().attr('id', 'newId_1')
.appendTo('body');
```

Making a Copy of an Element

```
// copy the first list item to the end of the list
$('#myList li:first').clone().appendTo('#myList');
```

Removing Elements

There are two ways to remove elements from the page: `$.fn.remove` and `$.fn.detach`. You'll use `$.fn.remove` when you want to permanently remove the selection from the page; while the method does return the removed element(s), those elements will not have their associated data and events attached to them if you return them to the page.

If you need the data and events to persist, you'll want to use `$.fn.detach` instead. Like `$.fn.remove`, it returns the selection, but it also maintains the data and events associated with the selection, so you can restore the selection to the page at a later time.

Note: The `$.fn.detach` method is extremely valuable if you are doing heavy manipulation to an element. In that case, it's beneficial to `$.fn.detach` the element from the page, work on it in your code, and then restore it to the page when you're done. This saves you from expensive "DOM touches" while maintaining the element's data and events.

If you want to leave the element on the page but simply want to remove its contents, you can use `$.fn.empty` to dispose of the element's inner HTML.

Creating New Elements

jQuery offers a trivial and elegant way to create new elements using the same `$()` method you use to make selections.

Creating New Elements

If you create a string of text with the HTML for an element, then pass it to `$()`, a DOM element will be created. The method returns a single-element collection containing the new element. You can then use any of the methods discussed earlier to add it to the document.

You can also pass an HTML string to any of the methods that accept an element to insert into the DOM.

```
$('<p>This is a new paragraph!</p>').prependTo($('body'));
$('<li class="new">new list item</li>').appendTo($('#myList'));

$('body').append('<p>This is another new paragraph!</p>');
```

The above approaches will also work if you use a selector for the destination, instead of a jQuery object. The difference in the code below is that the parameters to `appendTo` and `prependTo` are simple strings, instead of return values from the `$` function.

```
$('<p>This is a new paragraph!</p>').prependTo('body');
$('<li class="new">new list item</li>').appendTo('myList');
```

You can pass a second parameter with options representing body content and attributes for the tag.

```
$('<a/>', {
  html : 'This is a <em>new</em> link',
  'class' : 'new',
  href : 'foo.html'
}).appendTo($('body'));
```

Note that the `'class'` property of the element attributes object is quoted -- this is to avoid conflicts with the reserved word `class`.

Also note the empty `<a />` tag -- it gets populated by the `html` property of the second parameter.

It is worth noting that this approach of creating an element with an attributes object only works when the HTML and attributes are passed to the `$` function. You cannot pass both an HTML string and an attributes object to the methods like `append`. For string parameters, it will only accept a single parameter with the HTML string, or a series of strings, each representing an element to create and add. But, you could instead pass it a jQuery object, including one you created from a string.

```
// Won't work
$('#myDiv')
.append('<h2/>', { html: 'New Heading', 'class': 'yellowBg' });

// Will work
$('#myDiv')
.append($('#<h2/>', { html: 'New Heading', 'class': 'yellowBg' }));
```

Code Sample:

jqv-concepts/Demos/creating-new-elements.html

```
---- C O D E   O M I T T E D ----
<script>
$(document).ready(function() {
  // simple elements
  $('<p>A new paragraph!</p><p>Another new paragraph!</p>')
  .prependTo($('body'));
  $('<li class="new">new list item</li>').appendTo($('#myList'));
  $('body').append('<p>This is another new paragraph!</p>');

  // element with attributes object
  $('<a/>', {
    html : 'This is a <em>new</em> link',
    'class' : 'new',
    href : 'foo.html'
  }).appendTo($('body'));
});
</script>
</head>
<body>
<ul id="myList">
<li>First</li>
<li>Second</li>
<li>Third</li>
<li>Fourth</li>
</ul>
<hr />
</body>
</html>
```

The syntax for adding new elements to the page is so easy, it's tempting to forget that there's a huge performance cost for adding to the DOM repeatedly. If you are adding many elements to the same container, you'll want to concatenate all the HTML into a single string, and then append that string to the container instead of appending the elements one at a time. You can use an array to gather all the pieces together, then join them into a single string for appending.

```
var myItems = [], $myList = $('#myList');

for (var i=0; i<100; i++) {
  myItems.push('<li>item ' + i + '</li>');
}

$myList.append(myItems.join(''));
```

Solution:

jqv-concepts/Solutions/js/sandbox-manipulating.js

```

$(document).ready(

function() {
    // Add five new list items to the end of the unordered list #myList.
    for (var i = 0; i < 5; i++) {
        $('<li/>', { html: 'new item ' + i, 'class': 'current' })
            .appendTo('#myList');
    }

    // Remove the odd list items from #myList.
    $('#myList>li:odd').remove();

    // Add another h2 and another paragraph to the last div.module.
    $('div.module:last')
        .append('<h2>New Heading</h2>').append('<p>New Paragraph</p>');

    // Add another option to the select element;
    // give the option the value "wednesday",
    // and inner HTML of "Wednesday".
    $('#specials select[name=day]')
        .append('<option value="wednesday">Wednesday</option>');
    // another way
    $('<option>Thursday</option>')
        .val('thursday')
        .appendTo('#specials select[name=day]');

    // Add a new div.module to the page after the last one;
    // put a copy of one of the existing images inside of it.
    // Note that the images have ids, which should be removed.
    $('<div class="module">')
        .append($('img:first').clone().removeAttr('id'))
        .insertAfter('div.module:last');
    }
);

```

Lesson 1, Activity 14: **Manipulating**

Duration: 10 to 20 minutes.

Open the file [index.html](#) in your browser. Use the [sandbox.js](#) or work in Firebug to accomplish the following:

1. Add five new list items to the end of the unordered list `#myList`. (Hint: `for (var i = 0; i<5; i++) { ... }`)
2. Permanently remove the odd list items from `#myList`.
3. Add another `h2` and another paragraph to the last `div.module`.
4. Add another option to the select element; give the option the value "wednesday", and inner HTML of "Wednesday".
5. Add a new `div.module` to the page after the last one; put a copy of one of the existing images inside of it. Note that the images have `ids`, which should be removed from the copy.