CSCE 120: Learning To Code

Module 7.0: Manipulating Data II
Introduction to jQuery

Introduction

This module is designed to get you started working with manipulating HTML elements and their styles programmatically by using the jQuery JavaScript library in order to start creating more dynamic, interactive applications.

jQuery

jQuery is a JavaScript library/framework that was created to make common tasks in HTML and JavaScript easier. It is the most popular JavaScript library in use today and has hundreds of available plugins and other libraries built on top of it. Most notably, the companion jQueryUI library provides functionality for many common User Interface (UI) widgets and functionality while jQuery Mobile (https://jquerymobile.com/) provides a touch interface-optimized library for mobile devices such as tablets and phones.

We've already seen jQuery in action in previous hacktivities. In previous projects we "hot-linked" the jQuery library by connecting to a Content Delivery Network (CDN). To recap, you can include the jQuery library in your HTML document by including the following in the `<head>` (or you may include an alternative version):

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
```

Note that this is the quick way of including the library. In an actual application it would be better to download (after any possible customization and configuration) the jQuery file(s) and load them from your local copy.

Using jQuery

jQuery allows you to apply one or more actions to a *set* of DOM elements. The first step is to *select* the elements that you want to manipulate. To do this, you use the dollar sign and and provide a selection rule (as a string) that is very similar to the CSS selectors/combinators we have used previously. Some examples:

```
//This matches all table cells:
$("td")

//This matches all paragraphs that appear *inside* a div element:
$("div p")

//This matches all td elements AND all paragraph elements:
$("div, td")

//This matches the element whose id is leadParagraph:
$("#leadParagraph")

//This matches all elements whose class is greeting
$(".greeting")
```

The examples above will match all elements in the document tree. That is, jQuery traverses the DOM and finds all elements that match the criteria that we provide. We didn't actually specify what should be done with them. We now describe how we can. Each of the calls above returns a set (think of it as an array) of objects which we can apply other function calls to. There are many functions that jQuery provides and we will only look at a small subset of them. To get started, we describe how we can change the stylistic elements using the function `css()`.

When changing the style of elements, the `css()` function takes two arguments: the first one is a string of the CSS property that you want to set, and the second is the value you want to set it to. Some examples:

```
//this sets the font of all paragraph elements to courier
$("p").css("font-family", "courier");

//this sets the background color of all table cells to green
$("td").css("background-color", "green");

//this sets the color of the element whose id is leadParagraph
$("#leadParagraph").css("color", "red");

//this makes all elements whose class includes the class
//"strong" to have bold font
$(".strong").css("font-weight", "bold");
```

Chaining

jQuery is designed with what is known as a *fluent* design pattern. This means that you can *chain* calls to functions in jQuery. Consider the following example.

```
1  //we wish to set three different style elements to the lead
2  //paragraph; one way is to make 3 calls:
3  $("#leadParagraph").css("font-size", "150%");
4  $("#leadParagraph").css("font-weight", "bold");
5  $("#leadParagraph").css("color", "red");
6
7  //A better way to do the same thing is to *chain* all three calls:
8  $("#leadParagraph").css("font-size", "150%")
9                     .css("font-weight", "bold")
10                    .css("color", "red");
```

Chaining provides a more convenient way to invoke multiple function calls on the same set of results. Not only is it more convenient, it is potentially more efficient. In the example above, the first way of doing it would mean that the DOM is traversed three times. The second way of doing it means that the DOM is only traversed once. The chaining avoids re-traversing the DOM to apply the second two function calls.

Common Useful Functions

The `css()` function

In addition to being able to *set* CSS properties, the `css()` function can be used to *get* CSS properties of DOM elements. To do this, you only provide *one* parameter to the function that is a string representation of the CSS property you wish to retrieve. The function returns the property's value as a string. An example:

```
1  var lpColor = $("#leadParagraph").css("color");
2  //now we can use this value:
3  $("#secondParagraph").css("color", lpColor);
```

Note: if your selector returns a set of results, the `css()` function will only return the value of the CSS property of the *first* element.

The `addClass()` and `removeClass()` functions

Another way to modify how HTML elements are styled is by giving them classes. Recall that an element can have any number of classes, so we can add and/or remove any number as well using the `addClass()` and `removeClass()` functions. Each one takes a string that can contain one or more (space-delimited) classes.

```
//remove fooClass from all paragraphs:
$("p").removeClass("fooClass");
//remove fooClass AND barClass from all paragraphs:
$("p").removeClass("fooClass barClass");

//add fooClass to all div elements:
$("div").addClass("fooClass");
```

Another useful function is `toggleClass()` which works the same way but "toggles" the class. If the element(s) have the class, it is removed; if the element(s) do not have the class, it is added.

The `text()` function

The `text()` function is used to get or set the inner text of an HTML element. Consider, for example, the following HTML document snippet

```
...
<p id="leadParagraph">Hello World!</p>
...
```

```
//get the text in the lead paragraph element:
var pText = $("#leadParagraph").text();
//pText would contain the string "Hello World!"

//set the text in the lead paragraph element:
$("#leadParagraph").text("Welcome to my home page...");
//The "Hello World!" paragraph would be changed to
//"Welcome to my home page..."
```

If the HTML element(s) have child elements, then when using the `text()` function to get the contents, it *concatenates* the inner text of *all* the children. If you set the text, it essentially eliminates all the children, so be careful.

The `html()` function

The `html()` function is similar to `text()`, but instead of just the inner text, it retrieves the actual HTML (as a string). If the element has children, they are included in HTML format.

The real power comes when using the `html()` function to (re)set the HTML. If you provide a well-formatted HTML string, the function essentially creates and adds new HTML elements as children.

```
//screw up the first row in a table:
$("tr:first").html("foo");

//modifies the first row with two new columns:
$("tr:first").html("<td>col 1</td><td>col 2</td>");
```

The `attr()` function

You can also change the attributes of a DOM element using the `attr()` function. As with other functions, you can provide one argument to retrieve the attribute's value and two arguments to set the attribute value.

```
//reset what a hyperlink links to:
$("#myLink").attr("href", "http://cse.unl.edu");

//reset which picture is displayed by modifying src:
$("#photo").attr("src", "other.jpg");
```

Documentation: http://api.jquery.com/attr/

The `val()` function

When a user enters information into an HTML form element (say a text box or checkbox), it is not part of the element's HTML or inner text. Instead, the value is an attribute. To get the value using jQuery we can use the `val()` function.

```
var firstName = $("#firstNameInput").val();
```

Like the other functions, `val()` can be used to set the contents of a text box if you provide a string as an argument.

```
$("#firstNameInput").val("Please Reenter!");
```

Documentation: http://api.jquery.com/val/

Adding and Removing DOM Elements

The `html()` function can be used to add HTML elements to the document. However, a better way to do this is by using the following functions.

- `append()` – adds the content to the end of each element in the set
- `prepend()` – adds the content to the beginning of each element in the set

- `after()` – adds the content to the DOM *after* each element in the set (append and prepend operate *inside* the matched elements)
- `before()` – adds the content to the DOM *before* each element in the set
- `empty()` – empties the content of the matched elements (the internal text and all of its children are deleted)
- `remove()` – removes the matched element and all its content and children from the DOM

As an example, consider the following.

```
1  //adds a period to the matched element
2  $("#lead").append(".");
3  //<p id="lead">Hello</p>
4  //would become:
5  //<p id="lead">Hello.</p>
6
7  //adds a period to the matched element
8  $("#lead").after("<p>World</p>");
9  //<p id="lead">Hello</p>
10 //would become:
11 //<p id="lead">Hello</p><p>World</p>
12
13 //empties the content of the paragraph:
14 $("#lead").empty();
15 //<p id="lead">Hello</p>
16 //would become:
17 //<p id="lead"></p>
18
19 //removes the paragraph entirely
20 $("#lead").remove();
21 //<p id="lead">Hello</p>
22 //would cease to exist entirely
```

It is important to understand that jQuery functions operate on the DOM loaded in memory only. That is, they operate on the page *after* it has been loaded in a browser. These functions only modify the document in memory. The original HTML source document is *not* affected. If you refresh/reload the document will return to its original rendering.

Effects & Animation

The jQuery library also provides several visual effects that can be applied to DOM elements. Full documentation can be found here: http://api.jquery.com/category/effects/. Some examples:

- `hide()` and `show()` allow you to make DOM elements invisible/visible
- `fadeIn()`, `fadeOut()`, `fadeToggle()` allow you to show/hide elements but with a fade-in or fade-out effect
- `slideUp()`, `slideDown()`, `slideToggle()` allow you to show/hide elements but with a "slide" effect

These basic effects can be augmented with more advanced effects in the jQuery UI library (see here for full documentation: http://jqueryui.com/effect/).

```
$("#lead").fadeOut();
//the paragraph is now hidden, but is still part of the DOM

//you can make it reappear, slower than it faded
//out, and remove it after the animation completes
$("#lead").fadeIn("slow");

//quickly slide the element out and remove it after
//the animation completes
$("#lead").slideDown("fast").remove();
//you can no longer make it reappear as it is no longer
//part of the DOM
```

The `ready()` function

When a request is made to a webpage, a web server responds by sending the initial HTML page. Based on the elements in the document, the browser then makes other requests—to load images, style sheets, scripts, etc. However, this is all done *asynchronously*. Each request is sent at the same time and resources may be received in a nondeterministic (random) order. Still, we may want to execute some code (such as adding functionality to certain elements or loading other data), but only after the entire page and all of its content has been loaded and rendered properly. jQuery provides the `ready()` function for doing this. For now we observe the syntax, later on we'll discuss functions and callbacks in more detail.

```
$(document).ready( function() {
  //code to be executed after the document is ready...
});
```

The code within the function will only execute *after* the page is fully loaded, rendered, and "ready". This is especially useful if, for example, we are loading multiple scripts or libraries. If we want to execute some code to initialize some elements using code in a library, we would want to ensure that the library code is properly downloaded and available. Otherwise, if we attempted to execute the code before the library had loaded, it could lead to errors and incorrect rendering.

jQuery UI

jQuery UI is a library built on top of jQuery that provides functionality of User Interface (UI) elements, usually called *widgets*. jQuery provides common "dynamic" elements such as accordions (sections that expand and contract), a datepicker (a popup that allows a user to click a date on a calendar), tooltips (popups that provide additional information or content), autocompletes (that provide suggestions to users as they type in an input box), among many others.

Each widget provides extensive ways to customize widgets. However, each widget can easily be integrated with your document through a few simple calls. First, we need to ensure that the library and its style sheet are included in our document. For this, we use the same Google CDN as before.

```
<link rel="stylesheet"
    href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/themes/smoothness/jquery-ui.css">
<script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/jquery-ui.min.js"></script>
```

Then, we can add a widget's functionality by calling the relevant function on the DOM element that we want to apply the widget/functionality to. This is usually done within the `ready()` function. Examples:

```
<script>
  $(document).ready( function() {
    //attach a date picker widget to the birthday element:
    $("#birthday").datepicker();
    //attach an accordion to the information div:
    $("#mainInformation").accordion();
    //add a tooltip to the lead paragraph;
    //the tool tip content is specified in the title attribute
    //by default but can be customized
    $("#leadParagraph").tooltip();
  });
</script>
```

Complete documentation on the features that jQuery UI provides along with live demonstrations of each are provided on its homepage, http://jqueryui.com/.