JOHNS HOPKINS UNIVERSITY

Zanvyl Krieger School of Arts and Sciences
Advanced Academic Programs

410.712.81
Advanced Practical Computer Concepts For Bioinformatics

Topic: Javascript and jQuery

Instructor: Joshua Orvis

Overview

It's difficult to find a modern site that doesn't employ Javascript. To be a web developer, you simply must have experience in HTML, CSS, Javascript and some server-side programming language. We've done HTML and server-side programming so far, so where does Javascript fit?

It's useful to provide functionality on your site that doesn't require round trips to the server before the user gets feedback. Javascript provides this client-side scripting capability and a powerful means to creating rich, interactive web applications.

Classic form validations, creating draggable objects in the browser window, drawing on the HTML5 canvas, and fetching data from the server in the background while the user interacts with your page – all these interactions are possible because of Javascript.

In this week's lecture we'll cover the basics of Javascript before moving onto a very popular framework called jQuery. It serves as a layer on top of the core Javascript that makes common tasks a lot easier and hard interactions and visualizations much more accessible. It has extensive cross-browser support, is relatively lightweight, and even comes already available as part of the HTML5 Boilerplate we've been using in class.

Browser compatibility

All modern browsers support Javascript, though there are both browser-specific and platform-specific differences in the 'features' they support. We will learn some of these, but the differences between these implementations is one of the great reasons to use frameworks like jQuery – you don't have to constantly write different methods for different browsers since the logic for handling each is already built into the framework.

You also might have to consider what to do if users have Javascript completely turned off in their browsers. This really depends on what sort of application you're writing, your level of funding, and how much time you're willing to spend on supporting this small fringe of users. In most academic settings, it's common to simply display a warning message that the site will be unusable if users have Javascript turned off.

For even business applications, writing server-side fall backs for the less than 5% of users who have Javascript turned off may not be worth the investment considering that it usually doubles (or more) the development time.

See the last slide for more resources and browser compatibility lists for specific Javascript features.

Getting started

The two main methods for adding Javascript to your page are embedding the code into the header or, better yet, linking to an external file.

Example of embedding in the header:

```
<head>
    <title>Title goes here</title>
    <script type=”text/javascript”>
        // your code goes here
    </script>
</head>
```

Linking to an external file:

```
<head>
    <title>Title goes here</title>
    <script type=”text/javascript” src=”/js/common.js” />
</head>
```

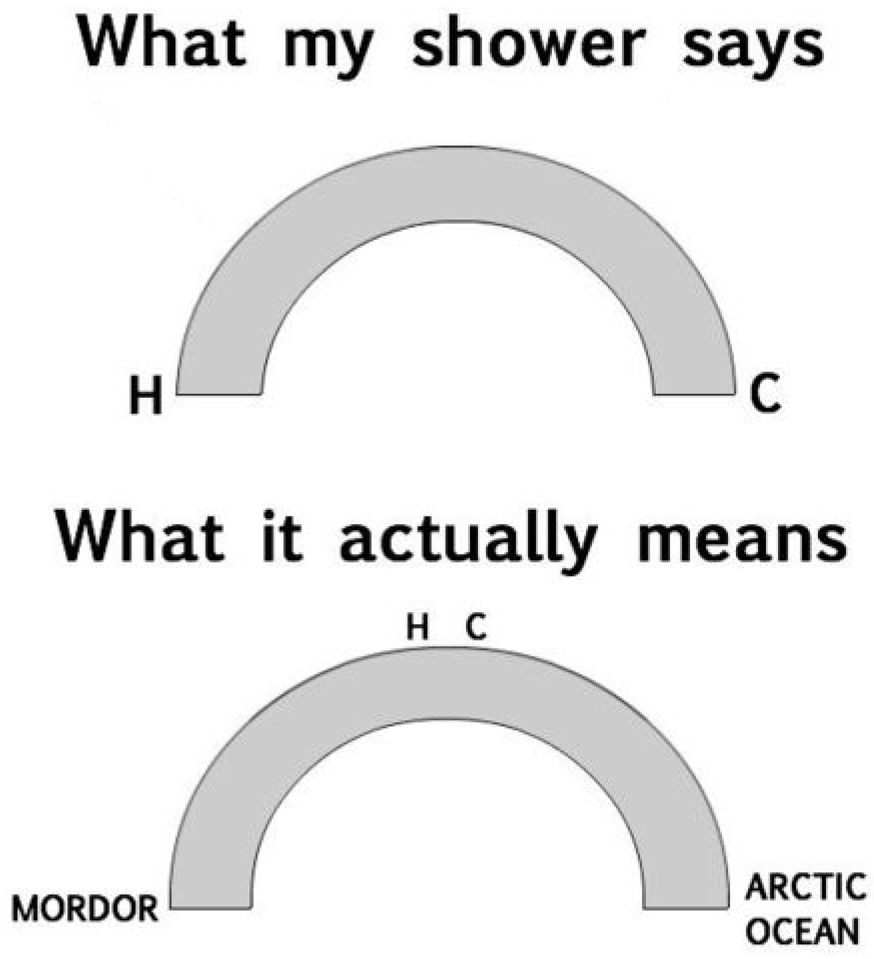
A quick aside

All of you are already familiar with programming languages, since you do Perl in this class already. Most of you know several others, and by now you've learned that the first step in learning a new language is just seeing how it does some of the things you're already familiar with differently.

With that in mind, we're going to go through a lot of the Javascript concepts with the understanding that you already know how programming works, you're just learning a new syntax.

Of course there will be new (and sometimes strange) things that are unique to Javascript, and I'll point these out, but for the most part you should be able to leverage your abilities in Perl and other languages to get moving with Javascript relatively quickly.

Sometimes documentation differs from the actual implementation.



Comments

Javascript has two different methods for commenting:

```
// Double forward slashes begins a comment, and everything to the end of the line is ignored


/*
    You can have multiple lines of comments using
    this syntax, where a slash+star begins a comment
    that is ended by a star+slash.
*/
```

As with all other languages, you should be sure to document your code well for your own sanity and the happiness of others.

Variables

Like Perl, variables in Javascript are not strongly typed. That means all you have to do is declare one and you can store anything you want in it, from integers to references to entire paragraphs of text. There are a *few* rules on their names:

- Case sensitive with upper, lower-case letters, numbers and the underscore
- They cannot begin with a number

You can declare variables with or without assigning them an initial value using the *var* keyword:

```
var seqName;
var seqLength = 50;
var accession = 'AT8234982';
var isProtein = false;
```

You'll notice from the above that numbers don't require quotes around them as strings do. Also, the keywords 'true' and 'false' can be used without quotes to indicate boolean values.

Variable types

While Javascript doesn't make the developer deal with strongly typed variables, they are still typed and handled in the background. You'll need to occasionally check a variables type or force conversions between types if you're trying to perform operations on multiple variables.

The **typeof** function will return the type of any given variable, and there are an array of type conversion functions when you need to force a value to be considered as a particular type:

```
var year = Number( “2010” );
var truthiness = Boolean(“42”);
var theAnswer = String(42);
```

In some cases Javascript will do this sort of *type casting* automatically, but it's best practice to do it explicitly.

Arrays

Arrays are collections of values accessed by their index position. You can declare them with (optionally) an initial size and then access and set their values using this 0-based ordinal positon:

```
var genes = new Array(3);
genes[0] = 'recA';
genes[1] = 'tonB';
genes[2] = 'dnaA';
```

You can also declare an array with the initial values instead:

```
var genes = new Array( 'recA', 'tonB', 'dnaA' );
```

The value of any given array position can be another array to create multidimensional data structures, with each tier accessed by index.

```
var matrix = new Array(2);
matrix[0] = new Array(3);
matrix[1] = new Array(3);
matrix[1][2] = 'foo';
```

The code above creates a 2x3 array and then shows the syntax for accessing the last element.

Object attributes and methods

Methods in Javascript are invoked on the object using '.' notation:

Object.method(parameters);

Attributes of objects also use the same notation, but without the () symbols.

Javascript allows you to create your own objects, and we'll cover this a bit later. I wanted to introduce the syntax now so you'd understand it for the built-in things in Javascript as we go along.

Array methods

Arrays in Javascript are a type of object and have attributes and methods.

```
var genes = ('recA', 'tonB', 'dnaA');


var geneCount = genes.length;   // returns the number of elements in an array
var genesSorted = genes.sort();  // sorts an array's elements (only the 1st dimension)
var geneStr = genes.join(“ “);     // merges array elements into a string


var gene = genes.shift();     // removes the first element from an array
var gene = genes.pop();       // removes the last element from an array
var gene = genes.unshift();       // adds an element to the front of an array
var gene = genes.push();      // adds an element to the end of an array
```

Fail.

Numeric operations

There's nothing really unique about how Javascript handles mathematical operations in terms of operators, orders of precedence, etc.

```
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
% (modulus)

++ (increment)
-- (decrement)
```

Comparisons / conditionals

Unlike Perl, Javascript uses the same comparison operators whether you're comparing number or string types. They are:

```
==    !=    <    >    <=    >=
```

There are also the usual logical operators to combine expressions:

&& (and) || (or)

The full structure of a conditional statement is:

```
if ( condition ) {
    // code here
} else if ( condition )  {
    // code here
} else {
    // code here
}
```

Loops

Javascript has familiar looping constructs like the C-style *for* loop and *while* loops.

```
for ( var i=0; i <= 10; i++ ) {
    // code here
}

var geneCount = 10;

while ( geneCount > 0 ) {
    // code here
    geneCount--;
}
```

If you want to make any loop skip immediately to the next iteration, you use the **continue** keyword. If you want to make a loop terminate, use the **break** keyword.

continue;	// like Perl's *next*
break;	// like Perl's *last*

Functions

Defining your own functions is relatively straightforward.

```
// absurdly simple function example
function calcSum( num1, num2 ) {
    var sum = num1 + num2;
    return sum;
}

var sum1 = calcSum( 10, 50 );
```

Of course, the function can just perform operations or write content to the browser, it doesn't have to return a value.

Printing content

This is actually a pretty large topic, as you could write out nearly all of your page content with just Javascript using several different methods. From manual writes to DHTML scripting, there are tons of resources online that go into these in detail. Since we're going to primarily use JQuery in this course, I'll demonstrate basic writes and debugging alerts and I'll save the rest of content generation discussion for the JQuery section ahead. Here's an example that prints content to a page you can use when developing:

```
<html>

<head><title>Test page</title></head>

<body>
    <script type="text/javascript">

    var hoursPerDay = 24;
    var msg = "There are " + hoursPerDay + " hours per day";

    // this writes content to the browser window
    document.write(msg);

    // this adds content to a pop-up window
    alert(msg);

    </script>
</body>

</html>
```

Other methods/functions

Here are some useful functions that didn't fit neatly into the previous slides:

isNaN(*var*)	// returns true if the passed var is not a number
eval(*string*)	// evaluates the passed string as an expression

Scratching the surface

There are semester-long classes devoted to Javascript, so I certainly can't summarize all the functionality in one series of slides. What I've tried to show so far is the core of the language, how expressions are written, and the basics you'll need to get started.

We'll use all this as we cover jQuery next, which has its own methods and syntax but still requires vanilla Javascript once you get into writing your own methods. We'll cover any new elements of syntax as they come up.

Javascript frameworks

There are a lot of Javascript frameworks with overlapping functionality. It's not at all uncommon for most web developers to be familiar with using several of them and use them regularly. Some of the most popular are below (with their demo pages):

Dojo - http://dojocampus.com/explorer/

ExtJS (now called Sencha) - http://dev.sencha.com/deploy/dev/examples/

Google Web Toolkit (GWT) - http://code.google.com/webtoolkit/examples/

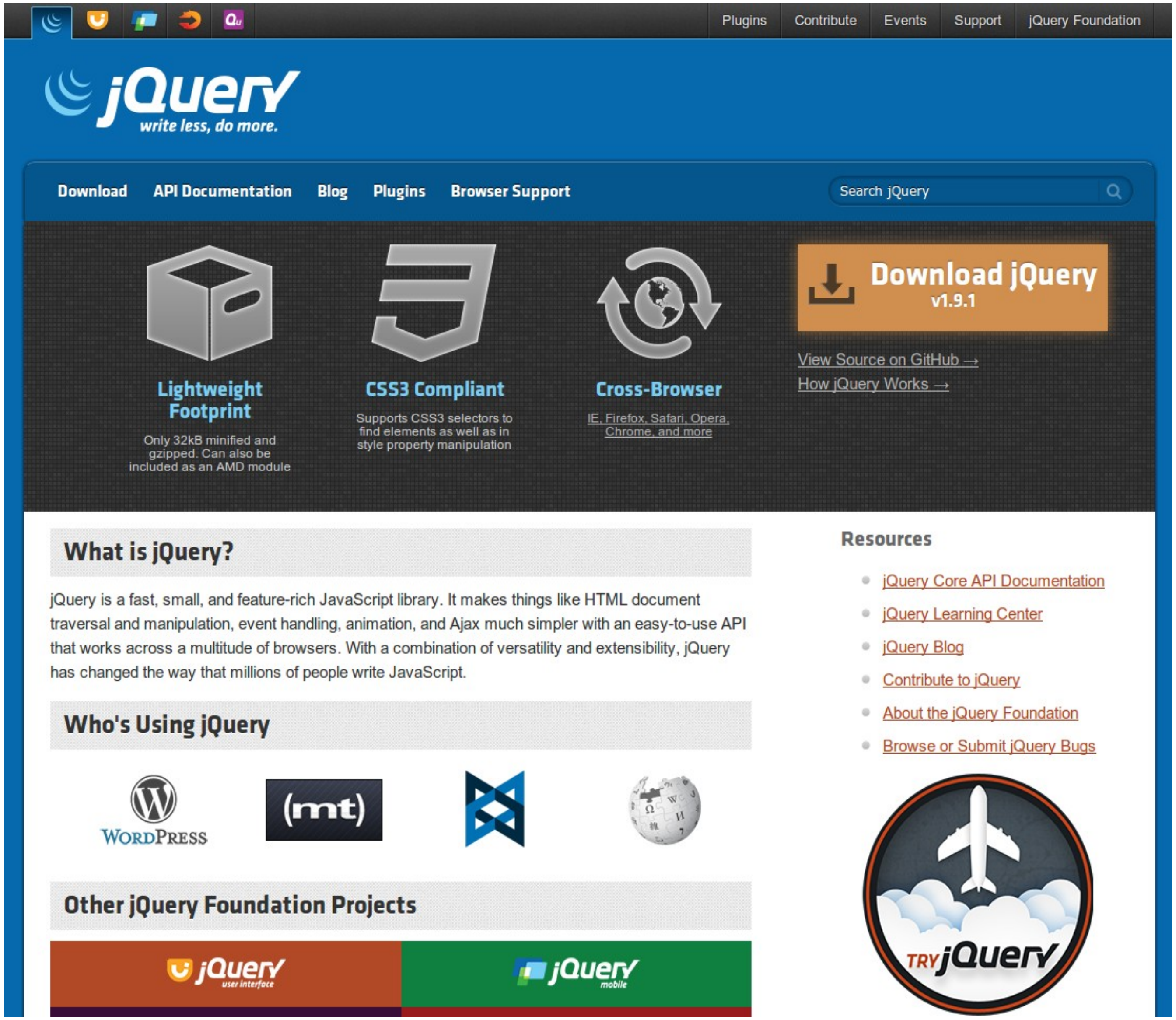
jQuery - http://ui.jquery.com/demos

MooTools - http://mootools.net/demos

Prototype + script.aculo.us - http://github.com/madrobby/scriptaculous/wikis/demos

YUI Library - http://developer.yahoo.com/yui/examples/

jQuery



Only 32kB minified and gzipped. Can also be included as an AMD module

Supports CSS3 selectors to find elements as well as in style property manipulation

IE, Firefox, Safari, Opera, Chrome, and more

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

- jQuery Core API Documentation
- jQuery Learning Center
- jQuery Blog
- Contribute to jQuery
- About the jQuery Foundation
- Browse or Submit jQuery Bugs

jquery.com

I settled on jQuery for this course because it has a mix of simple, accessible utility functions as well as more advanced features such as form validation, event handling, graphical layouts, AJAX development, widgets, etc.

It is very well documented and has a large user base.

Using jQuery

If you want to incorporate jQuery into a site you already have you can use the download links on their homepage. It has a builder that lets you get only the parts you're interested in so you don't have to deal with the overhead of the entire library if you don't need it (although even the whole thing is acceptably small.) Once you have your file downloaded you can bring it in using the <script> tag as we saw earlier. If you're using the HTML5 Boilerplate you should have it already. Look at this code found within the index.html there:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.js"></script>
<script>
    window.jQuery || document.write('<script src="js/libs/jquery-1.9.1.min.js">\x3C/script>')
</script>
```

That checks to see if it can pull jQuery hosted by Google's Libraries API and, if not, falls back to using a local copy distributed with the boilerplate. After that, you're ready to use it.

Example: jQuery selectors

One of the most basic things developers have to do in Javascript is get a reference to a particular HTML element on their page, either using their identifier or class. Using the ID was easier, but if you wanted to be safe you still had to try two different methods:

```
// the W3C way
if (document.getElementById) {
    return document.getElementById(obj);

// the way some other older browsers did it
} else if (document.layers) {
    return eval("document." + obj);
}
```

Doing it by class name wasn't really supported without a lot of trickery, which I'll post here in a super-tiny font:

```
function getElementsByClassName(oElm, strTagName, strClassName){
    var arrElements = (strTagName == "*" && oElm.all)? oElm.all : oElm.getElementsByTagName(strTagName);
    var arrReturnElements = new Array();
    strClassName = strClassName.replace(/\-/g, "\\-");
    var oRegExp = new RegExp("(^|\\s)" + strClassName + "(\\s|$)");
    var oElement;
    for(var i=0; i<arrElements.length; i++){
        oElement = arrElements[i];
        if(oRegExp.test(oElement.className)){
            arrReturnElements.push(oElement);
        }
    }
    return (arrReturnElements)
}
```

jQuery supports both of these access methods in a single line of code using jQuery *selectors*, since all the dirty work happens behind the scenes within the library ….

jQuery selectors

Instead of all the nonsense that used to be required on the previous slide, you can access elements on a page using jQuery the same way that CSS selectors work (in case you know that already):

```
var fooElm = $('#foo');    // fetches a reference to any element with an id of 'foo'
var geneElms = $('section.gene');  // gets refs to all <section> elements of class 'gene'
```

Let's look at this a bit closer. The first tricky thing is this little construct:

$(*selection*)

As you use jQuery your fingers will get used to typing that over and over again. When you include the jQuery library the $ symbol is hijacked to be an alias to jQuery *class* to save yourself typing, and so $() constructs a new jQuery object. When you pass that constructor values, it returns one or more elements that match that selection string.

What exactly are these values that you pass to select elements? They are modeled directly after CSS and Xpath selectors. These are described for jQuery in detail here, but I'll show some examples on the next slide:

http://docs.jquery.com/DOM/Traversing/Selectors

jQuery selectors

*	all elements
E	an element of type <E>
E.warning	an E element whose class is "warning"
E#myid	an E element with ID equal to "myid".
E,F,G	select all E elements, F elements, and G elements
E F	an F element descendant of an E element
E > F	an F element child of an E element
E + F	an F element immediately preceded by an E element
E ~ F	an F element preceded by an E element
E:nth-child(n)	an E element, the n-th child of its parent
E:first-child	an E element, first child of its parent
E:last-child	an E element, last child of its parent
E:only-child	an E element, only child of its parent
E:empty	an E element that has no children (including text nodes)
E:enabled	a user interface element E which is not disabled
E:disabled	a user interface element E which is disabled
E:checked	a user interface element E which is checked (for instance a radio-button or checkbox)
E:selected	a user interface element E which is selected (one or more option elements inside a select) - not in the CSS spec, but nonetheless supported by jQuery
E:not(s)	an E element that does not match simple selector s

So underground

“If a tree falls in the forest, and there is no one around to hear it …. a hipster will buy the soundtrack (and a matching forest-themed case for his iPad.)”

Role of Javascript

For a long time the majority of a page's appearance and layout was determined by the HTML alone. Later CSS became more common and controlled most of the appearance. The page was loaded, CSS applied, and then Javascript served a supplemental role for interactivity.

This is rapidly changing. CSS still defines the appearance and a layout of a page, but it is often now selectively controlled and applied by a Javascript layer or framework. The web server issues the content (HTML), the appearance definitions (CSS), and the functional code (Javascript) and it is only *after the HTML DOM loads* that the Javascript takes over, transforming the content layer and applying styles.

Of course, this makes the three technologies tightly linked – it is difficult to develop a modern site without having expertise in all three. It's not at all uncommon for the initial HTML portion of a page to be very minimal, and have then 90% of the visual content and appearance of the page generated on the client-side after the page load.

Because of this, you'll often find yourself attaching much of your code to an *event handler* executes once the page is ready for it. jQuery makes this easy.

jQuery ready function

The following complete page shows an example of the ready() function, which will only execute once the DOM is completely loaded. We use it here to attach an event to any <a> elements on the page to display an alert when clicked.

```
<html>
<head>

<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
        $("a").click(function() {
            alert("Hello world!");
        });
    });
</script>

</head>

<body>
    <a href="">Link example</a>
</body>

</html>
```

The syntax just uses a selector for the document as a whole that has as its argument a function definition:

$(document).ready(*function here*);

Effects example

Here's another example, this time using some built-in effects methods in jQuery. We create a link and a <section> element, then use the link to toggle the visibility of the section.

```
<html>
<head>

<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
        $("a").toggle(function(){
            $("section").hide('slow');
        },function(){
            $("section").show('fast');
        });
    });
</script>

</head>

<body>
    <a href="">Click me</a>
    <section>
        <p>Here is some content</p>
    </section>
</body>

</html>
```

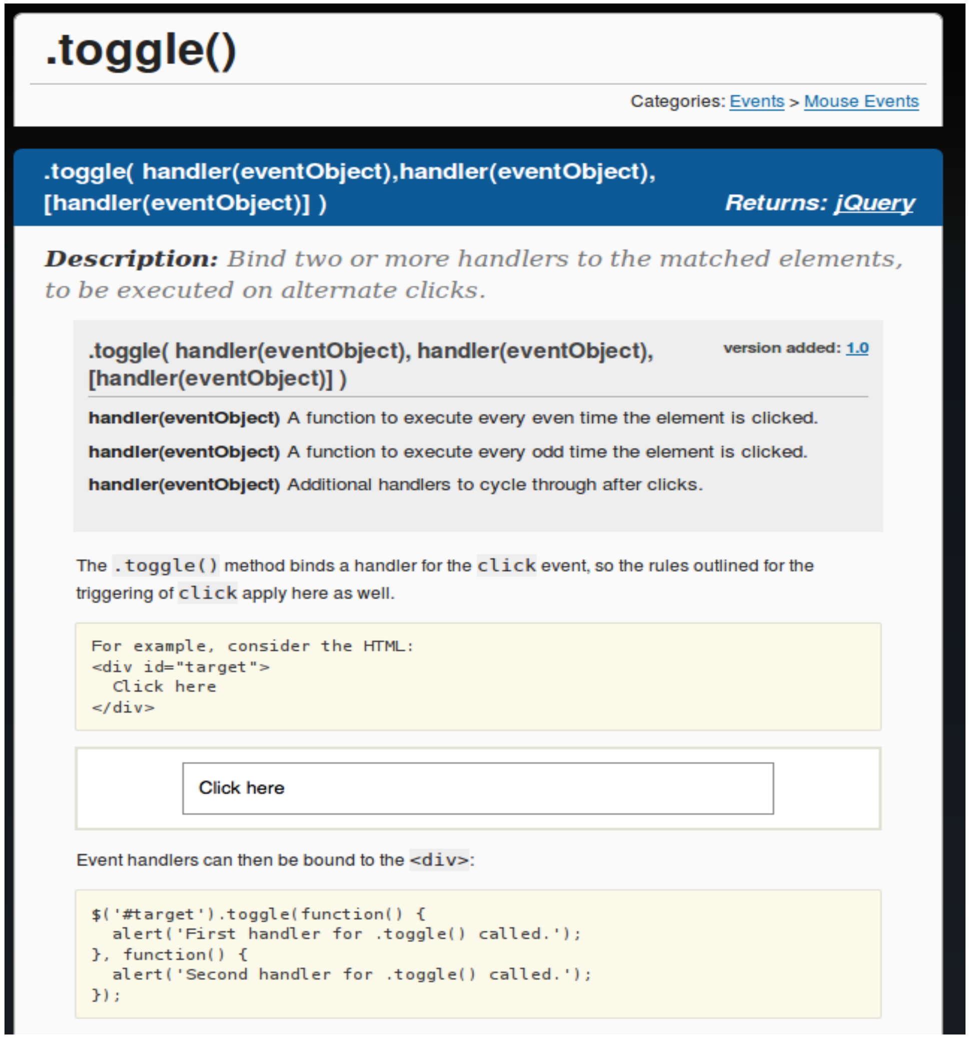
Effects example and jQuery docs

In the example on the previous slide we attached a toggle() method to attach function handles to alternate clicks of the link. The first time it is clicked the hide() method is called, and the show() method is called the next time. The easy use of selectors illustrates that we can perform actions on any element when any other element is clicked (or many other events.)

How do we learn all this? You could certainly buy a book, but most people find it more beneficial to browse the excellent documentation for the framework. Have a look at the API docs for the toggle event:

http://api.jquery.com/toggle-event/

The page has the formal syntax, examples, user comments, demos, etc.



.toggle()

Categories: Events > Mouse Events

.toggle(handler(eventObject),handler(eventObject), [handler(eventObject)]) Returns: jQuery

Description: Bind two or more handlers to the matched elements, to be executed on alternate clicks.

.toggle(handler(eventObject), handler(eventObject), [handler(eventObject)]) version added: 1.0

handler(eventObject)	A function to execute every even time the element is clicked.
handler(eventObject)	A function to execute every odd time the element is clicked.
handler(eventObject)	Additional handlers to cycle through after clicks.

The `.toggle()` method binds a handler for the `click` event, so the rules outlined for the triggering of `click` apply here as well.

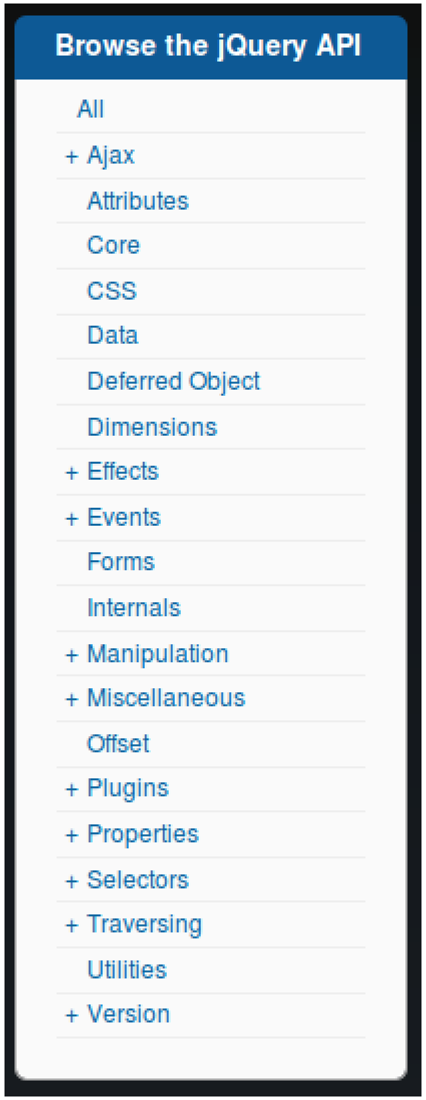
```
For example, consider the HTML:
<div id="target">
  Click here
</div>
```

Click here

Event handlers can then be bound to the `<div>`:

```
$('#target').toggle(function() {
  alert('First handler for .toggle() called.');
}, function() {
  alert('Second handler for .toggle() called.');
});
```

jQuery API



Documentation can often be dry, barely accessible reading one has to trudge through to tease some functionality out of code – often avoided unless absolutely necessary and certainly not used as a primary learning tool.

But the space of Javascript frameworks is a competitive one, and if users aren't quickly guided toward the right solution for their problem they'll simply find another framework.

Browsing the jQuery API pages is not only a great way to learn to use functionality you were specifically interested in but also to find features you didn't know were there.

The examples and user comments are probably some of the biggest strengths, and you can copy them into your code and modify as needed for your application.

The tree browser on the left highlights the general documentation sections for browsing.

http://api.jquery.com/

AJAX example

Several students have expressed interest in using AJAX in their final projects, but wanted to see an example of how it was done. I've created a demo that I'll describe here and then refer you to the code on the class server for the fine details.

What is AJAX? It stands for Asynchronous Javascript and XML and is, to simplify it, a method and protocol for sending a request to the server in the background while a user views the page and then process the data returned – all without exiting or reloading the page the user was on. At first a novelty, this technique is now an expected ability in the web developer's skill set.

In our example, we had a gene search interface from previous weeks' homework that queried a Chado database for a search term and displayed the results using HTML templating. This example modifies this, by dumping the template and passing data back to the browser using a format called JSON, then using Javascript to parse that JSON data and display the search results in a table.

The next slide has a comparison of the two different data flows:

AJAX example, data flows

Old way:

Our 'search.html' page held our form only and submitted our request to 'search_product.cgi'. This queried the database and populated an HTML template, displaying the results to the user directly on a new page.

New way:

Our 'search.html' has been expanded to have both the form and an initially-hidden result section. When the user submits a form, the 'search_product.cgi' still queries the database but now prints a JSON structure directly instead of loading a template. Javascript within the same search.html page reads this JSON file, parses it, and uses it to create the results table rows. No page reloads occur.

You can find the code on the server here:

```
/var/www/jorvis/unit09
```

You should know already how to use view this page in your browser as well. Give it a try, then step through it, paying attention to the comments as you go along. Understanding this flow is critical with modern web applications.

JSON format

The Javascript Object Notation format is a simple alternative to XML that provides nested data structures storing key/value pairs. A simple example:

{ “name” : “jorvis”,   “city” : “Tulsa”,   “state” : “OK }

Aside from key/value pairs, any value can also be an array of key/value pairs or other arrays.

The perl module JSON is used in our example to turn any arrayref or hashref into a JSON representation, and then we parse this using Javascript on the client side.

For more information see the JSON perldoc or even the wikipedia page here:

http://en.wikipedia.org/wiki/JSON

Soldier on

It is highly recommended to spend some time walking through the AJAX gene search example mentioned on the previous slides. From the HTML form to the server side perl code, database connectivity and querying, JSON formation, Javascript parsing and content generation – all these together form the groundwork of much more advanced web applications.

Mastering these and really getting a handle on how these technologies work together looks great on a resume, and opens a lot of opportunities for better data visualization and user interaction.

Again, go through this demo and use the jQuery API documentation when you see methods that are new. This is a fast way to learn the framework and get started using it immediately.

Further reading

Javascript event compatibility tables
    http://www.quirksmode.org/dom/events/index.html

“30 CSS selectors you Must memorize”
    http://net.tutsplus.com/tutorials/html-css-techniques/the-30-css-selectors-you-must-memorize/