Purpose of jQuery

- HTML – used to describe content of a page (.html file)
- CSS – used to describe how content should be displayed (.css file)
- Javascript – used for interaction with the page (.js file)
- jQuery – a Javascript library that makes javascript easy and manageable
Getting started with jQuery

- include reference to jQuery library in HTML
- jQuery interacts with DOM to access and modify HTML

- $(() = jQuery()
- document tells us that we're about to work our magic on the HTML
- .ready()'s parentheses is the jQuery event that occurs as soon as the HTML document is ready
Getting some elements

$( '#header' );  // select the element with an ID of 'header'

$( 'li' );     // select all list items on the page

$( 'ul li' );  // select list items that are in unordered lists

$( '.person' ); // select all elements with a class of 'person'
Did my selection get anything?

if ( $( '#nonexistent' ) ) {
  // Wrong! This code will always run!
}

if ( $( '#nonexistent' ).length > 0 ) {
  // Correct! This code will only run if there's an element in your page
  // with an ID of 'nonexistent'
}

if ( $( '#nonexistent' ).length ) {
  // This code will only run if there's a matching element
}
Getters, setters, and implicit iteration

• There are many methods you can call once you've made a selection. These methods generally fall into two categories:
  • getters
    • retrieve a piece of information from the selection
    • getters operate only on the first element in a selection
  • setters: setters alter the selection in some way
    • alter the selection in some way
    • operate on all elements in a selection, using what's known as implicit iteration
Examples for Setters

- Setters

  ```javascript
  $( 'li' ).html( 'New HTML' );
 
  $( 'li' ).html(function( index, oldHtml ) {
    return oldHtml + '!!!'
  });
  ```

  ```javascript
  $( 'li' ).each(function( index, elem ) {
    // this: the current, raw DOM element
    // index: the current element's index in the selection
    // elem: the current, raw DOM element (same as this)
    $( elem ).prepend( '<b>' + index + ':</b>' );
  });
  ```
Chaining

```javascript
$( 'li' )
  .click(function() {
    $( this ).addClass( 'clicked' );
  })
  .find( 'span' )
    .attr( 'title', 'Hover over me' );

• can call a series of methods on a selection
• Extensive chaining can make code extremely difficult to read
```
Creating new elements

- If you pass an HTML snippet to $(), it will create a new element in memory
- It won't be placed on the page until you place it on the page

```javascript
$( '<p>Hello!</p>' ); // creates a new <p> element with content

$( '<p>', {
  html: 'Hello!','
  'class': 'greet'
});
```
Traversal

• make an initial selection
• move through the DOM relative to that selection
Filtering selections

var listItems = $( 'li' );

// filter the selection to only items with a class of 'special'
var special = listItems.filter( '.special' );

// filter the selection to only items without a class of 'special'
var notSpecial = listItems.not( '.special' );

// filter the selection to only items that contain a span
var hasSpans = listItems.has( 'span' );
Tree Traversal Recap
Finding elements relative to a selection

// get the first list item on the page
var listItem = $( 'li' ).first(); // also: .last()

// get the siblings of the list item
var siblings = listItem.siblings();

// get the next sibling of the list item
var nextSibling = listItem.next(); // also: .prev()

// get the list item's parent
var list = listItem.parent();

// get the list items that are immediate children of the list
var listItems = list.children();

// get ALL list items in the list, including nested ones
var allListItems = list.find( 'li' );

// find all ancestors of the list item that have a class of "module"
var modules = listItem.parents( '.module' );

// find the closest ancestor of the list item that has a class of "module"
var module = listItem.closest( '.module' );
Getting back to your original selection

- jQuery stores a reference to your initial selection in case you want to get back to it
  
  ```javascript
  $( '#my-unordered-list' ).find('li')
  .addClass('special')
  .end()
  ```
  
  // now we're working with the list items
  
  ```javascript
  .addClass('super-special');
  ```

- use the jQuery .end() method to get back to your original selection

  ```javascript
  // now we're back to working with the list
  ```

- use it sparingly
Altering elements

- Whenever possible, you should use classes combined with CSS rules to affect the presentation of elements, and use jQuery only to add and remove those classes as shown above

  ```javascript
  $( 'li' ).addClass( 'hidden' );
  $( 'li' ).eq( 1 ).removeClass( 'hidden' );
  $( 'li' ).eq( 1 ).toggleClass( 'hidden' );
  ```
Placing elements in the document

- consider the case where you want to move the first list item in a list to the end of the list. There are several ways to achieve this:
  - `appendTo()`
  - `.append()`
  - `.insertAfter()`
  - `.after()`

```javascript
var listItem = $( '#my-unordered-list li' ).first();
listItem.appendTo( '#my-unordered-list' );
$('#my-unordered-list').append(listItem);
```
Removing elements

- `.remove()`
  - used to remove elements permanently

- `.detach()`
  - temporarily removing elements from the document

- `.replaceWith()`
  - replaces an element or elements with the element or HTML passed as an argument

```javascript
var removedListItem = $( '#my-unordered-list li' ).first().remove();

var detachedListItem = $( '#my-unordered-list li' ).first().detach();

var replacedListItem = $( '#my-unordered-list li' ).first().replaceWith( '<li>new!</li>' );
```
Events and Event Delegation

- selects all list items on the page, then binds a handler function to the click event of each list item using jQuery's .click() method.

```javascript
$('thingToAffect').click();

$( 'li' ).click(function( event ) {
    console.log( 'clicked', $( this ).text() );
});
```
Some jQuery methods

<table>
<thead>
<tr>
<th>Native Event Name</th>
<th>Shorthand Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>click</td>
<td>.click()</td>
</tr>
<tr>
<td>keydown</td>
<td>.keydown()</td>
</tr>
<tr>
<td>keypress</td>
<td>.keypress()</td>
</tr>
<tr>
<td>keyup</td>
<td>.keyup()</td>
</tr>
<tr>
<td>mouseover</td>
<td>.mouseover()</td>
</tr>
<tr>
<td>mouseout</td>
<td>.mouseout()</td>
</tr>
<tr>
<td>mouseenter</td>
<td>.mouseenter()</td>
</tr>
<tr>
<td>mouseleave</td>
<td>.mouseleave()</td>
</tr>
<tr>
<td>scroll</td>
<td>.scroll()</td>
</tr>
<tr>
<td>focus</td>
<td>.focus()</td>
</tr>
<tr>
<td>blur</td>
<td>.blur()</td>
</tr>
<tr>
<td>resize</td>
<td>.resize()</td>
</tr>
</tbody>
</table>
On() event handler

• What if you interact with items that weren’t there when the DOM was loaded?

• We will need a new event handler: .on(). You can think of .on() as a general handler that takes the event, its selector, and an action as inputs.

```javascript
$(\'.item\').click(function() {
    $(this).remove();
}); // will not work

$(document).on('event', 'selector', function() {
    Do something!
});

$(document).on('click', \'.item', function() { $(this).remove() });
```
Namespace events

- Not using namespace events
  ```javascript
  ```
  ```javascript
  $( 'li' ).on( 'click', function() {
    console.log( 'a list item was clicked' );
  });
  ```
  ```javascript
  $( 'li' ).on( 'click', function() {
    registerClick();
    doSomethingElse();
  });
  ```
  ```javascript
  $( 'li' ).off( 'click' ); // unbind all click handlers on all li elements
  ```

- Using Namespace events, allows for finder control
  ```javascript
  $( 'li' ).on( 'click.logging', function() {
    console.log( 'a list item was clicked' );
  });
  ```
  ```javascript
  $( 'li' ).on( 'click.analyFcs', function() {
    registerClick();
    doSomethingElse();
  });
  ```
  ```javascript
  $( 'li' ).off( 'click.logging' ); // will leave analytics-related click untouched
  ```
Event object

• Whenever an event is triggered, the event handler function receives one argument, an event object that is normalized across browsers

```javascript
$( document ).on( 'click', function( event ) {
  console.log( event.type );  // The event type, eg. "click"
  console.log( event.which );  // The button or key that was pressed.
  console.log( event.target );  // The originating element.
  console.log( event.pageX );  // The document mouse X coordinate.
  console.log( event.pageY );  // The document mouse Y coordinate.
});
```
Review

```javascript
$(document).ready(function() {
  $('thingToTouch').event(function() {
    $('thingToAffect').effect();
  });
});
```
Inside the event handler

• When you specify a function to be used as an event handler, that function gets access to the raw DOM element that initiated the event as this

```
$( 'input' ).on( 'keydown', function( event ) {
  // this: The element on which the event handler was bound.
  // event: The event object.

  // Change the input element's background to red if backspace was pressed, otherwise green.
  $( this ).css( 'background', event.which === 8 ? 'red' : 'green' );
});
```
Preventing the default action

- Often, you'll want to prevent the default action of an event; for example, you may want to handle a click on an a element using AJAX, rather than triggering a full page reload.

```javascript
$( 'a' ).on( 'click', function( event )
{
 // Prevent the default action.
 event.preventDefault();
 // Log stuff.
 console.log( 'I was just clicked!' );
});
```
Event bubbling

• What happens when you click on an a element that's nested inside other elements?
• In fact, the click event will be triggered for the a element as well as for all of the elements that contain the a — all the way up to the document and the window.

• `<a href="#foo"><span>I am a Link!</span></a>`
• When you click on "I am a Link!", you are not actually clicking on an a, but rather on a span inside of an a.
Event delegation

- it allows us to bind fewer event handlers than we'd have to bind if we were listening to clicks on individual elements, which can be a big performance gain

- it allows us to bind to parent elements — such as an unordered list — and know that our event handlers will fire as expected even if the contents of that parent element change

```javascript
$( '#my-unordered-list' ).on( 'click', function( event ) {
  console.log( event.target ); // logs the element that initiated the event
});

$('#my-unordered-list').on( 'click', 'p', function( event ) {
  console.log(event.target);
});
```
Resources

• Slides based on http://jqfundamentals.com/
• jQuery hands-on tutorial on code academy (3 hours)
• http://www.w3schools.com/jquery/jquery_examples.asp