# jQuery

*Notes open for creative commons use @ developer blog: https://unfoldkyle.com, github:* SmilingStallman*, email: kmiskell@protonmail.com*

**Learning Resources**
https://www.tutorialspoint.com/jquery/index.htm
https://api.jquery.com/
https://www.tutorialsteacher.com/jquery/jquery-tutorials

## Intro to jQuery

-JavaScript library that allows for selector shorthand, ajax shorthand, DOM manipulation, special effectes, event handling functions, etc.. One of the most widely used libraries, even in 2020.

-Easy to write, excellent API documentation, cross-browser short, shortens code

-jQuery all in one file. Can include in page either by downloading and including as *<script>* or linking to online hosted version:
*<script type="text/javascript" "src=https://code.jquery.com/jquery-3.3.1.min.js"></script>*

-Selector style matches CSS

**Document Ready**
-Since want jQuery to run after page has loaded and elms are available, suggested to put jQuery code in global function:

```
$(document).ready(function(){
        //jQuery
);
```

-Compared to core JS *window.onload, $document.ready(…)* provides improved speed, as it loads when full DOM hierarchy has been loaded vs entire window

**Basic Selectors**
-Selectors wrapped inside *factory function,* which is a shorthand similar to *document.querySelector('selector')*

-Syntax: *$('selector')*

-class: *$('.some-class')*
-id: *$('#some-id')*
-element: *$('some-element')* *//ex. $('p') selects all <p>*

-Multiple elements seperated by combos

-*$(this)* – used to select current element(s). Often used inside handling function

-ex. *$("li").click(function () {*
*if ($(this).is(":first-child"))*
*$("p").text("This is list item 1");*
*}*

## Selectors

**Advanced Selectors**
-class: *$('some-element.some-class')*
-element of class: *$('some-element.some-class')*
-ID: *$('#some-id')*

-elm with attribute of value: *$('elm[attr="something"]')* *//useful for selecting by <input> name*
-attribute exists: *$('["some-attr"]')*

-even/odd elms: *$('elm:odd')*
*-first of type: $('.some-elm:first')*
*-not: $('something:not(something)')*

-descendent: *$('some-elm another-elm')*
-child: *$('some-elm > another-elm')*

-inputs (ex. *text, password,* etc., as well as attributes, *selected, checked, hidden,* etc.):
*$(':input')* *$(':password')* *$(':checked')* *$(':hidden')* *etc.*

-other standards css, like *:nth-child(#),* also apply

## Attributes

**General**
-jQuery has functions to read, modify, etc. DOM node attributes (ex. *src, title, class,* etc.)

-Read attr value: *$('selector').attr()*

-Set attr value: *$('selector').attr('attr-name', 'val')*
ex. *$("#myimg").attr("src", "/jquery/images/jquery.jpg")*

-Remove attr: *$('selector').removeAttr()*

**Class**
-Manipulating classes for nodes useful for dynamically adding/removing existing styles defined to apply to *class-name* on some action

-Has class: *$('selector').hasClass("class-name")*
-Add class: *$('selector').addClass("class-name")*

-Remove class: *$('selector').removeClass("class-name")*

-Toggle class: *$('selector').toggleClass("class-name")*
-If has class, removes. If does not have class, adds.

**HTML, Text, & Vals**
-Inner HTML = content inside tags. ex. * <h4>inner html</h4> *
-* * inner html would be *<h4>inner html</h4>*

-Get inner HTML: *$('selector').html()*
-Set inner HTML: *$('selector').html('val')*

-*html()* treats string passed into it as html, meaning can use it to added children to nodes
-ex. *$("#div1").html('<a href="example.html">Link</a>hello');*

-*html()* methods returns string, and converts string to DOM node when setting html

-*text()* used to access inner html text of node. Returns string.
-ex. *<h4>inner html</h4>*

-*text()* methods return and interact with strings

-Get node text: *$('selector').text()*
-Set node text: *$('selector').text('some-string')*

-*val()* used to interact with nodes that hold value, but not enclosed in tags, mostly *<input>*. Returns string, num, or array, depending on value.

-Get value: *$('selector').val()*
-Set value: *$('selector').val('some-value')*

## CSS

-jQuery can set both individual CSS rules and multiple rule declarations

-Get CSS prop value: *$('selector).css('prop-name')*

-Set rule: *$('selector').css('prop-name', 'prop-value')*
-Set rule block: *$('selector).css('{'propA':'valA', 'propB':'valB', …})*

-Set width shorthand: *$('selector').width(optional-n) //no arg returns width*
-Set height shorthand: *$('selector').width(optional-n)*

-Also additional functions to work with inner height/width (excludes border/padding), outer height/width, and get scroll top/lef offsets (for use with scroll bar position with scrolling)

# DOM Interaction

**DOM Traversal**
-jQuery lets you traverse child nodes by index, where first child is 0
-Access child of index *n*: *$('selector').eq(n)*

-Get all direct children: *$('selector').children('optional-filter-selector')*

-Get parent: *$('selector').parent('optional-filter-selector')*

-Get all siblings: *$('selector').siblings('optional-filter-selector')*

-Also additional methods for selecting all parents up the DOM tree, only previous/next siblings, etc.

**DOM Filtering**
-*filter()* filters out all nodes of a given selector, returning only those that match the filter selector
-Syntax: *$('selector').filter('another-selector')*
-Useful to *filter()* then call some modification, etc. method on results

-*filter()* can also take in a function that takes in *index* arg (the current matching selected node) and returns boolean. *False* nodes will not be returned from *filter()* call, which *true* will.

-*find()* used to return all child elements of a specific type from a parent
-Syntax: *$('selector').find('some-tag')*

-*is()* used for "*if selector matches*," returning boolean
-Syntax: *$('selector').is('another-selector')*

-*not()* for not selector
-Syntax: *('selector').not('another-selector')*

-*slice()* for child slice x:y (exclusive y)
-Syntax: *('selector').slice('x, y')*
ex. *('.my_list li).slice(2, 5).addClass('new')*

**DOM Removal**
-See above *html()* notes for manipulating inner HTML of elements

-Replace element: *$('selector').replaceWith('html')*

-*remove()* removes single element, as well as bound events
-Syntax: *$('elm-selector').remove('optional-filter')*

-*empty()* removes element and all children
-Syntax: *$('elm-selector').empty()*

**DOM Insert**
-Insert before selector: *$('selector').before('html content')* *//arg becomes sibling*
-Insert after selector: *$('selector').after('html content')*

-*wrap()* wraps selected nodes with specified single element. Arg becomes parent of selector.
-Syntax: *$('selector').wrap('html element')*
-ex. *$('#my_image').wrap(' ')* *//results in <img> *

-*wrap()* wraps each matching instance of selector in specified html/element. *wrapAll()* wraps all grouped (sibling) matching elm instances in one html/element.
-ex. *$('li').wrapAll('<ul></ul>')* *//group of 10 <li> now wrapped in <ul></ul>*

-*append()* is reverse of wrap, where instead of wrapping selected with arg, arg is added as a child for selector, wrapped by selector. Arg becomes child of selector.
-Children arranged in 1, 2, 3, 4, 5 order, where children added as "next" sibling
-Syntax: *$('selector').append('html content')*
-ex. *$('#my_span').append(' ')*
*//if call twice:    *

-*prepend()* does same as *append()* with same syntax, but children arragned in 5, 4, 3, 2, 1, where child added as "previous" sibling, hence first child is most recent added

-*clone()* used for duplication existing element(s). Can tree of elements.
-Syntax: *$('selector').clone(optional-boolean).someInsertMethod('selector')*
-Clones first selector, then call *before()*, *after()*, *replaceWith()*, etc. on return
-If pass in *true* also copies event handlers for element(s)

-*wrapInner()* wraps inner html content (ex. text inside * )*
-Useful for if want to wrap in *<bold></bold>*, etc.
-Syntax: *$('selector').wrapInner('html content')*

-Also methods such as *$('selector').insertAfter($('selector'))*, *prependTo($('selector'))*, etc., that are called on *selector* and and appended, etc. this to arg *selector*

## Event Handling

**Event Binding**
*$('selector').bind('eventType', optional-data, handlingFunction);*

-Handling function should take in single *event* arg, if wish to use *event*
-if passed in *optional-data* can access from inside *handlingFunction* via *event.data*
-Standard event methods can also be called on *event ( .preventDefault()*, etc.)

-If referencing *event.target* from inside handling function and with to call jQuery method on *target*, reference via *$(event.target)*

**Event Removal**
*$(selector).unbind('eventType', optional-handler)*

-If passed in function by name as arg during bind, can remove specific function by passing name in as *optional-handler* arg

**Event Triggering**
-*trigger()* triggers event
-Syntax: *$('selector').trigger('eventType', optional-data)*

-Other methods, such as *hover()* to simulate hover

**Event Types**
-All following can be passed into *bind()*, etc.

*blur change click dblclick focus keydown keyup*
*keypress mousedown mousemove mouseout mouseover mouseup*
*scroll select submit unload*

*error //error in loading or unloading*
*load //doc finishes loading*
*mouseenter mouseleave //mouse enters/exists elm box*
*resize //window resized*

**Event Props**
-Global props useful for event handling, accessing by calling *event.propName* from handling function

*altKey //true if alt key was pressed when event triggered*
*ctrlKey //"" but with ctrl key*
*metaKey //"" but with meta key*
*shiftKey //"" but with shift key*

*keyCode //key pressed*
*pageX //horizontal coordinate of mouse at event time relative to page origin*
*pageY //vertical coordinate " "*
*screenX //horizontal coordinate of mouse at event time relative to screen origin*
*screenY //vertical coordinate " "*
*target //node elm that triggered event*
*timeStamp //time in ms event occurred*
*type //type of event*
*which //numeric code for keyboard or mouse (1 left, 2 center, 3 right)*

**Shortcut methods**
-Shortcuts to *bind()* where don't need to pass in *eventType*.
-Cannot take in *optional-data* like *bind()* can
-If called with no args, trigger event. If called with function, bind event of that method type to selector
-ex. *$('#myButton').click(someFunction)*
*$('#myButton').click()*

*blur(optional-funct) change(optional-funct) click(optional-funct)*
*dblclick(optional-funct) error(optional-funct) focus(optional-func)*
*keydown(optional-funct) keypress(optional-funct) keyup(optional-funct)*

| | | | |
|---|---|---|---|
| *load(funct)* | *mousedown(funct)* | *mouseenter(funct)* | *mouseleave(funct)* |
| *mousemove(funct)* | *mouseout(funct)* | *mouseover(funct)* | *mouseup(funct)* |
| *resize(funct)* | *scroll(funct)* | *select(optional-funct)* | |
| *submit(optional-funct)* | | *unload(optional-funct)* | |

# AJAX

**AJAX 101**
-Asynchronous JavaScript and XML

-Major browsers implement *XMLHttpRequest* object, which allows object holding data to be sent on page without page reload

-JavaScript library AJAX uses *XMLHttpRequest API* to send such data. JavaScript sends request to specified url (ex. api url, php file), which gets data sent by Ajax, then sends back response.

-jQuery provides standard library of Ajax *get*, *post*, etc. methods for simplified Ajax use.
-jQuery Ajax can also send/receive plain html and json, which is more commonly done than xml

-GET - Use when only getting data from server, but not modifying it.
-POST – use when changing data on the server.

-Available data types:
- *-text* – transporting strings
- *-html* – transporting blocks of html
- *-script* – adding a new script to the page
- *-json* – transporting json formatted data
- *-jsonp* – transporting json formatted data to external domain
- *-xml* – transporting *xml* data

-Ajax is asynchronous, Ajax functions thus also must take a callback, which gets the *response* as the arg for handling *response* when asynchronous call finishes:

```
$.get( "foo.php", function( response ) {
    console.log( response );
});
```

**Ajax Methods**
**-**95% of the time only need to use $.*ajax()* method.

-Syntax:

```
$.ajax({
        url: ‘myPath.php’,
        data: myData,           //ex. { id: 23 }
        type: ‘httpMethod’,     //ex. POST
       dataType: ‘dataType’     //ex. json
```

```
        //additional options here
}).done( function( response ){
        //runs if successful response
}).fail( function( xhr, status, errorThrown ){
        //runs if call fails
        //above attributes available to handling logic
}).always( function( xhr, status ){
        //runs regardless of fail or success
});


$.ajax({
    type: 'POST',
    url: 'myurl.com/api'
    data: $('#my_form').serialize(),
    dataType: 'json',
    success: function(data){
        //success logic
    },
   error: function(response){
        //failure logic
    },
    always: function(response){
        //always logic
    }
});
```

-GET: *$.get(location, optional-data, callback-on-success(data), optional-data-type-returned')*
-ex. *$.get("result.php", { name: "Zara" }, data => $('#stage').html(data), json);*

-GET: *$.post(location, opt-data, callback-on-success(data), opt-data-type-returned')*
-ex. *$.post("updateCustomer.php", { name: "Neil" }, data => $('#stage').html(data), json);*

-optional data type returned: *'xml','html', 'script', 'json', 'jsonp', 'text'*

-LOAD: *$('selector').load(location, optional-data, optional-callback(data))*
-similiar to GET except, loads data returned directly into *selector* element(s)
-Useful for filling in text elements with simple strings, etc.

-GETJSON: *$.getJSON(location, optional-data, optional-callback(data))*
-Gets JSON data from server location, then passes into *callback*, then can access JSON via *data.propName* inside *callback*
-Useful for filling out text elements with multiple JSON props

-GETSCRIPT: *$.getScript('location.js', optional-callback)*
-Gets javascript, loads, and executes

**Ajax Options**
-Can pass in additional options to ajax besides required *url*, *type*, etc.

-*async: false* – AJAX runs without asynch calling

-*crossDomain: true* – forces cross domain call. For use with *jsonp*, etc.
-*dataType: 'type'* - expected response type from server. x*ml, json, script, html.*

-*headers: {//object}* – holds key/value header pairs sent with request
-*password: 'someString'* - sends pass to be used in auth during call
-*username: 'someString* – sends username to be used in auth during call

-*always: someFunction(response){}* - runs when ajax completes
-success*: someFunction(response){}* - runs if ajax request fails
-*error: someFunction(response){}* - runs if ajax request fails
-*timeout: someNum* – how long to wait in ms before considering request failed

**Ajax & Forms**
-*serialize()* - jQuery method can be called on a form (ex. *$('#edit_vdp_goal_form').serialize() )* to convert form inputs into query string

-*serializeArray()* - jQuery method can be called on form to convert inputs to array of objects, each object with *name* and *value* key-value pairs

-For client-side JS validation, call event handler *submit* on form and check validation logic only calling *$.ajax(….)* if validation passes

```
$( "#form" ).submit( function( event ) {
    if( //validation condition ) { event.preventDefault() }
    else { $.ajax(….) }
}
```

## Effects

**Showing & Hiding**
*$('selector').show(optional-ms-speed, optional-completion-callback)*

*$('selector').hide(optional-ms-speed, optional-completion-callback)*

*$('selector').toggle(optional-ms-speed, optional-completion-callback)  //hide if shown and vice versa*

-No *speed* causes to immediately hide or show

**Fading**
-Methods take in ms and change opacity from 1 to 0 (or vice versa). When 0 opacity, sets to *display: none*

*$('selector').fadeOut(speed-ms, optional-completion-callback)*

*$(‘selector’).fadeIn(speed-ms, optional-completion-callback)*

*$(‘selector’).fadeTo(speed-ms, opacity, optional-completion-callback)*
-fades to *opacity* over *speed-ms*

-If fadeded out using *fadeTo()* also need to fade in using

**Sliding**
-When slid up, element is display none. Once slide down, element slides by setting *overflow: none*, then expanding height to full height of element over *ms*. Result is that more of element is “revealed” as height increases and hidden *overflow* is shown.

*$(‘selector’).slideDown(speed-ms, optional-completion-callback)*

*$(‘selector’).slideUp(speed-ms, optional-completion-callback)*

*$(‘selector’).slideToggle(speed-ms, optional-completion-callback)*

**Animate**
-For creating custom animations. Take in object with CSS rules and speed, then applies rules over given time
-useful for items with *absolute*, etc. positions, moving them around with *left, tosp*, etc., as well as size, opacity, changing display, etc.

*$(‘selector’).animate({cssName: ‘cssVal’}, optional-speed-ms, optional-completion-callback)*

**Disable Animations**
-To disable/enable all jQuery animations, set boolean for global jQuery variable *jQuery.fx.off*
-*false* to enable, *true* to disable