# Introduction to jQuery

This workbook provides a basic introduction to jQuery. jQuery is a JavaScript library that simplifies JavaScript programming. jQuery is most effective when combined with JavaScript. jQuery provides ways to select and modify Elements in the DOM Tree and to handle events. It also avoids issues associated with cross-browser compatibility and inconsistent handling of whitespace nodes, which we have come across in earlier Workbooks.

This Workbook assumes that you are familiar with JavaScript. It assumes that you have already completed the Introduction to JavaScript, JavaScript and the Document Object Model and JavaScript Events and Forms Workbooks.

This Workbook includes explanations, code descriptions and code examples, which you should read. It also includes activity sections which ask you to carry out a set of actions, for example to create or modify some code. The activities reinforce what you have read, but also give you vital coding practice. The workbook is design to be read in sequence, if you skip parts it may make it harder to understand later parts.

## Resources

Whilst the workbook includes everything you need to cover, you may find it useful to refer to other sources of information. The w3Schools website has a course on jQuery, as does the codeacademy website. Our library also has books on jQuery.

You may also find the following resources useful:

http://api.jquery.com

https://learn.jquery.com/

## Including the jQuery library

Because jQuery is a JavaScript library, we need to explicitly include it in our HTML so that we can make use of it. There are two ways we can do this.

First, we can download the library from https://jquery.com/download/ and host it on our server. We can then link to it from our HTML page. Note: it is best to include the link to the library at the end of the <body>, but before any link to our own code. This means that the DOM Tree is mostly complete and the jQuery library is loaded before any of our code runs. The current version of the jQuery library is jquery-3.3.1.min.js

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>jQuery Library Example</title>

</head>

<body>

<!-- my HTML in here -->

**<script src="jquery-3.3.1.min.js"></script>**

<script src="mycode.js"></script>

</body>

</html>

The alternative is to make use of a copy of the library hosted on an external Content Delivery Network (CDN). This can lead to faster access and improved performance, particularly where users are distributed across the globe. There are several different CDNs which we can use, including Google and Microsoft. The example used here is from StackPath.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>jQuery Library Example</title>

</head>

<body>

<!-- my HTML in here -->

**<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous"></script>**

<script src="mycode.js"></script>

</body>

</html>

A potential disadvantage of using a CDN is that the CDN may be unavailable, or access may be restricted from some countries. For this reason, the best approach is to use a CDN if possible, but to provide a local fallback if required.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>jQuery Library Example</title>

</head>

<body>

<!-- my HTML in here -->

<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous"></script>

**<script>**

**window.jQuery || document.write('<script src="jquery-3.3.1.min.js "><\/script>')**

**</script>**

<script src="mycode.js"></script>

</body>

</html>

Here we can see that the script to load the library from the CDN is first. There is then a small script which executes. If the jQuery library has not been loaded, it writes

<script src="jquery-3.3.1.min.js "></script>

into the HTML, which is then executed and causes the local copy of the jQuery library to be loaded.

In some browser developer tools, this will generate a warning about potential vulnerability due to cross-site scripting (even though this is not cross-site). There is some debate about whether or not this warning should be generated and this still appears to be the preferred approach for providing this fallback.

## Basic jQuery notation

jQuery is very terse and looks rather cryptic at first encounter. The most important function is jQuery() which allows us to select one or more elements from the DOM Tree. The jQuery() function returns a jQuery Object which holds references to those elements. For example:

jQuery('li')

returns a jQuery Object containing **all** the <li> Element Nodes in the Dom Tree. The equivalent in JavaScript would be:

document.getElementsByTagName('li')

jQuery allows us to use $ as a shorthand, so rather than writing jQuery we would have:

$('li')

Once we have our jQuery Object we can manipulate it in various ways using jQuery methods. For example:

$('li').attr('class', 'full');

changes the class of all <li> Element Nodes to 'full'. The equivalent in JavaScript would be something like:

var els = document.getElementsByTagName('li');

var noOfElements = els.length;

for (var i = 0, i < noOfElements; i++) {

els[i].className = 'full';

}

which illustrates how compact jQuery can be, compared to JavaScript.

We can also *chain* jQuery update methods when they are working on the same jQuery Object. For example:

$('li').attr('class', 'full').hide().delay(500).fadeIn(1400);

changes the class of all <li> Element Nodes to 'full', then hides them (makes them non-visible), waits 500 milliseconds, then fades them back in over 1,400 milliseconds. In order to make chained methods more readable we can write them as:

$('li')

.attr('class', 'full')

.hide()

.delay(500)

.fadeIn(1400);

Don't forget that terminating semi-colon. Note also that the .delay() method only applies to certain jQuery items, effects items. We will discuss these more later.

**Activity**

1. Create a new HTML file containing the following mark-up:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Catalogue</title>

<link rel="stylesheet" href="jQcss.css">

</head>

<body>

<div id="page">

<h1 id="header">Course Catalogue</h1>

<h2>Course Status</h2>

<ul>

<li id="one" class="running new"><em>new</em> BSc Computer Networking</li>

<li id="two" class="running">BSc Computing</li>

<li id="three" class="full">BSc Informatics</li>

<li id="four">BSc Mechatronics</li>

<li id="five" class="new"><em>new</em> BSc Big Data</li>

</ul>

</div>

</body>

</html>

1. Create a new css, containing the following CSS rules:

body {

background-color: hsl(0, 0%, 0%);

font-family: sans-serif;

margin: 0;

padding: 0;}

#page {

background-color: hsl(290, 5%, 50%);

margin: 0 auto 0 auto;}

h1 {

color: hsl(197, 50%, 90%);

margin: 0 auto 0 auto;

padding: 30px 10px 20px 10px;

text-shadow: 2px 2px 1px hsl(290, 5%, 35%);}

h2 {

color: hsl(176, 50%, 90%);

margin: 0 0 10px 0;

padding: 0px 10px 20px 10px;

text-shadow: 2px 2px 1px hsl(290, 5%, 35%);}

ul {

background-color: hsl(290, 5%, 50%);

border: none;

padding: 0;

margin: 0;}

li {

background-color: hsl(290, 5%, 80%);

color: hsl(360, 100%, 100%);

border-top: 1px solid hsl(290, 5%, 90%);

border-bottom: 1px solid hsl(290, 5%, 70%);

list-style-type: none;

text-shadow: 2px 2px 1px hsl(290, 5%, 70%);

padding-left: 1em;

padding-top: 10px;

padding-bottom: 10px;}

.running {

background-color: hsl(155, 60%, 50%);

text-shadow: 2px 2px 1px hsl(155, 60%, 40%);

border-top: 1px solid hsl(155, 60%, 60%);

border-bottom: 1px solid hsl(155, 60%, 40%);}

.full {

background-color: hsl(12, 60%, 50%);

text-shadow: 2px 2px 1px hsl(12, 60%, 40%);

border-top: 1px solid hsl(12, 60%, 60%);

border-bottom: 1px solid hsl(12, 60%, 40%);}

1. Add JavaScript to load the jQuery library to your HTML page.
2. Make sure your CSS is linked to your HTML page.
3. Create a new JavaScript file, which uses jQuery to change the class of all <li> to 'running', hide all the <li> then reveal them using the .slideDown() method over 1000 milliseconds.
4. Link your JavaScript file to your HTML and check that it functions correctly.

We saw earlier that it is best to include our jQuery script at the end of the <body>, after we have loaded the jQuery library. This means that the DOM Tree is mostly complete and the jQuery library is loaded before any of our code runs. It is still possible that some of the other page assets, such as images will not have loaded at this point, but this tends to be less critical.

While this approach is fairly robust, it is generally good practice to ensure that our code checks that the page is loaded before it runs. This is particularly useful if the script might be called on other pages or by other people.

We can achieve this by wrapping our script in the following code:

$(document).ready(function() {

// our code here

});

Or we can use the shorthand version:

$(function() {

// our code here

});

**Activity**

1. Amend your jQuery code to include the above test.
2. Check that it still functions correctly.

## Selecting Element Nodes

We have already seen the basic jQuery selector:

$('li')

This returns a jQuery Object containing **all** the <li> Element Nodes. Note that even if there is only one such Element Node, it will still be returned as an Object. Remember also that the first element in the Object will have the index value 0.

We need to be aware that some jQuery methods only apply to the **first** element in the Object, whilst others will apply to **all** elements in the Object.

In addition to selecting by tag name, we can include class names in the selector:

$('li.running')

This returns a jQuery Object containing **all** the <li> Element Nodes that have class='running'.

We can include multiple class names in the selector:

$('li.running.new')

This returns a jQuery Object containing **all** the <li> Element Nodes that have class='running' **and** class='new'.

We can also include id names in the selector:

$('li#four')

This returns a jQuery Object containing the <li> Element Node that has id='four'.

We can also include multiple selectors:

$('li.running, li#four')

This returns a jQuery Object containing the <li> Element Nodes that have class='running' and the <li> Element Node that has id='four'. Note, the selectors are separated by a comma.

**Activity**

1. Create a JS file that contains the following code and connect it to your HTML file (remove the connection to the previous JS file if necessary)

$(function() {

console.log($('li.new'));

});

1. Looking at the script and your HTML, predict the contents of the Object that will be returned.
2. Reload the HTML page to run the script – you will need to open the developer tools to see the output (remember this is handy for debugging too). Check if your prediction was correct.
3. Modify your script to each of the following selectors in turn. Predict the output and check you were correct.

'li.running'

'li.running.new'

'li.new.running'

'li.new, li.running'

'li#five'

'li.new, li#four'

'li.new#four'

'li.new#five'

If we use a variable to store a jQuery Object its name is typically started with a $ to distinguish it from other types of variable, for example

var $myVariable;

## Filtering selections

jQuery provides a number of different ways to filter the selected Elements. We will look at some of these, but there are others which you can look up yourself.

We can exclude Elements based on a selector using :not(selector) for example:

$('li:not(".running")')

This returns a jQuery Object containing **all** the <li> Element Nodes that **do not** have class='running'. Note, the use of double quotes to avoid incorrect quote pairing. Note also the use of the colon : to indicate a filter.

We can pick the first :first or last :last Element, for example:

$('li:last')

This returns a jQuery Object containing the **last** <li> Element Node.

We can also combine filters, for example:

$('li:last:not(".running")')

This returns a jQuery Object containing the **last** <li> Element Node that **does not** have class='running'.

We can filter on the text content of an Element using :contains('string'), for example:

$('li:contains("Comp")')

This returns a jQuery Object containing **all** the <li> Element Nodes that have the string "Comp" as part of their text content.

We can filter on empty elements using :empty, for example:

$('li:empty')

This returns a jQuery Object containing **all** the <li> Element Nodes that have no text content.

We can also filter of HTML mark up using :has(selector), for example:

$('li:has(em)')

This returns a jQuery Object containing **all** the <li> Element Nodes that contain an <em> tag.

**Activity**

1. Modify your script to apply appropriate filters to retrieve <li> for each of the following in turn.

Courses that are not new (class="new")

Courses that do not have the id four

The first new course

Courses with tiles containing the string "at"

List items that do not contain <em> tags

All non-empty list items (add an empty list item to your HTML to check).

The last course that does not include the string "at"

## Working with Element content

jQuery provides a number of different methods for manipulating Element content.

There are two methods for retrieving Element content, .html() and .text(). They return different types of content and they behave in different ways.

.html() returns both HTML mark up and text content, in a string. It only does this for the **first** Element in the jQuery Object and any of its descendants. Consider the following function

$(function() {

var myListItems = $('li#five').html();

console.log(myListItems.length);

console.log(myListItems.charAt(3));

});

This retrieves the HTML and text content from the <li> with id="five", in our case the string '<em>new</em> BSc Big Data'. We are writing the length of the string (25, it includes spaces) to the console and then the character at position 3 in the string (remember the first character is at position 0), the character >.

.text() returns only the text content, in a string. It does this for **all** Elements in the jQuery Object, and any of their descendants. Consider the following function:

$(function() {

var myListItems = $('li#five').text();

console.log(myListItems.length);

console.log(myListItems.charAt(3));

});

This retrieves the text content from the <li> with id="five", in our case the string 'new BSc Big Data'. We are writing the length of the string (16, it includes spaces) to the console and then the character at position 3 in the string (remember the first character is at position 0), the space character.

**Activity**

1. Run the two functions above to verify their behaviour.
2. Run the two functions again but use $('li') instead of $('li#five') observe the differences in behaviour between .html() and .text().
3. Run the two functions again but use $('ul') instead of $('li') observe the differences in behaviour between .html() and .text() and between $('ul') and $('li').

.html() and .text() can also be used to change Element content.

.html() sets the content of **all** Elements in the jQuery Object. Any existing content, including mark-up and child Element Nodes is replaced. Any mark up in the new content will be treated as HTML mark up. Consider the following function:

$(function() {

$('li#three').html('<em>new</em> BA Games Design');

});

This will overwrite everything between <li id="three"> and </li> with the new string. The mark up in the new string will be treated as HTML mark up.

**Activity**

1. Run the function above to verify its behaviour.
2. Run the function again but use $('li') instead of $('li#three') observe the differences in behaviour.
3. Run the function again but use $('ul') instead of $('li') and use the following replacement string '<li><em>new</em> BA Games Design</li>'observe the difference in behaviour.

.text() sets the content of **all** Elements in the jQuery Object. Any existing content, including mark-up and child Element Nodes is replaced. Any mark up in the new content will be treated as text, **not** as HTML mark up. Consider the following function:

$(function() {

$('li#three').text('<em>new</em> BA Games Design');

});

This will overwrite everything between <li id="three"> and </li> with the new string. The mark up in the new string will be treated as text, not HTML mark up, so the <em> and </em> will be visible to the user.

**Activity**

1. Run the function above to verify its behaviour.
2. Run the function again but use $('li') instead of $('li#three') observe the differences in behaviour.
3. Run the function again but use $('ul') instead of $('li') and use the following replacement string '<li><em>new</em> BA Games Design</li>'observe the difference in behaviour.

.replaceWith() is another method which can be used to change content. It functions exactly the same as .html() but it returns the replaced contents as an Object. Consider the following function:

$(function() {

var oldContent;

oldContent = $('li#five').replaceWith('<li><em>new</em> BA Games Design</li>');

console.log(oldContent[0].textContent);

});

This will overwrite everything between <li id="five"> and </li> with the new string. The mark up in the new string will be treated as HTML mark up. oldContent will contain an Object holding the removed content. The text from the first Element of that returned Object is displayed to the console.

Note that .html() and .replaceWith() carry the same security risks with regards to untrusted content, as using the innerHTML property.

.remove() removes all of the elements in the jQuery Object. Consider the following code:

$(function() {

$('li.new').remove();

});

This will remove all <li> elements where class="new".

Consider the following code:

$(function() {

var oldContent;

var $workingObject;

$workingObject = $('li#four');

oldContent = $workingObject.text();

oldContent = '<em>new</em> ' + oldContent;

$workingObject.html(oldContent);

});

It stores a jQuery Object referencing the <li> with id="four" in the variable $workingObject. It stores the text content of that Object in the variable oldContent. It then adds the string '<em>new</em> ' to the start of oldCOntent. It then overwrites the existing contents of the <li> with id="four" with the new string, using the .html() method, so that the mark up in the string will be interpreted as mark up.

We can do this more neatly by using a function – we can do this with any of these methods.

Consider the following code:

$(function() {

$('li#four').html(function () {

return '<em>new</em> ' + $(this).text();

});

});

This achieves the same outcome as the previous code. Note that we are now calling .html() with an anonymous function rather than a string or a variable. The function is using the return keyword to return a value from the function. The function is also using the this keyword to refer to the current Object item.

**Activity**

1. Run the function above to verify its behaviour.
2. Run the function again but use $('li') instead of $('li#four') observe the differences in behaviour.
3. Write a new function which swaps the text content of the nodes with id="three" and id="four".
4. Write a new function that replaces 'new' with 'coming soon' in all new courses (class="new"). Use the string.replace() method.
5. Write a new function that causes the text of <li> with id="two" to be formatted as bold <b>

## Working with Elements

We have several other methods that allow us to insert content, in the form of jQuery Objects or strings, into different places in the DOM tree. Again we need to be aware of issues around untrusted content when using these methods.

|  |  |  |
| --- | --- | --- |
| .before() |  | .after() |
|  | <li>item</li> |  |
| .prepend() |  | .append() |

Consider the following code:

$(function() {

var $newListItem = $('<li>BSc Operating Systems Design</li>');

$('li:last').after($newListItem);

});

We are defining a new jQuery Object, $newListItem which contains the string '<li>BSc Operating Systems Design</li>'. We are then inserting $newListItem after the last <li> using the .after() method.

We can use these methods with jQuery Objects (Element Nodes), for example:

$(function() {

var $newListItem = $('<b> with BSc Operating Systems Design</b>');

$('li:last').append($newListItem);

});

Note the use of the jQuery ($) in this example to create the jQuery Object.

We can also us ethem with strings, for example:

$(function() {

var newListItem = 'with BSc Operating Systems Design';

$('li:last').append(newListItem);

});

Or, more succinctly:

$(function() {

$('li:last').append('with BSc Operating Systems Design');

});

**Activity**

1. Write code to add a new list item before the existing list items, containing the string 'BSc Operating Systems'.
2. Write code to add a new list item before the last list item, containing the string 'BSc Autonomous Agents'.
3. Write code the insert '<em>new</em> ' before the course title of the third course in the list. Use the :eq(*index*) filter to select the item with index value *index* (remember, index values start at 0).
4. Write code to insert ' <b> sandwich course</b>' after the course title of every course except the first two. Use the :gt(*index*) to select items with an index greater than *index*.
5. Write code the insert ' bursary available' after every other course. Use the :odd filter.

The order of the Elements in the .append() and .prepend() methods may feel a little counter-intuitive. If this is the case we can always use .appendTo() and .prependTo() instead:

|  |  |
| --- | --- |
| a.prepend(b) adds b to the start of a, ba | a.append(b) adds b to the end of a, ab |
| a.prependTo(b) adds a to the start of b, ab | a.appendTo(b) adds a to the end of b, ba |

## Working with Element attributes

We can use the .attr() method to retrieve or set an Element attribute, for example:

console.log($('li:first').attr('id'));

would write the id of the first <li> to the console, and

$('a#test').attr('href', 'http://www.cardiff.ac.uk');

would change the href value of the <a> with id="test" to the value "http://www.cardiff.ac.uk", so the link would point to a different destination. We could use the same technique to change the src for an image, video or audio file.

The .removeAttr() method allows us to remove an attribute and its value from an Element, for example:

$(function() {

$('li#two').removeAttr('class');

});

Would remove the class attribute and **all** class attribute values from the <li> with id="two".

If we only want to remove one of the class attribute values we can use the .removeClass() method, for example:

$(function() {

$('li#one').removeClass('running');

});

This would remove the class attribute running from the <li> with id="one", but would leave other values for the class attribute unchanged.

We can also add values to the class attribute using the .addClass() method, for example:

$(function() {

$('li#four').addClass('running');

});

Note that this adds the new class attribute value, preserving any existing values. It does **not** overwrite them. If we used the .attr() method, any existing class attribute values would be overwritten.

**Activity**

1. Write code to add a new class value 'running' to all <li>s.
2. Write code to replace any existing class values with class="new" on all <li>s.
3. Write code to remove the class value 'running' from all <li>s.
4. Write code to remove the class attribute from all <li>s.
5. Write code to add class="odd" to odd numbered <li> and class="even" to even numbered <li>. This can be handy when formatting tables, so we can colour each row or column a different colour.

## Working with CSS properties and rules

Changing the class attribute value of an Element is the best way to change the styling of that element, but we can also use the .css() method to retrieve and set css values.

We can use the .css() method to retrieve the value of a CSS property, for example:

$(function() {

console.log($('li').css('background-color'));

});

Note if the jQuery selector returns more than one Element, the .css() method is only applied to the first Element.

We can also use the .css() method to set the value of a CSS property, for example:

$(function() {

$('li').css('background-color', 'hsla(270, 100%, 50%, 1)');

});

If the CSS property has numerical dimensions expressed in pixels, we can increase and decrease their values with the += and -= operators, for example:

$(function() {

$('li').css('padding-left', '+=20');

});

Will increase the value of padding-left by twenty pixels.

We can also set multiple properties using object literal notation, for example:

$(function() {

$('li').css({

'background-color': 'hsla(270, 100%, 50%, 1)',

'color': 'hsla(270, 100%, 80%, 1)',

'padding-left': '3em'

});

});

Note the curly braces {} around the set of property: value pairs. Note also the colon separating the property from the value and the comma separating each property: value pair from the next. There is no comma after the last property: value pair.

**Activity**

1. Write code to change the background-color of all <li>s to hsla(270, 100%, 50%, 1) and then write the background-color of the second <li> to the console.log – note that the browser is returning rgb colour values because that is the colour model it works with.
2. Write code using object literal notation to change the style of all <li>s with class='running' so that border-radius is 10px, background-color is hsla(270, 100%, 50%, 1) and color is hsla(270, 100%, 80%, 1)

## Working with .each(), this and $(this)

We have already seen how some jQuery methods can update all the selected Elements as if we were using a loop, for example:

$('li').attr('class', 'full');

changes the class of all <li> Element Nodes to 'full'. The equivalent in JavaScript would be something like:

var els = document.getElementsByTagName('li');

var noOfElements = els.length;

for (var i = 0, i < noOfElements; i++) {

els[i].className = 'full';

}

This is very convenient, unless we want to get information from each Element or perform a series of operations on each Element. If we want to do this, then we need to use the .each() method, for example:

$(function() {

$('li').each(function(){

var ids = this.id;

$(this).append(' <b>' +ids +'</b>');

});

});

The .each() method allows us to take each member of the jQuery Object in turn. In this example we are applying the anonymous function to each of the <li> in turn. We are using this to refer to the specific member, so we can use JavaScript properties, in this case .id. We are then using $(this) to create a jQuery object of this so that we can apply the jQuery methods, in this case .append().

**Activity**

1. Write a version of the above code, but instead of displaying the id, display the class of each <li>.
2. Write a version of the above code, but instead of displaying the id, display the number of characters of each <li> content (for example 'BSc Computing' would be 13).

## Working with Events

In a previous workbook we used the addEventListener() method to add event handlers to Element Nodes. This method works on newer browsers, but has some issues with cross-browser compatibility with older browsers. If we use the jQuery .on() method instead, we will avoid these compatibility issues.

$(function() {

$(li).on('click', function() {

$(this).addClass('wasClicked');

});

});

As we can see in the above example, the .on() method takes two parameters, the event (in this case 'click') and a function that is applied if the event fires (in this case an anonymous function). We can add multiple event handlers to the same element and these can be used to trigger the same function or different functions, for example:

$(function() {

var ids = '';

var $listItems = $('li');

$listItems.on('mouseover click', function(){

classes = this.className;

$(this).children('.temp').remove();

$(this).append('<span class="temp"> classes: ' + classes +'</span>');

});

$listItems.on('mouseout', function(){

$(this).children('.temp').remove();

});

});

In the above example we are attaching an event handler to all <li>s - $('li'). Because we are using this jQuery Object more than once, we are storing in a variable, $listItems. We are then using the .on() methods to attach two event handers to the items in the Object. The first one will fire on 'mouseover' and on 'click', note how we can include more than one event. When it fires it first removes all children on the target Element whose class is .temp. It then appends a new span with class="temp" containing the class names of the target <li> to the content of the <li>. The second event handler fires on 'mouseout' and simply removes all children on the target Element whose class is .temp. This has the effect of removing the additional content we appended on 'mouseover' or 'click'.

**Activity**

1. Implement the above code and verify its behaviour.
2. Write a version of the above code, but instead of displaying the class, display the id of each <li>.
3. Write a version of the above code that allows the user to select one of the list items. When the mouse is over a list item, leaves a list item, or clicks on a list item, the list item will change colour, display a message and may be selected or unselected. The selected item is distinguished by colour. Only one list item may be selected at a time, selecting a second list item unselects the first. You will probably want to add two classes to your CSS, one for mouse-over items and one for selected items.

|  |  |  |  |
| --- | --- | --- | --- |
| Current item state | Mouse over | Mouse leave | Mouse click |
| unselected | Change colour to *over.*  Add 'click to select'. | Revert to original colour.  Remove 'click to select'. | Unselect any other.  Select self.  Change colour to *selected.*  Add 'click to unselect'. |
| selected | Maintain *selected* colour.  Add 'click to unselect'. | Maintain *selected* colour.  Remove 'click to unselect'. | Unselect self.  Change colour to *over.*  Add 'click to select'. |

You will remember that when we were using the addEventListener() method, we could pass it an Event Object so that we could access properties and methods of the event, rather than the Element to which it was attached. For example:

(function() {

$('li').on('click', function(e){

$('.temp').remove();

$('h2').append('<span class="temp"> x:' + e.pageX + ' y:' + e.pageY + '</span>');

});

});

In this example we are attaching an event to <li>s that fires on a 'click'. We are passing the Event Object as a parameter (e), which means we can access its methods and properties. We are accessing the .pageX and .pageY properties, which give the mouse position from the left edge and top edge of the viewport respectively. We can only access these properties if we explicitly pass the Event Object. There are a number of useful methods and properties associated with the Event Object.

The .on() method must have an event and a function. The .on() method has two additional, optional parameters which we might want to use. One parameter allows us to filter the descendent elements of the jQuery Object using the filters we saw earlier. The other parameter allows us to pass additional data to the function that is called when the event is triggered.

.on('events', 'selector', {data}, function(e));

For example:

$(function() {

var listItem, itemStatus, eventType;

$('ul').on(

'click mouseover',

':not(#four)',

{status: 'important'},

function(e){

listItem = 'Item:' + e.target.textContent + '<br>';

itemStatus = 'Status: ' + e.data.status + '<br>';

eventType = 'Event: ' + e.type;

$('.temp').remove();

$('h2').append('<span class="temp"><br>' + listItem + itemStatus + eventType + '</span>');

}

);

});

We have attached an event handler that will fire on 'mouseover' and on 'click'. It will only fire on Elements that do not have id="four". The anonymous function which fires when the event occurs is passed additional data status: 'important' in literal notation (it could contain several property: value pairs separated by commas). We access this data in the function using the e.data property, in our case e.data.status to get the value of status.

**Activity**

1. Write a version of the above code, where the event only fires if the user clicks on a running course. When they do so a message is displayed saying 'Spaces: 30' where the 30 is passed to the function as a data item.
2. Write a version of the above code, where the event only fires if the user clicks on a course whose title contains the word 'new'. When they do so a message is displayed saying 'Enrolment: September 2020' where the month and the year are passed to the function as separate data items.

## Working with Effects and Animation

jQuery has some basic effects and animation capabilities.

.show() .hide() .toggle() displays, hides or toggles the visibility of the selected elements.

.fadeIn() .fadeOut() .fadeTo() .fadeToggle() fades elements to visible, to invisible, to a specified opacity, or toggles the visibility of the selected elements via fading.

.slideUp() .slideDown() .slideToggle() makes selected elements appear, vanish or toggle between visible and invisible with a sliding motion.

These methods all take parameters, for example to control the speed over which they happen. See here for further details: <https://api.jquery.com/category/effects/>

We can also do some simple custom animation via CSS properties. However, we can only do this for properties that have numerical values. The CSS properties are written in camel case and with no punctuation, for example border-top-left-radius would become borderTopLeftRadius. For example

$(function() {

$('li').on('click', function(){

$(this).animate({

fontSize: '+=20',

opacity: 0.5

});

});

The method has three optional parameters, a speed, an easing value and a callback function.

Speed specifies the duration of the animation in milliseconds, or the keyword fast or slow.

Easing specifies how the speed of the animation varies. It can be linear (the speed of the animation is uniform) or swing (the animation is faster in the middle). The callback function is called when the animation is over. For example:

$(function() {

$('li').on('click', function(){

$(this).animate({

fontSize: '+=20',

opacity: 0.5

}, 'slow', 'swing', function(){

$(this).hide();

});

});

});

**Activity**

1. Experiment with the above code, try the different effects and their parameters.
2. Experiment with custom animations on different CSS properties and with different parameters.

## Working with the DOM Tree

jQuery also provides us with methods for traversing the DOM tree. These methods avoid cross-browser inconsistencies, such as those caused by whitespace nodes.

.find(*selector*) returns all elements in the current jQuery Object which match the specified selector.

.closest(*selector*) returns the nearest ancestor (parent or child) that matches the selector.

The following methods have an optional selector. If a selector is specified then the method returns all elements which match both the method and the selector.

.parent() returns direct parent of the current selection.

.parents() returns all parents of the current selection.

.children() returns all children of the current selection.

.siblings() returns all siblings of the current selection.

.next() returns the next sibling of the current selection.

.nextAll() returns all the subsequent siblings of the current selection.

.prev() returns the previous sibling of the current selection

.prevAll() returns all previous siblings of the current element.

## Adding and Filtering Elements

Once we have our selected elements in a jQuery Object we may want to add additional elements to it, or filter the elements in some way.

.add(selector) adds a new selection to an existing one. The selector can take many forms including selectors, elements and HTML fragments. For example $('p').add('div')

.filter(selector) finds elements in the original object that match a second selector.

.find(selector) finds descendants of elements in the original set that match the selector.

.not(selector) :not(selector) find elements in the original set which do not match the selector. Note we can express this in two different ways: $('li').not('.running') or $('li:not(.running')

.has(selector) :has(selector) finds elements in the original set that have a descendent which matches the selector.

:contains(text) selects all elements that contain the text (note, this is case sensitive).

.is() checks whether the current selection matches a condition – returns Boolean. For example if($this.is('.running')){…}

## Working with Form Elements

jQuery also provides us with selectors for Form Elements.

:button selects <button> and <input> elements whose type is button.

:checkbox selects <input> elements whose type is checkbox.

:checked selects checked elements from checkboxes and radio buttons.

:disabled selects all elements that have been disabled.

:enabled selects all elements that are not disabled.

:focus selects the element that currently has focus.

:input selects all <button>, <input>, <select> and <textarea> elements.

:password selects all password elements

:radio selects all radio button inputs. To select a specific group of radio buttons we can use $('input[name= "gender"]:radio').

:reset selects all elements that are reset buttons.

:selected selects all elements that are selected.

:submit selects all <button> and <input> elements whose type is submit

:text selects all <input> elements whose type is text

The .val() methods is also useful to retrieve the value on an <input>, <select> or <textarea> element. It returns the value of the first element. It can also be used to set the value of all of the elements.

**Activity**

1. Add the following <div> to your HTML, immediately after the </ul> and before the </div>.

<div id="newCourseButton"><button href="#" id="showForm">New Course</button></div>

<form id="newCourseForm">

<input type="text" id="courseTitle" placeholder="Enter title...">

<input type="submit" id="addButton" value="add">

1. Add the following jQuery code.

$(function() {

var $newCourseButton = $('#newCourseButton');

var $newCourseForm = $('#newCourseForm');

var $textInput = $('#input:text');

$newCourseButton.show();

$newCourseForm.hide();

$('#showForm').on('click', function(){

$newCourseButton.hide();

$newCourseForm.show();

});

$newCourseForm.on('submit', function(e) {

e.preventDefault();

var newText = $('input:text').val();

$('li:last').after('<li>' + newText + '</li.');

$newCourseForm.hide();

$newCourseButton.show();

$textInput.val('');

});

});

1. Experiment with this to see what it does.
2. Read through the jQuery code carefully, noting particularly the use of the Form selectors, effects and event handlers.

This workbook has provided a basic introduction to jQuery, a JavaScript library that simplifies JavaScript programming. jQuery also avoids issues associated with cross-browser compatibility and inconsistent handling of whitespace nodes, which we have come across in earlier Workbooks.

As we have seen, jQuery covers many different aspects of JavaScript programming, including DOM Tree manipulation, event handling and simple effects.

This Workbook has not covered all the details, so you will probably need to do some additional reading or look up specific details when you need them.