Learning Outcome 6: Create Dynamic HTML (DHTML) web pages

DHTML refers to pages that change over time and/or based on user actions; more interesting, user-friendly, and responsive pages.

See <https://en.wikipedia.org/wiki/Dynamic_HTML>

We’ve already created dynamic web pages, but jQuery adds power, simplicity, efficiency, and browser compatibility.

# Learning Step 6.1: Develop web pages that are dynamic

**Adding event handlers with jQuery**

jQuery provides a number of methods for adding event handlers to an element. For instance, to add a handler for the click event, we call the click() method on a jQuery object. The click method takes a single argument that is a function, and when the click event occurs, the function will be executed.

To ensure that the object exists before we add the event handler to it:

* Use the **onload** attribute of the **window** object; waits until all images and frames are rendered
* jQuery – we can use jQuery so that our code is only loaded when the web page is ready
  + **$(document).ready( function() {** */\* code to execute \*/* **} );** // Note that no quotes are needed for document
  + **$( function() {** */\* code to execute \*/* **} );** // callback function that jQuery executes when the document is loaded
  + See <https://learn.jquery.com/using-jquery-core/document-ready/>

**Adding CSS classes to elements**

We have already used:

$("p").css( "font-style", "italic" ); // adds italics directly

jQuery also has an addClass() method that allows us to add a specified class or classes to elements identified in a jQuery object:

$("p").addClass("special");

*And in the CSS file:* **.special { font-style: italic; }**

If we want to add more than one class to an element, we specify a space-separated list as the argument:

$("p").addClass("special error");

**Practice:** Using **jqueryevents-done.html**, **scripts/jqueryevents.js**, and **styles/jqueryevents.css**, update the web page so that when the visitor clicks the Bold button, the paragraphs are changed to bold (as well as any other formatting applied to them). To do this, you should do the following:

* Ensure that there is a CSS class that sets a bold font weight for members of its class.
* Add a click handler to the “Bold” button that adds the class to all paragraphs

**Removing CSS classes from elements**

We can also remove CSS classes from elements. To remove a class, we call the removeClass() method. removeClass() takes a single argument that is a string that contains a space-separated list of classes to be removed. If the element is not currently a member of the class being removed, there is no change made to the element’s classes.

**Practice:** Add a click handler to the “Undecorate” button. The click handler should remove both the “special” and the “bold” class from all the paragraphs on the page.

**Adding mouse events with jQuery**

Other event handlers include:

* mouseover() or mouseenter()
* mouseout() or mouseleave()

**Practice:** Add a mouseenter and mouseleave event handler (or a mouseover and mouseout event handler) to the first paragraph to add and remove the “green” class from that element (where the “green” class changes the text color to green).

There is a shortcut to define mouseenter and mouseleave at the same time; the hover() method takes two arguments – first, a function to be called when mouseenter occurs; and second, a function to be called when mouseleave occurs.

**Practice:** Use hover() on the second paragraph to make it green when the mouse is over the text and remove the green class when the mouse is not over the text.

**Other jQuery event handler registration methods**

See <http://api.jquery.com/category/events/> or [http://api.jquery.com/category/events/**mouse-events**/](http://api.jquery.com/category/events/mouse-events/) or [http://api.jquery.com/category/events/**form-events**/](http://api.jquery.com/category/events/form-events/) or [http://api.jquery.com/category/events/**keyboard-events**/](http://api.jquery.com/category/events/keyboard-events/)

There is also a generic method that you use to register event handlers called “on”. In most cases, it takes two arguments: the first is a string describing the event that you are registering an event handler for (like “click” or “mouseover” or “blur”), and the second argument is the event handler for the first argument.

$("#italic").click( function() { … } );

$("#italic").on( "click", function() { … } );

To remove an event handler, there is an off() method. It takes a string as an argument, which contains a space-separated list of event handlers to remove. If you don’t pass in any arguments, all event handlers will be removed from that object.

$("#italic").off( "click" ); // removes click handler from the button with the id italic

**Practice:** Add a click event handler to both paragraphs using the on() method. When you click on the first paragraph, it will be hidden and the second paragraph will be unhidden. When you click on the second paragraph, it will be hidden and the first paragraph will be unhidden. Hide the paragraphs using a CSS class that sets the “visibility” style to “hidden”, adding or removing this class to hide/unhide the paragraph.

**Bonus practice:** Add an input element with the type “button” with a corresponding click event handler that will remove *all* the event handlers from the first paragraph.

**Changing the contents of an element**

If we call **html()** with no arguments, it returns the HTML contents of the first element that is found; if we pass in a string to **html()**, it sets the HTML contents of all elements to that string.

$("#firstPara").on("click", function() {  
 $(this).html("You <em>clicked</em> me");  
 });

Similarly, we can call the **text()** method to return or change the inner text of all matching objects; we can call the **val()** method to return or change the value of an input element.

**Practice:** Change the functionality of the “Italic” button. When the button is clicked, it currently adds the “special” class to all paragraph elements. What we want to do now is, when the button is clicked, continue to do that, but also change the change the text of the button to “UnItalic”. When the button is pressed again, you will remove the class “special” from all of the paragraphs, then change the contents of the button back to “Italic”.

*Hint:* If you check the current contents of the button, you can tell what changes to make to the paragraph and the button.

**Bonus practice:** After clicking the button to remove all the event handlers from the first paragraph, change the text on that button to “Done”.

**Changing the value of an attribute**

jQuery also provides a method to access or change the value of an attribute. The method is called **attr()** and it takes one or two arguments. If it takes one argument, it returns the value of the attribute that is passed in as the first string. If it takes two arguments, it sets the value of the attribute that is passed in as the first argument to the value passed in by the second argument.

$("#alink").attr( "href" ); // get the href attribute’s value

$("#alink").attr( "href", "http://random.org" ); // sets the href attribute

**Practice:** Add an image to the bottom of your web page with the src attribute of “media/HappyFaceNoSmile.jpg”. When the website visitor rolls over the image, we will change the attribute to “media/HappyFaceSmile.jpg”; when they roll off the image, the “src” attribute will be changed back to its original value.

**Bonus practice:** Preload the “HappyFaceSmile.jpg” image.

**Extra bonus practice:** When the button to remove the event handlers is called, disable the button. Note that you can chain calls to jQuery objects, since jQuery calls return the jQuery object that they called on. You could try changing the alt attribute of the image as well.

# Learning Step 6.2: Create animation effects

**hide()** method – called on a jQuery object, it hides the specified elements (equivalent to setting the display property to none). Can be applied to a number of elements all at once.

$("li").hide();

**show()** method – opposite of hide; makes the element appear again.

**Practice:** Using **einstein.html**, add a click event handler to the Hide button so that when the button is clicked, Einstein’s picture will be hidden with the hide() method. Add a click event handler to the Show button so that when the button is clicked, the show() method is called to show Einstein’s picture.

**toggle()** method – causes a hidden object to display, or a displayed object to hide. (Works in combination with the hide() and show() methods as well.)

**Practice:** Add a click event handler to the “Toggle” button to toggle display of Einstein’s picture.

**fadeout()** method – causes an element to fade out

**fadeIn()** method – causes an element to fade in

**Practice**: Add click handlers to the “Fade Out” and “Fade In” buttons to fade out or fade in the image.

**Changing the speed of an effect**

We can call the effect’s method with a single argument for the speed:

* No argument – default speed (0 for hide/show/toggle, 400 ms for fadeIn/fadeout)
* "slow" – the effect takes 600 ms to complete
* "fast" – the effect takes 400 ms to complete
* A number specifying how many milliseconds for the event to take place

**slideUp, slideDown,** and **slideToggle** methods – cause the element to disappear or reappear by shrinking or growing the items vertically – NOTE that the width of an image must be set to a specific value in HTML or CSS; otherwise, the sliding effect will change both height and width

**Practice:** Change the Slide Up, Slide Down, and Slide Toggle buttons so that their event handlers call the appropriate method. For slideUp, specify an argument or “slow”; for slideDown, “fast”, and for slideToggle, no argument.

**Practice:** Add a mouseover event handler to the picture of Einstein. When the website visitor rolls the mouse over top of the picture, make the picture fade out over 5 seconds.

**Practice:** Change the Slide Up, Slide Down, and Slide Toggle event handlers so that they apply to the h1 element on the page instead of the image.

**Additional effects** – you can create your own effects using the **animate()** method; you can also use additional effects that other people have created using various plugins such as jQuery User Interface library, also known as jQuery UI (see <http://jqueryui.com>), which includes additional controls (tabs, date pickers, widgets) and additional effects.

**Practice:** Download **learning\_letters\_A\_start.html** from the OneDrive folder. Add functionality to this web page as described in **“LO6 LS2 learning\_letters\_A.docx”** (in the OneDrive folder as well). A solution will be posted in **learning\_letters\_A.html**.

Note that you can have multiple selectors in a jQuery call using a comma to separate the selectors, like:

**$("span", this).*methodToCall…***

This is read as “Select all **span** elements *within* **this** current element.”

You can try this out in the learning letters file – first, any complex selector like $("#game header") could be changed to $("header", "#game"); second, instead of changing/formatting/sliding the table cells with the class ".target", you could put the target letter "A" in a **span** element, then hide the specific **span** element for the cell that has been clicked using the selector **$("span", this)** – the comma-separated selectors are particularly useful when combined with **this**! (This second example will also require some additional code changes to keep everything working as before – more good practice!)

# Learning Step 6.3: Create navigation utilities

We will create an unordered list, and each <li> will be a separate menu. Then within each list will be another unordered list and within those, each list item will be a menu item.

Apply CSS styling: <https://www.w3schools.com/css/css_navbar.asp>

Another example: <https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_horizontal_float_advanced>

Apply jQuery: Note that **$("ul", this)** means look for **"ul"** elements inside **this** current element. The same results can be achieved using **$(this).find("ul")** if you prefer that notation.

**Goal:** Create dropdown menus

**Our approach:**

* Create nested **ul**’swith the menu bar (outer **ul**) and options (inner **ul**’s) with our options in links (**a** tags) in the **li** tags
* Apply CSS to format the **li**’sto look the way we want menu items to look
  + Hide the inner **ul**’s so that the options are hidden
  + Make the inner **ul**’s have absolute positioning and a high z-index so that when displayed, it will be over the web page content
* When one of the menu bar choices (main **li**) is hovered over, use jQuery to display (using some cool effect) the corresponding hidden options

**Practice:** To start, copy **faq-start.html** from **J:\CST\CST1\CWEB190\lo6**, and edit **faq.html** so that it has nested bulleted lists inside the **nav** element as follows:

* Main 1
  + Sub 1
  + Sub 2
* Main 2
  + 2.1
  + 2.2
  + 2.3

Add **<a href="#">***Main 1***</a>** around each list item (in both bulleted lists).

Apply the id **"menubar"** to the *first* (outer) **ul** tag.

Then apply **menu.css** and **menu.js** to the **faq.html** page.

**Bonus practice:** Do the extra practice in the HTML comments in **faq.html**.

*Some additional challenges:*

* Can you add one more menu (Main 3) with options below it (3A, 3B)? What does this involve?
* Can you change the formatting so that the menu bar options (in the first bulleted list) have a different look than the options that appear below it when they are hovered over? For instance, you could make the menu bar options have a different background color, and you could make the options left-aligned instead of centered. What does this involve?
* Can you change the menu bar options so that if they are clicked, the menu options below (in the nested bulleted list) stay on the screen, even when the menu bar is no longer hovered over, until the same menu bar option is clicked (or another menu bar option is clicked)?