**Introduction to Mouse Events**

The jQuery library provides a collection of methods that serve one of two purposes.

* To listen for an event — an event (e.g. clicking a button) is something the user does to trigger an action.
* To add a visual effect — a visual effect (e.g. a drop-down menu appearing when a user clicks a button) is something that changes the appearance of the web page. Events are often responsible for triggering a visual effect.

In this lesson, you will learn how to link a user event to a visual effect using *event handlers*.

There are two parts to an event handler: an *event listener* and a *callback function*.

* An *event listener* is a method that listens for a specified event to occur, like a click event.
* A *callback function* is a function that executes when something triggers the event listener.

Both the event listener and callback function make up an event handler.

In code, this looks like:

$('.example-class').on('click', () => { // Execute code here when .example-class is clicked });

Let's consider the example above step-by-step:

* $('.example-class') selects all HTML elements with a class of example-class.
* The on('click') method is the event listener. It checks if the user has clicked an .example-class HTML element.
* The second argument to .on() is a callback function. When a 'click' occurs on an example-class element, this function executes.

**On 'mouseenter'**

Another popular jQuery event listener is mouseenter. The mouseenter event triggers a callback function when a user enters the area that a targeted element occupies.

To listen for a mouse enter event, we can use the mouseenter event listener. Take a look at this code and compare it with the click function you learned in the previous exercises:

$('.example-class').on('mouseenter', () => { // Execute code here when mouse enters .example-class });

* As before, .on() is called on a jQuery object that selects .example-classelements.
* The first argument for .on() is the 'mouseenter' event handler. The code above will trigger the callback function when the mouse enters an .example-classelement's area.

# On 'mouseleave'

One issue with the behavior we added to our Sole Shoes website in the last exercise is that the menu remains in the DOM after the mouse leaves the menu area.

The mouseleave event listener can detect when a user's mouse leaves the area that an element occupies. The syntax looks like:

$('.example-class').on('mouseleave', function() { // Execute code here when mouse leaves .example-class });

In the example code above, a user will trigger the callback function when their mouse leaves the .example-class area.

# Chaining Events

jQuery also allows us to chain multiple methods. Instead of re-declaring the HTML element you're selecting, you can append one event to another. Let's see how we can use chaining to add hover functionality to .example-classelements.

$('.example-class').on('mouseenter', () => { $('.example-class-one').show(); }).on('mouseleave', () => { $('.example-class-one').hide(); });

In the example above, we chain a mouse enter event to a mouse leave event. Both of the event handlers target .example-class elements.

When a user's mouse enters an .example-classelement's area we show .example-class-oneelements. When a user's mouse leaves an .example-class element's area, we hide .example-class-one elements.

**currentTarget**

Have you noticed in our Sole Shoes website that when you mouse over one photo, all of the images zoom. That's because .product-photo is a class on all the product photos.

One way to solve this issue is to give each HTML element a unique class and to write three mouseenter and mouseleave functions. That, however, would result in a lot of repetitive code. Luckily there's a better way.

The solution is in the callback function and selector we're using when we add a new class. Instead of selecting $('.product-photo') in each callback function, we need to pass event information into the function and call the currentTarget attribute:

$('.example-class').on('mouseenter', event => { $(event.currentTarget).addClass('photo-active'); });

The points below explain some of the features of the event.currentTarget selector in the example above.

* When a user triggers the mouseenter event listener above, the .on() method generates an event object that we pass into the callback function.
* Inside the callback function, we select event.currentTarget. The currentTargetattribute refers to only the .example-classelement that the learner has moused over.
* Since $(event.currentTarget) refers to the element that an event is detected on, you will only use it inside a callback function.

In our Sole Shoes website, we will use $(event.currentTarget) to target the one image a user mouses over.

**Review: Event Handlers**

In this lesson, you learned a few of the most common mouse events in the jQuery library.

* Event handlers are comprised of an event listener and a callback function. An event listener specifies the type of event that will be detected. The callback function executes when the event happens. Everything together is the event handler.
* An event listener is set up using the .on()method.
* The events listened for included: 'click', 'mouseenter', and 'mouseleave'.
* In addition to event handlers, you learned how to use event.currentTarget to refer to the individual element that an event occurred on.

Well done! At this point, you understand the purpose of an event, and how to link it to a callback function. Use [jqapi.com](http://jqapi.com/) to learn about some other events that jQuery has to offer. Select **Events** in the left navigation bar to browse these other events.