jQuery:

The JavaScript community contributes to a collection of libraries that extend and ease its use. In this course, you will learn about jQuery, a JavaScript library that makes it easy to add dynamic behavior to HTML elements.
Let's look at an example of how JavaScript is used to add dynamic behavior to a web page

```
const login = document.getElementById('login');
const loginMenu = document.getElementById('loginMenu');

login.addEventListener('click', () => {
  if(loginMenu.style.display === 'none'){
    loginMenu.style.display = 'inline';
  } else {
    loginMenu.style.display = 'none';
  }
});
```

The code below accomplishes the same behavior with jQuery.

```
$('#login').click(() => {
  $('#loginMenu').toggle()
});
```

jQuery setup:

To use the jQuery library, **index.html** must load it with the other dependencies. Take a look at the attached diagram to see where various dependencies load in an HTML document.

The document is loaded from top to bottom. So the style dependencies in the `<head>` will load first, then the structural elements in the `<body>` will load next. It has become common practice to link the main JavaScript file at the bottom of the HTML document because a good deal of the content of the script will require that the dependencies, style sheets and elements exist before the browser can run the JavaScript that uses and references those things.

In this example, the jQuery library is loaded from the jQuery *content delivery network (CDN)*. A CDN is a collection of servers that can deliver content.

```
<!DOCTYPE html>
<html>

<head>
  <link href='https://fonts.googleapis.com/css?family=Source+Sans+Pro:400,700'
rel='stylesheet'>
  <link href='https://fonts.googleapis.com/css?family=Roboto+Condensed:400,700'
rel='stylesheet'>
  <link rel='stylesheet' type='text/css' href='css/styles.css'>
</head>

<body>

  <footer>
    <div class='container'>
      <object type='image/svg+xml'
data='https://s3.amazonaws.com/codecademy-content/courses/jquery/audit/images/sole-logo.sv
g'></object>
      <div>Contact Us</div>
    </div>
  </footer>

  <script
  src="https://code.jquery.com/jquery-3.2.1.min.js"
  integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
  crossorigin="anonymous"></script>

  <script src='js/main.js'></script>
</body>

</html>
```

.ready()

The jQuery .ready() method waits until the HTML page's DOM is ready to manipulate. You should wrap all JavaScript behavior inside of the .ready() method. This will make sure the web page is rendered in the browser before any jQuery code executes.

```
// main.js
```

```
$(document).ready(() => {

});
```

Targeting by Class:

In the example above, document is a special keyword that we use to target the HTML document and create a jQuery object.
We can use the same $() syntax to create jQuery objects for elements on a web page. Typically, we pass a string into $() to target elements by id, class, or tag. Once targeted, we can use . notation to attach a handler method that triggers a callback function.

```
// main.js

$(document).ready(() => {
    $('.product-photo').show()
});
```

Targeting by id:
While classes allow us to target multiple elements at once, we can also target single elements by id.

```
$(document).ready(() => {
        $('#nav-dropdown').hide()
});
```

jQuery objects:

You've probably noticed the $ symbol before anything we target. The $ symbol is an alias for the jQuery function.
The jQuery function takes a parameter that targets an element, like '#navMenu', and turns it into a jQuery object. Then, you can call any jQuery method on a jQuery object.
Developers often save jQuery objects in variables, like so:

```
$(document).ready(() => {
  const $navDropdown = $('#nav-dropdown')
  $navDropdown.hide()
});
```

Event handlers:

The jQuery .on() method adds event handlers to jQuery objects. The method takes two parameters: a string declaring the event to listen for (the handler) and a callback function to fire when the event is detected.

```
$(document).ready(() => {
  const $menuButton = $('.menu-button');
  const $navDropdown = $('#nav-dropdown');
          $menuButton.on('click',() => {
    $navDropdown.show()
  })
})

// final main.js
$(document).ready(() => {
  const $menuButton = $('.menu-button');
  const $navDropdown = $('#nav-dropdown');

  $menuButton.on('click', () => {
    $navDropdown.show();
  })

  $navDropdown.on('mouseleave', () => {
    $navDropdown.hide();
```

```
  })
})
```

---

jQuery Effects

jQuery Effects are a group of methods in the jQuery library that are responsible for adding dynamic behavior to websites.

.hide()

It helps in disappearing an element from the page and the space that it was taking up will disappear as well.

```
$('.hide-button').on('click', () => {
   $('.first-image').hide();
  });
```

.show()

Luckily, the .show() method does the opposite of .hide(). If an element on your page is hidden, .show() will make it appear.

```
$('.show-button').on('click', () => {
   $('.first-image').show();
  });
```

.toggle()

It is common for web pages to have one button that will either hide or show elements depending on their current state. We can achieve this by using the `.toggle()` method.

```
$('.toggle-button').on('click', () => {
   $('.first-image').toggle();
 });
```

.fadeIn() & .fadeOut()

Instead of instantly showing the element, `.fadeIn()` and `.fadeOut()` make the element appear or disappear over a given period of time. You can think of this as an animation. The transition between being visible and invisible happens over a duration of time.

Both `.fadeIn()` and `.fadeOut()` take an optional parameter that specifies how long the animation will take. For example, in the code below, all `<div>` elements will fade out over a period of 1000 milliseconds (or one second).

This number represents the number of milliseconds it takes for the animation to complete. If no argument is given, the default animation duration is 400 milliseconds.

```
$('.fade-out-button').on('click', () => {
   $('.fade-image').fadeOut(500)
 });

 $('.fade-in-button').on('click', () => {
   $('.fade-image').fadeIn(4000)
 });
```

.fadeToggle()

`.fadeToggle()` is the third and final method in this trio of fade methods. This method is similar to `.toggle()`. If you call `.fadeToggle()` on an element that is invisible, it will fade in. And if the element is already visible, then `.fadeToggle()` will make it fade out.

```
 $('.fade-toggle-button').on('click', () => {
   $('.fade-image').fadeToggle('fast');
 });
```

.slideUp(), .slideDown() & .slideToggle()

By using sliding effects, an element on your web page will slide up or down into place instead of appearing or disappearing. Just like with the other effects, sliding can be applied to any element on your page whether it be an image, a video, or text.

Sliding methods are animations; they happen over a period of time. As a result, sliding methods have an optional parameter to determine how long the animation will take.

```
 $('.up-button').on('click', () => {
        $('.slide-image').slideUp(100);
 });

 $('.down-button').on('click', () => {
   $('.slide-image').slideDown('slow')
 });

 $('.slide-toggle-button').on('click', () => {
   $('.slide-image').slideToggle()
 });
```

-

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.

```
$(document).ready(() => {
  $('.login-button').on('click', () => {
    $('.login-form').show();
  })
});
```

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.

```
  $('.menu-button').on('mouseenter', () => {
    $('.nav-menu').show()
  })
```

mouseleave

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

```
 $('.nav-menu').on('mouseleave', () => {
   $('.nav-menu').hide()
 })
```

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.

```
 $('.product-photo').on('mouseenter', () => {
   $('.product-photo').addClass('photo-active')
 }).on('mouseleave', () => {
   $('.product-photo').removeClass('photo-active')
 })
```

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:

```
$('.product-photo').on('mouseenter', (event) => {
   $(event.currentTarget).addClass('photo-active')
```

```
  }).on('mouseleave', (event) => {
    $(event.currentTarget).removeClass('photo-active')
  })
```

Code:

// main.js

```
$(document).ready(() => {

  $('.login-button').on('click', () => {
    $('.login-form').show();
  });

  $('.menu-button').on('mouseenter', () => {
    $('.nav-menu').show()
  })

  $('.nav-menu').on('mouseleave', () => {
    $('.nav-menu').hide();
  })

  $('.product-photo').on('mouseenter', (event) => {
    $(event.currentTarget).addClass('photo-active')
  }).on('mouseleave', event => {
    $(event.currentTarget).removeClass('photo-active')
  })

});
```

-

CSS & jQuery

At this point, you likely know that jQuery can link user events to dynamic effects on a web page. One of the most powerful toolsets in jQuery allows you to edit CSS property values. With these tools, you can change multiple visual aspects of an element at once

.css()

To modify CSS properties of an element, jQuery provides a method called `.css()`. This method accepts an argument for a CSS property name, and a corresponding CSS value.

```
 $('.menu-button').on('mouseenter', () => {
   $('.nav-menu').show();
   $('.menu-button').css('color', 'red')
 })

 $('.nav-menu').on('mouseleave', () => {
   $('.nav-menu').hide();
   $('.menu-button').css('color', '#EFEFEF')
 })
```

Changing multiple css :

In addition to changing one property at a time, the `.css()` method can accept many CSS propery/value pairs at once. You must pass the `.css()` method a JavaScript object with a list of key/value pairs like this:

```
$('.menu-button').on('mouseenter', () => {
   $('.nav-menu').show();
```

```
  $('.menu-button').css({
    color: '#C3FF00',
    backgroundColor: '#535353'
  })
 })
```

.animate()

The jQuery .animate() method enhances the visual possibilities by making CSS value changes over a period of time.

The first argument passed to .animate() is a JavaScript object of CSS property/value pairs that represent an element's end state. This is identical to the .css() method. For example, the following object could be passed to the .animate() method to change an element's height, width, and border thickness

The second parameter of the .animate() method determines how long the animation takes to complete. It is optional; if you do not provide an argument, the default value is 400 milliseconds. You can use a number (in milliseconds) or the strings 'fast' or 'slow'

```
$('.menu-button').on('mouseenter', () => {
  $('.nav-menu').show();
  $('.menu-button').css({
    color: '#C3FF00',
    backgroundColor: '#535353'
  })

  $('.menu-button').animate({
    fontSize: '24px'
  }, 200)
```

```
})
```

.addClass()

A JavaScript file can quickly get overloaded with styles if you regularly use the `css`method to modify element styles. It's a best practice to group all of the style code in a CSS file, and use jQuery to add and remove the classes from elements — this approach aligns to a design principle called *separation of concerns*.

Separation of concerns is a design principle stating that code should be separated based on its purpose in a program. In web development, that generally means the structure of a page is defined in an HTML document, styles are stored in a CSS file, and code that defines dynamic behavior is stored in a JavaScript file.

To keep CSS properties in a CSS file, jQuery can add a CSS class to an element with a method named `addClass`. It's syntax looks like this:

```
$('.menu-button').on('mouseenter', () => {
  $('.nav-menu').show();
  $('.menu-button').addClass('button-active')

  $('.menu-button').animate({
    fontSize: '24px'
  }, 200)
})
```

.removeClass()

Similar to `.addClass()`, the jQuery `.removeClass()` method can remove a class from selected elements.

```
$('.nav-menu').on('mouseleave', () => {
```

```
  $('.menu-button').removeClass('button-active')

  $('.menu-button').animate({
    fontSize: '18px'
  }, 200)
 })
```

.toggleClass()

Similar to other effects methods, you can use a toggle method instead of chaining the .addClass() and .removeClass() methods together.
The .toggleClass() method adds a class if an element does not already have it, and removes it if an element does already have it.

```
 $('.menu-button').on('click', () => {
  $('.nav-menu').toggleClass('hide')
  $('.menu-button').toggleClass('button-active')
 })
```

-

.children()

```
<div class='parent' id='holder'>
 <div>Child <span>1</span></div>
 <div>Child <span>2</span></div>
 <div>Child <span>3</span></div>
</div>

const $kids = $('#holder').children();
$kids.on('click', event => {
```

```
  $(event.currentTarget).css('border', '1px solid black');
});
```

In the example above, the $kids variable refers to all children of the element with id 'holder' (the divs inside '#holder'). The .on() method adds the click event handler to these $kids. When one of them is clicked, jQuery will add a border around it that is 1px wide and solid black.

It is important to note that only the direct descendants of '#holder' are considered children. Any elements inside the child elements of '#holder', such as the elements, will not be targeted by the .children() method.

Parent & Siblings:

There are two methods you can use to select the parent and siblings of an element.

```
$('.choice').on('click', event => {
  $(event.currentTarget).parent().hide();
});
```

In the example above, the .parent() method targets the parent element of '.choice' elements and removes it from the DOM.

```
$('.choice').on('click', event => {
  $(this).siblings().removeClass('selected');
  $(event.currentTarget).addClass('selected');
});
```

In the code above, the .siblings() method targets elements adjacent to the clicked '.choice' and removes the 'selected' class from any previously clicked '.choice's. Then the 'selected' class is added only to the clicked '.choice'.

Closest

To select an element close to the current element, we can use jQuery's .closest()method.
The .closest() method will travel up the DOM tree to find a specified selector closest to it. Its syntax looks like:
$('.example-class-one').closest('.another-class');
In the example above:

- The .closest() method is called on '.example-class' elements.
- The method then targets the element selected by the .closest() method with a class of '.another-class'.

```
<div class='.another-class'>
  <p class='.example-class-one'>1</p>
</div>
<div class='.another-class'>
  <p class='.example-class-two'>2</p>
</div>
```

Given this HTML, the jQuery above would select the element that wraps the paragraph with a value of 1, because it is the closest element, up the DOM tree, with the class .another-class.

Next

Sometimes you don't want to target all the siblings of an element — you just want to target the next one. That's where the aptly-named .next() method comes in!
The code below is HTML for a menu. The list of food types is hidden, <ol style='display:none'>.

```
<div class='heading'>MENU</div>
<ol style='display: none'>
  <li>Appetizers</li>
  <li>Entrees</li>
  <li>Salads</li>
  <li>Sides</li>
  <li>Desserts</li>
</ol>
```

Since the div and <ol> exist on the same level of the DOM, they are siblings. Since there are no elements between them, the <ol> is the next sibling of '.heading'. We can add an event handler to the div element and use the .next() method to show and hide the <ol> using the .toggle() method.

```
const $heading = $('.heading');
$heading.on('click', () => {
  $(event.currentTarget).next().toggle();
});
```

In the example above, the .on() method attaches the click event handler to $heading. Then the callback function will toggle the class of the $heading's next sibling, the ol element.
It's important to note that jQuery also has a method called .prev() that can look at the previous sibling.

-

Find

Sometimes we want to target an element that lives inside another, but we don't want to use the .children() method, since that might target more elements than we need. That's where the .find() method comes in. This method finds and targets singular or multiple elements that are descendants of an element. Unlike the .children() method, it traverses all descendants of the specified element, not just the first level down.

```
const $items = $('.myList').find('li');
```

The .find() method takes a parameter that specifies how to filter results. This parameter is just like anything you might use to select a jQuery object, ('#id', '.class', tag, etc.). .find() will return all descendants that match the passed in selector. In the example above, the .find() method will return all <li> child elements inside the '.myList'element.

-