* jQuery is a JavaScript library that makes it easy to add dynamic behavior to HTML elements.
* To include jQuery in your page use a **<script>** tag as follows:

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

* The integrity and cross origin properties in the example ensure the file is delivered without any third-party manipulation.
* a web page must be rendered in a user's browser before it's possible to have any dynamic behavior. 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. **$(document).ready(() => { });** document is a special keyword that we use to target the HTML document and create a jQuery object.
* We can use the same **$()** syntax to create jQuery objects for elements on a web page. Typically, we pass a string into $() to target elements by id, class, or tag. Once targeted, we can use . notation to attach a handler method that triggers a callback function.
* target by class **$(‘.className).method()**, target by Id **$(‘#idName).method()**
* You can save jQuery objects in variables. when doing so prefix with $ like: **const $jQueryObject = $('.someClass');**
* 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.

**$('#login').on('click', () => { $loginForm.show(); })**

* 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.
* 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. It's syntax looks like: **$('.example-class').css('color', '#FFFFFF');**
* 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: **{ color: '#FFFFFF', backgroundColor: '#000000', fontSize: '20px' }**
* When referencing CSS properties in an object, the property names are *camelCased* — they are modified to have no quotes or spaces, and to start each new word with a capital letter. Therefore, background-color becomes backgroundColor, and 'font-size'becomes fontSize.
* 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.

**{ height: '100px', width : '100px', borderWidth : '10px' }**

* 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. 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: **$('.example-class').addClass('active');**
* Similar to .addClass(), the jQuery .removeClass() method can remove a class from selected elements. Its syntax is similar to .addClass():

**$('.example-class').removeClass('active');**

* 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. Its syntax looks like: **$('.example-class').toggleClass('active');**
* Document Object Model or DOM is a tree of objects based on the HTML document that is created by the browser when it loads a page. Every element in this page exists on some branch of the tree, with elements above it, and possibly next to or below it.
* According to the DOM tree, the outermost element is the *parent* of all elements inside of it. Therefore, the HTML elements inside of the outer element are *children*. The jQuery **.children()**method allows us to target these elements. **const $kids = $('#holder').children(); $kids.on('click', event => { $(event.currentTarget).css('border', '1px solid black'); });**
* jQuery makes it easy to target HTML elements by tag name, class, and id. We can also dynamically target a single element in a given class by accessing an event's .currentTarget attribute.
* .siblings() to target elements adjacent to an element.
* Sometimes you don't want to target all the siblings of an element — you just want to target the next one or previous one. That's where the aptly-named .next() and .prev() methods come in
* .parent() to target an element's parent.
* .closest() travels up the DOM tree from the current element to target the closest element with a given selector.
* 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');**