**Why jQuery?**

JavaScript is the most widely-used language for adding dynamic behavior to web pages. 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 (don't worry about understanding the code).

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'; } });

In this example, JavaScript is used to apply behavior to an HTML element with id login. The behavior allows a user to click a **LOGIN**button that toggles a login form.

The code below accomplishes the same behavior with jQuery.

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

In this example, the same toggle functionality is accomplished using just three lines of code.

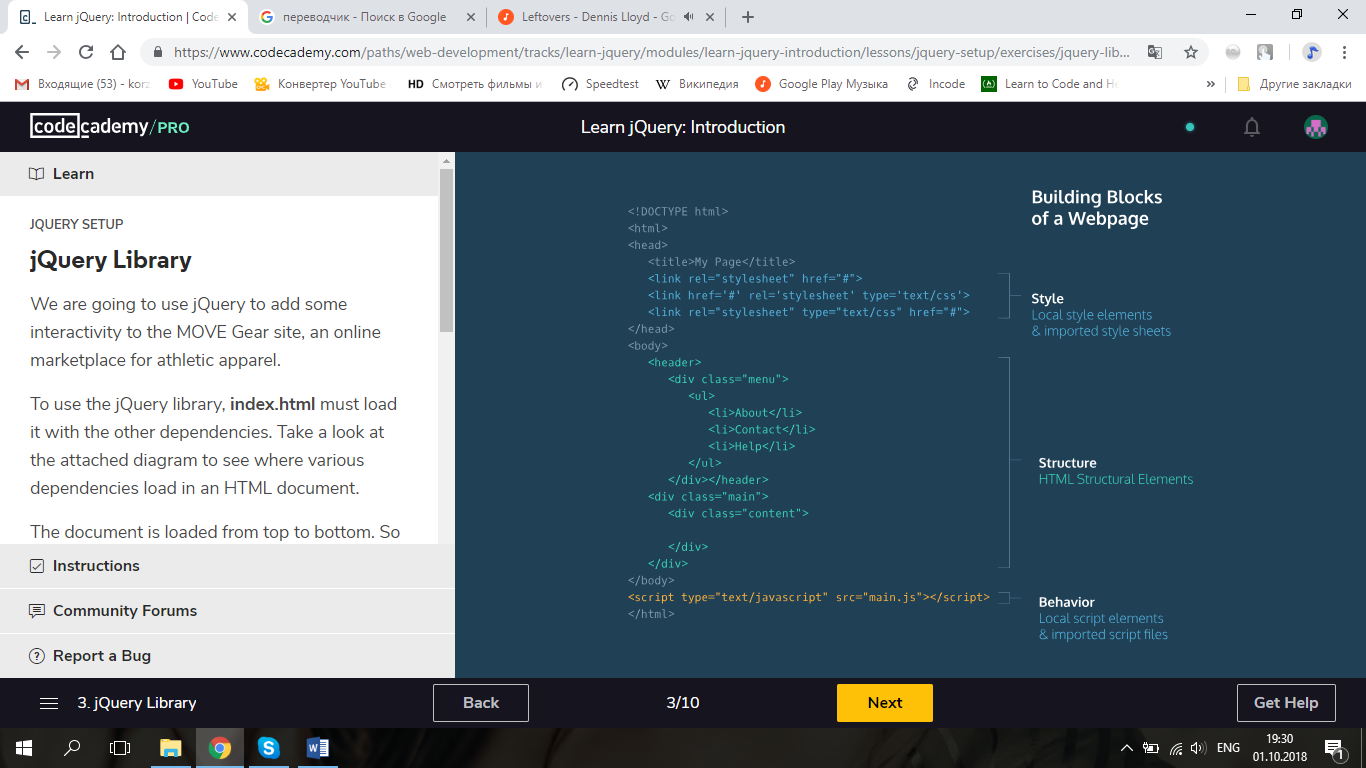
# jQuery Methods

In the last exercise, you used jQuery to cut twelve lines of code down to three.

Let's use Legos as an analogy for understanding how jQuery works. With an infinite number of Legos, you could build an entire city — of course, this would take a long time. What if you were given pre-made Lego buildings, Lego roads, Lego parks, etc? You could build a city much faster.

The JavaScript language represents an infinite supply of Lego blocks — the possibilities are endless but time-consuming. The pre-made Lego structures are like jQuery methods. You can use these methods to add dynamic behavior, such as .hide(), .show(), .fadeIn(), .fadeOut() etc., to HTML elements.

The example in the last exercise took twelve lines of JavaScript, but was achieved with only three lines of jQuery's .click() and .toggle()methods.



# jQuery Library

We are going to use jQuery to add some interactivity to the MOVE Gear site, an online marketplace for athletic apparel.

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.

When you add the jQuery library to your project in the next exercise, you will do so on the line before the </body> tag. Because HTML files load from top to bottom, adding the jQuery library at the bottom of your project will ensure that it will not affect the HTML (structure) and CSS (style) load times.

# .ready()

The jQuery library makes it quick and easy to add visual effects and interactivity to your web page. However, a web page must be rendered in a user's browser before it's possible to have any dynamic behavior. To solve this problem, we will use our first jQuery method.

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(() => { });

In the example above, the .ready() method waits until the browser finishes rendering the HTML document before triggering a callback function. We will write all of our jQuery behavior inside this callback function.

# Targeting by Class

Let's look at the code we just pasted into our **main.js** file:

$(document).ready(() => { });

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.

Let's consider how we can target elements by class. We can reference elements by class name with the following syntax:

$('.someClass').handlerMethod();

In the example above, every element with a class of 'someClass' is targeted. Note, we prepend the class name with a period (.someClass). Then, we call the .handlerMethod() on all of the referenced items.

Let's give this a shot on our MOVE Gear site.

# Targeting by id

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

To select by id, we prepend the element's id name with the # symbol.

$('#someId').hide();

In the example above, we target the #someIdelement and call the .hide() method on it. The .hide() method hides the someId element when the web page renders to a client's browser.

Let's target by id to hide the navigation menu in our MOVE Gear site.

# jQuery Objects

Now that you've seen some jQuery in action, let's dive a bit deeper into syntax. You've probably noticed the $ symbol before anything we target. The $ symbol is an alias for the jQuery function. The $ symbol and jQuery are interchangeable.

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:

const $jQueryObject = $('.someClass');

Notice our variable name, $jQueryObject, starts with the ($) character. It is best practice to name jQuery object variables with a leading $. It is a naming convention that reminds you and lets others know that a given variable is a jQuery object.

With this in mind, let's improve our code from the previous Exercise.

# Event Handlers

Now that we've got jQuery up and running, let's give it a little gas! 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(); })

In the example above, we use .on() to add the click event handler to the #login element. Inside of the callback function, we use the .show() method to show the jQuery object saved in the $loginForm variable.

When a user clicks the login element, the login form appears.

Let's try it out on our site!

# Review: jQuery Introduction

That's just a taste of what you can do with jQuery. While there's still a lot to learn, in this unit you used jQuery to add dynamic behavior to a website. With jQuery objects, you can quickly target elements, handle events and add visual effects with just a few lines of code.

That said, when using any jQuery tools, it is important to consider how events and effects will appear on different platforms.

In this lesson, you learned:

* About JavaScript libraries.
* How to add jQuery to your web page.
* How to avoid loading errors with the .ready() method.
* About jQuery objects and their methods.
* How to target elements by id and class.
* Naming conventions for storing jQuery objects in JavaScript variables.
* How to add event handlers with .on().
* A couple jQuery effects, such as .hide()and .show().

Nice work!