There are three essential data types to know:

1. *String*: Any grouping of words or numbers surrounded by single quotes: ' ... ' or double quotes " ... ".
2. *Number*: Any number, including numbers with decimals, without quotes: 4, 8, 1516, 23.42.
3. *Boolean*: This is always one of two words. Either true or false, with no quotations.

var myString = "Iris";

var myNumber = 129;

var myBoolean = true;

console.log("Name: " + myString);

console.log("Lucky Number: " + myNumber);

console.log("Good joke? " + myBoolean);

We can ask JavaScript to print words to the console with this line of code:

console.log('Your message here.');

We can do math with operators like +, -, \*, and /. We can find the remainder after dividing two numbers with a modulus: %.

Math.floor(Math.random() \* 50);

In this case:

1. Math.random will generate a random number between 0 and 1.
2. We then multiplied that number by 50, so now we have a number between 0 and 50.
3. Then, Math.floor will round the number down to the nearest whole number.

We write a single line comment with // and a multi-line comment with /\* and \*/.

Here is how you declare a variable:

var myName = 'Arya'; console.log(myName); // Output: Arya

You can almost read it aloud: "Variable myName is equal to Arya."

Let's dissect that statement and look at its parts:

1. var, short for variable, is the JavaScript *keyword*that will create a new variable for us.
2. myName is chosen by a developer (that's you!). Notice that the word has no spaces, and each new word is capitalized. This is a common convention in JavaScript, and is called *camelCase*.

After the variable is declared, we can print the variable with: console.log(myName). This will print 'Arya'to the console.

We can use the + operator from earlier to interpolate (insert) a variable into a string, like this:

var myPet = 'armadillo'; console.log('I own a pet ' + myPet + '.'); // Output: 'I own a pet armadillo.'

In programming, making decisions with code is called *control flow*.

Note: Why is there a \ in 'I lead a muggle\'s life.'? Since the string is surrounded by single quotes, we can use a back slash to add a single quote within the string. This is called *escaping* a character.

1. To check if two things equal each other, we can use === (three equals in a row).
2. *Logical Operators*, like &&, ||, !==, and !, can compare two variables to see if a certain condition exists

switch statements look like this:

var groceryItem = 'papaya'; switch (groceryItem) { case 'tomato': console.log('Tomatoes are $0.49'); break; case 'lime': console.log('Limes are $1.49'); break; case 'papaya': console.log('Papayas are $1.29'); break; default: console.log('Invalid item'); break; }

Take a look at this code:

var calculatorOn = false; function pressPowerButton() { if (calculatorOn) { console.log('Calculator turning off.'); calculatorOn = false; } else { console.log('Calculator turning on.'); calculatorOn = true; } } pressPowerButton(); // Output: Calculator turning on. pressPowerButton(); // Output: Calculator turning off.

Parameters are variables that we can set when we call the function. For example:

function multiplyByThirteen(inputNumber) { console.log(inputNumber \* 13); } multiplyByThirteen(9); // Output: 117



* *Scope* is the idea in programming that some variables are acessible/inaccessible from other parts of the program.
* *Global Scope* refers to variables that are accessible to every part of the program.
* *Functional Scope* refers to variables created inside functions, which are not accessible outside of its block.

Arrays are JavaScript's way of making lists. These lists can store different data types and they are ordered, meaning the position of each list item is numbered by JavaScript.

We can select the first item in an array like this:

var bucketList = ['Rappel into a cave', 'Take a falconry class', 'Learn to juggle'];

var listItem = bucketList[0];

console.log(listItem); // Output: 'Rappel into a cave'

Notice that you get undefined when you try to print position 3. That's because the array does not have an item at position 3, which is the fourth slot.

Fun fact: You can also access each individual character in a string the same way you do with arrays. For instance, you can write:

var hello = 'Hello World';

console.log(hello[6]); // Output: W

'W' will be the output since it's the character in the 6th position. This works because JavaScript internally stores strings in a similar way that it stores arrays.

We can find this out by using one of an array's built in *properties*, called .length.

First, push() allows us to add items to the end of an array. Here is an example of how this is used:

var bucketList = ['item 0', 'item 1', 'item 2'];

bucketList.push('item 3', 'item 4');

pop() is similar to push(), except that it deletes the last item of an array.

No function peek() used in JavaScript.

There are two kinds of loops we will learn in this lesson:

1. for loops, which let us loop a block of code a known amount of times.
2. while loops, which let us loop a block of code an unknown amount of times.

We can do this with for loops.

The syntax looks like this:

var animals = ["Grizzly Bear", "Sloth", "Sea Lion"];

for (var i = 0; i < animals.length; i++) {

console.log(animals[i]);

}

for loops are great, but they have a limitation: you have to know how many times you want the loop to run.

That's the purpose of the while loop. It looks like similar to a for loop. Check it out:

while (condition) { // code block that loops until condition is false }

We can link a JavaScript file to HTML by including it as the src of a <script> tag inside of an HTML file, like this:

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

Inside **main.js**, write:

alert('Hello JavaScript!');

Note: alert is a JavaScript function that will create a pop-up window with text inside it.

The *Document Object Model*, commonly referred to as the *DOM*', is the term for *elements* in an HTML file. Elements are any HTML code denoted by HTML tags, like <div>, <a>, or <p>.

We can select an HTML element with JavaScript by selecting its class attribute, like this:

var header = document.getElementsByClassName('example-class-name');

This would find an element like this in the HTML:

<div class='example-class-name'> ... </div>

We've just covered how to select HTML elements using the syntax: document.getElementsByClassName.

A great library for interacting with the DOM is jQuery.

jQuery is a library written in JavaScript. The syntax and functions it contains will help us interact with DOM efficiently.

In order to use jQuery, we need to:

1. Include jQuery in our project. jQuery is a library, which means it is a set of code in a file, therefore we will need to link that file in our HTML in order to access it.

Once we link it in our HTML file, we can use its functions and syntax in our **js/main.js** file.

1. Once linked, we'll need to make sure our HTML is loaded before we run our jQuery and JavaScript code.

This will prevent our jQuery and JavaScript code from running before the elements they select are rendered.

<script src='https://code.jquery.com/jquery-3.1.0.min.js'></script>

The link to jQuery needs to be above the link to the **js/main.js** file, which will give **main.js** access to the jQuery library.

Note: If you're curious, you can see all the code that makes up jQuery [here](https://code.jquery.com/jquery-3.1.0.js). You'll notice that jQuery is just JavaScript!

jQuery has a built in function to check if the page is ready before it will run our code. After the mainfunction, write this code:

$(document).ready(main);

Notice that we put main inside the parentheses of ready. main here is a callback, which means that our code will wait until the document (in other words, the DOM) is loaded, or ready. When it is, then it will execute the main function. jQuery calls back to the main function, therefore it's a callback.

In the event that our HTML and CSS took 5 minutes to load, this code would wait until it loaded completely before running

With plain JavaScript we selected an HTML element with this code:

document.getElementsByClassName('skillset');

With jQuery we can select the same element with:

$('.skillset');

1. We can wrap any CSS selector, like class, id, or tag, with $('.example-class') to select it with jQuery.
2. The selectors jQuery uses are the exact same as CSS selectors. For instance, if there's an element with a class of supporting-text, you could select it with $('.supporting-text'). Another example, if an element had an id of 'header', you could select it with $('#header').
3. Note: It is a common convention to name variables that hold jQuery selectors with a dollar sign $.

First off, it would be nice to make the page fade in when loaded.

To make a page fade in, it must first be hidden. We can hide elements with jQuery with a function named hide.

$('.my-selector').hide();

1. We attached the hide function directly to the jQuery selector.
2. The hide function will add the CSS property display: none to the DOM element from the page, which will hide it.

True to its name, fadeIn will fade an element in over a period of time in milliseconds. It looks like this:

$('.example-class').fadeIn(400);

1. Just like before, we can attach fadeIn directly to a jQuery selector.
2. Within fadeIn's parentheses, we can specify how long we want the fade to last in milliseconds. 400is the default.
3. The example code will fade in the '.example-class' element over 0.4 seconds. Not to fade after 0.4s.

The next feature we'd like to build is making the 'Recent Projects' clickable. When clicked, the button should show the individual projects, and when clicked again, it should hide the projects.

When clicked, the button should show the individual projects, and when clicked again, it should hide the projects.

jQuery can do this with an event listener function named on('click').

This function will wait for a click event, and when one occurs, it will execute a provided function.

$('.example-class').on('click', function() { // execute the code here when .example-class is clicked. });

1. $('.example-class') selects an HTML element with the class example-class.
2. .on('click', function() { ... }) adds a click listener to the selector. When it's clicked the function will execute the code within its block.

Query provides a function named show, which is the opposite of hide.

$('.example-class').show();

1. show is attached directly to the jQuery selector.
2. show will change the CSS attribute display: none to a visible display property, therefore showing the element.

jQuery provides a function named toggle that is helpful in this situation. toggle will hide or show an element, each time it is triggered.

$('example-class').toggle();

1. toggle can be called directly on an jQuery selector.
2. When toggle is executed, it will hide or show the element that the selector points to. If the element is currently hidden, toggle will show the element, and vice versa.

We can toggle a CSS class with a jQuery function named toggleClass. The syntax looks like this:

$('.example-class').toggleClass('active')

1. .toggleClass is a function that will toggle a CSS class on the jQuery selector it's connected to. If the element has the class applied to it, toggleClass will remove it, and if the element does not have the class, it will add it.
2. 'active' is the class that we will toggle on and off. Notice that toggleClass does not require us to include the period before 'active' since it's already expecting a CSS class.

We can select the specific element we clicked on with the jQuery selector $(this).

$('.example-class').on('click', function() { $(this).toggleClass('active'); });

1. $(this) selects the clicked element. If there are multiple elements with a class of .example-class, this will only toggle the class of the one that is clicked on.
2. Notice that $(this) does not require quotes around it, since it is not a CSS class. Instead, thisis a JavaScript keyword.
3. $(this) behaves just like our other selectors. We can attach toggleClass or toggle to it in the same way.

jQuery has a function named next to help us select elements that are next to another element. If we have this in our HTML:

<div class='item-one'> ... </div> <div class='item-two'> ... </div>

If we wanted to hide item-two, we could write:

$('.item-one').next().hide();

1. We can attach next to any jQuery selector to select the next direct element.
2. Then, we can attach any jQuery function to next(). In this case, we attached hide, which would hide the next element after the $('.item-one') element.

We can change the text of an element with the jQuery function text.

$('.my-selector').text('Hello world!');

1. text attaches directly to a jQuery selector.
2. Inside of text's parentheses, we can provide text that will become the text of our DOM element. The text we supply will replace any existing text, and if the element has no pre-existing text, text will add it.

Let's instead make the projects slide onto the page when we click the 'Recent Projects' button and then slide off the page when we click the button again.

jQuery provides a method named slideToggle that can animate an element's entrance and exit.

$('.example-class').slideToggle(400);

1. slideToggle can be called directly on a jQuery selector.
2. slideToggle also takes a parameter of milliseconds that the animation should last. The default is 400 milliseconds, or 0.4 seconds.