JAVASCRIPT FUNDAMENTALS QUICK LABS

V 1.3 – October 2022

CONTENTS

[Quick Labs Environment Set-Up 3](#_Toc115169919)

[**Code Editing** 3](#_Toc115169920)

[**NodeJS** 3](#_Toc115169921)

[**Alternative method for outputting results of JavaScript files** 3](#_Toc115169922)

[Quick Lab 1 - JavaScript Development Environment 4](#_Toc115169923)

[Quick Lab 2 - JavaScript Types 8](#_Toc115169924)

[Quick Lab 3 - JavaScript Operators 10](#_Toc115169925)

[Quick Lab 4a - JavaScript Conditionals 12](#_Toc115169926)

[Quick Lab 4b - JavaScript Loops 14](#_Toc115169927)

[Quick Lab 5 - Error Handling and Debugging 16](#_Toc115169928)

[Quick Lab 6 - JavaScript Arrays 17](#_Toc115169929)

[Quick Lab 7 - JavaScript Functions 20](#_Toc115169930)

[Quick Lab 8 - JavaScript Maps 24](#_Toc115169931)

[Quick Lab 9 - JavaScript Objects 25](#_Toc115169932)

[Quick Lab 10 - JavaScript and the DOM 27](#_Toc115169933)

[Quick Lab 11 - JavaScript Manipulating Styles 29](#_Toc115169934)

[Quick Lab 12 - JavaScript Events Styles 31](#_Toc115169935)

[Quick Lab 13a - JavaScript Form Validation 33](#_Toc115169936)

[Quick Lab 13b - JavaScript and Regular Expressions 35](#_Toc115169937)

[Quick Lab 14 - JavaScript Modules 36](#_Toc115169938)

[Quick Lab 15a - Object Oriented JavaScript 37](#_Toc115169939)

[Quick Lab 15b - Object Oriented JavaScript 39](#_Toc115169940)

[Quick Lab 16a - Asynchronous JavaScript - JSON 41](#_Toc115169941)

[Quick Lab 16b - Asynchronous JavaScript - Promises 43](#_Toc115169942)

[Quick Lab 16c - Asynchronous JavaScript - Fetch 44](#_Toc115169943)

[Quick Lab 16d - Asynchronous JavaScript - async/await 46](#_Toc115169944)

Quick Lab 2 - JavaScript Types

Objectives

* To understand how types work in JavaScript.

Activity

1. In VSCode, open the file index.js from the QuickLabs/02\_JavaScriptTypes/starter/src folder.
2. Add code to declare a number and a console.log to output it:

let numTest = 45.324568;

console.log(numTest);

1. Save the file (note the • that appears next to the file name in the tab to show that the file is not saved yet and changes to × when the file is saved).
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/02\_JavaScriptTypes/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

1. When the browser opens, view the console and check that the number is displayed.

Press F12 to see the developer tools and choose the Console tab. You should see the value of the numTest variable displayed.

You have created a Number data type object in the program stack, this is a 64bit number. Next, we will explore the Number type a little more with some of its methods.

We will create a new variable called twoDecimalPoints and use a method to truncate the number.

1. Under the last code you wrote in index.js, add the following code:

let twoDecimalPoints = numTest.toFixed(2);

console.log(twoDecimalPoints);

1. Save the file and your browser should automatically refresh to display the value of twoDecimalPoints as 45.32. Notice that this number is now BLACK not BLUE? The toFixed function converts the number to a string!
2. Under your last line of code in index.js, create a stringTest variable, as shown in the code segment below.

(Please be sure to add the text exactly as it appears; otherwise, the notes will not match up to what you will see in the console.)

let stringTest = `I am the very model of a modern major general`;

let indexOfM = stringTest.indexOf(`m`);

console.log(indexOfM);

1. Save the code and observer the browser console.

You will see a value of 3, examine the string and you will see that the m is the fourth character, so there are three characters before the first m.

1. Change the m within the indexOf method call to a capital M, save and observe the output in the browser again.

This time, the console.log will return a -1 value. The -1 value is telling us that there is no match within the string at all proving that string searches are case sensitive.

What if we convert the string to upper case?

1. Before the indexOfM line, add the following code:

stringTest = stringTest.toUpperCase();

1. Save and observe the output in the browser again.

The output will, once again, give a value of 3. Behind the scenes, the string is an indexed collection of characters and the search function is making its way through the letters character by character until it makes a match.

With that concept in mind, we will use the principals to learn how to slice a string.

1. Add the following code under the last line, then capture start and end in a console.log, save and observe the output.

let start = stringTest.indexOf("MODEL");

let end = stringTest.lastIndexOf('MAJOR');

This time, we have matched based upon words, but you could search for file paths or extensions; for instance, if we were reading from a form.

The two integer values held can be used to create a substring from the longer one using string's substring method.

1. Add the following lines of code to the end of your code, save and observe the output.

let subStr = stringTest.substring(start, end);

console.log(subStr);

The console should now return a value of "MODEL OF A MODERN".

Let's finish up this exercise by writing this content to the browser window using the document.write method (we will look at the document object in more depth later on).

1. Add the following lines of code to the end of your code, save and observe the output.

document.write("<p>" + subStr + "</p>");

We have used an operator here: the + sign, which we have used to concatenate the string together and mix our string value with some hard-coded HTML, to create new content for the page. With that done, let’s learn some more about operators.

This is the end of Quick Lab 2

Quick Lab 3 - JavaScript Operators

Objectives

* To understand how operators work in JavaScript.

Activity

1. In VSCode, open the file index.js from the QuickLabs/03\_JavaScriptOperators/solution/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/03\_JavaScriptOperators /solution folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

1. Examine the following operations and write the result you expect before you look at any code output!

|  |  |
| --- | --- |
| Arithmetic operators | |
| Operation | Result |
| console.log(5 + 5); |  |
| console.log(5 \* 10); |  |
| console.log(10 % 3); |  |
| console.log(5 + 10 / 2 \* 5 - 10); |  |
| console.log((6 + 10) / 2 \* 5 - 10); |  |

1. Uncomment the Arithmetic Operators section of the code - highlight the required lines to uncomment and press CTRL + / (or CMD + / on MacOS).
2. Save the file and check the console output against what you expected.

Instructions continue on the next page…

1. Repeat for assignment operators, assuming x is initialised as 0 and the statements are processed sequentially.

|  |  |
| --- | --- |
| Assignment operators | |
| Operation | Result |
| console.log(x = x + 1); |  |
| console.log(x+= 1); |  |
| console.log(x++); |  |
| console.log(++x); |  |

1. Now, we will move onto relational operators. Every expression will evaluate as either true or false.

|  |  |
| --- | --- |
| Relational operators | |
| Operation | Result |
| console.log(5 > 3); |  |
| console.log(3 != 3); |  |
| console.log(3 <= 2 && 5 >2); |  |
| console.log(!5>3); |  |

1. Finally, we will explore what happens with mismatched types.

|  |  |
| --- | --- |
| Mismatched types | |
| Operation | Result |
| console.log(5 + "5"); |  |
| console.log(5 + true); |  |
| console.log(5 \* "5"); |  |
| console.log(1 == true); |  |
| console.log(1 === true); |  |

This is the end of Quick Lab 3

Quick Lab 4a - JavaScript Conditionals

Objectives

* To investigate the JavaScript flow of conditional statements.

Activity

1. In VSCode, open the file index.js from the QuickLabs/04a\_JavaScriptConditionals/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/04a\_JavaScriptConditionals/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

1. Declare a variable called age and initialise it to be 15.
2. Enter the following code to create the if below the variable declaration:

if (age <= 17) {

console.log("Underage");

} else {

console.log("18 or over");

}

1. Save the page and observe the output on the console of the developer tools (F12).
2. Change the value to 42 and check that the output has updated.

We now have a simple if statement in place. Now it’s time to take our code a step further and add an else if caveat into our code. The else if is a further, logical check delivering a Boolean value. Additional checks are examined in order; when a true is evaluated, the program leaves the if statement, so the order is important.

1. The code from the previous part has been slightly amended and now needs to be included under the comment for part 2.

if (age <= 17) {

console.log("Underage");

} else if (/\*Remove me and insert your code here\*/) {

console.log("Insurable");

} else {

console.log("out of range");

}

The else if statement is going to check if the age variable is between 18 and 65, so:

do we need to use an and statement to achieve this or will a simple check for <=65 suffice?

1. Replace the code in the else if statement to achieve the desired result. Refer back to your notes or ask your instructor for help if you get stuck.
2. Save the page and observe the output in the browser.
3. Test that your code works by setting age to the following values:

* 10
* 50
* 80

1. Use a ternary statement to achieve the same results that was gained in parts 6-8.

This is the end of Quick Lab 4a

Quick Lab 4b - JavaScript Loops

Objectives

* To investigate JavaScript looping statements.

Activity

1. In VSCode, open the file index.js from the QuickLabs/04b\_JavaScriptLoops/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/04b\_JavaScriptLoops/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

The for loop has three arguments: the counter, the condition and the iterator. You are going to code in a simple for loop where the following properties need to be set:

|  |  |
| --- | --- |
| parameter | value |
| counter | Variable name i set to 1 |
| condition | i is less than 10 |
| iterator | Each loop must add 1 to the value of i |

1. Enter the following code, amended appropriately to achieve this:

for (counter; condition; iterator) {

console.log(i);

}

1. How many times do you expect the loop to execute? \_\_\_\_\_\_\_\_\_\_\_
2. Save the file and observe the output in the browser to check your assumptions.
3. Write a while loop that has the following rules:

|  |  |
| --- | --- |
| parameter | value |
| initial conditions | Variable name x set to 2 and loopCounter set to 0 |
| condition | x is less than 10000 |
| iterator | Each loop must square the value of x and add 1 to loopCounter |
| action | Each loop must log out the value of x and loopCounter |

1. Once you have resolved this, save the code and observe the output in the browser.

Further activities

Only attempt the further activities, if there is sufficient time remaining. Your instructor can help you with this if you need it, but we hope you can start ranging out on your own.

1. Alter the for loop you created in Step 8 of the exercise to count down rather than up.

This is the end of Quick Lab 4b

Quick Lab 5 - Error Handling and Debugging

1. There is no QuickLab for this module.

This is the end of Quick Lab 5

Quick Lab 6 - JavaScript Arrays

Objectives

* To investigate JavaScript arrays and their functions.

Activity

1. In VSCode, open the file index.js from the QuickLabs/06\_JavaScriptArrays/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/06\_JavaScriptArrays/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

1. Declare an array called quote that contains four strings, "I", "am" "your" and "friend".
2. Log the array to the console.
3. Save the file and observe the browser to check the output - you should see details of an array and expanding it will show the values and their indexes.
4. Access the index of the array that contains the string "your" and log the array element to the console.

console.log(quote[2]);

1. Save the file and observe the browser to check the output.
2. Using the pop function, remove the string "friend" from the end of the array.
3. Using the push function, add the string "father" to the end of the array.
4. Log the array to the console again.
5. Save the file and observe the browser to check the output.
6. Use the unshift function to add the string "Luke" to the start of the array.
7. Log the array to the console again.
8. Save the file and observe the browser to check the output.

There are two things wrong with the output. The first is that it the string is concatenated by commas and the second is that the 'quote' is actually a misquote! We're going to generate an output in a different way by looping through the array and creating a new string. We're also going to fix the misquote by detecting the erroneous word in the array and replacing it with the correct word! Let's do the latter first:

To do this, we are going to detect if indeed the erroneous word is in the array. If it’s, we are going to find the index, that the word is at and then, use this information to replace that index with the correct word.

1. Declare a variable called erroneousWord and set it to a string with the misquoted word from the array (it's Luke if you didn't know!).
2. Set a variable called lukeIsHere using the find() function to see if the quote array contains the erroneousWord. The code is:

let lukeIsHere = quote.find(n => { return n === erroneousWord});

The syntax inside the find function will feel a little alien at the moment but go with it as it’s explained later in the course.

1. Declare a variable called lukeIsAt without assigning it.
2. If lukeIsHere has been set to true, find the index that the erroneousWord sits at using the findIndex() function and set lukeIsAt to the value of the index. The code is:

lukeIsAt = quote.findIndex(n => { return n === erroneousWord});

1. Still inside the if block, use the value of lukeIsAt to set that index in the quote array to the string "No".

if (lukeIsHere) {

lukeIsAt = quote.findIndex(n => {

return n === erroneousWord

});

quote[lukeIsAt] = "No";

}

1. Log out of the array and ensure that the expected result is outputted in the browser.

To sort out the display of the quote, we need to create a new string, then loop through the array, adding a space into the string after each word, apart from the first word, (to which we'll append a comma and a space) and the final word, (to which we will append an exclamation mark).

1. Declare a variable called output and set it to be an empty string.
2. Create a for loop that:
3. Loops through the quote array.
4. Executes when the loop counter is less than the length of the array.
5. Adds an exclamation mark to the output string, if we are at the last element in the array.
6. Adds a comma and a space to the output string, if the current element is 'No'.
7. Otherwise adds a space to the output string.

for (let i = 0, j = quote.length; i < j; i++) {

if (i === j - 1) {

output += quote[i] + '!';

} else if (quote[i] === 'No') {

output += quote[i] + ', ';

} else {

output += quote[i] + ' '

}

}

1. Log out the output string.
2. Save the file and then check your browser output to ensure that the correct quote is displayed.

No, I am your father!

This is the end of Quick Lab 6

Quick Lab 7 - JavaScript Functions

Objectives

* To investigate JavaScript functions.

Activity

1. In VSCode, open the file index.js from the QuickLabs/07\_JavaScriptFunctions/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/07\_JavaScriptFunctions/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

Part 1 – Defining and using functions and understanding scope

In this part of the exercise, you will practice writing a function to perform some actions and investigate the scope of the variables within it.

In addition, you will look at the scope of variables -notice that the script already contains an array of objects, that contain some details about different films.

1. Under the comment for Part 1, declare a function called findMovie that takes an argument called movieTitle.
2. In the body of the function create a for…of loop of the movies array where:
3. The loop body should:
4. Check to see if the current movie title is the same as the movieTitle passed into the function and if it’s, log out details of the movie in a suitable string;
5. Log out the value of movie before the loop's closing brace;
6. The value of movie should be logged before the function closes.
7. Call the findMovie function with an argument of "Star Wars".
8. Log out the value of movie.
9. At this point, save your file and check the output.

The expected outcome is that there is a Reference Error - but which console.log is, or console.logs are, causing it/them?

1. Comment out the offending console.log(s) and check your output.

You should see all 5 movies logged, with the string you wrote for a found movie being outputted before the movie object - for it’s logged itself.

Two of the console.log statements added, produced a Reference Error. This is because of the scope of the variable movie. As it’s declared as part of the for…of loop, its scope is limited to inside the body of this block (i.e. between the { } that immediately follows the for). As long as execution remains inside this loop, the variable movie is in scope.

Once the loop finishes and execution returns to the level above (i.e. back to the body of the function) and the variable movie is no longer in scope - referring to it in the code will cause the Reference Error.

It follows that if movie is not available here, it will also not be available after the line that calls the function has completed execution - again causing a Reference Error.

Note: Because movieTitle is part of the function block, it’s accessible throughout the execution of the function, including inside any blocks that are used within the function body (i.e. in the for and if blocks).

Note: Because the const movies is declared at script level (i.e. inside the script tag) and at the top of it, it’s available to all blocks of code that live inside this script tag.

1. Under the last line, define a variable called movie set to the value of   
   "Thor: Ragnorok".
2. Uncomment the console.logs of movies and add another log of the value of movie under the declaration of the variable from step 11.
3. Observe the results.

What you should see this time is two undefined values.

1. In index.html, comment out the first script tag and uncomment the second.
2. Save and reload the page.

You will notice that Reference Errors as before are present.

1. Change the declaration of movie to have the var keyword in front of it (rather than let) and make sure that all console.logs are uncommented.
2. Observe the results.

What you should see this time is two undefined values again. The differences are all to do with concepts called hoisting and 'temporal dead zones'. More details of which can be found, with a good explanation of let at:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

This error was not shown when working in the module due to the scope rules creating a new scope for each module.

1. Swap the comments around the script tags in index.html again.
2. Back in index.js, add a call to the findMovie with the argument set to movie, saving and observe the output.

There is no output from the function call for findMovie - using the defined movie variable. This is because there is no movie in the movies array with the title   
"Thor: Ragnorok" and therefore, the loop completes without ever entering the if condition.

Part 2 – Creating functions that return data

Functions rarely just execute code and then the program continues. It’s more usual that a function will manipulate some data and then return some data which can then be used further. This part of the exercise will allow you to experiment with returning data from a function.

1. Comment out the section of code for Part 1, leaving the movies array intact and work under the comment for Part 2.
2. Declare a function called returnMovie that takes movieTitle as an argument and has a function body that:
3. Uses a for…of loop on the movies array with a loop body that:
4. Checks to see if the title property of the current movie matches the movieTitle supplied to the function;
5. If it does, it should simply return the current movie;
6. Logs out the current value of movie;
7. Logs out "Any text, any text at all".
8. In the body of the script, declare a variable called myMovie and set it to the result of calling returnMovie with an argument of "Avengers: Infinity War".
9. Log out the value of myMovie, save and observe the output.

If you have created your returnMovie function you should observe the following:

* Each of the movies that appear BEFORE the selected movie are logged out -as the loop has executed for each of these movies
* The movies that are AFTER the selected movie are not logged out -because the presence of the return statement stops the execution of the loop and indeed the function (so that "Any text, any text at all" is also not shown)
* The execution 'returns' to its call point with the value of whatever is returned

1. Access the properties of myMovie to produce and log a string as a sentence with them in it, saving and observing your output.

What happens if we try to pass a movie title that doesn't exist in the movies array into returnMovie? Let's find out!

1. Declare a variable myOtherMovie and set its value to a call to returnMovie with an argument of "Thor: Ragnorok".
2. Log out the value of myOtherMovie and observe the output.

The first thing that we notice is that the whole of the movies array has been logged out and the text "Any text, any text at all". This is because the title was not found and the function completed its execution fully and never returned a value

…or did it?

The next thing that we notice is that the console.log of myOtherMovie has outputted undefined. It looks like we've never set the value of myOtherMovie - because we haven't! Let's fix that…

1. Comment out the logging of "Any text…" and add a line that returns the string `Movie not found`.
2. Save and observe the output.

The logging of myOtherMovie now outputs "Movie not found".

The code is still not very reusable as if I want to log out the details of a movie, I have to supply the string inside a console.log. Also, what happens if it’s already a string (because it’s a movie not in the array)? Our output would be very messy! Step up another function!

1. Create a function called myMovieDetails that takes a variable anyMovie as an argument.
2. Check that the typeof anyMovie is an 'object' and return a suitable string if it’s and simply return anyMovie if it’sn't.
3. Inside a console.log, call myMovieDetails with an argument of myOtherMovie.
4. Observe the results.

It should output: Movie not found.

Can we use a function as the argument to another function? Yes we can!

1. Repeat the last instruction instead passing in returnMovie with an argument of "Jaws" as the argument to the myMovieDetails function.
2. Observe the results.

It should output the details for Jaws in your defined string.

This is the end of Quick Lab 7

Quick Lab 8 - JavaScript Maps

Objectives

* To create a Map in JavaScript

Activity

1. In VSCode, open the file index.js from the QuickLabs/08\_JavaScriptMaps/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/08\_JavaScriptMaps /starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

For clarity of instructions, the steps to save and observe the browser have been omitted after each instruction that affects the output.

1. Create a new Map object called hanSolo and add the following key/value pairs to it:
2. vehicle - Millenium Falcon;
3. bff - Chebacca;
4. sweetheart - Leia.
5. Access the properties that you have just declared by logging out the following details:
6. The size of the map hanSolo;
7. Han Solo's vehicle name (HINT - use the Map.get() method);
8. If Han Solo has a sweetheart (HINT: use the Map.has() method);
9. If Han Solo is a (has) Jedi.
10. Add another key/value pair to hanSolo that sets a key son to Ben and log this new property to the console.
11. Iterate over hanSolo using a for…of loop that uses both the key and the value of each pair, logging out each pair.
12. Manipulate the map by:
13. Changing the value of bff to Luke and log out hanSolo;
14. Deleting the key/value pair son and log out hanSolo;
15. Clearing the Map and logging it out.

This is the end of Quick Lab 8

Quick Lab 9 - JavaScript Objects

Objectives

* To understand how to declare and destructure objects

Activity

1. In VSCode, open the file index.js from the QuickLabs/09\_JavaScriptObjects/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/09\_JavaScriptObjects/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

For clarity of instructions, the steps to save and observe the browser have been omitted after each instruction that affects the output.

1. Create a new Object called darthVader and add the following key/value pairs to it:
2. allegiance - Empire;
3. weapon - lightsabre;
4. sith - true (boolean value).
5. Access the properties that you have just declared by logging out the following details:
6. DarthVader's allegiance;
7. Darth Vader's weapon;
8. If Darth Vader is a sith;
9. The value of Jedi from Darth Vader; (even though it’sn't defined in the object)
10. The number of properties Darth Vader has (see the line of code below for this)

console.log(Object.keys(darthVader).length);

Quick explanation - Object.keys is a function that takes an object and returns an array of the keys in it. By appending .length to it, we return the number of keys in the object.

1. Add key/value pairs to darthVader that:
2. Sets a key of children to 2;
3. Sets a key of childNames to the array ['Luke', 'Leia'];

And then log the children property and the value of the first element in the childNames array.

1. Iterate over darthVader using a for…in loop that uses both the key and the value of each pair, logging out each pair.
2. Manipulate the object by:
3. Changing the value of allegiance to The light side and log out darthVader;
4. Deleting the key/value pair children and log out darthVader;

Hint: use the code below:

delete darthVader.children;

1. Destructuring the object, setting a variable for each of the keys in the object to the corresponding value in the object:

let{allegiance, weapon, sith, childNames} = darthVader;

1. Console each individual variable out to ensure that they have been set.
2. Clearing the object and logging it out.

This is the end of Quick Lab 9

Quick Lab 10 - JavaScript and the DOM

Objectives

* To be able to manipulate the DOM by creating and adding content.

Activity

1. In VSCode, open the file index.js from the QuickLabs/10\_JavaScriptDOM/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/10\_JavaScriptDOM/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view!

1. Under the array declaration, create a function called buildP that takes 1 parameter called placeholder.

The parameter placeholder will hold a reference to the DOM element which we want to append new content to. The DOM programming mechanism allows us to create new elements by type using the createElement function. Any new elements have to be added to the DOM or it will not display on the page and it will not have any content, unless we give it some! When DOM programming, it’s important to work in the correct order:

1. Create the new element;
2. Create any text nodes that need to be attached to the new element;
3. Attach the text nodes to the new element;
4. Attach the new element to the appropriate existing DOM element;
5. Inside the function declare a variable called p and set it to a new element of type p using the createElement function:

document.createElement('p')

1. Create a text node to attach to the new paragraph called text with any string you want in it using the line of code below (the text given is an example):

document.createTextNode(`Have you tried turning it off and back on again?`)

1. Append the text node to the paragraph p using the appendChild function.
2. Append the paragraph p to the placeholder that was passed into the function.

The function is now complete and ready to go. All we need to do is call it and pass in the correct parameters. The new <p> element we are going to create needs to be appended to a container in the page. This could be another element on the page or even the document root itself. We will add it to the <div id=”placeholder”> found in index.html.

1. Under the function definition, call the function setting the placeholder argument with the element found by using:

document.querySelector('#placeholder')

When we created the buildP function, we added a second parameter to the function that we have not used, or even set in the function so far -this is one of JavaScript’s nice features. You can either add none, the first, or all parameters.

1. Save your code and refer to your browser.

You should see that the text and paragraph you created with JavaScript are now part of the page. Verify this by inspecting the elements in the Developer Tools.

A second parameter of the buildP function is going to be used to create and append multiple paragraph elements. We will always want at least 1 paragraph but maybe more - time to use the do…while loop.

1. Add a second parameter called num to the arguments for buildP.
2. At the top of the buildP function, declare a variable i initialised to 0.
3. Surround the rest of the code in the buildP function with a do…while loop that exits when i reaches the number supplied to the function (remember to increment i).
4. Amend the call to the buildP function to add a second argument of any number you choose.
5. Save and refer to your browser, checking the output is as expected.
6. Try a few different values and observe the page.

The final piece in this part of the exercise is to randomise the colours used to display our text in. To do this, we will use the array of colours already defined and a randomly generated value in the range of the indexes of the array.

**Math.random** gives us a value somewhere between 0 and 1. Indexes of arrays are integer values as are their length. The parseInt function will be used to return the nearest integer to a value which we are going to calculate, using the randomly generated number and the length of the colours array.

1. Before the line that appends the text node to the paragraph add the following line:

p.style.color = colours[parseInt(Math.random() \* colours.length)]

1. Save and refer to the output of the browser.

This is the end of Quick Lab 10

Quick Lab 11 - JavaScript Manipulating Styles

Objectives

* To be able to and examine and manipulate CSS styles using JavaScript.

Activity

1. In VSCode, open the file index.js from the QuickLabs/11\_JavaScriptStyles/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/11\_JavaScriptStyles/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view except the provided code!

The are 3 lines of code already provided. These are to enable the buttons on the webpage to work when clicked.

1. Under the provided lines of code but before the blueToRed function, declare a constant called blueParagraph and set this to be the element that has the id blueParagraph.
2. console.log the color obtained from its style object.
3. console.log the color obtained from the result of running it through getComputedStyle function:

console.log(getComputedStyle(blueParagraph).color);

1. Populate the function blueToRed with code that:
2. Sets the color value in the style object to red;
3. Logs out the new value of the color from the style object;
4. Logs out the value of color from the getComputerStyle function;
5. Sets blueParagraph's textContent to be its own value when it has the replace function applied to it to remove 'blue' and replace it with 'red';

blueParagraph.textContent = blueParagraph.textContent.replace('blue', 'red');

1. Save the file and check that the page behaves as expected.

You should notice that there are blank lines on the console where the style.color value should be from the first call. This is because the style.color is undefined as it has been set by a CSS class. When the button is clicked, red and the rgb value are printed as the color value has been set in-line.

1. Immediately after the blueToRed function, declare a constant called greenParagraphs and set this to be an array of elements with class greenBg applied to them.
2. Log out the style.backgroundColor and the computed style background colour of the first element in the array:

console.log(greenParagraphs[0].style.color);

console.log(getComputedStyle(greenParagraphs[0]).color);

1. Populate the greenToPink function with a for…of loop that takes each greenParagraph of greenParagraphs and:
2. Sets the className of greenParagraph to hotpinkBg;
3. Logs out the backgroundColor from the style object and the getComputedStyle function
4. Sets greenParagraph's textContent to be its own value when it has the replace function applied to it to remove 'green' and replace it with 'pink';
5. Save the page and observe the results when the button is clicked.

The value for backgroundColor from the style object is missing initially and when the button is clicked here because, in both cases, the value is set by a CSS class.

1. Immediately after the greenToPink function, declare a constant called tnrParagraph and set this to be the element with an id of tnrParagraph.
2. Log out the style.fontFamily value and the computed fontFamily.
3. Populate the tnrToArial function with code that:
4. Sets the style.fontFamily to arial;
5. Logs out the style.fontFamily value and the computed fontFamily;
6. Sets tnrParagraph's textContent to be its own value when it has the replace function applied to it to remove 'Times New Roman' and replace it with 'Arial';
7. Sets textContent of greenParagraph to newText.
8. Save the file and check that it behaves as expected.

This is the end of Quick Lab 11

Quick Lab 12 - JavaScript Events Styles

Objectives

* To be able to add and remove JavaScript events.

Activity

1. In VSCode, open the file index.js from the QuickLabs/12\_JavaScriptEvents/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/12\_JavaScriptEvents/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

The browser window will open, but there will be nothing to view except the provided code!

1. Under the comment, add event listeners to the elements, using the functions shown:

|  |  |  |
| --- | --- | --- |
| Element | Event | Handler |
| button with id of textCOLOUR | click | blueToRed |
| Button with id of bgcolour | click | greenToPink |
| Button with id of fonts | click | tnrToArial |

1. Save the file and make sure that the page functions as it did at the end of the last QuickLab.
2. Add an event listener to tnrParagraph that has an event of mouseover and a handler of mouseOver.
3. Add mouseOver as an arrow function that takes an argument of event:

const mouseOver = event => {

// function body here

}

1. Populate the function body:
2. Change the background colour of the event target to limegreen;

event.target.backgroundColor = 'limegreen';

1. Check to see if the textContent of the event target contains the word background
2. If it doesn't then add I have had my background colour changed on mouse over to it;
3. If it does then replace the word out with over in it.
4. Save the page and check to see if the new mouse over functionality works.

You should find that it doesn't. The explanation for this is in JavaScript's hoisting rules. Normal variables and functions are hoisted by JavaScript, meaning that it knows about them and their implementation no matter where they are declared. Arrow functions (and Classes - when using Object Oriented JavaScript) are not. Therefore, the page fails silently.

1. Move the registering of the event listener to beneath the declaration of the arrow function.
2. Save the file and you should see that the event now fires correctly.
3. Add another event handling function to tnrParagraph, using the previous steps as a template, to change the background colour to yellow when the mouse leaves the element and add/change the text accordingly - don’t forget to register the event on tnrParagraph!

The way that these new event handlers have been written means that they are reusable across any elements as they are not tightly bound to the element. When we pass in the event, event.target gives the reference to the element which fired the event.

1. Add an arrow function called elementClick. It should:
2. Receive event as an argument;
3. Set the background colour of the element that raised the event to white;
4. Change the text content of the element that raised the event to I have no event listeners attached to me now;
5. Remove the click event listener on the button with id of fonts;
6. Remove the click, mouseover and mouseout event listeners from the element that raised the event;
7. Checks the id of the element that raised the event and if it was tnrParagraph:
8. Changes the textContent of blueParagraph to Event listeners enabled;
9. Adds event listeners to blueParagraph for click, mouseover and mouseout.
10. If the element that raised the event didn't have an id of tnrParagraph it should:
11. Change the textContent of tnrParagraph to Event listeners enabled;
12. Adds event listeners to tnrParagraph for click, mouseover and mouseout.
13. Register the click event and handling function with tnrParagraph.
14. Save the file and check that all of the events are fired at the appropriate time. Use the developer tools and add console.logs if you wish to examine further.

This is the end of Quick Lab 12

Quick Lab 13a - JavaScript Form Validation

Objectives

* To be able to use JavaScript to validate fields on a form and intercept form submission.

Activity

1. In VSCode, open the file index.js from the QuickLabs/13a\_JavaScriptAndFormValidation/starter/src folder and the file index.html from the QuickLabs/13a\_JavaScriptAndFormValidation/starter Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
2. Ensure that the terminal is pointing to the QuickLabs/13a\_JavaScriptAndFormValidation/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

1. The browser window will open - click Submit and notice the change in the URL.
2. Enter some values and then click Submit again - notice that your form data is now part of the URL.
3. In index.html, remove novalidate from <form>.
4. Add the property required to the elements named name, email and gender.
5. Save the file, refresh and try to Submit an empty form.
6. Add a name to the input for Name and then click Submit again.
7. Add a name to the input for Email (not an email address) and click Submit.
8. Add [name@email.com](mailto:name@email.com) to the input for Email and click Submit.
9. Choose a value from the list for Gender and then click Submit

Default validation is applied to all fields that have the required property. However, this validation may not be exactly as we want. The form generates a GET request when the Submit button is clicked and this behaviour is often intercepted.

1. In index.js, write an arrow function called formSubmit that:
2. Takes an argument of event;
3. Stops the generation of the GET request;
4. Pops up an alert box to say that the form has been submitted.
5. Register a submit event with the form element using a handler of formSubmit.
6. Save the file and enter data onto the form, then click Submit.

This code should have stopped the page from refreshing with a new URL.

1. Add a loop to the formSubmit function that outputs the first 3 value**s** of the array included in the event target generated by the submit.

for(let i = 0; i < 3;i++) {

console.log(event.target[i].value);

}

1. Declare a constant called nameInput and set it to be the element with a name attribute of name. (Good practice - define all variables and constants at the top of the code)
2. Add an arrow function called validateNameLength that:
3. Receives an event as an argument;
4. Checks the length of the value to see if it’s at least 2:

if(event.target.value.length < 2) { }

1. Alerts Name not long enough if it’s not and
2. Puts the focus back on the input.

nameInput.focus();

1. Register a change event with nameInput, calling the function you have just defined.
2. Save the file and enter a single character in the name field, then move on to the Email field. You should get an alert message.
3. Verify that the alert message does not show when the name is at least 2 characters long.

This is the end of Quick Lab 13a

Quick Lab 13b - JavaScript and Regular Expressions

Objectives

* To be able to use regular expressions within JavaScript to validate fields on a form.

Activity

1. In a change from the norm, this QuickLab continues in the 13a QuickLab files, so make sure that you have got index.js open from QuickLabs/13a\_JavaScriptAndFormValidation/starter/src.
2. Declare a constant and set it to be the element for the email input.
3. Add an arrow function called validateEmailFormat that receives an event as an argument.
4. Within the function body test the value of the target of the event against the following regular expression:

/^\w+([\.-]?\w+)[\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/](mailto:*@\w+(%5b\.-%5d?\w+)*(\.\w%7b2,3%7d)+$/)

HINT: use the !([regExp].test(event.target.value)) as the condition for the if statement.

1. If the value of the event target does not meet this regular expression, produce an alert to warn that the email address is not in a valid format and set the form focus back into the email input.
2. Add a change event listener on the email input that uses the validation function to handle it.
3. Save the file and try moving on from the Email input at various stages of input. You should find that the minimum accepted format is [name@email.com](mailto:name@email.com).

This is the end of Quick Lab 13b

Quick Lab 14 - JavaScript Modules

Objectives

* To be able to use JavaScript Modules in development.

Activity

1. In VSCode, open the file index.js from the QuickLabs/14\_JavaScriptModules/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/14\_JavaScriptModule/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

1. The browser window will open showing the provided form.
2. Create 3 new files in the src folder:
3. formSubmit.js
4. validateName.js
5. validateEmail.js
6. Place any appropriate functions and variables in the files from index.js
7. Remember to include import statements ,where they need to be used, but are not in the file.
8. Remember to include export statements, where they are needed externally, to the file.
9. Save all files and check that the validation still functions as before.

This is the end of Quick Lab 14

Quick Lab 15a - Object Oriented JavaScript

Objectives

* To be able to use the class syntax in JavaScript and instances of it.

Activity

1. In VSCode, open the file index.js from the QuickLabs/15\_JavaScriptClasses/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/15\_JavaScriptClasses/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

1. In the src folder create a new file called motorvehicle.js.
2. Inside this file create a class, MotorVehicle and export as default.
3. Add a constructor that takes the following arguments:
4. make;
5. model;
6. wheels;
7. engineSize.
8. Inside the constructor, set the values of each - use the 'private' notation.
9. Add a \_speed property set to 0.
10. Add a 'getter' for each of the 5 properties.
11. Add an accelerate method that takes an argument of time and sets the speed to:

this.\_speed = this.\_speed + ((0.25 \* this.\_engineSize/this.\_wheels) \* time);

1. Add a brake method that takes and argument of time and sets the speed if it’s greater than 0 to:

this.\_speed =   
(this.\_speed - ((0.3 \* this.\_engineSize/this.\_wheels) \* time) > 0) ?  
this.\_speed - ((0.3 \* this.\_engineSize/this.\_wheels)\*time) : 0;

1. In the index.js file, import the MotorVehicle class.
2. Create an instance of MotorVehicle using myMake, myModel, 4, and 2000 as constructor arguments:

const myVehicle = new MotorVehicle("myMake", "myModel", 4, 2000);

1. Log out myVehicle and then it's speed.
2. Make the vehicle accelerate for 10:

myVehicle.accelerate(10);

1. Log out its speed again - it should be 1250.
2. brake for 5 and log out the speed - it should be 500.
3. brake again for 5 and log out the speed again - it should be 0.

This is the end of Quick Lab 15a

Quick Lab 15b - Object Oriented JavaScript

Objectives

* To be able to use the class extends syntax in JavaScript and instances of it.

Activity

Continue working in the folder QuickLabs/15\_JavaScriptClasses/starter/src

1. Create a file called car.js and import the MotorVehicle class.
2. Add an export default class called Car that extends MotorVehicle.
3. Add a constructor that takes the arguments:
   1. make;
   2. model;
   3. engineSize;
   4. doors;
   5. satNav set as false by default;
   6. wheels.
4. Inside the constructor:
   1. Make a super call with:
      1. make;
      2. model;
      3. wheels set as 4 by default;
      4. engineSize.
   2. Set the values of doors and satNav.
5. Add 'getters' for doors and satNav.
6. Add a 'setter' for satNav, taking satNav as an argument and using this to set the value.
7. Save the file and open index.js.
8. import the Car class.
9. Create a new instance of the Car class, called myCar.
   1. Use any values you want here - don't include a value for wheels.
10. Log out myCar.
11. In the src folder, create a new file called motorbike.js and import MotorVehicle.
12. Add an export default class called Motorbike that extends MotorVehicle.
13. Add a constructor that takes the arguments:
14. make;
15. model;
16. engineSize;
    1. driveType;
    2. wheels.
17. Inside the constructor:
18. Make a super call with:
19. make;
20. model;
21. wheels set as 2 by default;
22. engineSize.
    1. Set the value of driveType.
23. Add a 'getter' for driveType.
24. Override the MotorVehicle implementation of accelerate by adding an accelerate method to the Motorbike class:

accelerate(time) {

this.\_speed = this.\_speed + ((0.5 \* this.\_engineSize /   
 this.\_wheels) \* time);

}

1. Save the file and open index.js.
2. import the Motorbike class and make a new instance using any values you like - we used Kawasaki Ninja with a 650 sized engine and a chain drive-type.
3. Compare the implementations of the accelerate method for the Car instance and the Bike instance.

We accelerated both for 10 and see who won the race in a console.log!

This is the end of Quick Lab 15b

Quick Lab 16a - Asynchronous JavaScript - JSON

Objectives

* To be able to create a properly formed JSON file.
* To be able to install and run json-server

Activity

1. In VSCode, open the file index.js from the QuickLabs/16\_AsynchronousJavaScript/starter/src folder.
2. Open VSCode’s in-built console by selecting View → Integrated Terminal (or use the shortcut key or icon on the bottom bar).
3. Ensure that the terminal is pointing to the QuickLabs/16\_AsynchronousJavaScript/starter folder and then install the dependencies using the command:

npm i

1. Once the installation is finished, run the application by using the command:

npm start

1. In the src folder, create a new file called reactrangers.json.
2. Start the file with an opening and closing set of curly-braces:

{

}

1. Add a key of results with the value of an empty array:

{

"results": [

]

}

1. Inside this array add at least 2 objects (separated by a comma) that have 5 key/value pairs (id should increment with each). The object should look like the example below:

{

"id": 1,

"home": "React Rangers",

"away": "Angular Athletic",

"homeScore": 2,

"awayScore": 0

}

1. Save the file.
2. In VSCode's terminal window, initialise another terminal by clicking the 🞧 button.
3. Install json-server globally using:

npm i json-server -g

1. Ensure that the terminal is pointing to the src folder for this exercise, then spin up json-server using the command:

npx json-server reactrangers.json

1. Open your browser at:

<http://localhost:3000/results>

1. You should see the data from the file presented on the screen.

This is the end of Quick Lab 16a

Quick Lab 16b - Asynchronous JavaScript - Promises

Objectives

* To understand how Promises work

Activity

1. In VSCode, create a new file promises.js in the QuickLabs/16\_AsynchronousJavaScript/starter/src folder.
2. Create a function called runPromise() that is exported by default.
3. Inside the function, declare a variable called aPromise that is a new Promise whose constructor has an arrow function that:
   1. Takes resolve and reject as arguments

let aPromise = new Promise((resolve, reject) => {  
  
}

* 1. Has a function body that:
     1. Declares a variable called delayedFunc that is set as follows:

…

let delayedFunc = setTimeout(() => {

//whether it resolves or rejects is unknown

let randomNumber = Math.random();

(randomNumber < 0.5) ? resolve(randomNumber) :   
 reject(randomNumber);

}, Math.random() \* 5000); //function will return sometime: 0-5s

…

The fact that we have used setTimeout here and the final argument Math.random() \* 5000 (which generates a random number between 0 and 1 and multiplies it by 5000) means that the arrow function will execute somewhere between 0ms and 5000ms. The arrow function itself generates a random number between 0 and 1 -and the Promise is resolved if the number is less than 0.5 and rejects otherwise.

1. Call aPromise with a .then chain and set data to be the resolved value and log this out.
2. Add a catch block and set error to be the rejected value and log this out.
3. Save the file and open index.js.
4. import runPromise and then call it.
5. Save the file and refer to the console of your browser - don't forget that after each refresh you will need to wait up to 5 seconds for the result to show. Refresh the browser several times to satisfy that the Promise resolves and rejects randomly.

This is the end of Quick Lab 16b

Quick Lab 16c - Asynchronous JavaScript - Fetch

Objectives

* To be able to use the Fetch API to be able to send and receive data.

Activity

1. Open the files getResultsUtils.js and formUtils.js - you will see that some functions have been provided here to allow you to concentrate on the Fetch part of the application.
2. In the src folder, create a new file called constants.js and declare a const with the name of resultsURL with the address of your results on json-server as a string value. Ensure that this is exported.
3. If you look at the page in the browser, you should notice that there is a placeholder for the results but there is nothing displayed.
4. Populate the getResults arrow function in the getResultsUtils.js file with code that:
   1. Returns a fetch call to resultsURL that in the first.then block:
      1. Takes an argument of results for an arrow function;
      2. Checks to see if results.ok is true, returning:

results=results.json();

1. Otherwise throws a new Error object with the message Data not fetched;
2. In the second .then block:
3. Takes an argument of results for an arrow function;
4. Sets a variable called reactRangersResults to results;
5. Calls the function populateResults with an argument of reactRangersResults.
6. In the .catch block:
7. Takes an argument of error for an arrow function;
8. Logs out the error message.
9. Save your work and open index.js.
10. Make a call to getResults, making sure you import it.
11. Save this file and check your browser window - the results should appear, perhaps after a short delay.

To make the form submit the data, another Fetch call is needed.

1. Open formUtils.js and locate submitResult.
2. The function body should:
3. Declare a variable called resultToSubmit and set it to the stringified version of results:

let resultToSubmit = JSON.stringify(result);

1. Return a fetch call to resultsURL (remember to import it!) with a configuration object that sets:
2. method to POST;
3. body to resultToSubmit;
4. mode to cors;
5. headers to:

"headers": {  
 "Content-Type": "application/json"  
}

1. In the first .then block:
2. Take response as the argument to the arrow function;
3. If response.ok is true, alert that Data submitted successfully;
4. Otherwise throw a new Error with the message Something went wrong, please try again.
5. In the .catch block:
6. Take error as the argument to the arrow function;
7. Alerts the error message.
8. Save the file and open index.js.
9. Make a call to the function registerEventListeners which should be imported from formUtils.js.
10. Save the file and enter data on the form. You should notice that unless React Rangers (in any case) is one of the teams, the form does not submit. If you successfully enter data and submit, the page refreshes and the new result is displayed along with the others.
11. Check the reactrangers.json file you created before - the new result(s) is stored in this file and an id has automatically been added.

This is the end of Quick Lab 16c

Quick Lab 16d - Asynchronous JavaScript - async/await

Objectives

* To be able to use async/await to be able to send and receive data.

Activity

A little bit of preparation first:

* Comment out the functions directly under the Fetch comment in getResultsUtils.js and formUtils.js.

1. In getResultsUtils, immediately under the function you have just commented out, export a new async function called getResults that:
   1. Sets a variable called results to await a fetch call to resultsURL;
   2. Sets a variable called reactRangersResults to await results.json();
   3. Returns reactRangersResults.
2. In index.js, add a .then clause to the call to getResults that:
   1. Takes results as an arrow function argument;
   2. Passes results to a call to populateResults in the function body (don't forget to import populateResults!)
3. Add a .catch clause to the call to getResults that:
   1. Takes error as an arrow function argument;
   2. Logs out the error message.
4. Save all files and check that the results still display as before.
5. In formUtils.js, immediately under the function that has been commented out, declare an async function called submitResult that takes an argument of result.
6. The function body should:
   1. Set resultToSubmit to the stringified version of result;
   2. Set response to await the fetch call to resultsURL with the same config object as before;
   3. Checks to see if the response was ok and alerts if it was, alerting the contrary if not.
   4. Returns response.
7. Save this file and then check that the form still submits correctly.

This is the end of Quick Lab 16d