INF221 Lecture 2 - Lab Practice Exercises

1) Introduction

   Read more: conditioanl operators, loops, conditional statements and data value types.

2) Variables

   A variable is a name that can reference a specific value. Variables are declared using var,let or const followed by the variable's name. Here are some examples:

```
var example;

let example;

const example;
```

   The above variable is declared, but it isn't defined (it does not yet reference a specific value). Here's an example of defining a variable, making it reference a specific value:

```
var number = 12;

let myString = "some string";

const example = "some thing";
```

   NOTE A variable is declared using var, let, const and uses the equals sign to define the value that it references. This is known as variable initialization.

The challenge:

A. Create a file named index.html with the following content.

```
<!DOCTYPE html>

<html>

   <head>

       <title>My website</title>

   </head>

   <body>

       <h1>My website</h1>

       <p>This website contains nothing</p>

   </body>

</html>
```

B. We execute JavaScript in the browser by linking to it from an external file, with a script tag, like so:

```
<!DOCTYPE html>

<html>

   <head>

       <title>My website</title>

       <script src="script.js" lang="javascript"
type="text/javascript">

           // Javascript code...

       </script>

   </head>

   <body>

       <h1>My website</h1>

       <p>This website contains nothing</p>

   </body>

</html>
```

C. Else you can do the same in the `index.html`. In the `index.html` file between the `body` and `html` **closing** tags, put the following script tags `<script type="text/javascript">/* javascript code... */</script>`. Between the js tags declare the above variables. Make the variables equal to the value of choice. Then use `console.log()` to print each value to the console when the page loads.

3) Functions

Javascript supports both named and anonymous functions.

- Named functions:
  1. Normal way of declaring functions.

```
function myFunction() {
   // some code here
   console.log("My named function");
}
```

  2. Second way.

```
const myFunction = function() {
   // some code here
   console.log("My second named function");
}
```

  3. As arrow functions, es6.

```
// es6 way
const myFunction = () => {
   // some code here
   console.log("My second named function");
}
```

❖ Anonymous functions:

```
// anonymous function
(function() {
   // some code here
   console.log(confirm("Anonymous function?"));
})(); // note how the function is called
```

Note that functions can accept parameter values.

1. Passing values to functions

```
// find the maximum of two numbers
const max = (val1, val2) => {
    if(val1 !== 0 && val2 !== 0) {
        if(val1 > val2) {
            return val1;
        } else if(val1 === val2) {
            return 'equal';
        } else {
            return val2;
        }
    }
};
```

2. Passing functions as arguments.

```
// take two numbers and perform addition on them
```

```
const addition = (a, b) => {
   return a + b;
}
// take two numbers and perform addition on them
const subtraction = (a, b) => {
   return a - b;
}
// passing functions as parameters
const perform = (callback) => {
   return callback;
}
```

Then call `perform` function with necessary parameter values. i.e.

```
console.log(perform(add(12, 78));
console.log(perform(subtraction(1, 8));
```

3. Recursion (read more)

   Is when a function calls itself many times. Recursion is a programming pattern that is useful in situations when a task can be naturally split into several tasks of the same kind, but simpler. Or when a task can be simplified into an easy action plus a simpler variant of the same task. Or, as we'll see soon, to deal with certain data structures. Ie. `fibonacci` sequence etc.

   ```
   const fibonacci = (element) => element <= 1 ? 1 :
   fibonacci(element-1) + fibonacci(element-2);
   ```

   I. How different is recursion from iteration? Write any two functions for each to show the difference (Hint: `loops`).

The challenge:

Inside the `script.js` file, modify the `perform` function to accept two parameter values, the first one a function and the second one a list of integers (array). The function should look as below:

```
/**
* Here we have created a function that takes
* some list of something and a function as argument.
* It will go through the list and apply the provided function on each element,
* storing the returned value from the function in results.
*/
const perform = (callback, list) => {
    let results = [];
    for (let index = 0; index < list.length; index++) {
        results.push(callback(list[index]))
    }
    return results;
}
```

I. Write a function to find the factorial of a number when passed to it. Then call the `perform` function on: the `factorial` and any number of elements in an `array` passed as arguments.
II. Use the function to find the `fibonacci` sequence of each element in an `array`.

4) Data structures: Arrays and Objects

Numbers, Booleans, and strings are the basic atomic values from which data structures are built. In javascript `Objects` and `Arrays` are some of the best ways of representing these data structures. Arrays stores sequences of values, written as a list between square brackets, separated by commas. JavaScript arrays are zero-indexed: the first element of an array is at index `0`, and the last

element is at the index equal to the value of the array's `length` property minus 1. Using an invalid index number returns `undefined`.

```
1.    // array definition and accessing elements

let names = ['John', 'James', 'Joyce', 'Jane'];

// length

console.log(names.length); // log  4

// access element at index 0

console.log(names[0]); // log John

// access element at index 2

console.log(names[2]); // log Joyce

// access element at index 5

console.log(names[5]); // undefined, why?


2.    // accessing elements using some of the default methods

let vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot'];

console.log(vegetables);

// ["Cabbage", "Turnip", "Radish", "Carrot"]

let pos = 1, n = 2;

let removedItems = vegetables.splice(pos, n);

// this is how to remove items, n defines the number of items
to be removed, note the use of splice.

// from that position(pos) onward to the end of array.

console.log(vegetables);

// ["Cabbage", "Carrot"] (the original array is changed)

console.log(removedItems);

// ["Turnip", "Radish"]
```

`Objects` allow us to group values, including other objects, to build more complex structures. These are a collection of key-value pairs.

```
// object definition
let course = {
   id: 0,
   name: 'INF221',
   description: 'Web Design and Development',
}
// access a variable
console.log(course.name);
// change value of variable
console.log(course.name = 'Web Development');
```

We can at any time add new variables to the object.

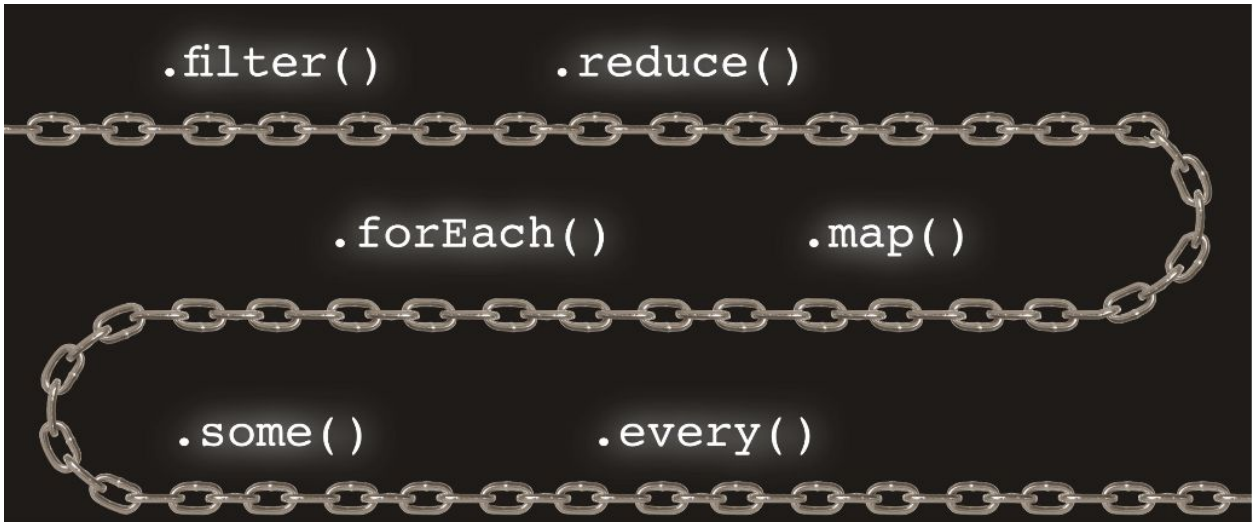
```
// add a new variable # of students and initialize it with 62
course.numberOfStudents = 62;
console.log(course);
```

You can define even more complex objects or arrays. The following is an array of objects.

```
// list of course objects
let courses = [
   {
       id: 0,
       name: 'INF221',
       description: 'Web Design and Development',
       numberOfStudents: 62
```

```
    },
    {
        id: 1,
        name: 'COM121',
        description: 'Intro. to Computer Programming',
        numberOfStudents: 200
    },
    {
        id: 2,
        name: 'COM222',
        description: 'Advanced Programming',
        numberOfStudents: 80
    },
];
```

In javascript, you will mostly work with `Arrays` or `Objects` and a combination of both to create very complex objects or arrays. ***Objects can also contain functions***. The following are some of the useful methods that can be called on arrays and objects.



The challenge:

Read more on the following ***before coming to class and lab tomorrow***: `forEach`, `map`, `reduce`, `filter higher-order functions` and `rest parameters, JSON, closures` and `array` or `object destructuring`.

I. Using `map` function print each of the objects contained in the array object.
II. Using a function, determine which course object has more students than the rest.
III. Calculate the total number of students for all courses by summing them together.
IV. Add a new function to `courses` and call it `summary`, the function should return a combination or concatenation of any variables in each object as brief summary.