WEEK 5:

JS:

Exercise 5.1:

Using Async/Await and Generators, create separate functions and achieve the same functionality. **(3 hours)**

Execute 3 callback functions asynchronously, for example doTask1(), doTask2() and doTask3().

Guidelines:

1. 2 functions should be created. One for Async/Await and the other one for Generators.
2. 3rd party libraries should not be used.
3. Custom Function should carry a meaningful name.
4. The program should execute without errors.
5. The program should achieve the desired result.
6. The program should take care of all 3 states of a Promise.
7. Should be committed to Git with meaningful commit messages.

Outcome:

1. Under the hood understanding of how a Generator works.
2. Under the hood understanding of how Async/await works.

Exercise 5.2:

Write a function called vowelCount which accepts a string and returns a map where the keys are numbers and the values are the count of the vowels in the string.

Guidelines:

1. JS function should have Map API implemented.
2. Map's set functionality should have been used.
3. Bonus - if space and time complexity is taken care.
4. Reference:

```
function isVowel(char){
  return "aeiou".includes(char);
}

function vowelCount(str){
  const vowelMap = new Map();
  for(let char of str){
    let lowerCaseChar = char.toLowerCase()
    if(isVowel(lowerCaseChar)){
      if(vowelMap.has(lowerCaseChar)){
        vowelMap.set(lowerCaseChar, vowelMap.get(lowerCaseChar) + 1);
      } else {
        vowelMap.set(lowerCaseChar, 1);
      }
    }
  }
  return vowelMap;
}
```

Outcome:

1. Understanding of Map API and its functionalities.

Exercise 5.3:

Write a function called hasDuplicate which accepts an array and returns true or false if that array contains a duplicate

Guidelines:

1. Reference

```
hasDuplicate([1,5,-1,4]) // false

const hasDuplicate = arr => new Set(arr).size !== arr.length
```

2. JS function should have Set API implemented.
3. Bonus - if space and time complexity is taken care.

Outcome:

1. Understanding of Set API and its functionalities.

Exercise 5.4:

Create a simple Javascript function code for addition, subtraction, and multiplication of 2 numbers and write the corresponding Jest based tests for it.

```
const mathOperations = {
  sum: function(a,b) {
     return a + b;
  },

  diff: function(a,b) {
     return a - b;
  },
  product: function(a,b) {
     return a * b
  }
}

module.exports = mathOperations
```

Guidelines:

1. Jest should’ve been installed.
2. Package.json file should have the config for running test cases.
3. Test case file should’ve been created. For example: calculator.test.js
4. BDD style tests for each function should’ve been created for the same.
5. All test cases should pass.
6. Reference: https://www.softwaretestinghelp.com/jest-testing-tutorial/

Outcome:

1. Understanding the importance of writing test cases.
2. How BDD works.
3. What are the packages required for writing test cases?
4. How to configure test cases in package.json file.