WEEK 4:

JS:

Exercise 4.1:
Implement a function named *getNumber* which generates a random number. If randomNumber is divisible by 5 it will reject the promise else it will resolve the promise. Let's also keep the promise resolution/rejection time as a variable.

1. JS promises should not be used.
2. A custom promise function should be created.
3. This function should be able to handle all 3 states Resolve, Reject and Fulfilled.
4. Should be able to accept callbacks as props.

Guidelines:

1. JS promises should not be used.
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 promise actually works.
2. Using "bind" to bind the callback functions sent as props.
3. Understanding what a polyfill is.
4. Error handling using functions.

Exercise 4.2:

Create an object called **Teacher** derived from the **Person** class, and implement a method called **teach** which receives a string called **subject**, and prints out:

`[teacher's name] is now teaching [subject]`

Guidelines:

1. The expected output should be achieved using the keyword .prototype.
2. Reference:

```
Var Person = function() {};
Person.prototype.initialize = function(name, age)
{
   this.name = name;
   this.age = age;
}

// TODO: create the class Teacher and a method teach

var him = new Teacher();

him.initialize("Adam", 45);
him.teach("Inheritance");
```

Outcome:

1. The candidates will understand how inheritance works in JS.
2. The candidates will understand what a prototype keyword is in JS.

Exercise 4.3:
Implement Fibonacci Series with Iterators

Sample output:

```
The Fibonacci Series is:
0
1
1
2
3
5
8
```

Guidelines:

1. The expected output should be achieved using Iterators only.
2. Reference: https://medium.com/@akshayshekokar/fibonacci-series-with-iterators-90a8b3dd0d92
3. For Loops, Maps should not have been used.
4. 3rd party libraries should not have been used.

Outcome:

1. Understanding how Iterators work.
2. Understanding ‘under the hood’ implementation of Iterators.