JavaScript ES6 02 – Async Flow

# Exercise 01: Generators

Your task is to use ES6 Generators to implement Object Spread

|  |
| --- |
| var numbers = {      // ..  };  // should print 0..100 by 1s  for (let num of numbers) {      console.log(num);  }  // should print 6..30 by 4s  for (let num of /\*..\*/) {      console.log(num);  } |

# Exercise 02: Async Flow

You are given a file called **numfiles.zip**

Unzip it and you will have a folder contains many small files like below:

|  |
| --- |
|  |

You need to do the following:

* Start from file 1024.
* Collect the binary digit enclosed in [].
* Proceed to get the next file and the next binary digit.
* Continue till the last file.

To read a file use the following API from NodeJS, which will read the file **1024** asynchronously

|  |
| --- |
| const fs = require('fs');  fs.readFile('1024', 'utf-8', function(err, data) {  if (err) {  throw err;  }  console.log(data);  }); |

You have to do the requirements using **Callback** first then **Promise** and lastly use **async/await**

You need to setup your project folder in order to do so.

Expected output:

|  |
| --- |
| 0101010001101000011001010010000001110011011011110110110001110101011101000110100101101111011011100010000001101001011100110010000001110100011010000110010100100000011011100110000101101101011001010010000001101111011001100010000001110100011010000110010100100000011011000110111101100011011000010111010001101001011011110110111000100000011000010111010000100000001100100011010100101110001101000011000001001110001000000011001100110010001011100011001100110101010001010010000001010100011010000110010100100000011100110110111101101100011101010111010001101001011011110110111000100000011010010111001100100000011101000110100001100101001000000110111001100001011011010110010100100000011011110110011000100000011101000110100001100101001000000110110001101111011000110110000101110100011010010110111101101110001000000110000101110100001000000011001000110101001011100011010000110000010011100010000000110011001100100010111000110011001101010100010100100000010101000110100001100101001000000111001101101111011011000111010101110100011010010110111101101110001000000110100101110011001000000111010001101000011001010010000001101110011000010110110101100101001000000110111101100110001000000111010001101000011001010010000001101100011011110110001101100001011101000110100101101111011011100010000001100001011101000010000000110010001101010010111000110100001100000100111000100000001100110011001000101110001100110011010101000101 |

# Exercise 03: Using Map

Given the following code which calculate the fibonaci number of n:

|  |
| --- |
| function fibonaci(n) {  if (n <= 1) {  return 1;  }  return fibonaci(n - 1) + fibonaci(n - 2);  } |

The problem with this code is we make to many duplicate calculate:

|  |
| --- |
|  |

In the figure above, we can see that f(2) is computed 3 times, f(3) is computed 2 times

Your task is to use Map to memorize the Fibonacci of lower value so we don’t have to recompute it again.

|  |
| --- |
|  |

For example: if we have computed Fibonacci for n – 2 and n – 3 (left branch) then we don’t need to compute anymore for the right branch.

# Exercise 04: Using Set

You are given a list of object represent a Person.

Each person have property: id, name, age.

Your task is to remove duplicates from that lists. We define a person is duplicate of another person if they have the same Id and we will keep the previous Person

Example:

|  |
| --- |
| [  {  id: 1,  name: 'Dung',  age: 20  },  {  id: 2,  name: 'Diu',  age: 20  },  {  id: 3,  name: 'Ky',  age: 20  },  {  id: 1,  name: 'Hai',  age: 22  }  ] |

In the above example, id = 1 is duplicated, and we will keep the person with id = 1 and name = ‘Dung’

Output

|  |
| --- |
| [  {  id: 1,  name: 'Dung',  age: 20  },  {  id: 2,  name: 'Diu',  age: 20  },  {  id: 3,  name: 'Ky',  age: 20  }  ] |