WEB700 Assignment 2

# Submission Deadline:

Sunday, Sep 28th @ 11:59 PM

# Assessment Weight:

9% of your final course Grade

# Objective:

The second assignment will focus on more advanced JavaScript skills including: promises, using / defining objects using the "class" keyword and passing functions as parameters (callback functions).

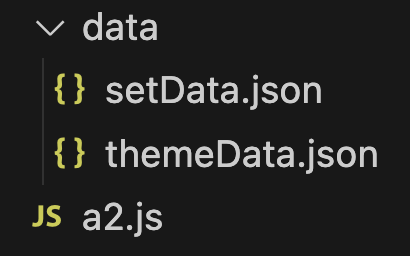
The main objective is to create and test a class that is responsible for pulling data from multiple files as well as to provide multiple promise-based methods that "resolve" with the data.

# Specification:

This assignment will consist of multiple files, including: a main "a2.js" file, and a "data" folder containing two files ("setData.json" & "themeData.json").

### **Step 1:** Create the Files & Directories

Whenever we start a new project or example for this course, we will always create a new folder to contain the code. For this assignment, start by creating a new folder somewhere on your local machine to house all of your code. Once you have done this, open it up in Visual Studio Code and create the following file / folder structure:



### **Step 2:** Obtaining the Data (setData.json & themeData.json)

The data for this assignment will exist in two separate files: setData.json and themeData.json. Follow the steps below to obtain the data. **Recall:** [JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON) stands for **JavaScript Object Notation** and is a way of representing JavaScript Objects in a plain-text format.

* Open your web browser and navigate to: [this link (setData.json)](https://pat-crawford-sdds.netlify.app/shared/winter-2025/web700/A2/setData.json) and copy the contents of the JSON file to your own setData.json file (within the "data" folder).
* Next, navigate to: [this link (themeData.json)](https://pat-crawford-sdds.netlify.app/shared/winter-2025/web700/A2/themeData.json) and copy the entire contents of the JSON file to your own themeData.json file (within the "data" folder)

### **Step 3:** Writing a2.js and the "LegoData" Class

Now that we have the data in place, we can begin to write a2.js. As a first step (at the top of the file), we will define and create a "legoData" class, which is responsible for managing with the data, as well as providing convenience methods to access / update it. This must be created according to the following specification:

Properties:

* **sets**  
    
  An array of objects

Methods:

* **constructor()**The constructor simply assigns the "sets" property to be an empty array. It does not accept any parameters
* **initialize()**

This purpose of this method is to load data from both setData and themeData files and populate our "sets" property with the results. To begin, add the following two lines of code:  
  
const setData = require("./data/setData");

const themeData = require("./data/themeData");

This will read the contents of both the setData.json and themeData.json files and create corresponding arrays from the data. We will learn more about the "require" statement, as well as JSON later in this course.

Next, we must fill the "sets" array by adding (ie: "pushing") copies of all the **setData** objects, ie:  
  
{

"set\_num": "001-1",

"name": "Gears",

"year": "1965",

"theme\_id": "1",

"num\_parts": "43",

"img\_url": "https://cdn.rebrickable.com/media/sets/001-1.jpg"

}

However, each of these objects must **also** include a new **"theme**" property. The value of the "theme" property should be the corresponding theme **name** from **themeData**, whose "id" value matches the "theme\_id" for the "setData" object.

For example, in the above "setData" object, we have a "theme\_id" value of "1". Therefore, we must add an additional "theme" property with the value of "Technic", since the "id" value for "Technic" is also "1" within **themeData**.

This will result in the following object being added (pushed) to the "sets" array:

{

"set\_num": "001-1",

"name": "Gears",

"year": "1965",

"theme\_id": "1",

"num\_parts": "43",

"img\_url": "https://cdn.rebrickable.com/media/sets/001-1.jpg"

**"theme": "Technic"**

}

As an additional example, consider the following object from a "setData" array:

{

"set\_num": "0011-2",

"name": "Town Mini-Figures",

"year": "1979",

"theme\_id": "67",

"num\_parts": "12",

"img\_url": "https://cdn.rebrickable.com/media/sets/0011-2.jpg"

}

This has a "theme\_id" of "67". From **themeData**, the name of the theme with id "67" is "Classic Town". Therefore, the new object to be added to the "sets" array should be:

{

"set\_num": "0011-2",

"name": "Town Mini-Figures",

"year": "1979",

"theme\_id": "67",

"num\_parts": "12",

"img\_url": "https://cdn.rebrickable.com/media/sets/0011-2.jpg"

**"theme": "Classic Town"**

}

**HINT:** Consider using the **.find()** and **.forEach()** Array methods for your solution

* **getAllSets()**

This method simply returns the complete "sets" array

* **getSetByNum(setNum)**

This method will return a specific "set" object from the "sets" array, whose "set\_num" value matches the value of the "setNum" parameter, ie: if getSetByNum("001-1") was invoked, the following set object would be returned:

{

"set\_num": "001-1",

"name": "Gears",

"year": "1965",

"theme\_id": "1",

"num\_parts": "43",

"img\_url": "https://cdn.rebrickable.com/media/sets/001-1.jpg"

"theme": "Technic"

}

**HINT:** Consider using the **.find()** Array method for your solution

* **getSetsByTheme(theme)**

The purpose of this method is to return an array of objects from the "sets" array whose "theme" value matches the "theme" parameter. However, it is important to note that the "theme" parameter may contain only part of the "theme" string, and case is ignored. For example:

getSetsByTheme(**"tech"**);

would return all the sets from your "sets" array whose "theme" property contains the string "tech" (ignoring case). For example, if your "sets" array contained the following objects, they would be returned:

[ {

"set\_num": "01-2",

"name": "Bulldozer Chain Links",

"year": "1982",

"theme\_id": "453",

"num\_parts": "50",

"img\_url": "https://cdn.rebrickable.com/media/sets/01-2.jpg",

**"theme": "Technic"**

},

{

"set\_num": "02-1",

"name": "Extra Large Tires & Hubs",

"year": "1982",

"theme\_id": "453",

"num\_parts": "4",

"img\_url": "https://cdn.rebrickable.com/media/sets/02-1.jpg",

**"theme": "Technic"**

} ]

**HINT:** Consider using the **.filter()** Array method as well as the **.toUpperCase() / .toLowerCase()** and **.includes()** String methods for your solution

### **Step 4:** Testing and Refactoring the "LegoData' class

Once you have completed the definition for the LegoData class, it's time to test it out and ensure that we're getting the expected data. To do this, add the following code to the bottom of your LegoData class definition in your a2.js file:

let data = new LegoData();

data.initialize();

console.log(`Number of Sets: ${data.getAllSets().length}`);

console.log(data.getSetByNum("0012-1"));

console.log(`Number of 'tech' sets: ${data.getSetsByTheme('tech').length}`);

Next, run your code by executing the command "node a2.js". In the console, you should see the following output if your code is working correctly:  
  
Number of Sets: 30

{

set\_num: '0012-1',

name: 'Space Mini-Figures',

year: '1979',

theme\_id: '143',

num\_parts: '12',

img\_url: 'https://cdn.rebrickable.com/media/sets/0012-1.jpg',

theme: 'Supplemental'

}

Number of 'tech' sets: 6

If the correct data is returned and you're satisfied that everything is working correctly, you can delete your testing code, leaving only the LegoData class definition.

Now, we can begin to refactor our functions to use "[Promises](https://web700.ca/Handling-Asynchronous-Code/promises-async-await)".

* Each of the 4 methods created (initialize(), getAllSets(), getSetsByNum(setNum), getSetsByTheme(theme)) must return a new Promise object that "resolves" either with data (if the function returns data) or "rejects" with an error, if the function encounters an error, for example:
  + **Initialize()** should resolve with no data, once the operation is complete (ie: the "sets" array is filled with objects)
  + **getAllSets()** should resolve with the completed "sets" array
  + **getSetsByNum(setNum)** should resolve with the found "set" object, and reject with an appropriate message (ie: unable to find requested set) if the set was not found
  + **getSetsByTheme(theme)** should resolve with the found "set" objects, and reject with an appropriate message (ie: unable to find requested sets)

Once is this is complete, write new test code using the same test data from above. However, this time your methods will return Promise objects, so adjust your code accordingly (**NOTE:** To ensure that your test code runs correctly, you must make sure that the **initialize()** method runs **before** any other methods. Once it has been resolved ***then*** it will be safe to test the other methods).

**Important:** You must leave your final testing code in your a2.js file to submit your assignment.

## Assignment Submission:

1. Add the following declaration at the top of your a2.js file:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* WEB700 – Assignment 02

\*

\* I declare that this assignment is my own work in accordance with Seneca's

\* Academic Integrity Policy:

\*

\* https://www.senecapolytechnic.ca/about/policies/academic-integrity-policy.html

\*

\* Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Compress your working folder (containing your a2.js file - including your test code, as well as your "data" folder) and submit it on My.Seneca

## Important Note:

* **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
* Submitted assignments **must**run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.