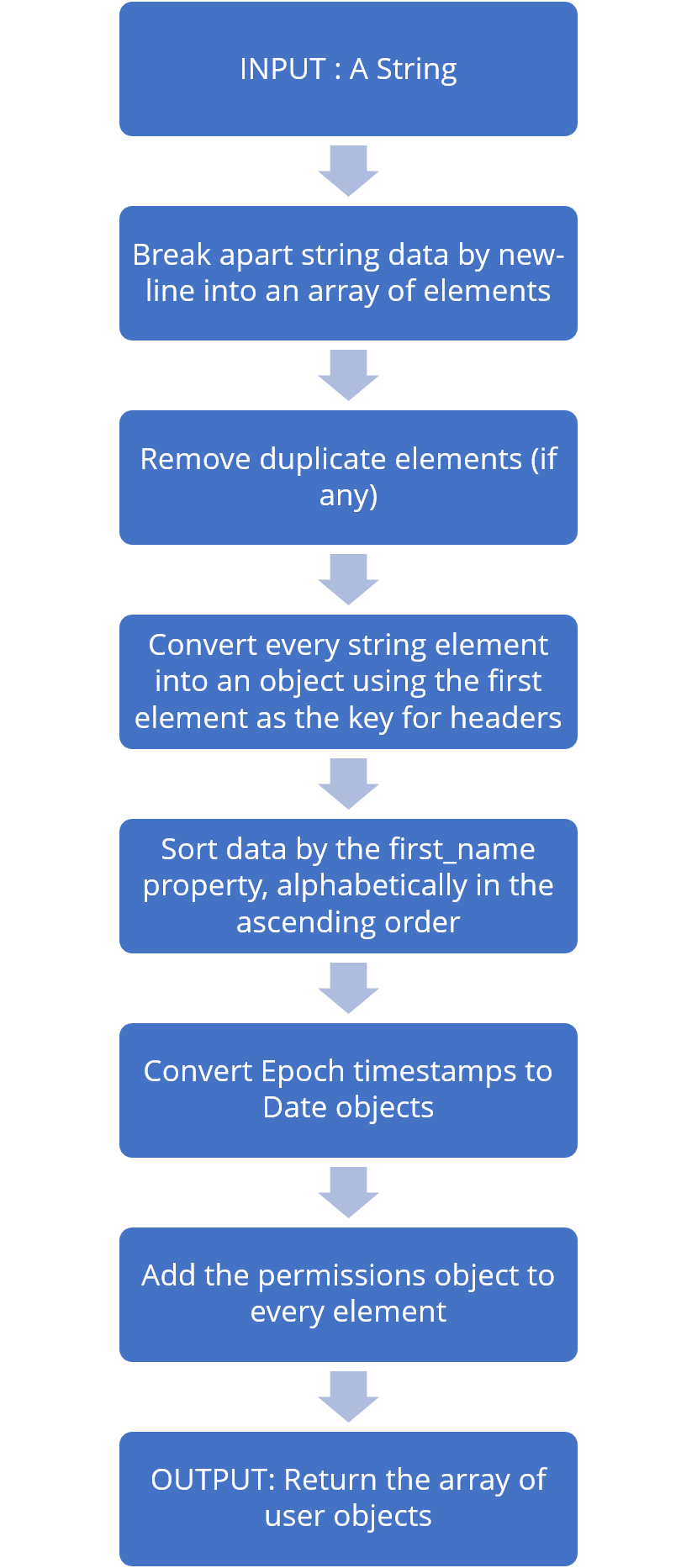
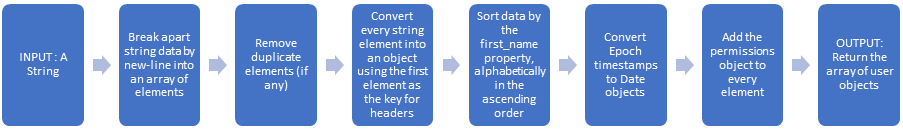
Welcome to your second JavaScript challenge. In this challenge, you will need to build a data processing pipeline that ingests comma separated string data and produces an array of formatted objects. In doing so you’ll need to build a collection of functions that do the heavy lifting at every stage of the processing pipeline.

As the input, you get a string with comma separated data representing a bunch of users with IDs, first names, last names, email IDs, designation and an EPOCH timestamp representing the moment they signed up.

 The data processing pipeline should look like the following:

  
In addition to this, you’ll also need to create a function that identifies bad/invalid e-mail IDs and returns the user objects.

**Key Skills**

The intent of this project is to give you ample practice and to evaluate your skill as you become a JavaScript developer:

1. Logic
2. Building functions
3. Creating, Accessing, Mutating, Filtering, Remapping & Sorting Arrays
4. Using loops
5. Array prototype methods
6. String prototype methods
7. Working with functions
8. Objects & prototype methods
9. Using Regular expressions
10. Working with Date & Time
11. Immutability

**Task 1: Converting comma separated data to rows of strings**

Your first function named ***csvToRows()*** in the main.js file is provided with a string input such as the one shown below:

*INPUT*

id,first\_name,last\_name,email,designation,registeredOn

1,Kata,Petrollo,Kata@jscorp.co,Budget/Accounting Analyst IV,1586786455

2,Rafaellle,Francescone,Rafaellle@jscorp.co,Quality Engineer,1591762193

3,Hort,Shufflebotham,Hort@jscorp.co,Paralegal,1604920379

4,Raimundo,Beddingham,Raimundo@jscorp.co,Librarian,1590530472

5,Jamaal,Hyde,Jamaal@jscorp.co,Quality Engineer,1606653657

The function should break down the above string by newline and return an array of rows as shown below:

*OUTPUT*

[

   'id,first\_name,last\_name,email,designation,registeredOn',

   '1,Kata,Petrollo,Kata@jscorp.co,Budget/Accounting Analyst IV,1586786455',

   '2,Rafaellle,Francescone,Rafaellle@jscorp.co,Quality Engineer,1591762193',

   '3,Hort,Shufflebotham,Hort@jscorp.co,Paralegal,1604920379',

   '4,Raimundo,Beddingham,Raimundo@jscorp.co,Librarian,1590530472',

   '5,Jamaal,Hyde,Jamaal@jscorp.co,Quality Engineer,1606653657',

]

**Task 2: Removing duplicate rows**

Your second task is to build the ***removeDuplicateRows()*** function in the main.js file. This function ingests the output of the *csvToRows()* function to remove duplicate rows. The output is an array of comma separated strings for every row of data.

*INPUT*

The output of ***csvToRows()*** function, an array of strings.

*OUTPUT*

An array of strings elements with duplicates removed.

**Task 3: Converting string elements to objects**

Your third task is to build the ***strRowsToObjects()*** function which ingests the outcome of the *removeDuplicateRows()*function to return back an array of objects with property names picked up from the first element in the input Array

*INPUT*

The output of **removeDuplicateRows()** function, an array of comma-separated string elements with the first one as the header.

*OUTPUT*

[

  {

    id: '1',

    first\_name: 'Kata',

    last\_name: 'Petrollo',

    email: 'Kata@jscorp.co',

    designation: 'Budget/Accounting Analyst IV',

    registeredOn: '1586786455'

  },

  {

    id: '2',

    first\_name: 'Rafaellle',

    last\_name: 'Francescone',

    email: 'Rafaellle@jscorp.co',

    designation: 'Quality Engineer',

    registeredOn: '1591762193'

  },

  {

    id: '3',

    first\_name: 'Hort',

    last\_name: 'Shufflebotham',

    email: 'Hort@jscorp.co',

    designation: 'Paralegal',

    registeredOn: '1604920379'

  }

]

**Task 4: Sort by the first\_name property**

Your fourth task is to sort the outcome of the ***strRowsToObjects()*** using the first\_name property, alphabetically in the ascending order in a function named ***sortByFirstName()***.

*INPUT*

The output of the **strRowsToObjects()** function, an array of unsorted objects.

*OUTPUT*

An array of user objects, sorted by the first\_name property, alphabetically in the ascending order.

**Task 5: Convert EPOCH timestamps to Date objects**

Your fifth task is to build the ***timeToDate()*** function. If you observe the user objects created thus far, there is a property named *registeredOn* which carries the EPOCH timestamp when the user registered in the system. These need to be converted to date objects.  
  
 TIP: To convert EPOCH timestamps to milliseconds, simply multiply by 1000

*INPUT*

The output of the***sortByFirstName()*** function, an array of user objects, sorted alphabetically by the first\_name property.

The input objects at this stage, bear the following shape:

{

    id: '3',

    first\_name: 'Hort',

    last\_name: 'Shufflebotham',

    email: 'Hort@jscorp.co',

    designation: 'Paralegal',

    registeredOn: '1604920379'

  }

*OUTPUT*

The function should return back an array of user objects where the registeredOn property carries date objects instead of EPOCH timestamps.

For the object sample shown above, here’s what the outcome should look like:

{

    id: '3',

    first\_name: 'Hort',

    last\_name: 'Shufflebotham',

    email: 'Hort@jscorp.co',

    designation: 'Paralegal',

    registeredOn: 2020-11-09T11:12:59.000Z

  },

**Task 6: Add user permissions**

Your sixth task is to build the ***addUserPermissions()*** function which should add a property named **permissions**to every user object, with the following shape:

{

      admin: false,

      profile: true,

      billing: true,

      supervisor: false,

}

*INPUT*

The output of the***timeToDate()***function, an array of user objects

*OUTPUT*

An array of user objects with the **permissions**property added with the contents shown above.

This function should not mutate the original array under any circumstances.

**Task 7: Constructing the data processing pipeline**

Your seventh task involves wiring up functions created in Tasks 1 - 6 and returning back the processed output, in a function named ***processData()***

*INPUT*

A string containing comma separated data as shown in the Task 1’s input.  
  
*PIPELINE ARCHITECTURE*

The architecture of this function is as follows:

*INPUT*→ csvToRows() →  removeDuplicateRows() →  strRowsToObjects() →  sortByFirstName() →  timeToDate() →  addUserPermissions() → OUTPUT

*OUTPUT*

The output of this function should be an array of user objects.

**Task 8: Finding users with invalid e-mail IDs**

Your concluding task is to build the ***findBadEmailIds()*** function. This function ingests data processed using the ***processData()*** function you created in the previous task and it should return back an array of user objects where email IDs are invalid.

For your convenience, a regular expression to validate e-mail IDs is provided to you in the *main.js* file.

*INPUT*

The outcome of processing data using the *processData()* function.

*OUTPUT*

An array of user objects where email IDs appear to be invalid.

**Important Points**

* Do not delete/modify the main.js other than defining the body of the eight functions.
* Spend time on figuring out the core logic behind the tasks before attempting to code.
* The incoming data is provided to you in a file named data.js. This project uses modules, a concept which you’ll study in the upcoming module. Modules help break down a large application into manageable components. In this case, the input data sits in the data.js file and is being imported in for use which produces the output you see on the terminal.