PER SCHOLAS

R-ALAB 308.4.1:

Working with Data Collections

Version 1.0, 10/13/23

Click here to open in a separate window.

Introduction

This **graded** assignment will have you create two different data processing algorithms. The first stage will have you transform raw data into a formatted array of objects according to a specification. The second stage will have you use the output from the first stage to provide a textual report on the data.

Objectives

- Use arrays to store ordered lists of data.
- Use objects to store keyed lists of data.
- Use conditional logic to process data.
- Use loops to handle repetitive tasks.
- Transform data according to specifications.

Submission

Submit your completed lab using the **Start Assignment** button on the assignment page in Canvas.

Your submission should include:

- A GitHub link to your completed project repository.

Instructions

Initialize a new git repository in a local project folder, and create a JavaScript file to contain your code. Complete the activities below. You may reference and repurpose any code you created for ALAB 308.3.1 - Loops and Iteration if you believe it would benefit you.

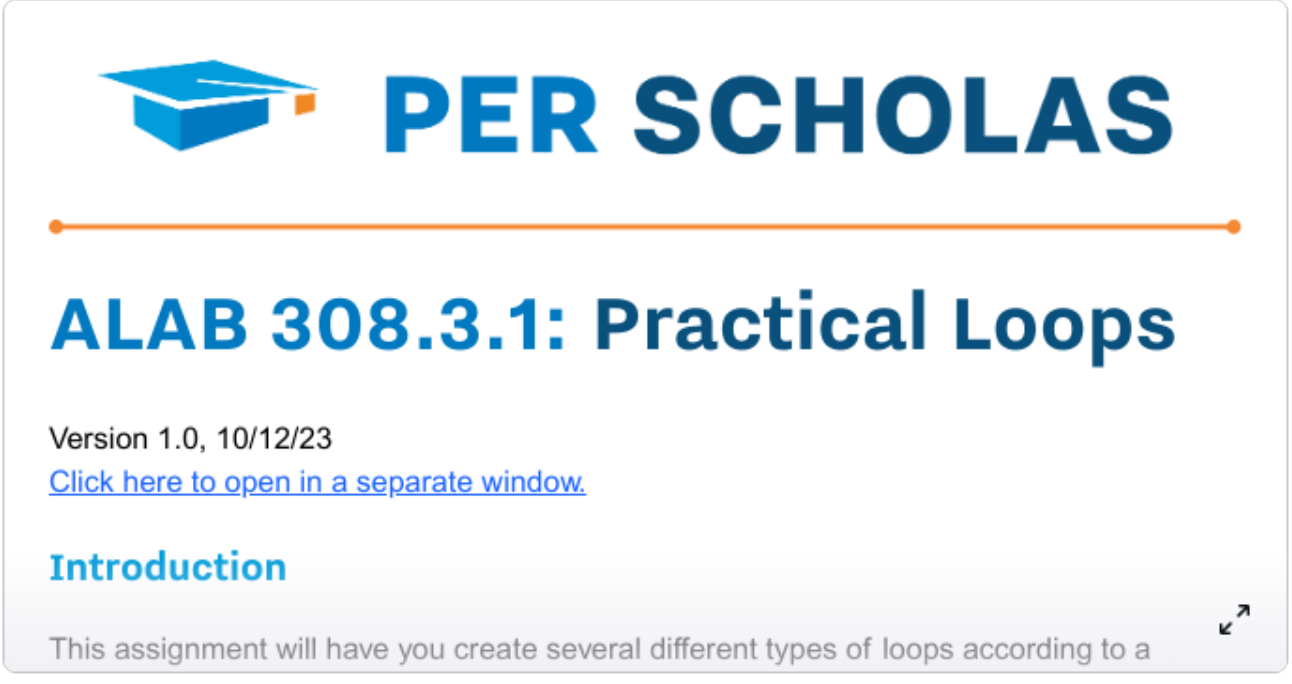
Commit frequently! Every time something works, you should commit it. Remember, you can always go back to a previous commit if something breaks.

Part 1: Refactoring Old Code

When code is outdated or inefficient, it often goes through a process called “refactoring.” Refactoring code is the process of restructuring that code without changing its original behavior.

For the first part of this assignment, revisit your code from ALAB 308.3.1, wherein you create a script that parsed CSVs. Now that you have knowledge of arrays and objects, how would you change your approach to this problem? Take a few minutes to examine and refactor the code before continuing.

For reference, ALAB 308.3.1 is embedded below. The section on CSV parsing is “Part 3.”



Part 2: Expanding Functionality

Now that you are familiar with your code, and perhaps have improved it, it is time to expand upon its functionality.

Begin with the following task:

- Declare a variable that stores the number of columns in each row of data within the CSV.
- Instead of hard-coding four columns per row, expand your code to accept any number of columns. This should be calculated dynamically based on the first row of data.

For example, if the first row of data (the headings) has eight entries, your program should create eight entries per row. You can safely assume that all rows that follow will contain the same number of entries per row.

After you have implemented the above:

- Store your results in a two-dimensional array.
  - Each row should be its own array, with individual entries for each column.
  - Each row should be stored in a parent array, with the heading row located at index 0.
- Cache this two-dimensional array in a variable for later use.

Using the original CSV example data, here is what the result of this step should look like:

```
ID,Name,Occupation,Age\n42,Bruce,Knight,41\n57,Bob,Fry Cook,19\n63,Blaine,Quiz Master,58\n98,Bill,Doctor’s Assistant,26
```

becomes

```
[["ID", "Name", "Occupation", "Age"],
 ["42", "Bruce", "Knight", "41"],
 ["57", "Bob", "Fry Cook", "19"],
 ["63", "Blaine", "Quiz Master", "58"],
 ["98", "Bill", "Doctor’s Assistant", "26"]]
```

Part 3: Transforming Data

While the data is now much more workable than it was in its string format, there is still a large amount of obscurity in the data itself. When we access an arbitrary index of the results array, it is impossible to know *what* that data is referring to without additional cross-referencing.

In order to make it more obvious what the data is, we will transform our rows into objects.

Implement the following:

- For each row of data in the result array produced by your code above, create an object where the key of each value is the heading for that value’s column.
  - Convert these keys to all lowercase letters for consistency.
- Store these objects in an array, in the order that they were originally listed.
- Since the heading for each column will be stored in the object keys, you do not need to create an object for the heading row itself.

For instance, the results of the example data above being passed through this step are as follows:

```
[["ID", "Name", "Occupation", "Age"],
 ["42", "Bruce", "Knight", "41"],
 ["57", "Bob", "Fry Cook", "19"],
 ["63", "Blaine", "Quiz Master", "58"],
 ["98", "Bill", "Doctor’s Assistant", "26"]]
```

becomes

```
[{ id: "42", name: "Bruce", occupation: "Knight", age: "41" },
 { id: "57", name: "Bob", occupation: "Fry Cook", age: "19" },
 { id: "63", name: "Blaine", occupation: "Quiz Master", age: "58" },
 { id: "98", name: "Bill", occupation: "Doctor’s Assistant", age: "26" }]
```

Important: While this functionality *can* be built into the original CSV parser you built in Part 2, we are intentionally creating two different algorithms to test different skillsets. Please leave these sections separate even if it would be more efficient to combine them.

Part 4: Sorting and Manipulating Data

It is important to know how to work with data in this format, an array of objects, as it is one of the most commonly used data formats in JavaScript.

Using array methods, accomplish the following tasks, in order upon the result of Part 3:

1. Remove the last element from the sorted array.
2. Insert the following object at index 1:
   - `{ id: "48", name: "Barry", occupation: "Runner", age: "25" }`
3. Add the following object to the end of the array:
   - `{ id: "7", name: "Bilbo", occupation: "None", age: "111" }`

So far, the results should look like this:

```
[{ id: "42", name: "Bruce", occupation: "Knight", age: "41" },
 { id: "48", name: "Barry", occupation: "Runner", age: "25" },
 { id: "57", name: "Bob", occupation: "Fry Cook", age: "19" },
 { id: "63", name: "Blaine", occupation: "Quiz Master", age: "58" },
 { id: "7", name: "Bilbo", occupation: "None", age: "111" }]
```

Finally, use the values of each object within the array and the array's length property to calculate the average age of the group. This calculation should be accomplished using a loop.

Part 5: Full Circle

As a final task, transform the final set of data *back* into CSV format.

There are a number of ways to do this; be creative!

Once complete, be sure to submit your work according to the submission instructions at the beginning of this document.