

Roann Cordova

3177313

SCTM-S3001-301: Comp Sci 2-Data & Algorithms

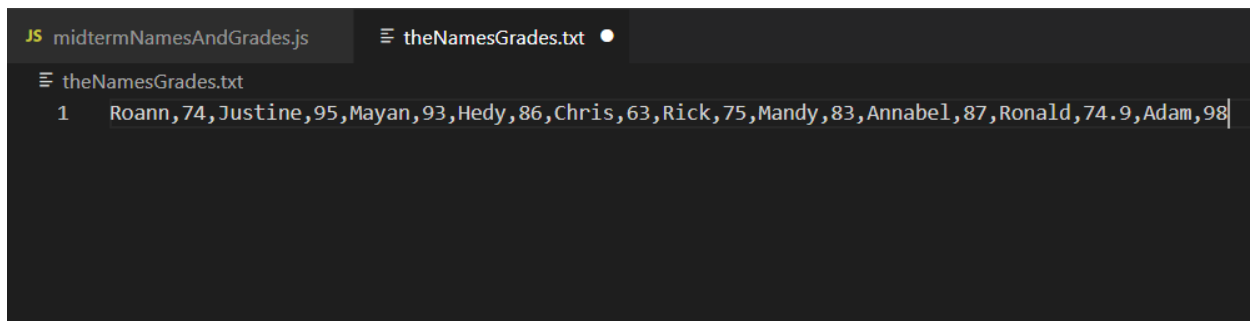
October 24, 2022

Dr. Adam Tindale

For my midterm assignment for this class, I have decided to implement different ways of sorting and searching through the arrays that I have learned so far in the class. I used these methods by choosing a specific theme for the school grading system of students. I will achieve my goal for the assignment by having information that is input by the user in the program with the ability to search, sort, and view the information in a more organized way.

The goal of my assignment is to have my program to:

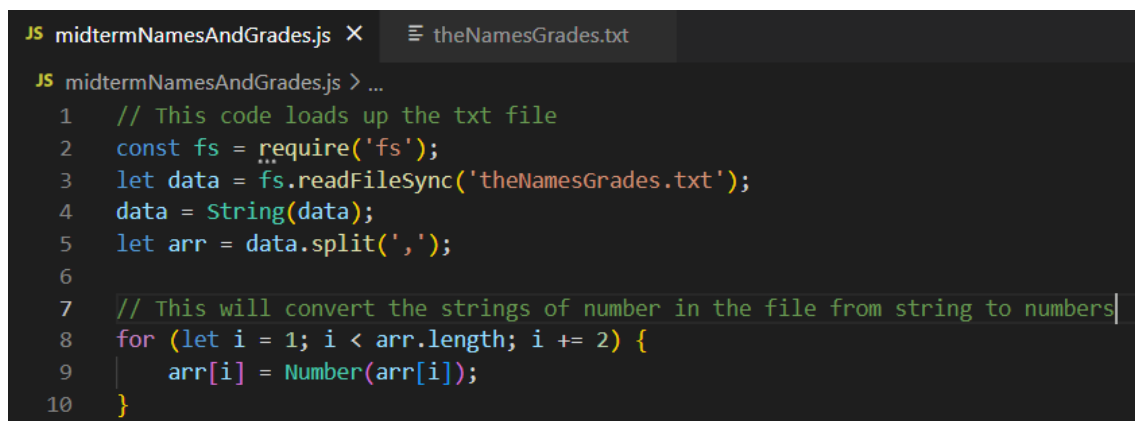
- Sort the name alphabetically
- Sort the name according to their grades
 - From lowest to highest
 - From highest to lowest
- Separate the results of the “passing” and “failing” students
 - And this will be determined by having a specific grade target that is set up by the instructor (user)
- Take the average/ mean of the of the students’ grades.
- Search for a specific name in the list and it will print their grade and tell the user whether the student passed the course or not.

A screenshot of a code editor with two tabs: 'midtermNamesAndGrades.js' and 'theNamesGrades.txt'. The 'theNamesGrades.txt' tab is active, showing a single line of text: '1 Roann,74,Justine,95,Mayan,93,Hedy,86,Chris,63,Rick,75,Mandy,83,Annabel,87,Ronald,74.9,Adam,98'. The text is highlighted in blue.

```
JS midtermNamesAndGrades.js  theNamesGrades.txt
theNamesGrades.txt
1  Roann,74,Justine,95,Mayan,93,Hedy,86,Chris,63,Rick,75,Mandy,83,Annabel,87,Ronald,74.9,Adam,98
```

Figure 1: The text file that contains names and their grade respectively.

I will initialize the program by requiring a text file that is created by the user and will consist of student names and their grade for the class respectively (Fig. 1). The program will read the text file and it will be converted into a bunch of strings and numbers. As I learned from the class, loading the text file and using the *split()* function to separate the names and numbers. I made sure to keep the names as strings and the numbers as number types, and I made this possible by figuring out a pattern in the file by incrementing the conversion of the arrays read by two (Fig 2).

A screenshot of a code editor showing the JavaScript code in 'midtermNamesAndGrades.js'. The code uses 'fs' to read the file 'theNamesGrades.txt', splits it by commas, and then iterates through the resulting array, converting every second element (starting from index 1) to a number.

```
JS midtermNamesAndGrades.js X  theNamesGrades.txt
JS midtermNamesAndGrades.js > ...
1  // This code loads up the txt file
2  const fs = require('fs');
3  let data = fs.readFileSync('theNamesGrades.txt');
4  data = String(data);
5  let arr = data.split(',');
6
7  // This will convert the strings of number in the file from string to numbers
8  for (let i = 1; i < arr.length; i += 2) {
9    arr[i] = Number(arr[i]);
10 }
```

Figure 2: Using functions to load the text file and using “+=2” to skip the names/strings for the numbers to be converted as number type.

The following lines of the program are collections of initializations that will play an important role in the code (Fig 3). For instance, a line that will assign a string to a variable its main purpose is to find the specific name in the list of students and print their name, grades, and determine if they passed or failed. Next, is an empty array that will later be filled with a collection of unique names and grades from the file. A line that will set the value of students' requirements to pass the course, and a simple for loop that will take the existing array to nested arrays, so it contains organized information in every single array with only one index value for each.

```
12 // The name that will be searched in the file that will be printed out.
13 let theNameSearch = "Mayan";
14 // Initializing the array that will hold both the name and their grades.
15 let stdNmGr = [];
16 // The value of the passing grade for the course.
17 let passingGrade = 75;
18 // placeholders for the temp values to swap strings and numbers
19 let temp;
20
21 // Making nested arrays to have different arrays for grades and names
22 function studentArray(theName, theGrade,) {
23     this.theName = theName;
24     this.theGrade = theGrade;
25 }
26
27 // Assigns the names and their grades respectively
28 let k = 0;
29 for (let i = 0; i < arr.length; i += 2) {
30     stdNmGr[k] = new studentArray(arr[i], arr[i + 1]);
31     k++;
32 }
```

Figure 3: Initializations of codes and placeholders that will play an important role in the program

For the following lines of the program, I will introduce the functions I made to achieve the goal of the assignment. We have learned different approaches when it comes to sorting different arrays in certain situations. Out of many options, I have chosen a method of sorting the array by using *Selection Sort* since I find the method convenient and easy to navigate when I work with nested arrays. By creating functions that will take the nested array as the parameter and I have the option to call “theGrade” and “theName” if I only ever want to call specific information in the array. For example, I have a function that sorts the students by their grades from highest to lowest (Fig. 4) and lowest to highest (Fig. 5) using *Selection Sort*. I took the information on the nested array as a whole to determine its length and only used “theGrade” within the array since I am only using that information to sort the students. Without being destructive by rearranging the information, I have the advantage of reorganizing the names as well since I have the information of both “theGrade” and “theName” in one array only using one index value.

```
34 // The function for sorting the highest grade to lowest
35 function sortGradesHighest(assignedArray) {
36     for (let i = 0; i < assignedArray.length; i++) {
37         for (let j = i + 1; j < assignedArray.length; j++) {
38             if (assignedArray[i].theGrade < assignedArray[j].theGrade) {
39                 temp = assignedArray[i];
40                 assignedArray[i] = assignedArray[j];
41                 assignedArray[j] = temp;
42             }
43         }
44     }
45 }
```

Figure 4: A function that sorts the students from highest to lowest according to their grades

```

47 // The function for sorting the lowest grade to highest
48 function sortGradesLowest(assignedArray) {
49     for (let i = 0; i < assignedArray.length; i++) {
50         for (let j = i + 1; j < assignedArray.length; j++) {
51             if (assignedArray[i].theGrade > assignedArray[j].theGrade) {
52                 temp = assignedArray[i];
53                 assignedArray[i] = assignedArray[j];
54                 assignedArray[j] = temp;
55             }
56         }
57     }
58 }

```

Figure 5: A function that sorts the students from lowest to highest according to their grades

The following function is another sorting method that implements the *Insertion Sort* that we have done in class. I made several changes to what we have in sections. The function (Fig. 6) will take the nested array as the parameter and compare the strings to whether the name of the student comes first or later based on the English alphabet. I have taken a similar approach to what I have done in the previous function where it is referencing the whole nested array and using its unique array within it.

```

59 // The function for sorting the names alphabetically from A to Z,
60 // implementing the method of insertion sort learned from the class
61 function insertionSortName(assignedArray) {
62     let temp2 = 0;
63     for (let i = 1; i < assignedArray.length; i++) {
64         temp = assignedArray[i].theName;
65         temp2 = assignedArray[i].theGrade;
66         let j = i - 1;
67
68         // The inspection of where the insertion sort is suppose to occur
69         while (j >= 0 && temp < assignedArray[j].theName) {
70             assignedArray[j + 1].theGrade = assignedArray[j].theGrade;
71             assignedArray[j + 1].theName = assignedArray[j].theName;
72             j--;
73         }
74
75         // swapping the grades and name until the current temp is in the right place
76         assignedArray[j + 1].theName = temp;
77         assignedArray[j + 1].theGrade = temp2;
78         // console.table(assignedArray);
79     }
80 }
81 }

```

Figure 6 A function that will compare the names and sort alphabetically using the Insertion Sort

The next function is a simple feature that will search for the specific name that is input manually by the user in the collection of information on students (Fig. 7). Using a simple search, the process of this function is rather not the most convenient but an effective way to gather information from a big database. Instead of scanning and searching the whole array, I have eliminated half of the process by narrowing the scope of where to look within the array by changing the reference point. Since the array has been already previously sorted alphabetically, I made a statement that will compare the target name to the array's median whether it is in the upper or lower half of the names. The process saves both memory and process time by eliminating half of the scope that needs to be scanned. A small detail added to the function is a statement that will determine if the student searched passed or failed the course and will print its grade along with it.

```
82 // The function for searching a name in the text file.
83 function searchName(assignedArray, targetName, targetGrade) {
84     let start = 0;
85     let end = assignedArray.length;
86     let middle = Math.floor(assignedArray.length / 2);
87     let theIndexofTargetName = 0;
88     let passFail;
89
90     // By doing a binary search and dissecting the group, this takes a less time scan and search instead of searching the whole array.
91     if (targetName < assignedArray[middle].theName) {
92         for (let i = start; i < middle; i++) {
93             if (assignedArray[i].theName == targetName) {
94                 theIndexofTargetName = i;
95                 break;
96             }
97         }
98     } else {
99         for (let i = middle; i < end; i++) {
100             if (assignedArray[i].theName == targetName) {
101                 theIndexofTargetName = i;
102                 break;
103             } else if (targetName == assignedArray[middle].theName) {
104                 theIndexofTargetName = middle;
105             }
106         }
107     }
108 }
109
110 if(assignedArray[theIndexofTargetName].theGrade <= targetGrade){
111     passFail = "failed";
112 } else{
113     passFail = "passed";
114 }
115 console.log('The name is: ${assignedArray[theIndexofTargetName].theName} and their grade is: ${assignedArray[theIndexofTargetName].theGrade} and they ${passFail} the course.');
```

Figure 7: A simple search function that is linear and timesaving method search feature

Printing the students' information on a table is the most convenient way of seeing a list of information organized. It is one of the main reasons why I implemented nested arrays in my program so that it could look aesthetically pleasing (Fig. 8). I made this idea possible by doing a simple *for loop* and assigning each corresponding student to their nested arrays (Fig. 9).

The name is: Mayan and their grade is: 93 and they passed the course.
 The average of the class is: 82.89%
 There are 7 student(s) who passed the course and 3 student(s) who failed.
 Passed:

(index)	theName	theGrade
0	'Adam'	98
1	'Annabel'	87
2	'Hedy'	86
3	'Justine'	95
4	'Mandy'	83
5	'Mayan'	93
6	'Rick'	75

Failed:

(index)	theName	theGrade
0	'Chris'	63
1	'Roann'	74
2	'Ronald'	74.9

Figure 8: A visual table of nested arrays and printed results from the previous functions

```

119 // The function for collecting all the names and grades of all those students who passed and failed the course
120 function passingStudents(assignedArray, passingGrade) {
121   let theListofPassed = [];
122   let theListofFailed = [];
123   let k = 0;
124   let l = 0;
125   for (let i = 0; i < assignedArray.length; i++) {
126     if (assignedArray[i].theGrade >= passingGrade) {
127       theListofPassed[k] = assignedArray[i];
128       k++;
129     } else {
130       theListofFailed[l] = assignedArray[i];
131       l++;
132     }
133   }
134   console.log(`There are ${theListofPassed.length} student(s) who passed the course and ${assignedArray.length - theListofPassed.length} student(s) who failed.`);
135   console.log("Passed: ");
136   console.table(theListofPassed);
137   console.log("Failed: ");
138   console.table(theListofFailed);
139 }

```

Figure 9: The function that will separate the passing and failing students that are determined by the value set manually by the user

```

141 // A function that calculates the average or the mean of the class
142 function theMean(assignedArray) {
143     let theMeanValue = 0;
144     for (let i = 0; i < assignedArray.length; i++) {
145         theMeanValue = theMeanValue + assignedArray[i].theGrade;
146     }
147     theMeanValue = theMeanValue/assignedArray.length;
148     console.log(`The average of the class is: ${theMeanValue.toFixed(2)}%`);
149 }

```

Figure 10: Using for loop to add every grades of the students and dividing it by the length of the array to get the class average.

```

152 // Prints out the "rough" array of the file loaded
153 console.table(arr);
154
155 sortGradesHighest(stdNmGr);
156 console.log("Highest Grade to Lowest: ");
157 console.table(stdNmGr);
158
159 sortGradesLowest(stdNmGr);
160 console.log("Lowest Grade to Highest: ");
161 console.table(stdNmGr);
162
163 insertionSortName(stdNmGr);
164 console.log("Alphabetical order A to Z: ");
165 console.table(stdNmGr);
166
167 searchName(stdNmGr, theNameSearch, passingGrade);
168
169 theMean(stdNmGr);
170
171 passingStudents(stdNmGr, passingGrade);
172

```

Figure 11: Calling all the functions and assigning parameters to gather the information needed to run the program

Lastly, this function (Fig. 10) will take the mean or the average of the whole class by using a simple *for loop* and printing the information gathered according to the array.

The program is overall quite straightforward and has a clear intention of what it wants to achieve. However, there are multiple roadblocks that I encountered before I got to decide that I have met the program's goal and purpose. For instance, I initially want to incorporate better visuals that show the result by incorporating P5.JS but unfortunately, the code needs to be fully translated with shapes and nested arrays do not work well with it. On the other hand, the code seems to be working fine and printing using *Code Runner* does the whole program justice when it comes to its visual aspect.