Name: Vuong Cat VO
Student #: N10049266
Date: 20-Aug-2024

IFQ714 Introduction to JavaScript Programming

Assigment 1: Programming Exercises

Introduction

This report will provide both an overview and rationale for the various programming approaches that were adopted in order to complete all five steps of the JavaScript programming exercises as set out by this assignment.

Step 1: Loading JSON data

```
const fs = require('fs');
const data = fs.readFileSync("Assignment1/NEOWISE_Dataset.json", "utf8");
const neowise = JSON.parse(data);
```

The code snippet above using Node.js for file handling.
The content of the JSON file "NEOWISE_Dataset" is read and stored in the variable "data" as a string.
The JSON.parse method converts the JSON string into an JavaScript object. Then store into the variable named "neowise".

Step 2: Basic functions

A total of 07 basic functions were created in order to complete the programming exercises.

Function 1:

```
// Display NEO index number
function displayNEOIndex(value){
   console.log('+------------------+');
   console.log('    NEO No: ', value);
   console.log('+------------------+');
}
```

The function above takes the index value of a NEO then displays the value in a user readable format.

Function 2:

```
/* Display NEO neoData in a readable format
If the value available then display the value otherwise display N/A
*/
// Export to Unit Test
//module.exports = displayNEOIndex;
function displayNEOData(neoData) {
   console.log(`   + Designation: ${neoData.designation ? neoData.designation : 'N/A'} `);
   console.log(`   + Discovery Date: ${neoData.discovery_date ? neoData.discovery_date : 'N/A'}`);
   console.log(`   + Observed Magnitude: ${neoData.h_mag ? neoData.h_mag : 'N/A'}`);
   console.log(`   + Minimum Orbit Intersection Distance (MOID): ${neoData.moid_au ? neoData.moid_au : 'N/A'}`);
   console.log(`   + Perihelion Distance: ${neoData.q_au_1 ? neoData.q_au_1 : 'N/A'}`);
   console.log(`   + Aphelion Distance: ${neoData.q_au_2 ? neoData.q_au_2 : 'N/A'}`);
   console.log(`   + Orbital Period (year): ${neoData.period_yr ? neoData.period_yr : 'N/A'}`);
   console.log(`   + Orbital Inclination: ${neoData.i_deg ? neoData.i_deg : 'N/A'}`);
   switch (neoData.pha) {
       case true:
           console.log('   + Potentially Hazardous Asteroid (PHA): YES');
           break;
       case false:
           console.log('   + Potentially Hazardous Asteroid (PHA): NO');
           break;
       case null:
           console.log('   + Potentially Hazardous Asteroid (PHA): N/A');
           break;
   }
```

```
    console.log(`    + ORBIT CLASS: ${neoData.orbit_class ?
neoData.orbit_class : 'N/A'}`);


console.log('============================================================');
}
```

The function takes a JS object as an argument. Then display all the NEO information in a user readable format.
In each console.log, the function checks whether the Neo's value is defined or not. If the value is present then displays the value. Otherwise displaying "N/A" string.
A switch case has been use to display the PHA data in a readable format

- If the PHA's value is true then displays YES
- If the PHA's value is false then displays NO
- If the PHA's value is undefined then displays "N/A"

Function 3:

```
// Find NEO based on its orbit_class then add to an array
function findNEO_OrbitClass(neoData, searchValue){
    let result = [];
    for (let i = 0; i < neoData.length; i++)
        if (neoData[i].orbit_class == searchValue){
            result.push(neoData[i]);
        }
    return result;
}
```

This findNEO_OrbitClass function searches and adds all NEOs that have the same Orbit_Class into an JS array. Then return the array.
This function uses a for loop to iterate through all the JS objects that are stored in the "neowise" variable. Which NEOs that have the same Orbit_Class as the searching value will be store in an JS array.

Function 4:

```
// Finding NEO based on its designation then return that NEO object
function findNEO_Designation(neoData, searchValue) {
    let tempNEO = [];
    neoData.forEach(element => {
        if (element.designation === searchValue) {
            tempNEO = element;
        }
    });
```

```
    return tempNEO;
}
```

The function finds a NEO that has the designation value the same as the searching value. Then return the NEO that NEO object.
A foreach loop is used to iterate through the JS objects. It compares the NEO's designation of each NEO with the search value. Then return the NEO's designation that matches the searching value.

Function 5: This function calculates and returns the maximum orbit value of the NEO.
(This function calculates and returns the wrong answer. I do not have time to fix the function to meet the assignment requirement.)

```
// Measure the maximum orbit of a NEO
function NEOMaxOrbit (neo){
    //IF the orbit of the NEO is null then the value is 0
    let maxOrbit = Math.max(neo.moid_au ? neo.moid_au : 0, neo.q_au_1 ?
neo.q_au_1 : 0, neo.q_au_2 ? neo.q_au_2 : 0);
    return maxOrbit;
}
```

This function takes the NEO object as an argument.
It uses Math.max() to calculate the maximum orbit value among the moid_au, q_au_1 and q_au_2. If any of the values is undefined then I consider it has value 0.
The function then returns the maximum orbit value of a NEO.

Function 6: This function calculates and returns the minimum orbit value of the NEO.
(This function calculates and returns the wrong answer. I do not have time to fix the function to meet the assignment requirement.)

```
// Measure the minimum orbit of a NEO
function NEOMinOrbit (neo){
    let minOrbit = 0;
    //If the orbit value of the NEO is null then exclude it from the
calculation
    switch (true) {
        case neo.moid_au == null:
            minOrbit = Math.min(neo.q_au_1, neo.q_au_2);
            break;
        case neo.q_au_1 == null:
            minOrbit = Math.min(neo.moid_au, neo.q_au_2);
            break;
```

```
        case neo.q_au_2 == null:
            minOrbit = Math.min(neo.moid_au, neo.q_au_1);
            break;
        default:
            minOrbit = Math.min(neo.moid_au, neo.q_au_1, neo.q_au_2);
    }
    return minOrbit;
}
```

This function takes the NEO object as an argument.
The function using the switch case to eliminate any undefined or null value from the calculation.
It then uses the Math.min() to calculate and return the minimum orbit value of a NEO.

Function 7: This function calculates the average orbit value of the NEO.
(This function also calculates and returns the wrong answer. I do not have time to fix the function to meet the assignment requirement.)

```
// Measure the average orbit of a NEO
function NEOAverageOrbit (neo) {
    let averageOrbit = 0;
    //If the orbit value of the NEO is null then exclude it from the
calculation
    switch (true) {
        case neo.moid_au == null:
            averageOrbit = (neo.q_au_1 + neo.q_au_2) / 2;
            break;
        case neo.q_au_1 == null:
            averageOrbit = (neo.moid_au + neo.q_au_2) / 2;
            break;
        case neo.q_au_2 == null:
            averageOrbit = (neo.moid_au + neo.q_au_1) / 2;
            break;
        default:
            averageOrbit = (neo.moid_au + neo.q_au_1 + neo.q_au_2) / 3;
    }
    return averageOrbit;
}
```

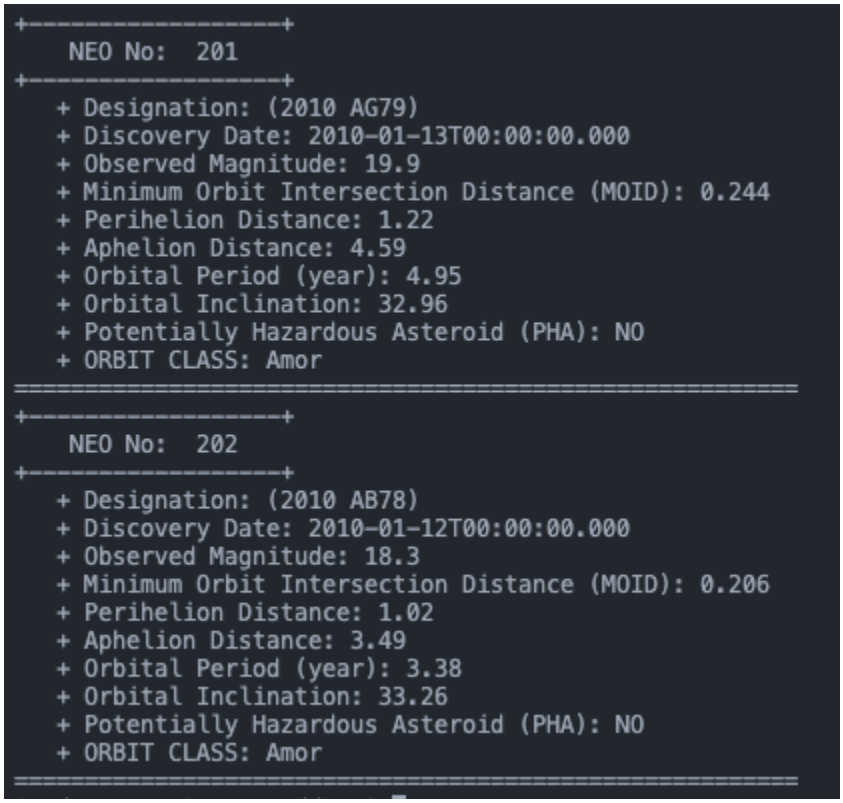
This function takes the NEO object as an argument.
The function using the switch case to eliminate any undefined or null value from the calculation.
Then it calculates and returns the minimum orbit value of a NEO.

Step 3: Analysis

All analyses were attempted and completed using those functions below.
Function 9: This function uses a for loop to iterate through all NEO objects and display all NEOs in a user readable format by calling 02 basic functions [displayNEOIndex] and [displayNEOData].

```
// Step 3: Analysis
// 3.1 Display information of all NEOs in the datasheet
function displayAllNEODataInfo(neoData){
   for (let i = 0; i < neoData.length; i++) {
       displayNEOIndex(i+1);
       displayNEOData(neoData[i]);
   }
}
// Testing display all NEO data
//displayAllNEODataInfo(neowise);
```

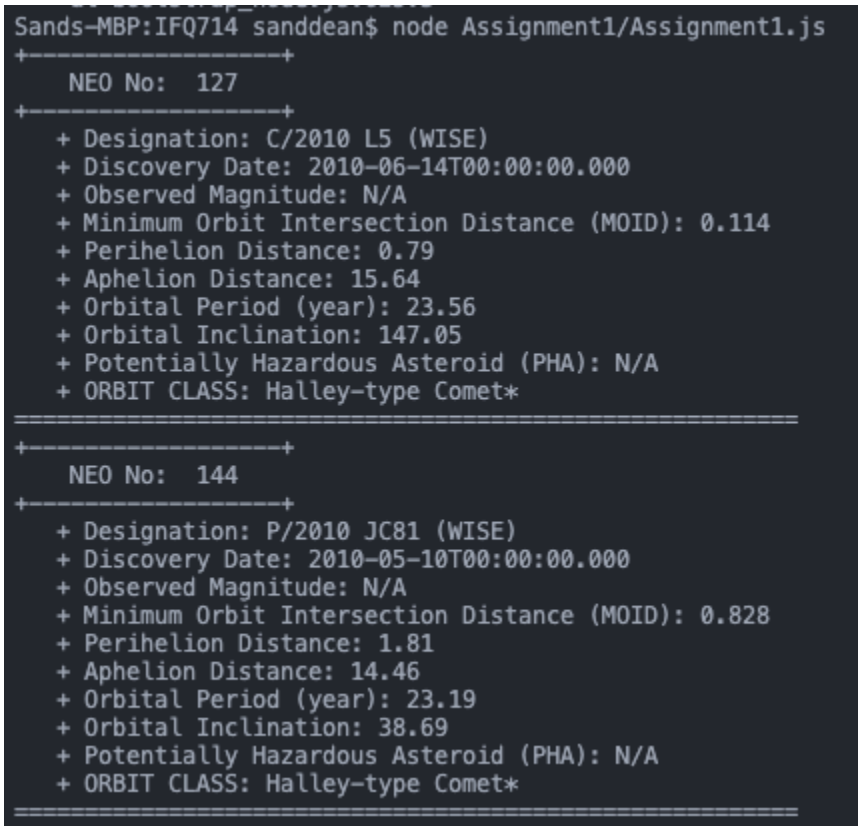
The output were logged to the console to display the results as below



Function 10 and 11: These function uses the for loop to iterate through all NEO objects. It displays all NEOs that have the same orbit class or PHA as the searching class. 02 Basic functions are used to display the result are [displayNEOIndex] and [displayNEOData]

```
// 3.2.a Display information on NEOs data that have certain criteria. Eg.
Certain Orbit Class
function displayNEO_OrbitClass(neoData, searchValue){
   for (let i = 0; i < neoData.length; i++)
       if (neoData[i].orbit_class == searchValue){
           displayNEOIndex(i);
           displayNEOData(neoData[i]);
       }
}
// Test finding NEO with orbit_class 'Halley-type Comet*'
displayNEO_OrbitClass(neowise,'Halley-type Comet*');
```

The output of the function was logged to the console with the result below

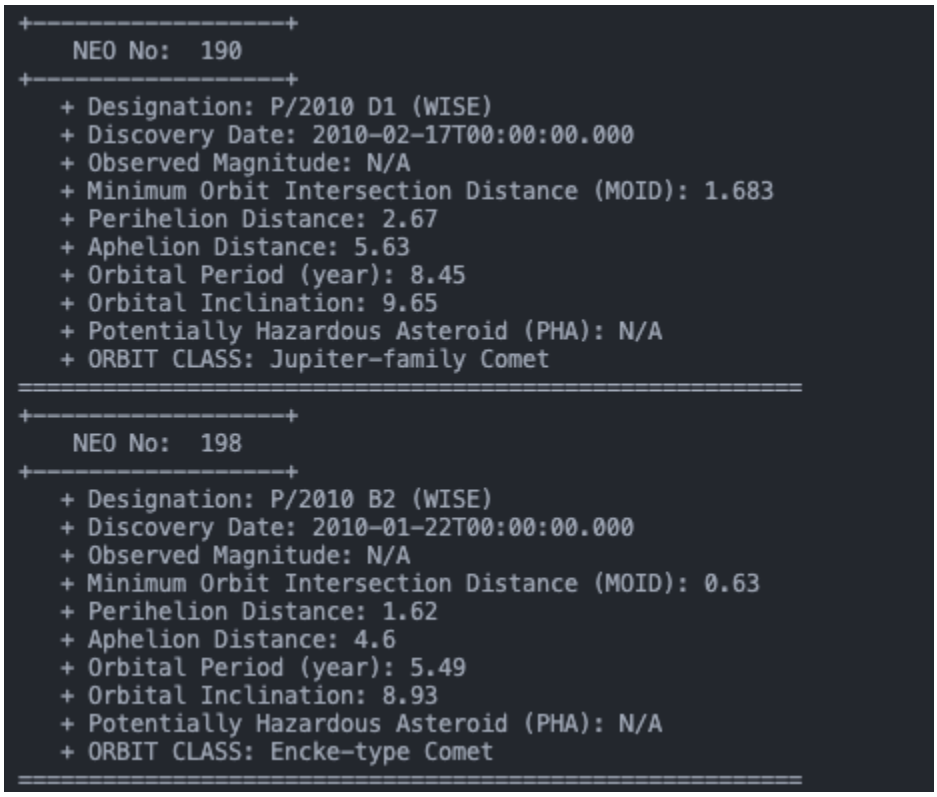


Function 11:

```
// 3.2.b Display information on NEOs data that have certain criteria. Eg.
Certain PHA
function displayNEO_PHA(neoData, searchValue){
   for (let i = 0; i < neoData.length; i++) {
       if (neoData[i].pha == searchValue) {
           displayNEOIndex(i);
```

```
            displayNEOData(neoData[i]);
        }
    }
}
// Test display NEOs with PHA [true, false, null]
displayNEO_PHA(neowise,null);
```

The result logged to the console as below



Function 12, 13, 14: These functions calculate and return the maximum/minimum orbit value of the NEOs that have the same orbit class.
(These functions calculate and return the wrong answer due to the misunderstanding of the Assignment requirement. I do not have time to fix the function to meet the assignment requirement.)
These functions take the NEO objects and a search value as arguments.
It searches all NEOs that have the same orbit_class and stores them in an JS array named "tempNEOs'.
It uses a for each loop to iterate through the tempNEOs array to add each NEO designation and its max orbit value to the 2 dimensional array named "tempNEO2DArray".
The function uses the array map method to extract the second value of the 2 dimensional array and store them in an array named "secondValues". It then uses Math.max() / Math.min() or a for loop to calculate the result.
These functions return the calculated result as a number for Unit Test purpose.

Function 12:

```
// Measure the maximum orbit value of NEOs in the same orbit_class
function MaxOrbitOfSameClassNEO (data, searchValue) {
   // Search all NEO with the same Orbit Class and add to an array
   let tempNEOs = findNEO_OrbitClass(data,searchValue);

   // 2D Array that hold NEO designation and Max orbit value
   let tempNEO2Darray = [];

   // Calculate the max orbit value of each NEO in the array above
   tempNEOs.forEach(element => {
       // Add each Neo with its max orbit to an 2D array
       tempNEO2Darray.push([element.designation, NEOMaxOrbit(element)]);
   });
   // Compare all the Max orbit value then return the orbit designation
   // Extract the second values of each sub-array
   const secondValues = tempNEO2Darray.map(element => element[1]);
   // Find the maximum value among the secondValues array
   const maxOrbitValue = Math.max(...secondValues);
   // Find the NEO Designation that has the max orbit in the array
   const result = tempNEO2Darray.find(element => element[1] ===
maxOrbitValue)[0]; // the [0] return the designation of the NEO in the 2D
array
   // Display the NEO with max orbit in the same Orbit Class
console.log('================================================================
======');
   console.log('The NEO with MAX orbit in the same Orbit Class: [',
searchValue, '] has the MAX Orbit value: ', maxOrbitValue, ' AUs');
console.log('================================================================
======');
   displayNEOData(findNEO_Designation(neowise, result));
   return maxOrbitValue;
}
```

Function 13:

```
// Measure the minimum orbit value of NEOs in the same orbit_class
function MinOrbitOfSameClassNEO (data, searchValue) {
   // Search all NEO with the same Orbit Class and add to an array
   let tempNEOs = findNEO_OrbitClass(data,searchValue);
   // 2D Array that hold NEO designation and Min orbit value
   let tempNEO2Darray = [];
   // Calculate the min orbit value of each NEO in the array above
   tempNEOs.forEach(element => {
       // Add each Neo with its min orbit to an 2D array
       tempNEO2Darray.push([element.designation, NEOMinOrbit(element)]);
   });
   // Compare all the Min orbit value then return the orbit designation
   // Extract the second values of each sub-array
   const secondValues = tempNEO2Darray.map(element => element[1]);

   // Find the minimum value among the secondValues array
   const minOrbitValue = Math.min(...secondValues);
   // Find the NEO Designation that has the max orbit in the array
   const result = tempNEO2Darray.find(element => element[1] ===
minOrbitValue)[0]; // the [0] return the designation of the NEO in the 2D
array
   // Display the NEO with min orbit in the same Orbit Class
console.log('================================================================
======');
   console.log('The NEO with MIN orbit in the same Orbit Class: [',
searchValue, '] has the MIN orbit value: ', minOrbitValue, ' AUs');
console.log('================================================================
======');
   displayNEOData(findNEO_Designation(neowise, result));
   return minOrbitValue;
}
```

Function 14:

```
// Measure the average orbit value of NEOs in the same orbit_class
function AveOrbitOfSameClassNEO (data, searchValue) {
   // Search all NEO with the same Orbit Class and add to an array
   let tempNEOs = findNEO_OrbitClass(data,searchValue);
   // 2D Array that hold NEO designation and Max orbit value
   let tempNEO2Darray = [];
   // Calculate the max orbit value of each NEO in the array above
   tempNEOs.forEach(element => {
       // Add each Neo with its max orbit to an 2D array
       tempNEO2Darray.push([element.designation,
NEOAverageOrbit(element)]);
   });
   // Calculate the average orbit value then return the orbit designation
   // Extract the second values of each sub-array
   const secondValues = tempNEO2Darray.map(element => element[1]);
   // Find the average value of the secondValues array
   let sum = 0;
   for (let i = 0; i < secondValues.length; i++){
       sum += secondValues[i];
   }
   const aveOrbitValue = sum / secondValues.length;
   // Display the NEO with max orbit in the same Orbit Class

console.log('================================================================
======');
   console.log('The AVERAGE orbit value of all NEOs in the same Orbit
Class: [', searchValue ,'] is: ', aveOrbitValue, 'AUs' );

console.log('================================================================
======');
   return aveOrbitValue;
}
```

The challenge I encountered was I did not fully understand the assignment requirements. So I did a wrong analysis. While reporting my work, I realized my mistake and came up with an idea on how to fix my mistake. But I don't have enough time to fully implement my idea nor creating

the Unit test case. However, I attempted to implement those functions below and used console.log() to test the function. The results seem correct.
Function 3a: this is the updated version of the function 3.

```
// Find NEO based on its orbit_class and PHA value then add to an array
function findNEO_OrbitClassPHA(neoData, searchOrbitClass, searchPHA){
   let result = [];
   for (let i = 0; i < neoData.length; i++)
       if (neoData[i].orbit_class == searchOrbitClass && neoData[i].pha ==
searchPHA){
           result.push(neoData[i]);
       }
   return result;
}
```

This function finds all NEOs based on its orbit class and the PHA value then add the NEO to an JS array named result.

Function 12a: this is the updated version of the function 12

```
//Updated function MaxOrbit_SameClass_SamePHA
function MaxOrbit_SameClassPHA (data, searchValue, searchValue2) {
   // Search all NEO with the same Orbit Class and add to an array
   let tempNEOs = findNEO_OrbitClassPHA(data, searchValue, searchValue2);
   // Find max moid_au
   let all_moid_au = [];
   // Add all H_mag to an array
   tempNEOs.forEach(element => {
       all_moid_au.push(element.moid_au);
   });
   let max_moid_au = Math.max(...all_moid_au);
   return max_moid_au;
}
/*
let testClass = 'Apollo';
let testPHA = true;
let testresult = MaxOrbit_SameClassPHA(neowise, testClass, testPHA); //
This return 0.049 AUs
console.log(`Max H_mag of all NEOs in Apollo class that has a true PHA
value is: ${testresult} AUs`);
```

This function

Function 13a: this function is the updated version of the function 13.

```
//Updated function MinOrbit_SameClass_SamePHA
function MinOrbit_SameClassPHA (data, searchValue, searchValue2) {
   // Search all NEO with the same Orbit Class and add to an array
   let tempNEOs = findNEO_OrbitClassPHA(data, searchValue, searchValue2);

   // Find max h_mag
   let all_moid_au = [];
   // Add all H_mag to an array
   tempNEOs.forEach(element => {
       all_moid_au.push(element.moid_au);
   });
   let min_moid_au = Math.min(...all_moid_au);
   return min_moid_au;
}

//let testresult2 = MinOrbit_SameClassPHA(neowise, testClass, testPHA); //
This return 0.0002 AUs
//console.log(`Min H_mag of all NEOs in Apollo class that has a true PHA
value is: ${testresult2} AUs`);
```

This function takes the NEO objects, orbit_class and PHA value as arguments
It called the function findNEO_OrbitClassPHA to search for all NEOs that have the same orbit class and PHA value and store them in a JS Array named "tempNEOs".
It uses a foreach loop to iterate through the JS array to extract all orbit values that need to be calculated then add to another JS array named "all_moid_au".
The function will use Math.min() or Math.max() method to calculate the minimum or maximum orbit value then return it.
The results after running the 02 updated function as below.

```
Sands-MBP:IFQ714 sanddean$ node Assignment1/Assignment1.js
Max Moid_au of all NEOs in Apollo class that has a true PHA value is: 0.049 AUs
Min Moid_au of all NEOs in Apollo class that has a true PHA value is: 0.0002 AUs
Sands-MBP:IFQ714 sanddean$
```

Step 4: Changing JSON format

Function 15: This function rearranges the NEO data and writes to a new JSON file.

```
// Step 4: Changing the JSON format
```

```
// Rearrange the NEO data
function rearrangedNEOs (neoData) {
   // create temp object to store the rearranged data
   let tempNEOs = {};
   // Go through each element of the JSON file
   neoData.forEach(element => {
       let orbit_class = element.orbit_class;
       // Create the new array to store the data if the particular orbit
class has not existed in the temp array yet
       if (!tempNEOs[orbit_class]) {
           tempNEOs[orbit_class] = [];
       }
       //add the element to the corresponding array.
       tempNEOs[orbit_class].push(element);
   });
   return tempNEOs;
}
```

This function takes the NEOs object as argument
The function uses the JS tempNEOs object to store the rearranged data.
The function uses a foreach loop to iterate through the whole given data.
Each NEO's orbit class will be stored in a temp variable called orbit_class.
The function will create a new object key if the orbit value has not existed.
Then it will add the NEO to the object with the object key according to the NEO orbit_class.
The function will return a JS object that will be written to a new JSON file by the code below

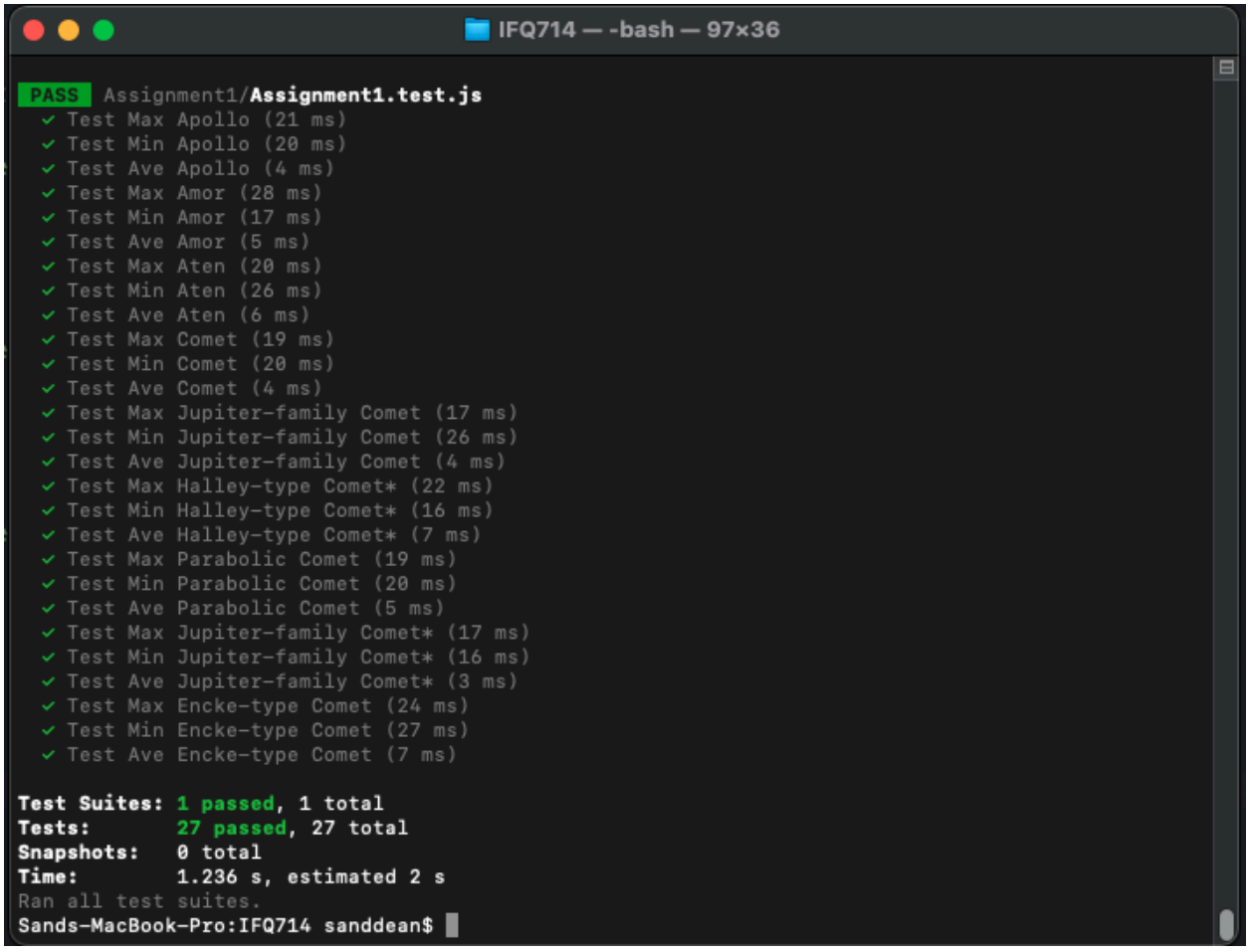
```
// This 2D array will hold the rearranged Neo Data. The data is
categorised based on the orbit_class.
const rearrangedNEOdata = rearrangedNEOs(neowise);

// Write the rearranged data to the new JSON file.
fs.writeFileSync('Assignment1/Rearranged NEO Data.json',
JSON.stringify(rearrangedNEOdata, null, 4));
```

The code below exports all functions to use in the User Test Case file.

```
//Export all functions to Test Case
module.exports = {
   displayNEOIndex,
   displayNEOData,
```

```
    findNEO_OrbitClass,
    findNEO_Designation,
    NEOMaxOrbit,
    NEOMinOrbit,
    NEOAverageOrbit,
    MaxOrbitOfSameClassNEO,
    MinOrbitOfSameClassNEO,
    AveOrbitOfSameClassNEO,
    rearrangedNEOs
};
```

The Test case result is below



Conclusion

Through this project, I learn how to work with JavaScript objects, arrays, 2D arrays. I also learned how to read and write JSON files. I learned how to use built-in methods to work with

data that is stored in JavaScript arrays. I also learned how to create Unit Test cases to test my code ensuring the code running with minimum error. Even though I did not successfully resolve all of the assignment questions. But I did came up with the idea to resolve the issue that meet the assignment criteria.