Christopher Horsfall

IFQ714 Introduction to Java Script

25/08/2024

Assignment 1: Programming Exercises

Report

## Solutions:

The first step in understanding the data provided on the Near Earth Objects was to parse the JSON data into an “object of objects”. This format was decided upon to facilitate the use of java script object methods. Earlier the data was parsed into an “array of objects” and corresponding functions were designed around the array, but it was concluded that object methods is a feature of the JavaScript language that should be utilized.

Because the data initially had properties that were abbreviations, it was decided to rename the properties to better help communicate the data. This resulted in some tabulated data not conforming to the size of the debug consol in VSCode. A work around was devised to print the output of functions to a separate text document so as to be viewed adequately.

As required by the assignment, several basic methods were constructed to interact with the data. These include: a “display object” method, a “get object by index” and a “find by orbit class” method. The “get object by index” method was not useful as the data no longer had a true index. This is because the data was an “object of objects” not an “array of objects”, so objects were accessed via their designation (name) instead of index. Accessing objects by their name appears more useful than accessing them by their index, the index of an item is likely to be arbitrary and could change in the future.

The findByOrbitClass method was used to create new arrays each containing a specific orbit class for later analysis.

A “find Potentially Hazardous Asteroid” method was created to arrange PHAs into their own array for later analysis.

## Analysis:

Now that all the Near Earth Objects were in separate arrays based on their Orbit class or if they were Potentially Hazardous Asteroids, some functions needed to be written to analyze the data.

A “sort NEO by property” function was created, this function was to be used to help find min and max values for object properties. Because some the data was missing a “remove Null entries” function was created. Properties with Null or Undefined values would break attempt to find min and max values.

The sorting function was used as a base to build a “find Max object by property” and “find Min Object by property” function. A “calculate Average of property” function was also written.

Finally, an “Analyze Class” function was written combining the previous functions into one output. The function can also be used to analysis PHAs.

Example below:

Analysis: Apollos

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Min | Max | Average |
| Observed Magnitude | 16.2 | 24.3 | 20.53 |
| Minimum Orbit Intersection Distance (AU) | 0.0002 | 0.703 | 0.13 |
| Perihelion (AU) | 0.14 | 1.01 | 0.78 |
| Aphelion (AU) | 1.08 | 9.89 | 3.20 |
| Orbital Period (Years) | 1 | 12.13 | 2.94 |
| Orbital Inclination (Degrees) | 0.82 | 64.75 | 22.39 |

Note the table as displayed in the console log of VSCode displays many more decimal figures for the values in the Average column.

## Observations:

Through the analysis functions it was observed that PHAs have a very low average MOID (Minimum Orbit Intersection Distance (AU)). This means that the asteroid’s orbit very nearly intersects the orbit of Earth. The max MOID was also low at only 0.05 AU.

Through the analysis it was observed that Apollo class objects have an orbit that ‘crosses’ the orbit of Earth. Specifically their perihelion is less than 1 AU and their aphelion is greater than 1 AU. This means that when viewed from above the orbital plane it appears that the orbits intersect, however this ‘intersection’ is not true in 3 dimensions generally.

Through the analysis it was observed that Amor class objects have an orbits entirely outside the orbit of Earth. Their Perihelion and Aphelion are both greater than 1 AU.

## Testing:

Since the ‘remove Null entries’ function and the sorting function were crucial for other functions to work correctly it was decided to write tests to test these functions first. Initially the ‘remove null entries’ function failed the tests, this was because of the strict nature of the .toEqual method. The function was not returning the items in the exact order the test was expecting. This was not necessary for the function to do its job so the test was altered to be less strict.

expect(result).toEqual(expect.arrayContaining(expected));