IFQ714 Introduction to JavaScript Programming

Assignment 1: Programming exercises

Jack CHEN n8597626

Introduction

This report will detail the number of approaches I used to complete the programming exercises for this assignment. It will also cover the rationale as to why I used these approaches and how I analyse the NEO data provided.

Step 1: Loading JSON data

I used the Node.js file system module to read the .json file and store it in a constant called data. To avoid assigning the data as a global variable, I created a function called parseData which takes in dataSet as a parameter. This is then called inside other functions which need to use the parsed data.

Step 2: Basic functions

I created a total of 9 helper functions to assist in analysing the data (found in functions.js).

Function 1: **getNeoByIndex(dataSet, index)**
This function returns an element (NEO object) in the dataset if a valid index is passed or null if there is no element found at that index.

Function 2: **getNeoByDesignation(dataSet, designation)**
This function returns an array containing the element/s (NEO object) that has an designation value which matches that one passed into the function or null if no matches are found. I used the filter() method (of Array instances) to filter the NEO objects that have the designation value passed into the function and return it. If the filtered array is empty (no NEO object has that designation value), then null is returned.

Function 3: **getNeoByMagnitude(dataSet, magnitude)**
This function returns an array containing the element/s (NEO object) that has an h_mag value which matches that one passed into the function or null if no matches are found. Refer to function 2 for the approach I used to achieve this.

Function 4: **getNeoByPha(dataSet, isPha)**
This function returns an array containing the element/s (NEO object) that has a pha value which matches that one passed into the function or null if no matches are found. Refer to function 2 for the approach I used to achieve this.

Function 5: **getNeoByClass(dataSet, orbitClass)**
This function returns an array containing the element/s (NEO object) that has a value which matches that one passed into the function or null if no matches are found. I used the replace method for Strings to remove the “*” found in some NEO class names to allow for accurate

string matching with the parameter (orbitClass) of this function.Refer to function 2 for the approach I used to achieve this.

Function 6: **getMinOrbitValue(dataSet, orbitProp)**
This function returns an array containing the element/s (NEO object) that has the min value of the orbitProp (period_yr, moid_au, q1_au etc.). I use the map() method of Arrays to create an array of the orbitProp values. I used Math.min() to get the min value from that array and used filter() method on the data set to return an array of NEO object/s that have this min orbitProp value. If no object has this orbit property in the data set, return null.

Function 7: **getMaxOrbitValue(dataSet, orbitProp)**
Refer to function 6 approach. Math.max() was used instead to get the max value from orbitProp array of values.

Function 8: **getAverageOrbitValue(dataSet, orbitProp)**
This function returns the average value of the orbits (orbitProp) for elements in the dataset. The average is determined using (sum of the values / total length of array). I used reduce() method to get the total sum of the array and array.length to get the total length of the array. If no object has the orbitProp key, return null.

Function 9: **displayNeoInfo(neo)**
This function logs the neo data in a readable format to console using console.table() and console.log() / template strings. If the neo parameter is null/undefined, return null.

Step 3: Analysis
The analysis of the NEO data was done using the functions in Step 2.

(index)	Values
min q1	0.48
max q1	1.04
avg q1	0.841
min q2	1.07
max q2	4.3
avg q2	2.458
min moid	0.0002
max moid	0.05
avg moid	0.025
min mag	18
max mag	21.9
avg mag	20.583

(index)	designation	discovery_date	h_mag	moid_au	q_au_1	q_au_2	period_yr	i_deg	pha	orbit_class
0	'(2010 DJ77)'	'2010-02-20T00:00:00.000'	21.6	0.05	0.75	1.16	0.93	24.98	true	'Aten'

(index)	designation	discovery_date	h_mag	moid_au	q_au_1	q_au_2	period_yr	i_deg	pha	orbit_class
0	'365424 (2010 KX7)'	'2010-05-16T00:00:00.000'	21.9	0.034	0.82	1.16	0.98	21.49	true	'Aten'

The maximum Minimum Orbit Intersection Distance (MOID) of all PHA in the dataset I found to be 0.05 AU. The minimum observed magnitude or brightness (h_mag) of all PHA in the dataset I found to be 21.9. Thus, NEOs with MOID value <= 0.05 AU and h_mag value <=

(index)	min q1	max q1	avg q1	min q2	max q2	avg q2	min moid	max moid	avg moid	min period	max period	avg period
Apollo	0.14	1.01	0.784	1.08	9.89	3.207	0.0002	0.703	0.138	1	12.13	2.941
Amor	1.02	1.3	1.126	1.45	41.78	4.19	0.043	1.617	0.225	1.39	99.82	5.235
Aten	0.31	0.85	0.612	1.04	1.6	1.24	0.007	0.316	0.07	0.72	1	0.891
Comet	1.59	7.15	3.542	122.19	23255.11	5153.052	0.65	6.373	2.839	520.06	1254179.62	238220.626
Jupiter-family Comet	1.2	3.66	2.179	4.65	10.42	6.681	0.204	2.945	1.225	5.01	17.3	9.654
Halley-type Comet	0.79	1.81	1.3	14.46	15.64	15.05	0.114	0.828	0.471	23.19	23.56	23.375
Parabolic Comet	1.09	2.27	1.68	null	null	null	0.307	1.546	0.926	null	null	null
Encke-type Comet	1.62	1.62	1.62	4.6	4.6	4.6	0.63	0.63	0.63	5.49	5.49	5.49

21.9 are classified as PHA.

The minimum perihelion distance (q_au_1) of all Amor NEOs I found to be 1.02 AU. The maximum perihelion distance of Amor NEOs is 1.3 AU. NEOs classified as Amor have q_au_1 with range: 1.02 AU <= q_au_1 <=1.3 AU.

The maximum perihelion distance of all NEOs with class Apollo I found to be 1.01 AU. NEOs classified as Apollo have q_au_1 <= 1.01 AU.

The minimum aphelion distance (q_au_2) of Aten is 1.04 AU. Aten classed NEOs have q_au_2 >= 1.04 AU.

Step 4: Changing JSON format

The rearranging of the data into a new format which contains arrays based on the NEO class was done using the function **arrangeNeoByClass(dataSet)**. This function iterates through the original data array and adds the array of NEOs of a certain class (I obtained this using **getNeoByClass(dataSet, orbitClass)**, refer to Step 2) as a value and the orbit_class property as key to an object which is returned. The function **exportDataToFile(filename, dataSet)** is used to stringify the object and writes it to a new .json file using fs.write(). I created a constructor called **NeoData** which takes a data array as a parameter and contains methods detailed in Step 2 to assist in the analysis.

Step 5: Unit tests

I tested 3 functions in total using jest:

- **getNeoByIndex(dataSet, index)**
- **getNeoByDesignation(dataset, designation)**
- **getNeoByClass(dataset, orbitClass)**

The results are displayed below:

```
PASS ./functions.test.js
  √ Input of data and index, output should return element at that index in data (2 ms)
  √ Input of data and invalid index, output should return null
  √ Input of data and designation value, output of element with matching designation (4 ms)
  √ Input of data and invalid designation value, output should be null
  √ Input of data and orbit_class, output of element with matching orbit_class in data
  √ Input of data and orbit_class, output should have length equal to number of elements with matching class

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.528 s, estimated 3 s
Ran all test suites.
```

Conclusion:

The methods used to analyse the NEO data throughout the assignment are from Array instances. I used filter() to find NEOs that have properties which match certain values, map() was used to create arrays of certain values for ease of analysis (e.g. h_mag, q_au_1 etc). These methods were included in the NeoData constructor to assist in analysis.

The results found from analysis are as follows:

- NEOs with MOID value <= 0.05 AU and h_mag value <= 21.9 are classified as PHA.
- NEOs with q_au_1 range: 1.02 AU <= q_au_1 <=1.3 AU are classified as Amor.
- NEOs classified as Apollo have q_au_1 <= 1.01 AU.
- NEOs classified as Aten have q_au_2 >= 1.04 AU.

Reference:

- *Neo basics* (no date) *NASA*. Available at: https://cneos.jpl.nasa.gov/about/neo_groups.html (Accessed: 13 August 2023).