WEB222 Assignment 2 Instructions

Introduction

This assignment will help you learn and practice JavaScript Arrays and Objects, as well as working with real data.

Please do this assignment on your own. You should not work in partners, and all code must be written by you. Gaining experience with JavaScript and the web takes a lot of personal practice, and working on these problems yourself will help build your skill.

Submission

To hand in your work, see the "**Submitting your Assignment**" section below.

Please read and follow all instructions below carefully. If you have problems or questions, talk to your professor.

Setup

Just as we did with assignment 1, this assignment once again uses "Unit Testing", and relies on a number of dependencies, which must be installed on your computer.

In order to install these dependencies, you must first install Node.js on your computer. See installation instructions at:

> https://nodejs.org/en/

You can install the LTS (Long Term Support) version of node.js, which is currently 12.18.3, although any 12.x.x version should work.

Install Dependencies

Open a command line terminal (e.g., cmd.exe) and navigate (i.e., `cd`) to the directory where you have unzipped the assignment files. When you type `dir` (Windows) or `ls` (Linux/macOS) you should see the `package.json` file.

In this directory, install the assignment dependencies using the Node.js Package Manager command, named `npm`. The `npm` command is installed along with node.js. In your terminal, type the following:

```
npm install
```

This will download and save all the necessary dependency files to a folder named `node_modules/` in the current directory. You should see a `node_modules` folder next to your `package.json`.

If you have trouble getting "npm install" to work:

- Did you install node.js?
- If you type "node –version" in your terminal, does it print the version?
- Are you in the right directory (you must cd into the correct directory)

If you need help, ask your classmates and/or talk to your professor.

Learn how to Navigate the Code

The assignment has a number of components:

- JSON (structured text) files, containing COVID case data for the city of Toronto.
- src/problem-0x.test.js, unit test files. You need to get these to all pass.
- src/cases.js, Part 1 of the assignment. Work on this first.
- src/analyze.js, Part 2 of the assignment. Work on this second after Part 1 is done.

Part 1: Implement all Functions in src/cases.js and Pass All Unit Tests

You are asked to complete the code in the file `src/cases.js`. A basic file layout has already been created with various functions and variables. Also, detailed comments have been left above each function you need to implement.

In addition, unit tests have been written for each function. They are named `src/problem-00.test.js`, `src/problem-01.test.js` and so on.

These tests will help you determine if your code is working correctly: running the tests should produce the output that the tests expect, and that tests will either pass or fail.

To run the tests, use the npm command:

```
npm test
```

Your goal is to get all of the tests to pass correctly. If a test fails, pay attention to the error messages that get produced, what was expected vs. what was actually returned, and make corrections to your code in `assignment1.js`.

Differnet Ways to Run Tests

If you are going to run your tests over and over as you make changes to `src/solutions.js`, you can run the tests so they automatically watch for changes, and re-run whenever you save your file:

```
npm run test-watch
```

You can stop the tests from running using `CTRL + c`.

Next, you can run a single test instead of all tests:

```
npm test problem-00
```

This will only run the tests in `problem-00.test.js`, making it easier to work on only one problem at a time.

You can also watch a particular test, and re-run it when the code is saved:

```
npm run test-watch problem-00
```

Part 2: Implement missing code in src/analysis.js

After you have finished Part 1, you may begin working on the file `src/analyze.js`. This is a command-line JavaScript program that will use some of the functions you wrote in `src/cases.js`. Most of the program is written already, but it is missing some key features, and has a few bugs.

Look for all the TODO comments in the code and follow the instructions to get the program to work correctly. Your program should produce the following output when it is run.

To run the program and analyze all data, use the following command:

```
npm run analyze --
```

Here is how the output should look:

```
$ npm run analyze --

Loading data...
Analyzing data...
Using all case data

Summary
-------------------------
Unknown 33 (0.18%)
19 and younger 1,311 (7.34%)
20 to 29 Years 3,049 (17.06%)
30 to 39 Years 2,724 (15.24%)
40 to 49 Years 2,416 (13.52%)
50 to 59 Years 2,720 (15.22%)
60 to 69 Years 1,813 (10.14%)
70 to 79 Years 1,133 (6.34%)
80 to 89 Years 1,557 (8.71%)
90 and older 1,116 (6.24%)
-------------------------
```

To run the program and analyze a neighbourhood (e.g., Niagara), use the following command:

```
npm run analyze -- --name=”Niagara”
```

Here is how the output should look:

```
$ npm run analyze -- --neighbourhood="Niagara"

Loading data...
Analyzing data...
Using Neighbourhood='Niagara' with a fuzzy match to limit the data

Summary
-------------------------
Unknown 0 (0%)
19 and younger 3 (2.42%)
20 to 29 Years 47 (37.90%)
30 to 39 Years 36 (29.03%)
40 to 49 Years 13 (10.48%)
50 to 59 Years 14 (11.29%)
60 to 69 Years 8 (6.45%)
70 to 79 Years 3 (2.42%)
80 to 89 Years 0 (0%)
90 and older 0 (0%)
-------------------------
```

To run the program and analyze an FSA (M6K), use the following command:

```
npm run analyze -- --fsa=”M6K”
```

Here is how the output should look:

```
$ npm run analyze -- --fsa="M6K"

Loading data...
Analyzing data...
Using FSA='M6K' to limit the data

Summary
-------------------------
Unknown 0 (0%)
19 and younger 5 (1.59%)
20 to 29 Years 45 (14.29%)
30 to 39 Years 46 (14.60%)
40 to 49 Years 31 (9.84%)
50 to 59 Years 38 (12.06%)
60 to 69 Years 39 (12.38%)
70 to 79 Years 39 (12.38%)
80 to 89 Years 43 (13.65%)
90 and older 29 (9.21%)
-------------------------
```

Learn how to Lint your Code

In addition to running unit tests, you can also run a linter called `eslint`. Linting helps to find and remove common errors in code, for example, missing a semi-colon, or forgetting to declare a variable.

To run eslint, use the npm command:

```
npm run eslint
```

If there are no problems, there will be no output. If there is any output, pay attention to what it says, so you can make corrections. For example:

```
assignment1/assignment1.js
  18:9  error  'x' is defined but never used  no-unused-vars
```

Here, we see a lint error, which has various information:

1. The filename is listed first, `assignment1/assignment1.js`
2. The line number is listed next: 18
3. The column number on line 18 is listed next: 9
4. The actual error or warning comes next: `error 'x' is defined but never used`
5. The rule name comes last: `no-unused-vars`. You can lookup how to fix these errors using the rule name, for example: https://eslint.org/docs/rules/no-unused-vars

Your code should have no lint errors when you submit it.

Learn how to Properly Format your Code

Source code needs to be properly structured, use consistent indenting, semi-colons, etc. Doing so makes it easier to understand, read, and debug your code.

Your code must be properly and consistently formatted. You can do it by hand, or, you can use Prettier (https://prettier.io/) to do it automatically.

There are two ways to use Prettier. First, using the npm command:

```
npm run prettier
```

This will rewrite your files to use proper formatting. NOTE: running this command will overwrite your file, so make sure you have saved your work before you run it.

The second way to run Prettier is using the Prettier extension, and format-on-save. If you install the recommended extensions and settings for this project, saving your file will result in Prettier automatically fixing your code for you.

Debugging Code and Tests

You can also use VSCode's built in debugging tools to run and debug your code, or the test code, within the editor, step through code, inspect variables, examine call stacks, etc. See the instructions at: https://github.com/microsoft/vscode-recipes/tree/master/debugging-jest-tests#debugging-all-tests

Submitting your Assignment

When you have completed your assignment, you need to prepare your submission. To do so, use the npm command:

```
npm run prepare-submission
```

This will do a number of things automatically for you:

- create a `submission/` directory, deleting an existing one if present
- run `prettier` on the source
- copy all files under `src/` to `submission/src`
- copy `package.json` to `submission/package.json`
- run `eslint` and write the output to `submission/eslint.log`
- run `npm test` and write the output to `submission/test.log`
- zip the `submission/*` directory to `submission.zip`

You can upload `submission.zip` to Blackboard for submission.