WEB222 Assignment 1 Instructions

Introduction

This assignment will help you learn and practice JavaScript syntax, and use built-in and user-defined functions. It will also help you set up your web development environment and learn some industry standard tooling for JavaScript.

You must do this and all future assignments on your own. You should **not** work in partners or groups, 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.

If you get stuck, please speak with your professor. Do not copy other student work or code from the web. It is important that you learn and understand the concepts in the assignment.

NOTE: you can also watch a video on YouTube that shows how to setup and work on this assignment, see:

https://www.youtube.com/watch?v=8RuLrmVDOw8

Submission

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

Setup

This assignment uses a technique called "Unit Testing," and relies on a number of dependencies, which must be installed on your computer. A dependency is a library, tool, or other piece of software that is needed in order to build, test, and run another piece of software.

JavaScript and web projects often rely on hundreds (or thousands) of dependencies, and getting used to installing and using them is an important step. Part of the philosophy of JavaScript is to keep the language small and rely on third-partys to extend it.

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 16.15.0, although any 16.x.x version should work. LTS versions are supported for a long time.

Install Dependencies

Open a command line terminal (e.g., cmd.exe or bash) 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 `src/` directory, `package.json` file, etc.

In this root 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 professor for help.

Install a Proper Editor

You will need a proper code editor to develop for the web. The most popular choice is Microsoft Visual Studio Code (aka, VSCode). VSCode runs on Windows, Linux, and macOS (fun fact: VSCode is actually written in HTML, CSS, and JavaScript. It's a web page you can also run in the browser, see https://vscode.dev/) . You can download and install it for free from:

https://code.visualstudio.com/

NOTE: VSCode is not the same as Visual Studio.

A number of extensions are available that will work automatically with the scripts and configuration in this project, and are recommended:

- eslint https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint
- prettier https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode

When you open this project's folder, VSCode should offer to **install these recommended extensions**. Once these are installed VSCode will automatically show you errors as you type, automatically format your code when you save, and show you spelling mistakes in your code comments and names.

If you want to use a different editor, that's fine. Confirm with your professor that it will work for our needs. You may not use Notepad, for example.

Learn how to Navigate the Code

The assignment has a mix of project configuration and source code. All of the config files are in the project root, and the source code is contained in the `src` directory (this is a common pattern). Your code goes in the `src/solution.js` file. The other files in this directory are unit tests, which you should read, but don't need to modify.

Learn how to Run the Assignment Tests

You are asked to complete the code in the file `src/solutions.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. A Unit Test tests the smallest possible unit of a software system or component, and we write many small Unit Tests to prove that our implementation works as we expect.

For example, we expect `2 + 2` to return `4`, not `5`. We can write a unit test for this to confirm that our code does what we expect. See http://softwaretestingfundamentals.com/unit-testing/ for more info about unit tests.

These tests will help you determine if your code is working correctly: running the tests should produce the output that the tests expect, and the 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
```

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

Your assignment is complete when all tests are passing, and there are no more linting errors. 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.

Discussion of Other Assignment Files

You may be wondering about some of the other files in this project. While you don't need to modify them, or use them directly, here is what each one does:

node_modules/ - the installed dependencies necessary to run prettier, eslint, etc., installed when your run `npm install`.

.eslintrc.js - configuration for eslint, see https://eslint.org/docs/user-guide/configuring

.prettierrc.js - configuration settings for prettier, see https://prettier.io/docs/en/configuration.html

package.json - node.js package info, dependencies, build scripts, see https://docs.npmjs.com/files/package.json

package-lock.json - a generated file with dependency version information, see https://docs.npmjs.com/files/package-lock.json