CS142 Project 2: JavaScript Calisthenics

Due: Thursday, April 14, 2022 at 11:59 PM

Setup

Although this project has you run code in your browser, you need to have Node.js installed on your system to run the code quality checker. If you haven't already installed Node.js and the npm package manager, follow the installation instructions now.

Once you have Node.js installed, create a directory `project2` and extract the contents of this zip file into the directory. The zip file contains the files `cs142-test-project2.html` and `cs142-test-project2.js` that act as a testing framework for the code you write in this project.

You can fetch the code quality tool (also known as a "linter"), ESLint, by running the following command in the `project2` directory:

```
npm install
```

This will fetch ESLint into the `node_modules` subdirectory. You will be able to run it on all the JavaScript files in `project2` directory by running the command:

```
npm run lint
```

The code you submit should start with `'use strict';` in each JavaScript file and running ESLint using the command above should not output any errors or warnings. **Any errors or warnings will be used to deduct style points.**

To run the assignment, you have two options:

1. **In your browser**: Open the file `cs142-test-project2.html` in your browser. The web page will load the JavaScript code you have written and run a few tests.
2. **In Node.js**: You can also run the tests without a browser using the command
   ```
   npm test
   ```

We recommend using the browser for development as the debugging environment is much better. However, we will use Node.js for grading purposes.

Note: These tests don't cover all the edge cases. They are there to help guide you and let you know when you have the basic functionality. It is your responsiblity to handle everything stated in the following specs that aren't explicitly tested in the test file we give you.

In this project we ask you to write or modify some JavaScript functions. The problems in this assignment are of a practical nature and functionality you develop will be useful in completing later class projects. Given the availability of JavaScript libraries to solve or help solve pretty much any JavaScript tasks you would be assigned, it is likely you could solve these with a couple of lines to call some library routine. Since the goal of the project is to learn JavaScript, **we forbid you to use any JavaScript libraries in your solutions.** Functions built-in to JavaScript, like Arrays and Date objects, are acceptable.

Problem 1: Generate a Multi Filter Function (10 points)

In your `project2` directory, make a new file named `cs142-make-multi-filter.js`.The code for your Multi Filter Function will go in this file.

Declare a global function named `cs142MakeMultiFilter` that takes an array (`originalArray`) as a parameter and returns a function that can be used to filter the elements of this array. The returned function (`arrayFilterer`) internally keeps track of a notion called `currentArray`. Initially, `currentArray` is set to be identical to `originalArray`. The `arrayFilterer` function takes two functions as parameters. They are:

1. `filterCriteria` - A function that takes an array element as a parameter and returns a boolean. This function is called on every element of `currentArray` and `currentArray` is updated to reflect the results of the `filterCriteria` function. If the `filterCriteria` function returns `false` for an element, that element should be removed from `currentArray`. Otherwise, it is left in `currentArray`. If `filterCriteria` is not a function, the returned function (`arrayFilterer`) should immediately return the value of `currentArray` with no filtering performed.
2. `callback` - A function that will be called when the filtering is done. `callback` takes the value of `currentArray` as an argument. Accessing `this` inside the `callback` function should reference the value of `originalArray`. If callback is not a function, it should be ignored. `callback` does not have a return value.

The `arrayFilterer` function should return itself unless the `filterCriteria` parameter is not specified in which case it should return the `currentArray`. It must be possible to have multiple `arrayFilterer` functions operating at the same time.

The following code shows how one might make use of the functions you define in this problem:

```
// Invoking cs142MakeMultiFilter() with originalArray = [1, 2, 3] returns a
// function, saved in the variable arrayFilterer1, that can be used to
// repeatedly filter the input array
var arrayFilterer1 = cs142MakeMultiFilter([1, 2, 3]);

// Call arrayFilterer1 (with a callback function) to filter out all the numbers
// not equal to 2.
arrayFilterer1(function (elem) {
  return elem !== 2; // check if element is not equal to 2
}, function (currentArray) {
  // 'this' within the callback function should refer to originalArray which is [1, 2, 3]
  console.log(this); // prints [1, 2, 3]
  console.log(currentArray); // prints [1, 3]
});

// Call arrayFilterer1 (without a callback function) to filter out all the
// elements not equal to 3.
arrayFilterer1(function (elem) {
  return elem !== 3; // check if element is not equal to 3
});

// Calling arrayFilterer1 with no filterCriteria should return the currentArray.
var currentArray = arrayFilterer1();
console.log('currentArray', currentArray); // prints [1] since we filtered out 2 and 3

// Since arrayFilterer returns itself, calls can be chained
function filterTwos(elem) { return elem !== 2; }
function filterThrees(elem) { return elem !== 3; }
var arrayFilterer2 = cs142MakeMultiFilter([1, 2, 3]);
var currentArray2 = arrayFilterer2(filterTwos)(filterThrees)();
console.log('currentArray2', currentArray2); // prints [1] since we filtered out 2 and 3

// Multiple active filters at the same time
var arrayFilterer3 = cs142MakeMultiFilter([1, 2, 3]);
var arrayFilterer4 = cs142MakeMultiFilter([4, 5, 6]);
console.log(arrayFilterer3(filterTwos)()); // prints [1, 3]
console.log(arrayFilterer4(filterThrees)()); // prints [4, 5, 6]
```

Problem 2: Template Processor (5 points)

In your `project2` directory, make a new file named `cs142-template-processor.js`. The code for your Template Processor will go in this file.

Create a template processor class (`Cs142TemplateProcessor`) that is constructed with a string parameter `template` and has a method `fillIn`. When invoked with an argument of a `dictionary` object, `fillIn` returns a string with the template filled in with values from the dictionary object. `Cs142TemplateProcessor` should be written using the standard JavaScript constructor and prototype structure.

The `fillIn` method returns the `template` string with any text of the form `{{property}}` replaced with the corresponding `property` of the `dictionary` object passed to the function.

If the template specifies a property that is not defined in the dictionary object, the property should be replaced with an empty string. If the property is between two words, you'll notice that replacing the property with an empty string will result in two consecutive whitespaces. Example: `"This {{undefinedProperty}} is cool"` → `"This  is cool"`. This is fine. You do not have to worry about getting rid of the extra whitespace.

Your system need only handle properly formatted properties. Its behavior can be left undefined in the following cases as we will not be checking explicitly for them.

- nested properties - `{{foo {{bar}}}}` or `{{{{bar}}}}` or `{{{bar}}}`
- unbalanced brackets - `{{bar}}}`
- stray brackets in any property string - `da{y` or `da}y`

The following code shows how one might make use of the functions you define in this problem:

```
var template = 'My favorite month is {{month}} but not the day {{day}} or the year {{year}}';
var dateTemplate = new Cs142TemplateProcessor(template);

var dictionary = {month: 'July', day: '1', year: '2016'};
var str = dateTemplate.fillIn(dictionary);

assert(str === 'My favorite month is July but not the day 1 or the year 2016');

//Case: property doesn't exist in dictionary
var dictionary2 = {day: '1', year: '2016'};
var str = dateTemplate.fillIn(dictionary2);

assert(str === 'My favorite month is  but not the day 1 or the year 2016');
```

Problem 3: Fix `cs142-test-project2.js` to not pollute the global namespace (5 points)

The test JavaScript file we give you (`cs142-test-project2.js`) declares numerous symbols in the global JavaScript namespace. For example, after the script is loaded the symbol `p1Message` appears in the global namespace. Another JavaScript file would then be able to access and change `p1Message`. Change `cs142-test-project2.js` to use the standard JavaScript module pattern using an **anonymous function** to hide symbols in the global namespace yet keep the same checking functionality.

Style Points (5 points)

These points will be awarded if your JavaScript code for the problems above is clean, readable, and ESLint warning-free.

Useful Hints

- In JavaScript, the convention for checking equality/inequality is `===` and `!==`, instead of `==` and `!=`.
- When running ESLint, make sure the `.eslintrc.json` (hidden file) is present, or ESLint will throw an error. You can use the command `ls -a` (`dir /a` in a Windows command line) to view all the files in your current directory, including hidden files.
- If ESLint output shows a lot of style issues, you can ask ESLint to fix many of them for you by running `npm run lint -- --fix`.
- When moving files around, make sure to do that using the command line. Dragging/dropping files usually does not move hidden files which is a common reason for students missing their `.eslintrc.json` file.
- The requirement that your solution to Problem 2 support multiple simultaneous uses of `cs142MakeMultiFilter` means that you should not be using global variables to store the `currentArray` state.
- In problem 3, you only need to address symbols created by the test file. Also, you will need to leave the variable `cs142Project2Results` as a global (i.e., property on the `window` object) so that you can continue to run the tests using the `run-tests-using-node.js` script.

Deliverables

Use the standard class submission mechanism to submit the `project2` directory. This directory should include the starter files we gave you along with code files you created or modified for the problems including `cs142-make-multi-filter.js`, `cs142-template-processor.js`, `cs142-test-project2.js`, and `cs142-test-project2.html`. Make sure your submission runs `npm run lint` and `npm test` with no errors or warnings.