Download lab5.zip, extract it to lab5 folder

1. In your lab5 directory, make a new file named make-multi-filter.js.The code for your Multi Filter Function will go in this file.

Declare a global function named MakeMultiFilter 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:

* 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.
* 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.

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 MakeMultiFilter() 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 = MakeMultiFilter([1,2,3]);

// call arrayFilterer1 (with 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) {

console.log(this); // printing 'this' within the callback function should print originalArray which is [1,2,3]

console.log(currentArray); // prints [1, 3]

});

// call arrayFilterer1 (without 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 = MakeMultiFilter([1,2,3]);

var arrayFilterer4 = MakeMultiFilter([4,5,6]);

console.log(arrayFilterer3(filterTwos)()); // prints [1,3]

console.log(arrayFilterer4(filterThrees)()); // prints [4,5,6]

2.

In your lab5 directory, make a new file named template-processor.js. The code for your Template Processor will go in this file.

Create a template processor class (TemplateProcessor) 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. TemplateProcessor 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 TemplateProcessor(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');